НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Кафедра комп'ютерних інтелектуальних систем та мереж

САЛАГОР Дмитро Вікторович

ДИПЛОМНА РОБОТА МАГІСТРА

ДОСЛІДЖЕННЯ ІНТЕЛЕКТУАЛЬНОЇ МОДЕЛІ КЛАСИФІКАЦІЇ ТЕКСТІВ НА ОСНОВІ АНАЛІЗУ КЛЮЧОВИХ СЛІВ

Спеціальність 123 – Комп'ютерна інженерія
Спеціалізація – Комп'ютерні системи та мережі

Керівник: Тішин Петро Метталінович,
кандидат технічних наук, доцент

Одеса - 2021

АНОТАЦІЯ

Салагор Д.В. Дослідження інтелектуальної моделі класифікації текстів на основі аналізу ключових слів – Магістерська дипломна робота. Одеса, 2021: 82 с., 30 рис., 14 табл., 1 додаток, 29 джерел.

Об'єкт дослідження – процеси классифікації текстової інформації.

Предмет дослідження – методи класифікації бінарного та багатокласового типу.

Мета роботи – розробка і реалізація класифікатора тексту на основі нечітких множин, який зможе визначити предметну область за допомогою ключових слів з точністю вище 55%.

В роботі проведено дослідження двох алгоритмів класифікації, які засновані на нечітких множинах. Алгоритми були реалізовані на мові програмування Java і протестовані. Отримано результати точності 79% для багатокласової класифікації та 75% для бінарної класифікації. Також проведено огляд нечітких множин. Проведено порівняння з іншими популярними алгоритмами (метод опорних векторів, нейронні мережі, дерева рішень). Популярні алгоритми були реалізовані в середовищі KNIME. В результаті порівняння алгоритми, розроблені автором роботи, дали найбільші значення точності. Розроблені алгоритми пропонується використовувати в базах даних для автоматичної класифікації текстів або в поєднанні з іншими алгоритмами розпізнавання.

НЕЧІТКІ МНОЖИНИ, КЛАСИФІКАЦІЯ, МЕТОД ОПОРНИХ ВЕКТОРІВ, ДЕРЕВА РІШЕНЬ, НЕЙРОННІ МЕРЕЖІ.

АННОТАЦИЯ

Салагор Д. В. Исследование интеллектуальной модели классификации текстов на основе анализа ключевых слов – Магистерская дипломная работа. Одесса, 2021: 82 с., 30 рис., 14 табл., 1 приложение, 29 источников.

Объект исследования – процессы классификации текстовой информации.

Предмет исследования – методы классификации бинарного и многоклассового типа.

Цель работы – разработка и реализация классификатора текста на основе нечетких множеств, который сможет определить предметную область с помощью ключевых слов с точностью выше 55%.

В работе проведено исследование двух алгоритмов классификации, которые основаны на нечетких множествах. Алгоритмы были реализованы на языке программирования Java и протестированы. Получены результаты точности 79% для многоклассовой классификации и 75% для бинарной классификации. Также проведен обзор нечетких множеств. Проведено сравнение с другими популярными алгоритмами (метод опорных векторов, нейронные сети, деревья решений). Популярные алгоритмы были реализованы в среде KNIME. В результате сравнения алгоритмы, разработанные автором работы, дали наибольшие значения точности. Разработанные алгоритмы предлагается использовать в базах данных для автоматической классификации текстов или в сочетании с другими алгоритмами распознавания.

НЕЧЕТКИЕ МНОЖЕСТВА, КЛАССИФИКАЦИЯ, МЕТОД ОПОРНЫХ ВЕКТОРОВ, ДЕРЕВЬЯ РЕШЕНИЙ, НЕЙРОННЫЕ СЕТИ.

ABSTRACT

Salahor D. V. Research of an intellectual model of text classification based on keyword analysis – Master's work. Odessa, 2021: 82 p., 30 fig., 14 tables., 1 appendix, 29 sources.

The object of research is the processes of classification of text information.

The subject of the research is binary and multiclass classification methods.

The aim of the work is to develop and implement a text classifier based on fuzzy sets, which will be able to determine the subject area using keywords with an accuracy of more than 55%.

The paper studies two classification algorithms that are based on fuzzy sets. The algorithms were implemented in the Java programming language and tested. Accuracy results of 79% for multiclass classification and 75% for binary classification were obtained. Fuzzy sets are also reviewed. A comparison is made with other popular algorithms (the method of reference vectors, neural networks, decision trees). Popular algorithms were implemented in the KNIME environment. As a result of the comparison, the algorithms developed by the author of the work gave the highest accuracy values. The developed algorithms are proposed to be used in databases for automatic classification of texts or in combination with other speech recognition algorithms.

FUZZY SETS, CLASSIFICATION, SUPPORT VECTOR MACHINES, DECISION TREES, NEURAL NETWORKS.

ЗМІСТ

ВСТУП

Існує потреба угруповання даних за родами діяльності, в разі, коли потрібно провести порівняльний аналіз різних груп. Поділ на групи або класи є важливою проблемою в багатьох областях діяльності людини, таких як статистика, економіка, біологія, соціологія.

Класифікація даних вручну призводить до збільшення часу обробки, збільшення витрат і людських ресурсів. Очевидно, що як тільки промисловий і науковий розвиток дозволив, класифікація даних була автоматизована.

При класифікації за допомогою комп'ютерних алгоритмів процес відбувається на комп'ютерній мові, а саме на машинному коді. В результаті при такій класифікації часто повертається бінарний результат у вигляді нуля або одиниці. У такому вигляді результати не завжди задовольняють завданням класифікації, і можуть призводити до великої кількості помилок у разі аварії.

У реальному світі, особливо в дикій природі, існує еластичність, завдяки якій складові живої природи - біоми, види тварин, рослини взаємодіють один з одним. Сам процес еволюції, при якому біологічні види плавно переходять в нові, пристосовуючись до нових умов, показує, що при описі природних явищ зручніше використовувати не дискретні і бінарні величини, а дійсні і наближені.

Людина, як і інші тварини, в процесі еволюції розвинула навички угруповання і обробки наборів даних, що надходять від органів чуття для визначення умов, в яких вона знаходиться. Незалежно від того, наприклад, чи є рослина їстівною, або чи знаходиться особа в безпеці, людський мозок обробляє і класифікує всі дані. Також потрібно відзначити, що людина може обробляти багато даних одночасно і класифікувати їх не тільки як «чорне» і «біле», а може виконувати і багатокласову класифікацію.

Спостерігаючи за природою, можна відмітити і використовувати методи, які вже були перевірені природним відбором і характеризуються великою еластичністю і надійністю. Це вже було зроблено і вже існує безліч таких методів класифікації (дерева рішень [1, 2, 3, 4, 5], метод опорних векторів (support vector machines - SVM) [6, 7, 8, 9, 10], деякі методи також використовують структури на основі мозку у тварин і людей (нейронні мережі [11, 12, 13, 14]). Опис таких алгоритмів знаходиться розділі 2.

Також дуже популярні алгоритми на основі нечітких множин, які замість дискретних значень оперують дійсними і розмитими значеннями. Опис нечітких множин знаходиться в підрозділі 1.1.

Об’єкт роботи – процеси классифікації текстової інформації.

Предмет роботи – дані наукових робіт авторів Люблінської політехніки в період 1980-2020 років, а саме ключові слова та предметні галузі.

Мета роботи – розробка і реалізація класифікатора тексту на основі нечітких множин, який зможе визначити предметну область за допомогою ключових слів з точністю вище 55%.

Темою роботи не є огляд існуючих методів розпізнавання тексту, так само як і огляд нечітких множин.

Вхідні дані роботи – дані наукових робіт авторів Люблінської політехніки в період 1980-2020 років, а саме ключові слова та предметні галузі.

Вихідні дані роботи – предметні області, що відповідають науковим роботам згідно вхідним даним.

Вхідні дані отримано з бази даних наукових статей Scopus [15]. Кількість отриманих прикладів даних - 6543. Дані отримано, використовуючи наступне сортування:

- рік публікації наукової роботи від 1980 до 2020;
- структурна організація Люблінська політехніка;
- формат файлу - значення, розділені комами.

Дані отримані окремо для кожної з предметних областей, а потім об'єднані в один загальний файл формату значень, розділених комами. У цьому файлі дані впорядковані в наступному порядку:

- список авторів роботи (один або більше);
- заголовок роботи;
- посилання на роботу в базі даних Scopus;
- ключові слова, зазначені автором роботи;
- ключові слова, отримані за допомогою індексування базою даних Scopus;
- список предметних областей (одна або більше).

Вихідні дані є строкою, яка містить одну чи більше предметних областей зі списку, наведеного нижче.

Список всіх предметних областей англійською мовою:

- engineering;
- materials science;
- physics and astronomy;
- mathematics;
- computer science;
- environmental science;
- chemistry;
- chemical engineering;
- energy;
- agricultural and biological sciences;
- social sciences;
- earth and planetary sciences;
- biochemistry, genetics and molecular biology;
- medicine;
- economics, econometrics and finance;
- business, management and accounting;
- decision sciences;
- multidisciplinary;

- pharmacology, toxicology and pharmaceutics;
- arts and humanities;
- health professions;
- neuroscience;
- immunology and microbiology;
- psychology;
- dentistry;
- nursing;
- veterinary.

Ключові слова, отримані за допомогою індексування використовуються, щоб у разі потреби використовувати їх замість ключових слів автора. Список авторів роботи, заголовок і посилання на роботу в базі даних Scopus при роботі алгоритмів не використовуються.

В розділі 1 проведена аналітика та критичний огляд робіт з подібною тематикою та проблемами, також проведений огляд методів, які використовуються при класифікації. Розділ 2 містить алгоритми, які були створені для вирішення завдання магістерської роботи, а саме:алгоритми для теоретичного розв’язання; алгоритми для практичного розв’язання на основі частково готових рішень; алгоритми для практичного розв’язання, створені автором роботи; В розділі 3 приведений опис реалізації моделей і методів з розділу 2, опис перевірки та отримання результатів, приведені результати роботи моделей та методів.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SVM	Support Vector Machines, метод опорних векторів
BOW	Bag of words, «мішок слів»
Backpropagation	Метод зворотного поширення помилки

1 АНАЛІТИЧНИЙ ОГЛЯД НАУКОВИХ РОБІТ З ОБ'ЄКТУ ДОСЛІДЖЕННЯ

Проблема класифікації - широко обговорювана і розглянута проблема. Серед безлічі існуючих можна виділити модифікований підхід до використання нечітких множин [16] при визначенні настрою у текстах малого розміру. Описана проблема, яка часто з'являється при використанні нечітких множин, а саме при перекладі нечітких множин в звичайні множини. Пропонується використання додаткового правила, що визначає поріг переходу. N-грами [17] були використані для структуризації вхідних даних. Отримані за допомогою описаного підходу результати були порівняні з іншими методами розпізнавання (метод опорних векторів, наївного класифікатора Баєса, дерев рішень, градієнтно-посиленного дерева, глибоких нейронних мереж), а також з методом, що використовує нечіткі множини, але без правила перекладу в звичайні множини. Автор статті пропонує використовувати отримані результати в системах визначення випадків ненависті в Інтернеті. На думку автора дипломної роботи, тема статті заслуговує на увагу, а вибір N-грам добре показує себе при роботі з великими і середніми текстами.

Іншим цікавим підходом є алгоритм, показаний в роботі [18], що описує підхід до класифікації документів при використанні модифікованої версії BOW («мішок слів»). Під час зберігання вектора слів використовується нечіткий підхід. При визначенні приналежності використовується косинусна функція. При класифікації використані методи наївного Баєса і опорних векторів. Модель була реалізована на мові програмування Python. Результати експерименту показали, що даний підхід працює краще, ніж інші класифікатори в разі великої кількості неоднозначних тренувальних даних. Автор дипломної роботи вважає, що

модифікування алгоритму зберігання даних може відкрити нові можливості обробки і виправити якість класифікації.

Метод, що об'єднує ймовірнісну модель з нечіткими множинами була описана в [10]. Метод полягає в поділі всіх слів на важливі, звичайні і неважливі слова. Для кожного типу слів створюються свої нечіткі множини. Метод протестований на даних індексації сторінок в Інтернеті і був отриманий середній результат точності в 90%. Описаний метод може бути використаний для поліпшення індексації сторінок в Інтернеті. Автор дипломної роботи звертає увагу на цікаву ідею використання класифікації для індексування.

Підхід до класифікації даних з використанням зменшення кількості вимірів і нечітких множин запропонований в [19]. Кожну область, яку потрібно розпізнати, описано ознаками. Багато ознак можуть бути малозначними або зашумленими, тому зменшення їх кількості може призвести до підвищення якості розпізнавання і зменшення кількості обчислювальних процесів під час класифікації. Описано різні методи вибору ознак (feature selection), такі як наближені, наближені-нечіткі, і нечіткі-наближені, які використовуються для попередньої обробки даних. До вже оброблених даних застосовуються класифікатори дерев рішень. Отримані результати вказують на те, що в більшості випадків зменшення кількості ознак не призводить до зменшення точності класифікації. Автора дипломної роботи зацікавило застосування і порівняння декількох варіантів нечітких множин.

Для категоризації текстової інформації в [20] використано нечіткі множини. Тренування для введення текстової інформації використовує "мішок слів". Далі дані розподіляються за категоріями. Результатом статті є фреймворк. Фреймворк був протестований на безлічі даних про спорт, а саме на 737 статтях з 5 різних категорій. Результуюча точність коливається в межах від 78% до 100%. Даний фреймворк також може бути використаний під час обробки текстових даних в різних предметних областях. Автор статті планує подальший розвиток і поліпшення алгоритму. Автор дипломної роботи звертає увагу на відмінний вибір алгоритму.

Підхід до отримання ключових слів з текстових даних представлений в [21]. На початку, використовуючи метод максимальної ентропії, всі фрази виділяються і представляються у вигляді нечіткої множини. Результатом будуть найбільш значущі слова в множині. Підхід був протестований на вибірці з 30 статей преси BBC. Такий підхід дає можливість виділення ключових фраз з тексту способом, подібним до ручного виділення людиною. Автор статті планує поліпшити даний підхід для використання в більш загальних алгоритмах, наприклад при виділенні речень. Автор дипломної роботи впевнений, що такий підхід ідеальний для виправлення або перевірки ключових слів після людини, або ж для автоматичного генерування ключових слів в статтях.

Підхід до визначення настрою в коротких текстових даних за допомогою нечітких множин описаний в [22]. Для експерименту використані дані з трьох різних джерел, а саме публічного мікроблогу, мобільного форуму та форуму авіаперельотів. Кожне з джерел має свою структуру і наповнення. Експеримент підтвердив, що отримана модель має більш високу точність, ніж поєднання методу опорних векторів і методу максимальної ентропії або поєднання методу опорних векторів і методу хі-квадрат. Запропонована модель добре працює з короткими текстами, визначаючи загальний настрій тексту. Автор дипломної роботи вважає, що використання декількох різних джерел це відмінний спосіб для збільшення узагальненості описаної в статті моделі.

Застосування різних методів, в тому числі і нечітких множин, для боротьби з тероризмом, описано в [23]. Використано методи підсумовування методів, бінарний метод, а також метод, що використовує нечіткі множини для отримання інформації з тексту. Для класифікації використані методи наївного Баєса і опорних векторів. В якості збору даних використані збірник прикладів CTE (боротьба з тероризмом і екстремізмом) і ASB (антисоціальна поведінка). Експеримент показав, що метод, що використовує нечіткі множини і метод опорних векторів дав найкращий результат. Автор дипломної роботи переконаний, що така висока точність отримана завдяки використанню нечітких множин.

1.1 Характеристика нечітких множин

Звичайні множини дозволяють описати елементи, що належать до певного класу. Однак в реальних випадках часто неможливо однозначно класифікувати елемент. У 1965 р. була опублікована стаття [24] за авторством Л. Задеха, в якій пропонується новий тип множин – нечіткі множини.

Нечітка множина може бути представлена як звичайна множина *A*, до кожного елементу якої застосована функція приналежності *f(x)* [24].

Функція приналежності *f (x)*, як показує назва, показує ступінь приналежності до множини або деякого класу в межах [0, 1], причому:

- якщо *f (x)* = 0, то *x* не належить множині *A*;
- якщо *f (x)* = 1, то *x* однозначно належить множині *A*;
- якщо *f (x)* = 1 для кожного *x* в *A*, то нечітка множина *A* вироджується в звичайну;
- якщо *f (x)* = 0 для кожного *x* в *A*, то нечітка множина *A* є порожньою.

Якщо функція приналежності приймає тільки два значення «0» і «1», тоді множину пропонується називати звичайною або простою [24]. Автор також звертає увагу на факт, що незважаючи на схожість між функцією приналежності (при кінцевому числі елементів) і функцією густини приналежності (при нескінченному числі елементів), нечіткі множини не є статистичними.

У [24] автор розширює операції на звичайних множинах для нечітких множин. В операціях для нечітких множин використовуються не елементи, а значення їх функцій приналежності для результуючої нечіткої множини. Після знаходження функції приналежності всі елементи з *f(x)* = 0 виключаються з множини. У [24] були визначені наступні операції:

- сума;
- перетин;
- доповнення;
- підмножина.

Сума *C* нечітких множин *A* і *B* визначається як (1.1):

$$f_C(x) = Max[\, f_A(x), f_B(x)] \qquad (1.1)$$

Рисунок 1.1 представляє операцію в графічному вигляді.

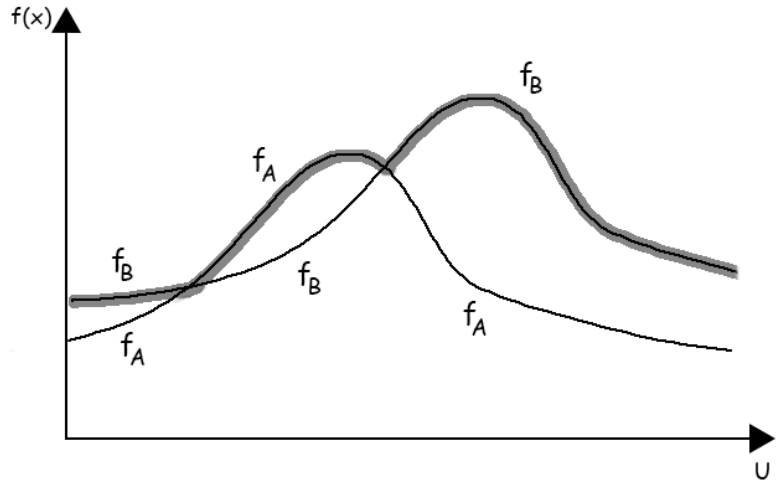

Рисунок 1.1 – Операція суми в нечітких множинах

Рисунок 1.1 містить: на осі X лежать елементи, що містяться в множинах *A* і *B*. Ось Y показує значення функції приналежності. *f(x)* є значеннями функції приналежності для елементів множин *A* і *B*. Товста сіра лінія на малюнку показує значення функції приналежності для операції суми між множинами *A* і *B*.

У разі операції перетину функція приналежності визначається наступним способом (1.2):

$$f_C(x) = Min[\, f_A(x), f_B(x)] \qquad (1.2)$$

Рисунок 1.12 представляє операцію в графічному вигляді.

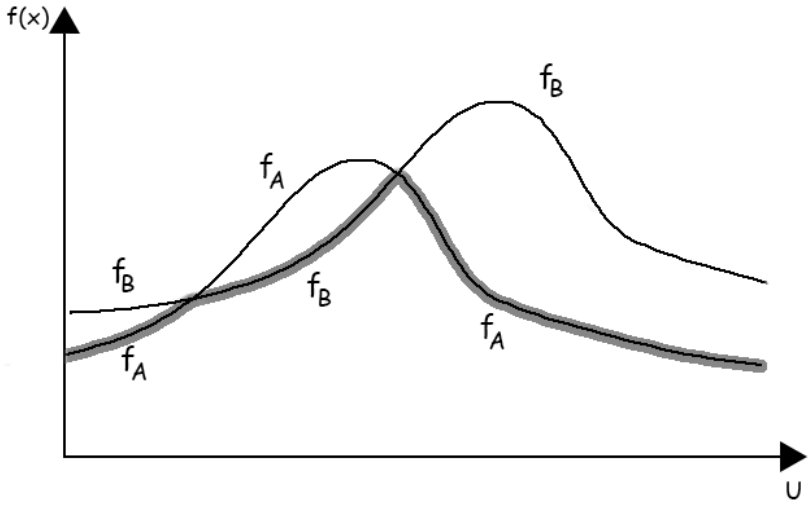

Рисунок 1.2 – Операція перетину в нечітких множинах

Рисунок 1.2 містить: на осі X лежать елементи, що містяться в множинах *A* і *B*. Ось Y показує значення функції приналежності. *f(x)* є значеннями функції приналежності для елементів множин *A* і *B*. Товста сіра лінія на малюнку показує значення функції приналежності для операції перетину між множинами *A* і *B*.

У разі операції доповнення функція приналежності визначається наступним способом (1.3):

$$f_{C'}(x) = 1 - f_A(x) \qquad (1.3)$$

Нечітка множина A є підмножиною B, коли функція приналежності *A* для кожного *x* менша або дорівнює функції приналежності *B* для кожного *x* (1.4):

$$A \subset B \Leftrightarrow f_A \leqq f_B \qquad (1.4)$$

Для операцій, вказаних вище, так, як і для звичайних множин, можна використовувати наступні правила:

1) Дистрибутивність

(1.5):

$$C \cap (A \cup B) = (C \cap A) \cup (C \cap B)$$
$$C \cup (A \cap B) = (C \cup A) \cap (C \cup B) \qquad (1.5)$$

2) Закон Де Моргана

(1.6):

$$(A \cup B)' = A' \cap B'$$
$$(A \cap B)' = A' \cup B' \qquad (1.6)$$

Зараз нечіткі множини широко використовуються. Застосування нечітких множин дає перевагу, наприклад, при класифікації. Автор [30] пропонує використання нечітких множин як спосіб опису відносин. У [25], нечіткі множини використовуються для класифікації зображень. Також нечіткі множини часто використовуються у поєднанні з алгоритмами класифікації для отримання більш детальних результатів [18, 23].

1.2 Дерева рішень

Дерево рішень має деревоподібну структуру, в якій з основного стовбура ростуть гілки з вузлами, що закінчуються листям. У цьому випадку вузли є умовними операторами, гілки - переходами між ними, а листя - призначенням певного класу. У дереві рішень, в залежності від вихідних даних, виконуються умови, і в кінці роботи алгоритму буде призначений певний клас. Приклад простого дерева рішень представлений на рис. 1.3.

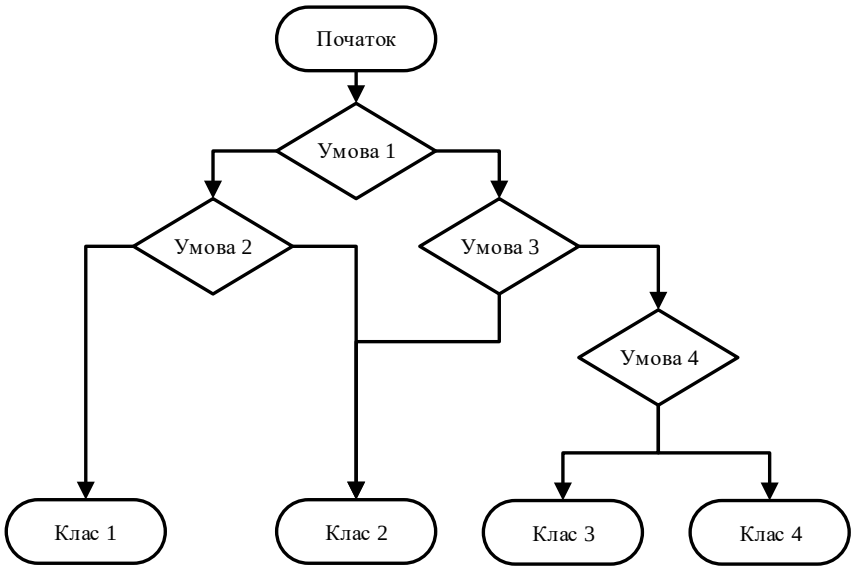

Рисунок 1.3 – Приклад дерева рішень

Існує два основних типи дерев рішень. Першим типом (який і був використаний в роботі) є класифікаційне дерево рішень. Класифікаційне дерево рішень повертає певний клас. Другий тип, дерево регресії, повертає в якості результату дійсне число (ціна, ймовірність, кількість) [1].

Створюючи дерево рішень для конкретних класифікацій, э можливість автоматизувати цей процес. Для цього існують різні алгоритми. Завдяки можливості створення великих умовних структур в деревах рішень можна реалізувати недвійкову класифікацію. Крім того, можна використати оператор "switch" замість умовного оператора "if" і тим самим зменшити розмір дерева.

Дерева рішень забезпечують описові та точні результати, але можуть не мати оптимізованої структури.

1.3 Метод опорних векторів

Метод опорних векторів опирається на модель машинного навчання. Принцип роботи моделі полягає в розміщенні навчальних зразків на площині і їх поділі для максимального зазору. Метод опорних векторів є двійковим класифікатором. Існують лінійні і нелінійні класифікатори. На малюнку 1.4 показана модель, що використовує метод опорних векторів, де «А3» не є рішенням, «А2» є рішенням, але неоптимальним через невелику відстань між елементами групи і лінією, що розділяє групи. Рішенням є «А1».

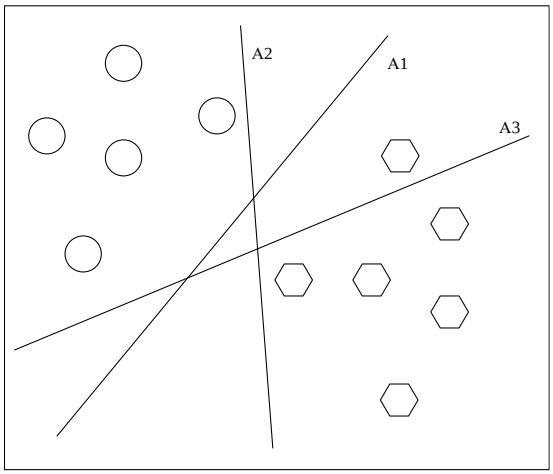

Рисунок 1.4 – Приклад класифікації методом опорних векторів

1.4 Нейронні мережі

Нейронна мережа - це мережа, яка складається з нейронів, що розміщаються шарами. Перший шар пов'язаний з входами мережі, останній-з виходами. Шари послідовно з'єднані між собою. При навчанні мережі використовується метод зворотного поширення помилки (backpropagation), який полягає в зворотній передачі вихідних даних на входи з корекцією ваг в зв’язках між нейронами. Якщо результат на виході нейронної мережі не коректний, то ваги в зв’язках, що впливали на цей результат зменшуються, в іншому випадку збільшуються. Цей

метод використовується в машинному навчанні для мінімізації помилок. На рисунку 1.5 показана схема нейронної мережі і спосіб роботи методу зворотного поширення помилки. Якщо на виході «y1» з'являється незадовільна відповідь, то ваги зв’язків буде зменшена для всіх з'єднань, які брали участь в отриманні такого результату.

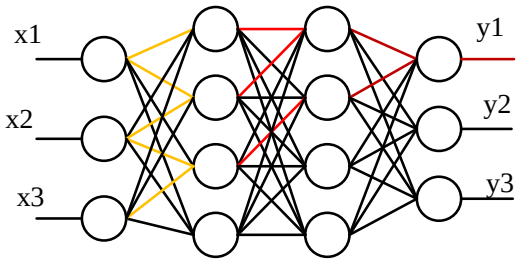

Рисунок 1.5 – Приклад простої нейронної мережі

Нейронні мережі широко використовуються у випадках, коли потрібна класифікація, розпізнавання чи автоматизація рішень.

1.5 Висновки до розділу

Алгоритми класифікації постійно удосконалюються, поєднуються одне з одним, використовуються в різних галузях та є актуальними в різних країнах та регіонах світу. Найвідоміші методи класифікації – це дерева рішень, нейронні мережі та метод опорних векторів. Ці методи часто використовуються для класифікації чи для оцінки роботи нових чи вже існуючих методів. В розділі був проведений огляд та аналітика стану знань на тему класифікації текстової інформації. Тема роботи є актуальною та вартою дослідження.

2 ТЕОРЕТИЧНІ ДОСЛІДЖЕННЯ МЕТОДІВ КЛАСИФІКАЦІЇ

В алгоритмах використовуються наступні позначення і скорочення:

- N - ціле число, що позначає кількість предметних областей;

- M - ціле число, що позначає кількість всіх записів, які передаються на вхід алгоритму;

- ключове слово - рядок, що зберігає одне слово (для модифікованої версії алгоритму) або кілька слів (для звичайної версії алгоритму).

У підрозділах 2.1 - 2.9 описано алгоритми та їх реалізацію з використанням частково готових рішень. У підрозділах 2.10 і 2.11 описано алгоритми та їх реалізацію, розроблені автором роботи.

Підрозділ 2.1 описує алгоритм попередньої обробки вхідних даних для отримання першого ключового слова, трьох перших, трьох останніх, трьох випадкових і всіх ключових слів.

У підрозділах 2.1 - 2.9 в якості частково готового рішення використано платформу аналізу даних KNIME [26]. KNIME це відкрита, безкоштовна платформа, заснована на середовищі розробки Eclipse. KNIME дозволяє використовувати дані з різних джерел, таких як бази даних (Oracle, mysql) або файли (Excel, значення, розділені комами). Отримані дані можна сортувати, модифікувати, додавати нові дані (у тому числі згенеровані). За допомогою елементів візуального програмування є можливість працювати з даними в стилі запитів SQL. Також можливо ручна обробка даних за допомогою скриптів, написаних на мовах програмування Java, Python, Javascript і тд. Платформа дозволяє генерувати вихідні дані у вигляді графіків, таблиць, файлів і тд.

У підрозділах 2.10 і 2.11 алгоритми були реалізовані на мові Java. Для порівняння реалізації мовою Java і реалізацію в KNIME, була використана версія

алгоритму, яка виконує бінарну класифікацію тільки для однієї конкретної предметної області.

2.1 Алгоритм попередньої обробки

Алгоритм використовується для підготовки вхідних даних до роботи з різними типами класифікаторів і множинами ключових слів в KNIME. Вхідні дані для алгоритму обробки даних представлений у вигляді текстового файлу у форматі значень, розділених комами. Кожен рядок у файлі має:

- список авторів роботи;
- заголовок роботи;
- посилання на роботу в базі даних Scopus;
- ключові слова, зазначені автором роботи;
- ключові слова, отримані за допомогою індексування роботи;
- список предметних областей.

Вихідні дані для алгоритму це таблиця з наступними колонками:

- список предметних областей;
- список ключових слів.

Список авторів роботи, заголовок роботи і посилання на роботу в базі даних не використовується в роботі алгоритму. Алгоритм попередньої обробки приведений в табл. 2.1:

Таблиця 2.1 – Алгоритм попередньої обробки даних

Алгоритм попередньої обробки даних:
Читання файлу в таблицю даних *table* Видалення колонки *Authors* Видалення колонки *Title* Видалення колонки *Link* Видалення першої колонки с назвами колонок Створення нової колонки *KeyWords* в *table* **for** кожна строка **in** *table*:   **if** *AuthorKeywords* і *IndexKeywords* не пусті: **if** *AuthorKeywords* дорівнює *IndexKeywords* :   *KeyWords = AuthorKeywords*   **else**

Продовження таблиці 2.1

```
        KeyWords = AuthorKeywords + '; ' + IndexKeywords
    end
  else if AuthorKeywords не пуста:
    KeyWords = AuthorKeywords
  else
    KeyWords = IndexKeywords
  end
  if KeyWords пустая:
    Видалення строки в table
  end
end
Видалення колонки AuthorKeywords з table
Видалення колонки IndexKeywords з table
```

На рисунок 2.1 представлений алгоритм попередньої обробки даних.

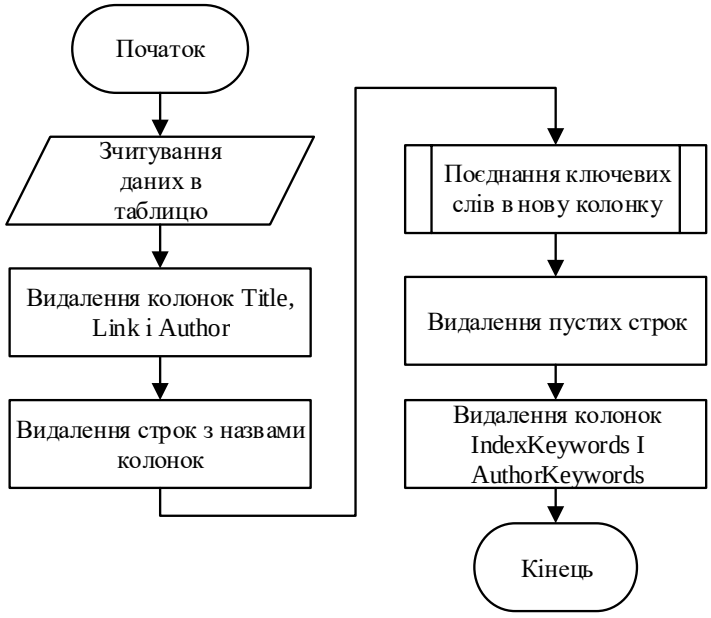

Рисунок 2.1 – Алгоритм попередньої обробки даних

Алгоритм був реалізований з-за допомогою наступних елементів: File Reader для читання даних, Column Filter для видалення колонок, Java Snippet для об'єднання ключових слів, Row filter і Rule-based Row Filter для видалення окремих строк строк. Реалізація алгоритму в KNIME представлено на рисунок 2.2.

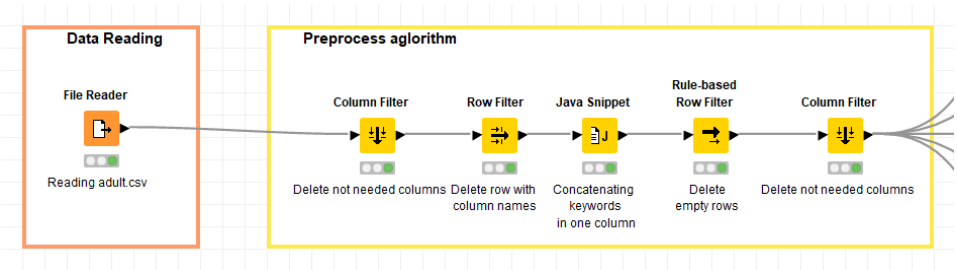

Рисунок 2.2 – Реализация алгоритма предварительной обработки данных в KNIME

2.2 Алгоритм обробки даних для отримання першого ключового слова

Алгоритм служить для підготовки до роботи з класифікаторами, які використовують одне ключове слово. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- список ключових слів;
- список предметних областей.

Вихідні дані це таблиця з наступними колонками:

- ключове слово;
- значення належності до предметної області "Engineering".

Алгоритм обробки даних для отримання першого ключового слова має структуру як в таблиці 2.2:

Таблиця 2.2 – Алгоритм обробки даних для отримання першого ключового слова

Алгоритм обробки даних для отримання першого ключового слова:
Створення нової колонки *KeyWord* в *table* **for** кожна строка **in** *table*:   *KeyWord* = перше ключове слово з *KeyWords* **end** Видалення колонки *KeyWords* з *table* Створення нової таблиці таблицы *table2* – копії *table* Видалення колонки KeyWord з *table2*

Продовження таблиці 2.2

Створення нових колонок з назвами з строки *SubjectArea* в *table2*
Транспонування *table2*
Групування строк по значенням в строках в *table2*
Транспонування *table2*
for кожна строка **in** *table2*:
  for кожна колонка **in** *table2*:
    if *KeyWords* пуста:
      Записати в колонку «0»
    end
  end
end
Об'єднати *table* і *table2* по *RowID*
Видалення непотрібних символів з назв колонок в *table*
Видалення всіх колонок крім *KeyWord* і *Engineering*

На рисунку 2.3 представлений алгоритм обробки даних для отримання першого ключового слова.

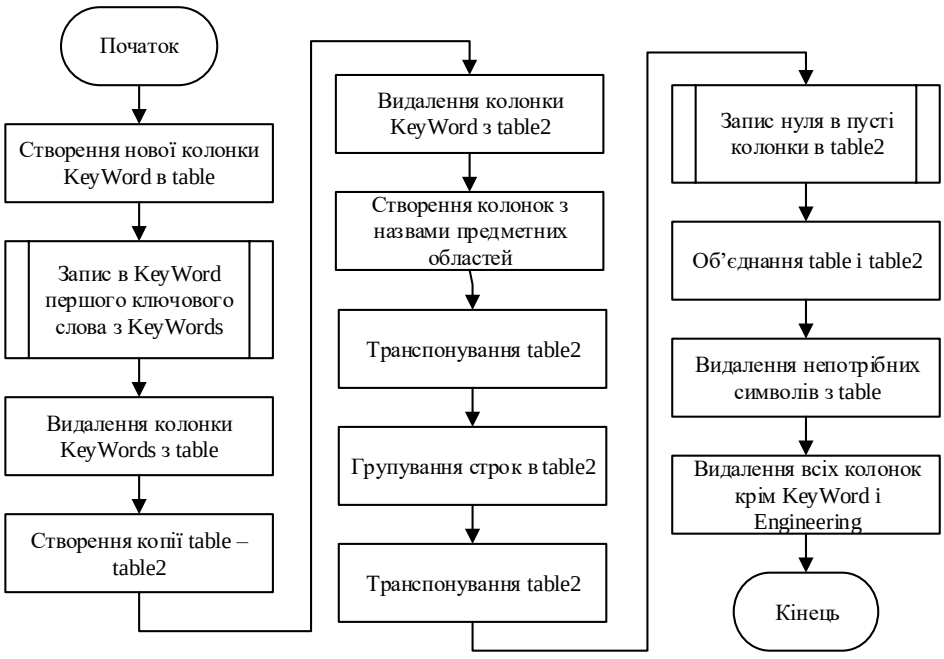

Рис. 2.3 – Алгоритм обробки даних для отримання першого ключового слова

Алгоритм був реалізований з-за допомогою наступних елементів: Java Snippet для виділення ключового слова, Column Filter для видалення колонок, Cell Splitter для створення нових колонок, Unpivoting і Pivoting для транспонування таблиці, GroupBy для групування даних в таблиці, Missing Value для запису в пусті строки нулів, Joiner для об'єднання таблиць, Column Rename (Regex) для зміни назв колонок. Реалізацію алгоритму в KNIME представлено на рисунку 2.4.

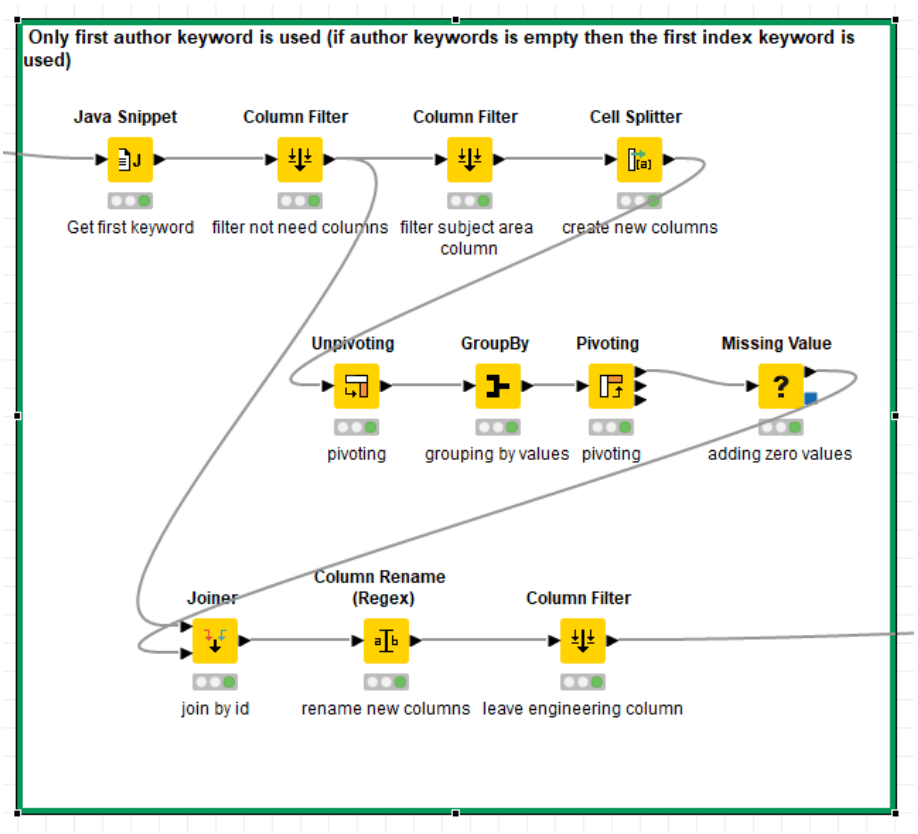

Рисунок 2.4 – Реалізація алгоритму обробки даних для отримання першого ключового слова в KNIME

2.3 Алгоритм обробки даних для отримання перших трьох ключових слів

Алгоритм служить для підготовки до роботи з класифікаторами, які використовують три перших ключових слова. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- список ключових слів;

- список предметних областей.

Алгоритм обробки даних для отримання трьох перших ключових слів має структуру як в таблиці 2.3:

Таблиця 2.3 – Алгоритм обробки даних для отримання трьох перших ключових слів

Алгоритм обробки даних для отримання трьох перших ключових слів:
Створення нової колонки *KeyWord1* в *table* Створення нової колонки *KeyWord2* в *table* Створення нової колонки *KeyWord3* в *table* **for** кожна строка **in** *table*:   *KeyWord1* = перше ключове слово з *KeyWords*   **if** *KeyWords* має більше, ніж 1 ключове слово: *KeyWord2* = друге ключове слово з *KeyWords*   **end**   **if** *KeyWords* має більше, ніж 2 ключових слова: *KeyWord3* = третє ключове слово з *KeyWords*   **end** **end** Видалення колонки *KeyWords* з *table* Створення *table2* - копії *table* Видалення колонки KeyWord1 з *table2* Видалення колонки KeyWord2 з *table2* Видалення колонки KeyWord3 з *table2* Створення нових колонок з назвами з строки *SubjectArea* в *table2* Транспонування *table2* Групування строк по значенням в строках в *table2* Транспонування *table2* **for** кожна строка **in** *table2*:   **for** кожна колонка **in** *table2*:   **if** *KeyWords* пуста: Записати в колонку «0»   **end**

Продовження таблиці 2.3

end
end
Об’єднати *table* і *table2* по *RowID*
Видалення строки з пустою колонкою KeyWord2 в *table*
Видалення строки з пустою колонкою KeyWord3 в *table*
Видалення непотрібних символів з назв колонок в *table*
Видалення всіх колонок крім *KeyWord1, KeyWord2, KeyWord3* і *Engineering*

Вихідні дані це таблиця з наступними колонками:

- перше ключове слово;

- друге ключове слово;

- третє ключове слово;

- значення належності до предметної області "Engineering".

На рис. рисунок 2.5 представлений алгоритм обробки даних для отримання трьох перших ключових слів.

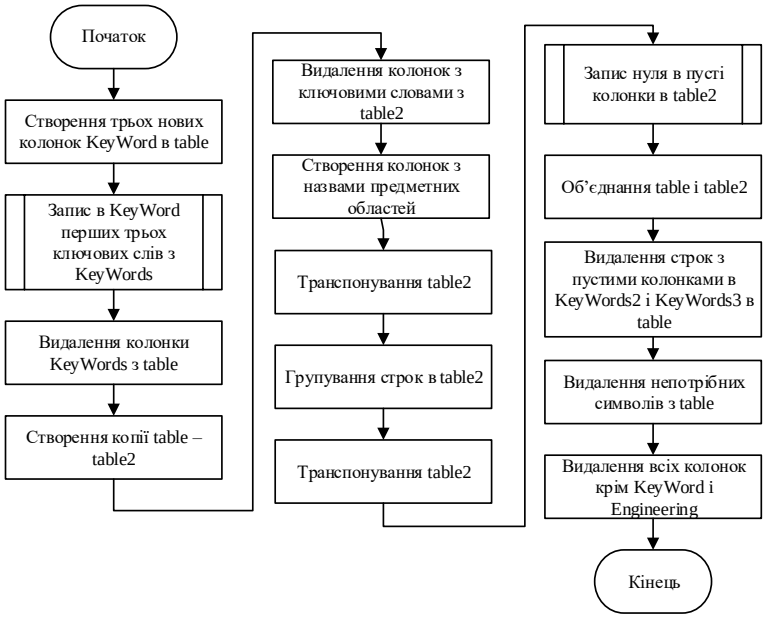

Рисунок 2.5 – Алгоритм обробки даних для отримання перших трьох ключових слів

Алгоритм був реалізований з-за допомогою наступних елементів: Java Snippet для виділення ключового слова, Column Filter для видалення колонок, Cell Splitter для створення нових колонок, Unpivoting и Pivoting для транспонування таблиці, GroupBy для групування даних в таблиці, Missing Value для запису в пусті строки нулів, Joiner для об'єднання таблиць, Row Filter для видалення строк, Column Rename (Regex) для зміни назв колонок. Реалізацію алгоритму в KNIME представлено на рисунку 2.6.

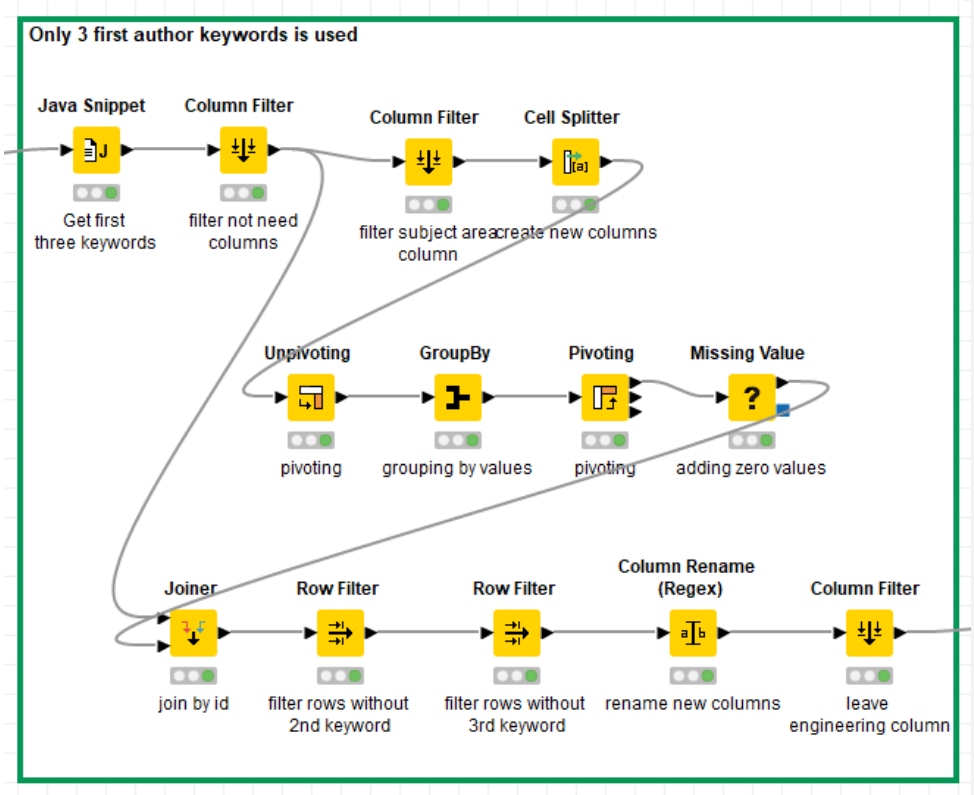

Рисунок 2.6 – Реалізація алгоритму обробки даних для отримання трьох перших ключових слів в KNIME

2.4 Алгоритм обробки даних для отримання трьох останніх ключових слів

Алгоритм служить для підготовки до роботи з класифікаторами, які використовують три останніх ключових слова. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- список ключових слів;

- список предметних областей.

Алгоритм обробки даних для отримання трьох останніх ключових слів має структуру як в таблиці 2.4:

Таблиця 2.4 – Алгоритм обробки даних для отримання трьох останніх ключових слів

Алгоритм обробки даних для отримання трьох перших ключових слів:
Створення нової колонки *KeyWord1* в *table*
Створення нової колонки *KeyWord2* в *table*
Створення нової колонки *KeyWord3* в *table*
for кожна строка **in** *table*:
KeyWord1 = перше ключове слово з кінця в *KeyWords*
if *KeyWords* має більше, ніж 1 ключове слово:
KeyWord2 = друге ключове слово з кінця в *KeyWords*
end
if *KeyWords* має більше, ніж 2 ключових слова:
KeyWord3 = третє ключове слово з кінця в *KeyWords*
end
end
Видалення колонки *KeyWords* з *table*
Створення *table2* - копії *table*
Видалення колонки KeyWord1 з *table2*
Видалення колонки KeyWord2 з *table2*
Видалення колонки KeyWord3 з *table2*
Створення нових колонок з назвами з строки *SubjectArea* в *table2*
Транспонування *table2*
Групування строк по значенням в строках в *table2*
Транспонування *table2*
for кожна строка **in** *table2*:
for кожна колонка **in** *table2*:
if *KeyWords* пуста:
Записати в колонку «0»
end

Продовження таблиці 2.4

end
end
Об’єднати *table* і *table2* по *RowID*
Видалення строки з пустою колонкою KeyWord2 в *table*
Видалення строки з пустою колонкою KeyWord3 в *table*
Видалення непотрібних символів з назв колонок в *table*
Видалення всіх колонок крім *KeyWord1*, *KeyWord2*, *KeyWord3* і *Engineering*

Вихідні дані це таблиця з наступними колонками:

- перше ключове слово;

- друге ключове слово;

- третє ключове слово;

- значення належності до предметної області "Engineering".

На рис. рисунок 2.7 представлений алгоритм обробки даних для отримання трьох останніх ключових слів.

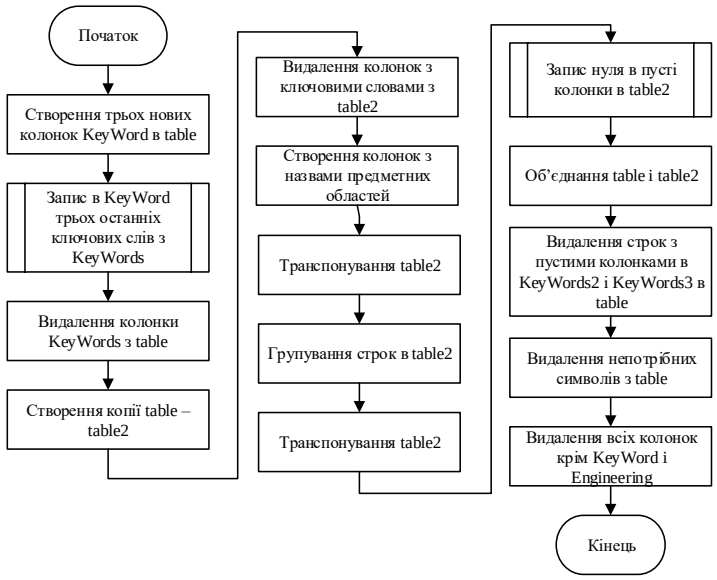

Рисунок 2.7 – Алгоритм обробки даних для отримання трьох останніх ключових слів

Алгоритм був реалізований з-за допомогою наступних елементів: Java Snippet для виділення ключового слова, Column Filter для видалення колонок, Cell Splitter для створення нових колонок, Unpivoting и Pivoting для транспонування таблиці, GroupBy для групування даних в таблиці, Missing Value для запису в пусті строки нулів, Joiner для об'єднання таблиць, Row Filter для видалення строк, Column Rename (Regex) для зміни назв колонок. Реалізацію алгоритму в KNIME представлено на рисунку 2.8.

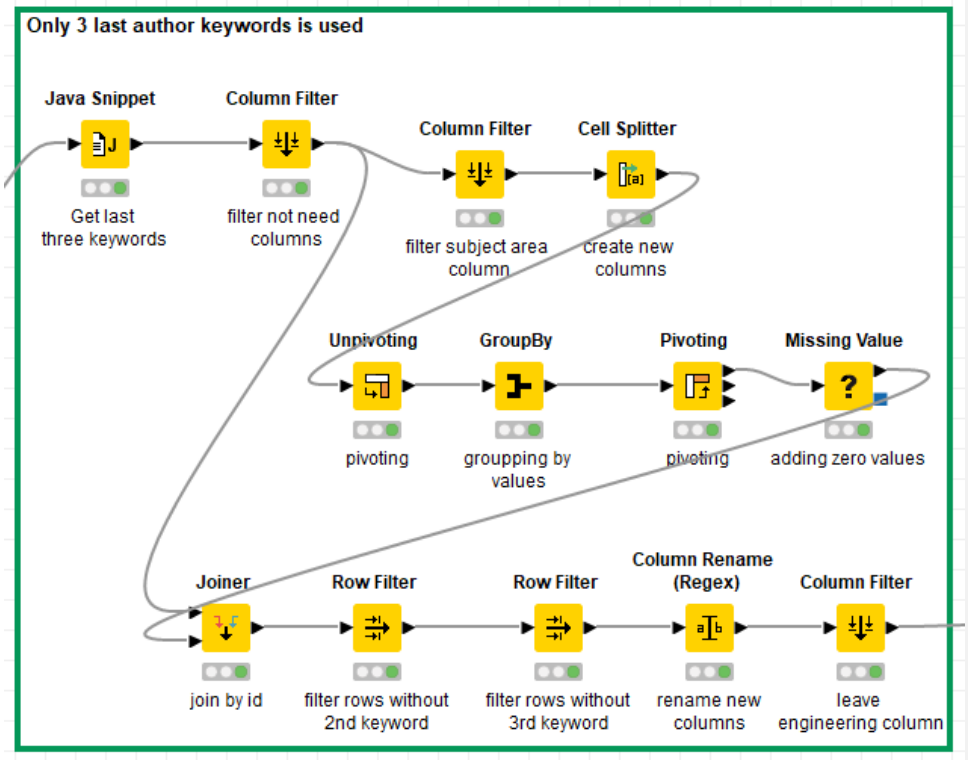

Рисунок 2.8 – Реалізація алгоритму обробки даних для отримання трьох останніх ключових слів в KNIME

2.5 Алгоритм обробки даних для отримання трьох випадкових ключових слів

Алгоритм служить для підготовки до роботи з класифікаторами, які використовують три випадкових ключових слова. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- список ключових слів;
- список предметних областей.

Вихідні дані це таблиця з наступними колонками:

- перше ключове слово;
- друге ключове слово;
- третє ключове слово;
- значення належності до предметної області "Engineering".

Алгоритм обробки даних для отримання трьох випадкових ключових слів має структуру як в таблиці 2.5:

Таблиця 2.5 – Алгоритм обробки даних для отримання трьох випадкових ключових слів

Алгоритм обробки даних для отримання трьох випадкових ключових слів:
Створення нової колонки *KeyWord1* в *table* Створення нової колонки *KeyWord2* в *table* Створення нової колонки *KeyWord3* в *table* **for** кожна строка **in** *table*:   *KeyWord1* = перше випадкове ключове слово з *KeyWords*   **if** *KeyWords* має більше, ніж 1 ключове слово: *KeyWord2* = друге випадкове ключове слово з *KeyWords*   **end**   **if** *KeyWords* має більше, ніж 2 ключових слова: *KeyWord3* = третє випадкове ключове слово з *KeyWords*   **end** **end** Видалення колонки *KeyWords* з *table* Створення *table2* - копії *table* Видалення колонки KeyWord1 з *table2* Видалення колонки KeyWord2 з *table2* Видалення колонки KeyWord3 з *table2* Створення нових колонок з назвами з строки *SubjectArea* в *table2* Транспонування *table2*

Продовження таблиці 2.5

Групування строк по значенням в строках в *table2*
Транспонування *table2*
for кожна строка **in** *table2*:
  for кожна колонка **in** *table2*:
    if *KeyWords* пуста:
      Записати в колонку «0»
    end
  end
end
Об’єднати *table* і *table2* по *RowID*
Видалення строки з пустою колонкою KeyWord2 в *table*
Видалення строки з пустою колонкою KeyWord3 в *table*
Видалення непотрібних символів з назв колонок в *table*
Видалення всіх колонок крім *KeyWord1*, *KeyWord2*, *KeyWord3* і *Engineering*

На рис. 2.9 представлений алгоритм обробки даних для отримання трьох випадкових ключових слів.

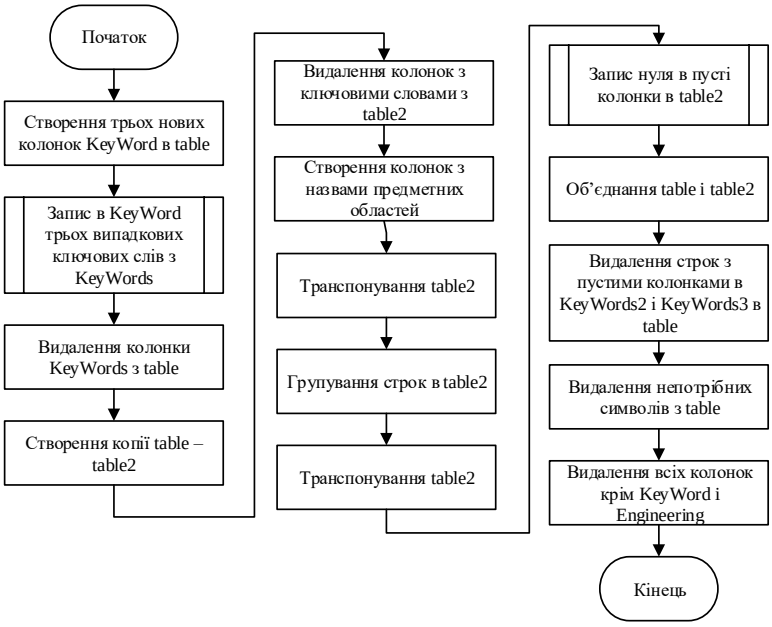

Рисунок 2.9 - Алгоритм обробки даних для отримання трьох випадкових ключових слів

Алгоритм був реалізований з-за допомогою наступних елементів: Java Snippet для виділення ключового слова, Column Filter для видалення колонок, Cell Splitter для створення нових колонок, Unpivoting и Pivoting для транспонування таблиці, GroupBy для групування даних в таблиці, Missing Value для запису в пусті строки нулів, Joiner для об'єднання таблиць, Row Filter для видалення строк, Column Rename (Regex) для зміни назв колонок. Реалізацію алгоритму в KNIME представлено на рисунку 2.10.

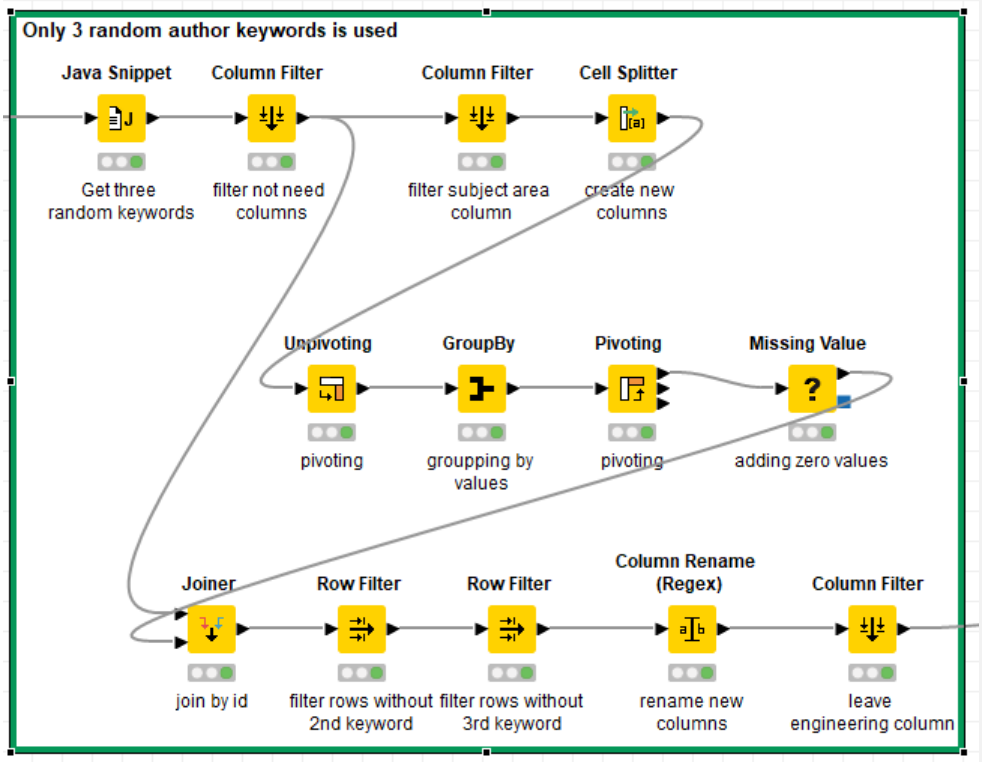

Рисунок 2.10 – Реалізація алгоритму обробки даних для отримання трьох випадкових ключових слів в KNIME

2.6 Алгоритм обробки даних для отримання всіх ключових слів

Алгоритм служить для підготовки до роботи з класифікаторами, які використовують всі ключові слова. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- список ключових слів;

- список предметних областей.

Вихідні дані це таблиця з наступними колонками:

- список ключових слів;

- значення належності до предметної області "Engineering".

Алгоритм обробки даних для отримання всіх ключових слів має структуру як в таблиці 2.6:

Таблиця 2.6 – Алгоритм обробки даних для отримання всіх ключових слів

Алгоритм обробки даних для отримання всіх ключових слів:
Видалення колонки *KeyWords* з *table* Створення *table2* - копії *table* Видалення колонки KeyWords з *table2* Створення нових колонок з назвами з строки *SubjectArea* в *table2* Транспонування *table2* Групування строк по значенням в строках в *table2* Транспонування *table2* **for** кожна строка **in** *table2*:   **for** кожна колонка **in** *table2*:   **if** *KeyWords* пуста: Записати в колонку «0»   **end**   **end** **end** Об’єднати *table* і *table2* по *RowID* Видалення строки з пустою колонкою KeyWords в *table* Видалення непотрібних символів з назв колонок в *table* Видалення всіх колонок крім *KeyWords* і *Engineering*

На рисунку 2.11 представлений алгоритм обробки даних для отримання всіх ключових слів.

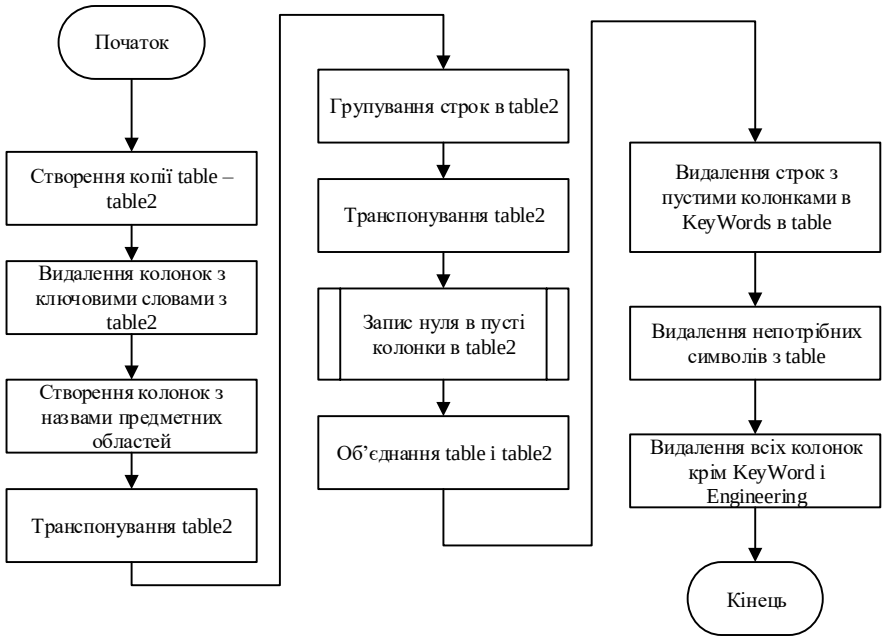

Рисунок 2.10 - Алгоритм обробки даних для отримання всіх ключових слів

Алгоритм був реалізований з-за допомогою наступних елементів: Column Filter для видалення колонок, Cell Splitter для створення нових колонок, Unpivoting и Pivoting для транспонування таблиці, GroupBy для групування даних в таблиці, Missing Value для запису в пусті строки нулів, Joiner для об'єднання таблиць, Row Filter для видалення строк, Column Rename (Regex) для зміни назв колонок. Реалізацію алгоритму в KNIME представлено на рисунку 2.12.

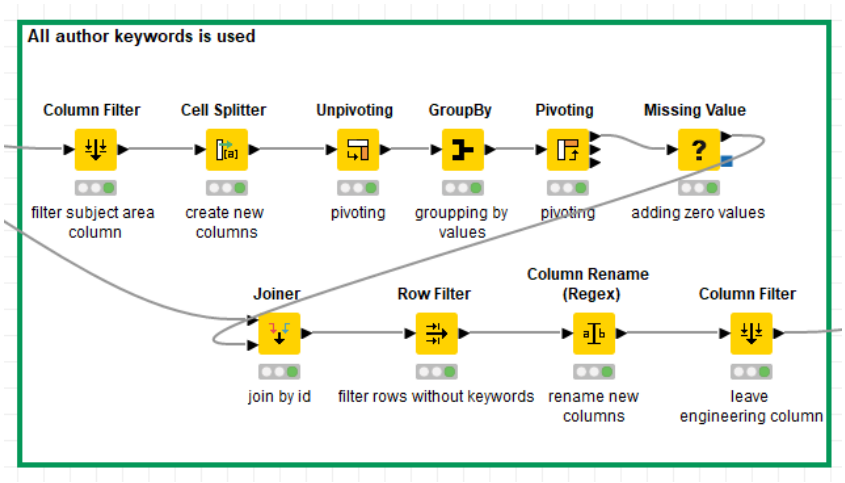

Рисунок 2.11 – Реалізація алгоритму обробки даних для отримання всіх ключових слів в KNIME

2.7 Алгоритм на основі дерева рішень

Алгоритм описує роботу класифікатора на основі дерева рішень. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- одна чи більше колонок з ключовими словами;
- належність до предметної області.

В даному випадку колонка, що зберігає належність до предметної області є результатом, який має визначити класифікатор, а всі інші колонки є параметрами для класифікації, які подаються на входи класифікатора. Вихідні дані це таблиця з наступними колонками:

- назва класифікатора;
- точність класифікації.

Алгоритм класифікації на основі дерева рішень має структуру як в таблиці 2.7.

Таблиця 2.7 – Алгоритм класифікації на основі дерева рішень

Алгоритм класифікації на основі дерева рішень
for кожна строка **in** *table*:
Тренування класифікатора на вхідних даних
end
Класифікація всіх даних
Отримання результатів класифікації (точності)

На рисунку 2.13 представлений алгоритм класифікації на основі дерева рішень.

Рисунок 2.123 – Алгоритм класифікації на основі дерева рішень

Алгоритм був реалізований з-за допомогою наступних елементів: X-Partitioner і X-Aggregator для збору даних, Decision Tree Learner і Decision Tree Predictor для тренування і класифікації, PMML To Cell і Scorer для отримання результатів в вигляді таблиці, Rule Engine і Constant Value Column для створення нових колонок, RowID для видалення колонок по id строки, Column Filter і Row

Filter для отримання статистики, Cross Joiner для об’єднання моделі і результатів. Реалізацію алгоритму в KNIME представлено на рисунку 2.14.

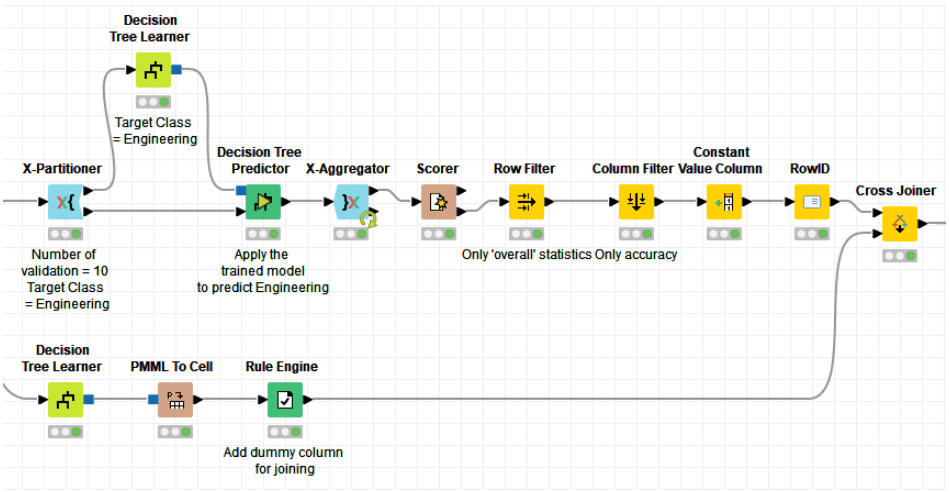

Рисунок 2.14 – Реалізація алгоритму на основі дерева рішень в KNIME

2.8 Алгоритм на основі методу опорних векторів

Алгоритм описує роботу класифікатора на основі методу опорних векторів. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- одна чи більше колонок з ключовими словами;
- належність до предметної області.

В даному випадку колонка, що зберігає належність до предметної області є результатом, який має визначити класифікатор, а всі інші колонки є параметрами для класифікації, які подаються на входи класифікатора. Вихідні дані це таблиця з наступними колонками:

- назва класифікатора;
- точність класифікації.

Алгоритм класифікації на основі метода опорних векторів має структуру як в таблиці 2.8.

Таблиця 2.8 – Алгоритм класифікації на основі методу опорних векторів

Алгоритм класифікації на основі метода опорних векторів
for кожна строка **in** *table*:
Тренування класифікатора на вхідних даних
end
Класифікація всіх даних
Отримання результатів класифікації (точності)

На рисунку 2.15 представлений алгоритм класифікації на основі метода опорних векторів.

Рисунок 2.15 – Алгоритм класифікації на основі методу опорних векторів

Алгоритм був реалізований з-за допомогою наступних елементів: X-Partitioner і X-Aggregator для збору даних, SVM Learner і SVM Predictor для тренування і класифікації, PMML To Cell і Scorer для отримання результатів в вигляді таблиці, Rule Engine і Constant Value Column для створення нових колонок, RowID для видалення колонок по id строки, Column Filter і Row Filter для

отримання статистики, Cross Joiner для об’єднання моделі і результатів. Реалізацію алгоритму в KNIME представлено на рисунку 2.16.

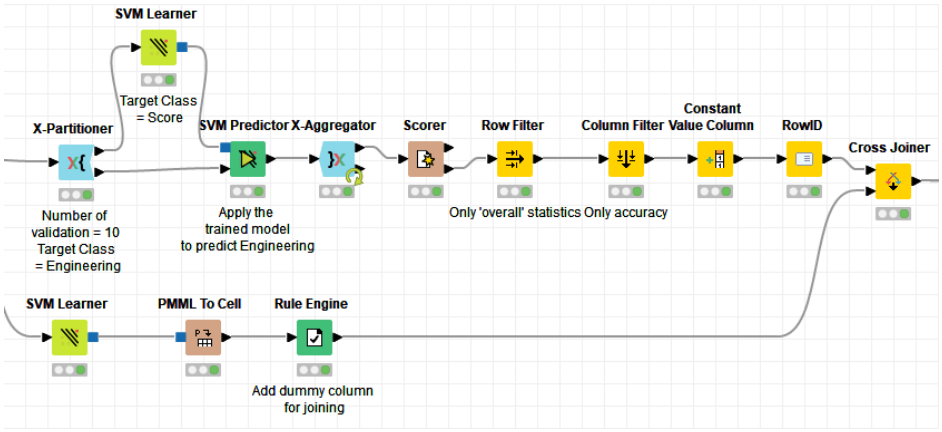

Рисунок 2.16 - Реалізація алгоритму на основі методу опорних векторів в KNIME

2.9 Алгоритм на основі нейронної мережі

Алгоритм описує роботу класифікатора на основі нейронної мережі. Вхідними даними для алгоритму є таблиця *table*, яка містить наступні колонки:

- одна чи більше колонок з ключовими словами;
- належність до предметної області.

В даному випадку колонка, що зберігає належність до предметної області є результатом, який має визначити класифікатор, а всі інші колонки є параметрами для класифікації, які подаються на входи класифікатора. Вихідні дані це таблиця з наступними колонками:

- назва класифікатора;
- точність класифікації.

Алгоритм класифікації на основі нейронної має структуру як в таблиці 2.9.

Таблиця 2.9 – Алгоритм класифікації на основі нейронної мережі

Алгоримт класифікації на основі нейронної мережі
for кожна строка **in** *table*: Тренування класифікатора на вхідних даних **end** Класифікація всіх даних Отримання результатів класифікації (точності)

На рисунку 2.17 представлений алгоритм класифікації на основі нейронної мережі.

Рисунок 2.17 – Алгоритм класифікації на основі нейронної мережі

Алгоритм був реалізований з-за допомогою наступних елементів: X-Partitioner і X-Aggregator для збору даних, RProp MLP Learner і RProp MLP Predictor для тренування і класифікації, PMML To Cell і Scorer для отримання результатів в вигляді таблиці, Rule Engine і Constant Value Column дляо отримання результатів в бінарному вигляді і для створення нових колонок, Double To Int для переводу в цілий формат числа, RowID для видалення колонок по id строки, Column Filter і

Row Filter для отримання статистики, Cross Joiner для об'єднання моделі і результатів. Реалізацію алгоритму в KNIME представлено на рисунку 2.18.

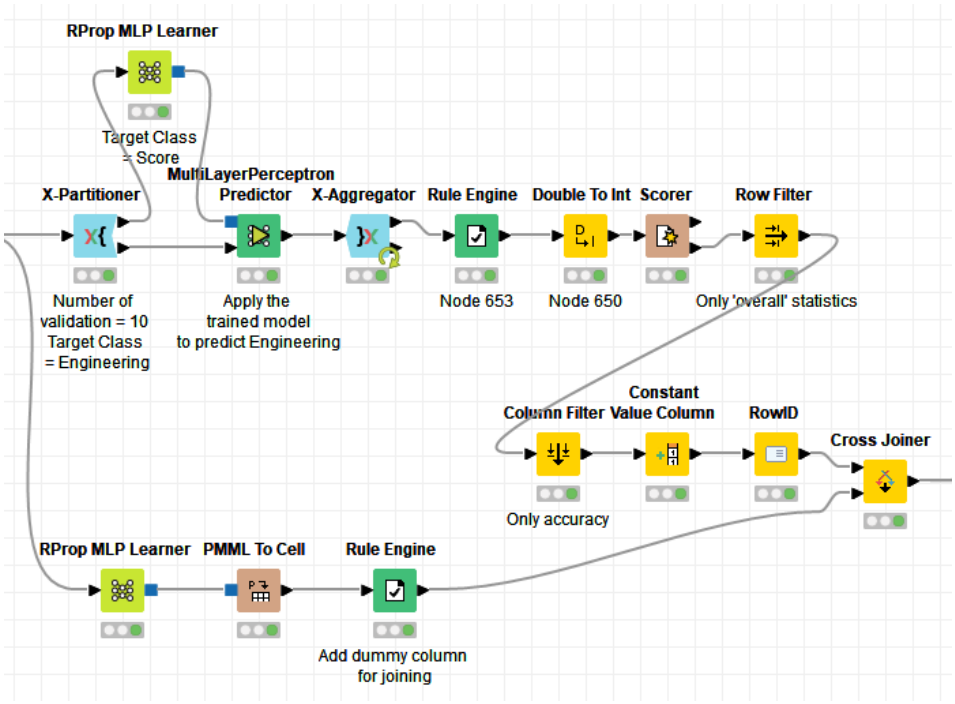

Рисунок 2.18 – Реалізація алгоритму на основі нейронної мережі в KNIME

2.10 Алгоритм класифікації, розроблений автором (звичайний)

У цьому підрозділі спочатку описується сам алгоритм. Потім описується структура і конкретна реалізація цього алгоритму. Цей алгоритм був раніше описаний в [27], автором якого є автор даної роботи. Тому за згодою автора нижче наведені теоретичні розрахунки.

Алгоритм можна розділити на дві частини - алгоритм тренування, в якому "мішок слів" заповнюється даними, і саме алгоритм класифікації, в якому за вхідними даними класифікація здійснюється за допомогою натренованого і заповненого "мішка слів".

Нехай дана множина *SubjectAreas,* що містить *N* предметних областей. Класифікація буде виконана по елементам з цьої множини.

Вхідні дані для алгоритму тренування це множина *TrainingInput*

(2.7), яка містить список *TrainingSet*

(2.8). *TrainingSet* містить:

- предметна область;

- список ключових слів, пов'язаних з предметною областю.

$$TrainingInput = \{TrainingSet1, TrainingSet2..TrainingSetM\} \quad (2.7)$$

де *TrainingSet1, TrainingSet2.. TrainingSetM* – множина вхідних даних *TrainingSet*.

$$TrainingSet = \{SubjectArea, \{KeyWord1, KeyWord2..KeyWordK\}\} \quad (2.8)$$

де *SubjectArea* – предметна область;

KeyWord1,KeyWord2..KeyWordK – ключові слова, пов'язані з предметною областю *SubjectArea.*

Під час передачі наборів даних наповнюється «мішок слів». «Мішок слів» реалізований в вигляді складної множини *WordBag* (2.9):

$$WordBag = \{WordData1, WordData2..WordDataJ\} \quad (2.9)$$

де *WordData1, WordData2.. WordDataJ* – множини *WordData*

(2.10) з даними про належність ключових слів до предметної області.

$$WordData = \{KeyWord, \left\{ \begin{matrix} \{SubjectArea1, Frequency1\}, \\ \{SubjectArea2, Frequency2\}..\{SubjectAreaN, FrequencyN\} \end{matrix} \right\}\} \quad (2.10)$$

де *KeyWord* – окреме ключове слово;

SubjectArea1.. SubjectAreaN – всі предметні області;

Frequency1.. FrequencyN – цілі числа, що показіють, скільки разів ключове слово *KeyWord* трапилось під час тренування в предметній області *SubjectArea*. *WordData* не може мати повторень K*eyWord*.

В результаті з ростом потужності множини *TrainingSet*, буде зростати потужність множини *WordBag* або будуть зростати значення *Frequency*.

Для алгоритму класифікації вхідними даними є множина *ClassificationInput* (2.11), яка містить ключові слова.

$$ClassificationInput = \{KeyWord1, KeyWord2..KeyWordK\} \quad (2.11)$$

Вихідні дані для алгоритму класифікації це множина *ClassificationOutput* (2.12), який містить:

- список предметних областей, по яким відбувається класифікація;

- список значень відношення вхідних ключових слів до предметних областей.

$$ClassificationOutput = \{\{SubjectArea1, Probability1\}, \{SubjectArea2, Probability2\}..\{SubjectAreaN, ProbabilityN\}\} \quad (2.12)$$

де *SubjectArea1*.. *SubjectAreaN* – предметні області, по яким відбувається класифікація;

Probability1.. *ProbabilityN* – вартості ймовірностей в форматі з плаваючою точкою в інтервалі [0..1], що показують належність до предметної області *SubjectArea*. Алгоритм класифікації представлений в таблиці 2.10.

Таблиця 2.10 – Алгоритм класифікації на основі нечітких множин

Алгоритм класифікації на основі нечітких множин
N = кількість предметних областей Множина *WordBag* **for** кожне *KeyWord*: Створити множину WordData з WordBag на основі KeyWord Отримати суму *Frequency* в усіх множинах *SubjectArea* Записати суму *Frequency* в *Probability* в множину *ClassificationOutput* **if** сума *Frequency* не рівна 0: Нормалізувати *Frequency*: розділити Frequency на суму всіх *Frequency* в *Probability* **end** **end** **for** кожного *Probability* в *ClassificationOutput*: Нормалізувати *ClassificationOutput*: розділити *Probability* на максимальний елемент в *Probability* **end** return *ClassifactionOutput*;

Реалізація була виконана на мові програмування Java в вигдляд консолької програми. В якості множин (звичайних і розмитих) були використані класи ArrayList [28] і HashMap [29]. При запуску програми потрібно подати параметри,

які вказують, який тип алгоритму (нормальний чи модифікований) використати, а також скільки ключових слів (одне, три перших, три останніх, три випадкових) будуть використані в програмі. Структура роботи програми представлена на рисунку 2.19:

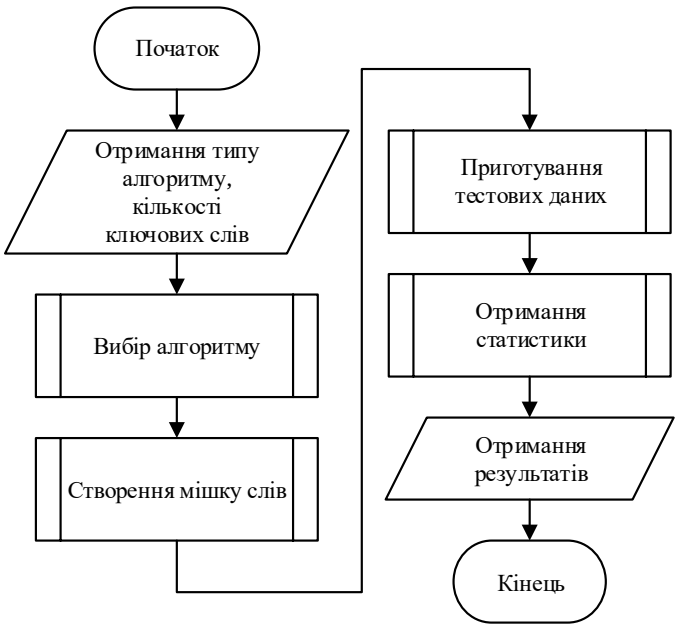

Рисунок 2.19 – Схема роботи реалізації програми

На початку алгоритму читаються вхідні дані (тип алгоритму, дані з файлу, імена тематичних областей). Потім, в залежності від типу алгоритму, реалізуються по порядку функції. Реалізації є для звичайного і модифікованного алгоритмів.

В функції створення "мішка слів" є створення і заповнення "мішка слів" і створення тестового зразка. Схема реалізації алгоритму показана на рис. 2.20.

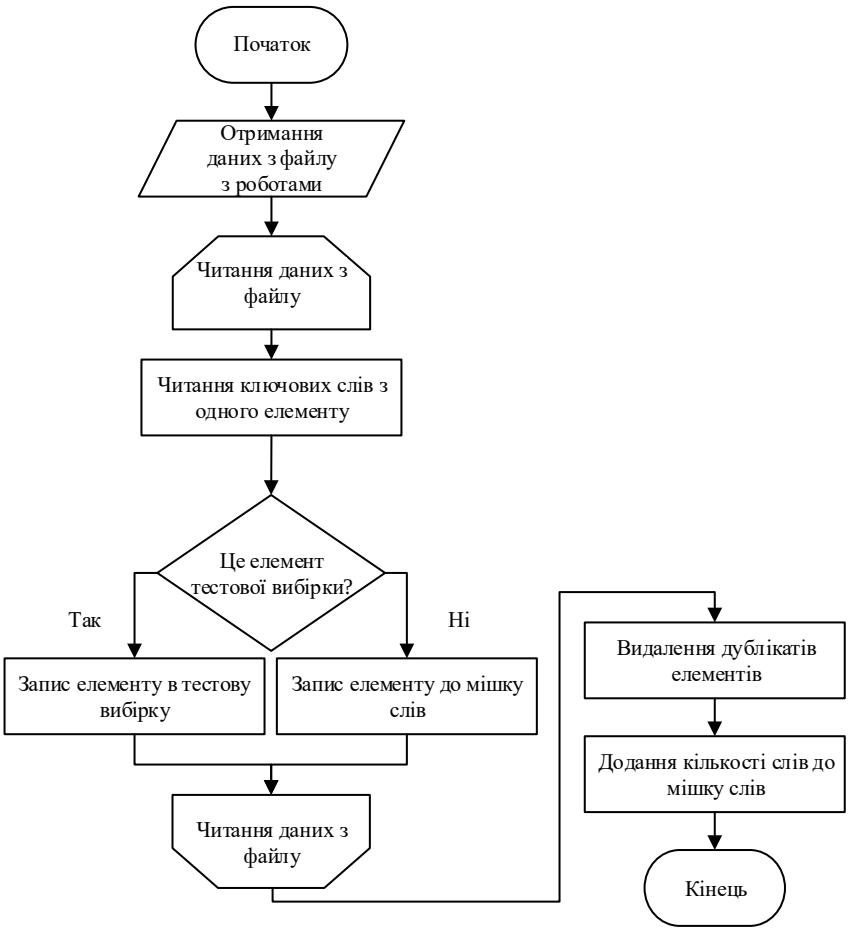

Рисунок 2.20 – Схема роботи реалізації створення і наповнення «мішку слів»

В залежності від кількості ключових слів, які будут використані під час класифікації, в тестовій вибірці будуть різні дані. Схема реалізації показана на рис. 2.21.

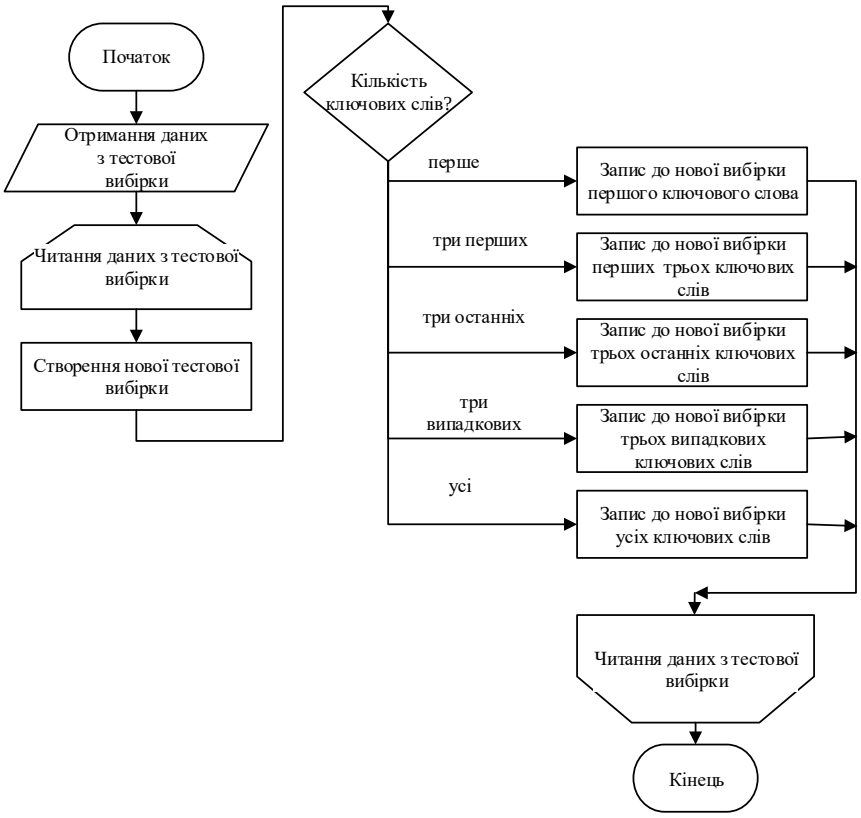

Рисунок 2.21 – Схема роботи реалізації підготовки тестової вибірки

Добавлено примечание ([D1]): есть пара схем для большего расписания раздела

2.11 Алгоритм класифікації, розроблений автором (модифікований)

Модифікований алгоритм практично ідентичний зі звичайним алгоритмом з підрозділу 2.10, але відрізняється від нього тим, що замість ключових слів в цілому беруться окремі слова (наприклад слова взяті з фрази "Класифікація текстів" будуть розділені на "класифікація" і "текстів").

2.12 Висновки до розділу

В розділі описано алгоритми та їх реалізації для вирішення завдання. Алгоритми і реалізації створені як на основі частково готових рішень, так і написані з нуля. Створені алгоритми на основі «мішку слів», нейронної мережі, дерева рішень, методу опорних векторів. Приведені схеми, лістинги алгоритмів, рисунки реалізацій. Реалізовані проекти в середовищі KNIME і консольна программа на Java.

3 ПРАКТИЧНЕ ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕНЬ

3.1 Перевірка результатів

Оскільки результат алгоритму класифікації представлений у вигляді нечіткої множини, для отримання конкретного результату необхідно перевести результати в бінарну версію. При переході до бінарного результату прийнято наступне правило - якщо значення ймовірності більше або дорівнює 0,5, то результат позитивний, в іншому випадку негативний. Результатом буде множина *ClassificationOutputBinary*

(3.1), яка вже не буде нечіткою.

$$ClassificationOutputBinary = \begin{Bmatrix} \{SubjectArea1, BinaryValue1\}, \\ \{SubjectArea2, BinaryValue2\}.. \\ \{SubjectAreaN, BinaryValueN\} \end{Bmatrix} \quad (3.1)$$

де *SubjectArea1..SubjectAreaN* – предметні області, по яким виконується класифікація;

BinaryValue1..BinaryValueN – бінарні цілі значення в інтервалі [0,1], вказуючі належність до предметної області *SubjectArea*.

Для перевірки результатів також потрібно отримати значення точності класифікації предметних областей. Тоді точність обчислюється як відношення кількості правильно класифікованих предметних областей до кількості усіх предметних областей, зв’язаних з тою тематикою. В таблиці 3.1 приведений алгоритм отримання точності.

Таблиця 3.1 – Алгоритм отримання точності класифікації

Алгоритм отримання точності класифікації
Множини *Classified, Real* **if** | *Classified* | ≥ | *Real* |**:**   *Точність* = | (*Real* ∩ *Classified*) | / | *Classified* |; **else**   *Точність* = | (*Real* ∩ *Classified*) | / | *Real* |; **end** return *Точність*

В табл. 3.1 | (*Real* ∩ *Classified*) | - кількість правильно класифікованих ключових слів, | *Classified* | - кількість класифікованих ключових слів, | *Real* | - множина ключових слів з правильною класифікацією.

3.2 Результати досліджень

В ході тестування реалізації алгоритму отримано результаті для наступних варіантів:

1) В залежності від кількості ключових слів:
   - тільки перше ключове слово;
   - тільки три перших ключоих слова;
   - тільки три останніх ключових слова;
   - тільки три випадкових ключових слова;
   - всі ключові слова.
2) В залежності від алгоритму:
   - звичайний алгоритм (реалізований на мові програмування Java);
   - модифікований алгоритм (реалізований на мові програмування Java);
   - дерево рішень (реалізовано в середовищі KNIME);
   - метод опорних векторів (реалізовано в середовищі KNIME);
   - нейронна мережа (реалізовано в середовищі KNIME).
3) В залежності від роду класифікації:
   - бінарна;
   - небінарна (багатокласова).

Також окремо отримано результати роботи алгоритмів (звичайного і модифікованого), реалізованих на мові програмування Java для всіх предметних областей (багатокласова класифікація). Таблиця 3.2 має всі значення точності для бінарної, де значення „1” означає 100%, значення „0,25” означає 25%, значення „0” означає 0%.

Таблиця 3.2 – Точність для бінарної класифікації

	Звичайний алгоритм	Модифікований алгоритм	Дерево рішень	SVM	Нейронна мережа
Перше ключове слово	0,538	0,586	0,665	0,585	0,585
3 перших ключових слова	0,564	0,611	0,626	0,586	0,586
3 останніх ключових слова	0,505	0,611	0,63	0,586	0,585
3 випадкових ключових слова	0,541	0,614	0,611	0,586	0,586
Всі ключові слова	0,750	0,771	0,586	0,585	0,585

Для одного ключового слова, перших трьох ключових слів і останніх трьох ключових слів максимальна точність досягається в випадку алгоритму дерева рішень. Для останніх трьох ключових слів і всіх ключових слів найбільша точність була досягнута з використанням модифікованого алгоритму. Точність при використанні алгоритмів, реалізованих в середовищі KNIME, практично не змінюється, точність алгоритму дерева рішень змінюється незначно. Точність звичайних і модифікованих алгоритмів змінюється в залежності від кількості і якості ключових слів. Графічне представлення значень точності для двійкової класифікації представлено на рис. 3.1.

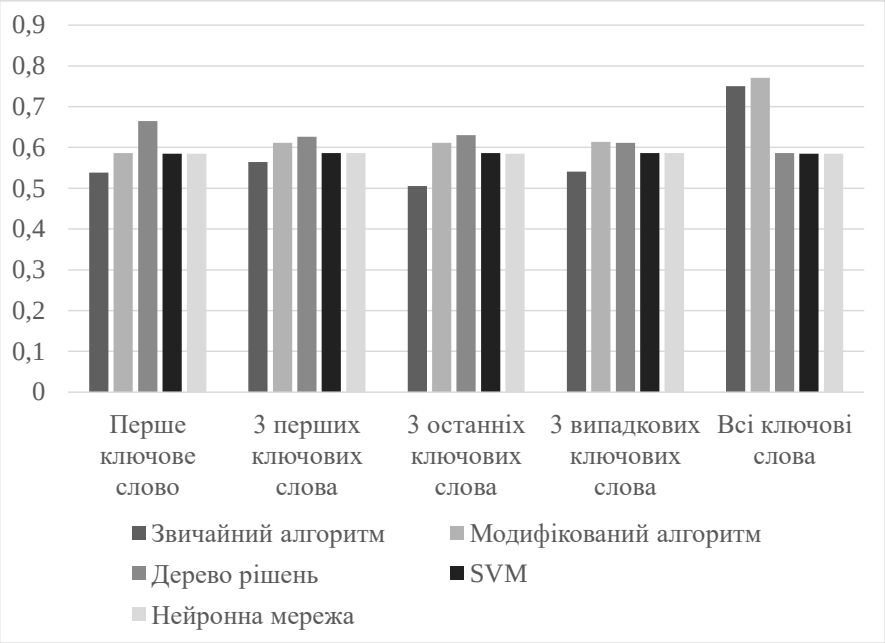

Рисунок 3.1 – Точність для бінарної класифікації

Рис. 3.1 показує, що точність алгоритму, створенному на основі дерева рішень, має більшу точність з невеликою кількістю ключових слів у порівнянні з іншими алгоритмами. Якщо використовуються всі ключові слова, звичайний і модифікований алгоритми мають велику точність.

Для небінарної класифікації приведені кількісні дані про точність класифікації. У таблиці 3.3 наведені значення, пов'язані з класифікацією для звичайного алгоритму, включаючи середні значення.

Таблиця 3.3 – Результати роботи класифікатора на основі звичайного алгоритму, реалізованого на мові програмування Java

	Кількість вхідних даних	Відношення кількості даних до вхідного набору даних, %
Повністю правильно класифіковано	118	18,02
Як мінімум одна предметна область правильно класифікована	520	79,39
Без правильно класифікованих предметних областей	135	20,61
Всього класифіковано даних	655	100
Середнє значення	-	47,38
Медіанне значення	-	50

З досліджуваного набору даних, який налічує 655 зразків, 118 були однозначно класифіковані, тобто всі класи були правильно розпізнані. У таблиці 3.3 також наведені значення, пов'язані з точністю розпізнавання тільки одного класу, оскільки цього достатньо для практичних завдань, а саме 520 зразків. Середнє і медіанне значення наведені для ясності точності розпізнавання. Таблиця 3.4 представляє класифікаційні значення для модифікованого алгоритму і середні значення.

Таблиця 3.4 – Результати роботи класифікатора на основі модифікованого алгоритму, реалізованого на мові програмування Java

	Кількість вхідних даних	Відношення кількості даних до вхідного набору даних, %
Повністю правильно класифіковано	56	8,55
Як мінімум одна предметна область правильно класифікована	513	78,32
Без правильно класифікованих предметних областей	142	21,68
Всього класифіковано даних	655	100
Середнє значення	-	40,26
Медіанне значення	-	33

З досліджуваного набору даних, що налічує 655 зразків, 56 були однозначно класифіковані, тобто всі класи були правильно розпізнані. У таблиці 3.4 також наведені значення, пов'язані з точністю розпізнавання тільки одного класу, оскільки цього достатньо для практичних завдань, а саме 513 зразків. Середнє і медіанне наведені для ясності точності розпізнавання. На малюнку 3.2 і малюнку 3.4 представлені згруповані значення точності для реалізованих алгоритмів (звичайного і модифікованого) на мові програмування Java.

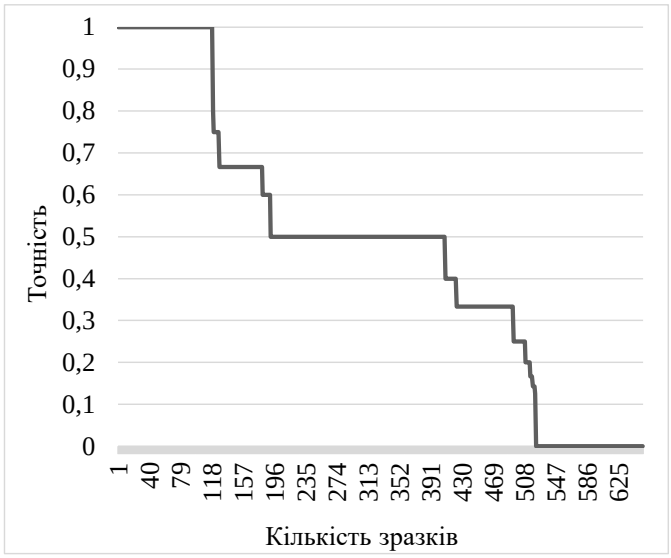

Рисунок 3.13 – Згруповані значення точності звичайного алгоритму, реалізованого на мові програмування Java

На рис. 3.2 представлені результати небінарної класифікації, де значення точності "1" означає, що всі класи були класифіковані правильно, значення точності "0,5" означає, що 50% класів були класифіковані правильно, значення точності "0" означає, що жоден з класів не був правильно класифікований. Як видно на малюнку, ~100 зразків були класифіковані правильно, близько 400 зразків були класифіковані з високою точністю, ~500 зразків були класифіковані з достатньою точністю. На малюнку чітко показана позитивна статистика класифікації. На рис. 3.3 і рис. 3.5 представлені згруповані значення кількості правильно класифікованих даних для алгоритмів (звичайного і модифікованого), реалізованих на мові програмування Java.

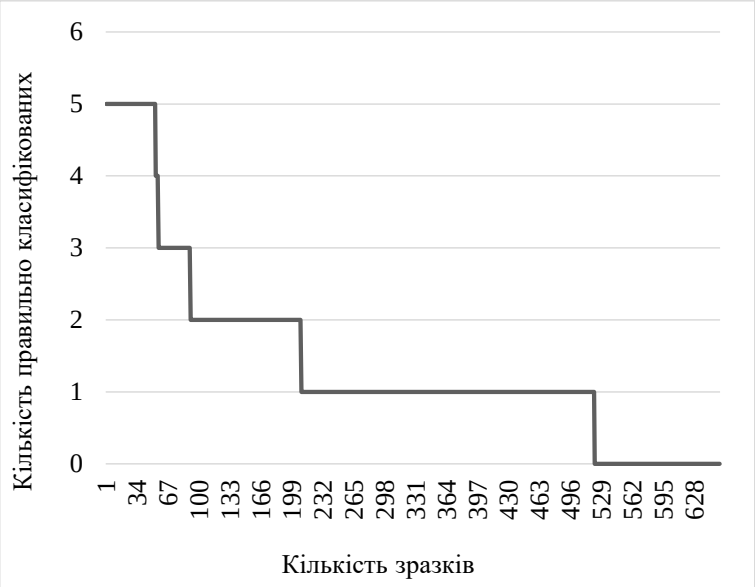

Рисунок 3.14 – Згруповані значення кількості правильно класифікованих зразків для звичайного алгоритму, реалізованого на мові програмування Java

На рис. 3.3 також представлені результати небінарної класифікації. Як видно на малюнку, 5 класів були правильно класифіковані в ~60 зразках, більш ніж 3 класи були правильно класифіковані в ~90 зразках, в ~200 зразках більш ніж 2 класи були правильно класифіковані і ~510 зразків було правильно класифіковано з більшою кількістю класів, ніж 1-й включно. На рисунку чітко показана позитивна статистика класифікації.

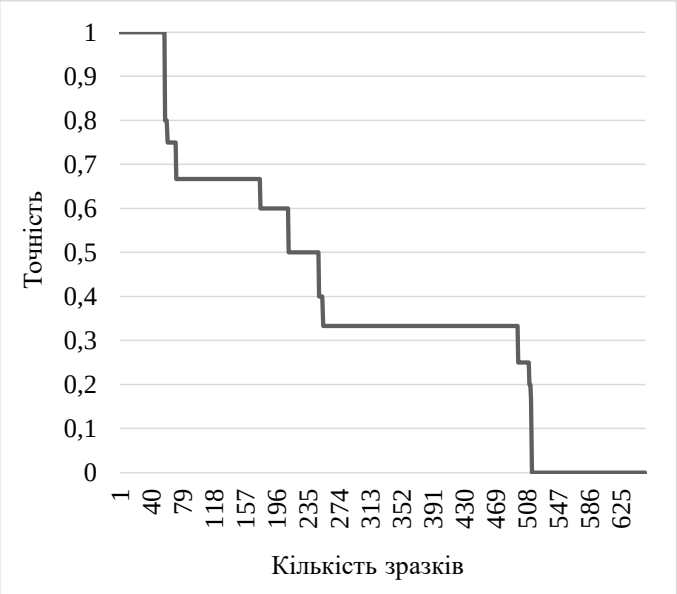

Рисунок 3.15 - Згруповані значення точності для модифікованого алгоритму, реалізованого на мові програмування Java

На рис. 3.4 представлені результати небінарної класифікації, де значення точності "1", означає, що всі класи були класифіковані правильно, значення точності "0,5" означає, що 50% класів були класифіковані правильно, значення точності "0" означає, що жоден з класів не був правильно класифікований. Як видно з рисунку, ~70 зразків були класифіковані правильно, ~250 зразків були класифіковані з високою точністю, ~500 зразків були класифіковані з достатньою точністю. На рисунку чітко показана позитивна статистика класифікації.

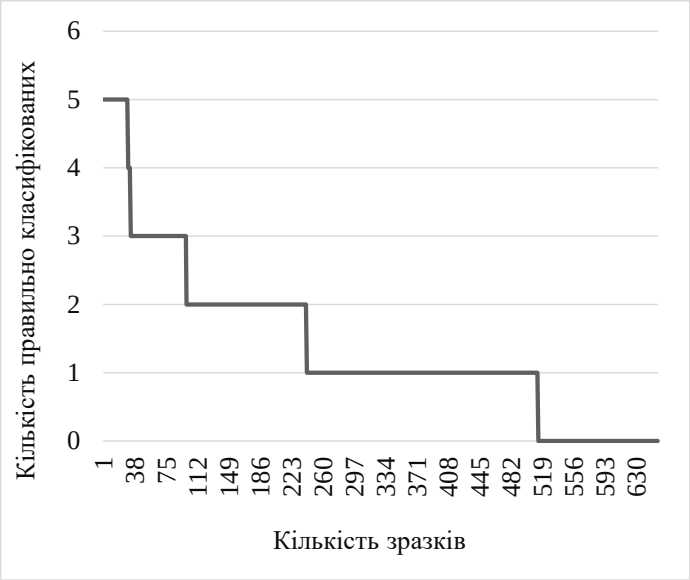

Рисунок 3.16 - Згруповані значення кількості правильно класифікованих зразків для модифікованого алгоритму, реалізованого на мові програмування Java

На рисунку 3.5 також представлені результати небінарної класифікації. Як видно на малюнку, 5 класів були правильно класифіковані в ~30 зразках, більш ніж 3 класи були правильно класифіковані в ~90 зразках, в тому числі в ~250 зразках більш ніж на 2 класи були правильно класифіковані, також в ~510 зразках було правильно класифіковано більше, ніж 1-й клас включно. На рисунку чітко показана позитивна статистика класифікації.

3.3 Висновки до розділу

В розділі описане проведення досліджень, використовуючи реалізовані моделі і алгоритми. Були перевірені всі моделі, описані в розділі 2, в тому числі розроблені автором були перевірені на якість роботи в багатокласовому режимі. Також було описане отримання результатів і їх збір. Отримано 79% точності для багатокласової класифікації і 75% для бінарної класифікації.

ВИСНОВКИ

Алгоритми класифікації постійно удосконалюються, поєднуються одне з одним, використовуються в різних галузях та є актуальними в різних країнах та регіонах світу. Найвідоміші методи класифікації – це дерева рішень, нейронні мережі та метод опорних векторів. Ці методи часто використовуються для класифікації чи для оцінки роботи нових чи вже існуючих методів. В розділі був проведений огляд та аналітика стану знань на тему класифікації текстової інформації. Також дається огляд нечітких множин, їх практичне застосування і основні операції з нечіткими множинами. Тема роботи є актуальною та вартою дослідження.

Описано алгоритми та їх реалізації для вирішення завдання. Алгоритми і реалізації створені як на основі частково готових рішень, так і написані з нуля. Створені алгоритми на основі «мішку слів», нейронної мережі, дерева рішень, методу опорних векторів. Приведені схеми, лістинги алгоритмів, рисунки реалізацій. Реалізовані проекти в середовищі KNIME і консольна программа на Java.

Описано проведення досліджень, використовуючи реалізовані моделі і алгоритми. Моделі були перевірені на якість роботи в багатокласовому режимі. Також було описане отримання результатів і їх збір. Окремо були реалізовані два алгоритми класифікації, які засновані на нечітких множинах і можуть виконувати бінарну і багтокласову класифікацію. Алгоритми були реалізовані на мові програмування Java і протестовані з ключовими словами в якості вхідних даних, правильно класифікуючи 79% для багатокласової класифікації і 75% для бінарної класифікації. В результаті порівняння алгоритми, розроблені автором, дали найвищі значення точності.

Розроблені алгоритми пропонується використовувати в базах даних для автоматичної класифікації текстів за атрибутами (за їх ключовими словами). Розроблені алгоритми можуть бути поліпшені, застосовуючи їх разом з іншими популярними алгоритмами або використовуючи нові методи для перекладу нечітких множин в звичайні. Також можливо об'єднати алгоритм з алгоритмами пошуку за ключовими словами, щоб отримати систему класифікації для всього тексту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Breiman L., Friedman R., Olshen R., Stone C., Classification and regression trees, Pacific Grove: Wadsworth & Brooks, 1984

2. Hunt E., Marin J., Stone P., Experiments in induction, New York: Academic, 1996

3. Kass G, An exploratory technique for investigating large quantities of categorical data, Applied Statistics(29), 1980

4. Michalski R., Carbonell J., Mitchell T., Machine learning, An artificial intelligence approach, 1983

5. Quinlan R., Induction of decision trees, Machine Learning(1), 1986

6. Boser B., Guyon I. i Vapnik V., A training algorithm for optimal margin classifiers, Proceedings of annual conference computational learning theory, 1992

7. Cortes C., Vapnik V., Support vector networks, Machine Learning, 3(20), 1995

8. Joachims T., Text categorization with support vector machines: Learning with many relevant features, In Proceedings of the European conference on machine learning, 1998

9. Shawe-Taylor J., Cristiani N., Margin distribution and soft margin, Cambridge: MIT Press, 2000

10. Wang H.-L., Yue L., Zhao P.-P., Cui Z.-M. Hierachical fuzzy set-based deep Web source classification, International Conference On Computer Design and Applications, 2010

11. Dreyfus S, Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure, Journal of Guidance, Control, and Dynamics, 5(13), 1990

12. Mizutani E., Dreyfus S., Nishio K., On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application, Proceedings

of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. Neural computing: New Chllenges and Perspectives for the New Millenium 2, 2000

13. Rosenblatt F., The perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review, 1958

14. Schmidhuber J., Deep Learning in Neural Networks: An Overview, Neural Networks(61), 2015

15. Scopus Open database, Elsevier B.V, [Електронний засіб] URL: https://www.scopus.com (дата доступу z 04.01.2021)

16. Liu H., Burnap P., Alorainy W., Williams M., A fuzzy approach to text classification with two-stage training for ambiguous instances, IEEE Transactions on Computational Social Systems, 2019

17. Shannon C., A mathematical theory of communication, The Bell System Technical Journal 27(3), 1948

18. Matthew A., Jamal S., Fuzzy mapping model for classification in supervised learning, IEEE Region 10 Conference (TENCON 2019), 2019

19. Mac Parthalain N., Jensen R., Unsupervised fuzzy-rough set-based dimensionality reduction, Information Sciences(229), 2013

20. Intarapaiboon P, An application of intuitionistic fuzzy sets in text classification, International Conference on Information Science, Electronics and Electrical Engineering, 2014

21. Modaresi P., Conrad S., From phrases to keyphrases: an unsupervised fuzzy set approach to summarize news articles, Proceedings of the 12th International Conference on Advances in Mobile Computing and Multimedia – MoMM, 2014

22. Jiang T., Yuan B., Jiang J., Yu H, Short text sentiment entropy optimization based on the fuzzy sets, 12th Web Information System and Application Conference, 2015

23. Nida Usel V., Sarac Essiz E., Ayse Osel S., Using fuzzy sets for detecting cyber terrorism and extremism in the text, Innovations in Intelligent Systems and Applications Conference (ASYU), 2018

24. Zadeh L. A., Fuzzy Sets, Information and Control, 1965

25. Kass G, An exploratory technique for investigating large quantities of categorical data, Applied Statistics(29), 1980

26. KNIME, KNIME AG, [Електронний засіб] URL: https://www.knime.com/, (дата доступу z 04.01.2021)zasoby z 06.05.2021

27. Salahor D., Smołka J., Model of the text classification system using fuzzy sets, Journal of Computer Sciences Institute, 2021

28. Oracle Java Documentation, Oracle, [Електронний засіб] URL: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html (дата доступу 06.05.2021)

29. Java Oracle Documentation, Oracle, [Електронний засіб] URL: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html (дата доступу z 06.03.2021)

ДОДАТОК А

Код програми реалізації класифікатора

```
import com.opencsv.CSVReader;
import java.io.IOException;
import java.io.Reader;
import java.net.URISyntaxException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;
import java.util.stream.Collectors;
import static java.util.Arrays.asList;

public class MagisterWorkClean {
    public static void main(String[] args) {
        AlgorythmType type;
        KeywordCount set;
        if (args.length < 2){
            printHelp();
            return;
        }else{
            if(args[0].equals("default")){
                type = AlgorythmType.DEFAULT;
            }else if(args[0].equals("advanced")){
                type = AlgorythmType.ADVANCED;
            }else{
                printHelp();
                return;
            }
            //ONE, THREE_FIRST, THREE_LAST, THREE_RANDOM, ALL
            if(args[1].equals("first")){
                set = KeywordCount.ONE;
            }else if(args[1].equals("three_first")){
                set = KeywordCount.THREE_FIRST;
            }else if(args[1].equals("three_last")){
                set = KeywordCount.THREE_LAST;
            }else if(args[1].equals("three_random")){
                set = KeywordCount.THREE_RANDOM;
            }else if(args[1].equals("all")){
                set = KeywordCount.ALL;
```

```
            }else{
                printHelp();
                return;
            }
        }
        //set with all data from csv file
        ArrayList<ArrayList<String>> allCsvs = new
ArrayList<ArrayList<String>>();
        //sets for storing model data (frequency maps) for default and advanced
algorithm
        HashMap<String, HashMap<String, Double>> freqCountMap = new
HashMap<>();
        HashMap<String, HashMap<String, Double>> advFreqCountMap = new
HashMap<>();
        ArrayList<ArrayList<String>> testKeyWords = new ArrayList<>();
        ArrayList<ArrayList<String>> testSubjectAreas = new ArrayList<>();
        ArrayList<ArrayList<String>> testAdvKeyWords = new ArrayList<>();
        ArrayList<ArrayList<String>> testAdvSubjectAreas = new ArrayList<>();
        //read data file
        Reader reader = null;
        try {
            reader =
Files.newBufferedReader(Paths.get(ClassLoader.getSystemResource("dbresult_concate
d" + ".csv").toURI()));
        } catch (IOException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
        try {
            //create frequency maps for default and advanced algorithms using reader
of data file
            createFrequencyMapAndTrainForCsv(reader, allCsvs,
freqCountMap,testKeyWords, testSubjectAreas, 20);
            createAdvancedFrequencyAndTrainMapForCsv(reader, allCsvs,
advFreqCountMap, testAdvKeyWords, testAdvSubjectAreas, 20);
        } catch (Exception e) {
            e.printStackTrace();
        }
        ArrayList<ArrayList<String>> preparedTestKeyWords = new
ArrayList<>();
        ArrayList<ArrayList<String>> preparedTestSubjectAreas = new
ArrayList<>();
        ArrayList<ArrayList<String>> preparedAdvTestKeyWords = new
ArrayList<>();
```

```
            ArrayList<ArrayList<String>> preparedAdvTestSubjectAreas = new
ArrayList<>();
            if(set != KeywordCount.ALL){
               //get need count of key words. for example one first, three first, three last,
three random, all.
        prepareTestData(testKeyWords,testSubjectAreas,preparedTestKeyWords,prepare
dTestSubjectAreas, set);
        prepareAdvancedTestData(testKeyWords,testSubjectAreas,preparedAdvTestKey
Words,preparedAdvTestSubjectAreas,set);
            }
            try {
               reader.close();
            } catch (IOException e) {
               e.printStackTrace();
            }
            //
            ArrayList<Double> accuracy = new ArrayList<>();
            ArrayList<Double> advAccuracy = new ArrayList<>();
            //
            ArrayList<Integer> fetchedCount = new ArrayList<>();
            ArrayList<Integer> advFetchedCount = new ArrayList<>();
            //
            ArrayList<Integer> noFetchedCount = new ArrayList<>();
            ArrayList<Integer> advNoFetchedCount = new ArrayList<>();
            if (type == AlgorythmType.DEFAULT){
               if (set == KeywordCount.ALL) {
                  //if u wanna use all keywords
                  getAllFrequencies(freqCountMap, testKeyWords, testSubjectAreas,
accuracy, fetchedCount, noFetchedCount);
               }else{
                  //if u dont wanna use all keywords
                  getAllFrequencies(freqCountMap, preparedTestKeyWords,
preparedTestSubjectAreas, accuracy, fetchedCount, noFetchedCount);
               }
               //print accuracy results for default algorithm
               printResults(accuracy, fetchedCount, noFetchedCount);
            }else if(type == AlgorythmType.ADVANCED){
               if (set == KeywordCount.ALL) {
                  //if u wanna use all keywords
                  getAllAdvFrequencies(advFreqCountMap, testAdvKeyWords,
testAdvSubjectAreas, advAccuracy, advFetchedCount, advNoFetchedCount);
               }else{
                  //if u dont wanna use all keywords
```

```
                    getAllAdvFrequencies(advFreqCountMap,
preparedAdvTestKeyWords, preparedAdvTestSubjectAreas, advAccuracy,
advFetchedCount, advNoFetchedCount);
                }
                //print accuracy results for advanced algorithm
                printResults(advAccuracy, advFetchedCount, advNoFetchedCount);
            }
        }

        private static void printHelp() {
            System.out.println("Use parameters: first parameters is type of algorythm,
second is type of keyword set.");
            System.out.println("Types of an algorythm: default, advanced.");
            System.out.println("Types of keyword set: one, three_first, three_last,
three_random, all.");
            System.out.println("Examples of use: MagisterWorkClean default all,
MagisterWorkClean advanced three_random.");
        }

        private static void printResults(ArrayList<Double> accuracy,
ArrayList<Integer> fetchedCount, ArrayList<Integer> noFetchedCount) {
            accuracy.stream()
                    .map(map -> map.doubleValue() + " ")
                    .forEach(System.out::print);
            System.out.println();
            fetchedCount.stream()
                    .map(map -> map + " ")
                    .forEach(System.out::print);
            System.out.println();
            noFetchedCount.stream()
                    .map(map -> map + " ")
                    .forEach(System.out::print);
            System.out.println();
            OptionalDouble average = accuracy
                    .stream()
                    .mapToDouble(a -> a)
                    .average();
            System.out.println("Average accuracy for algorithm: " +
average.getAsDouble());
            System.out.println("----------------");
        }

        private static void prepareTestData(ArrayList<ArrayList<String>>
oldKeyWords, ArrayList<ArrayList<String>> oldSubjectAreas,
```

```
ArrayList<ArrayList<String>> preparedTestKeyWords, ArrayList<ArrayList<String>>
preparedTestSubjectAreas, KeywordCount set) {
        if(oldKeyWords.size() != oldSubjectAreas.size()){
            throw new IllegalArgumentException("subject areas array and keywords
array sizes not match!");
        }
        for (int i = 0; i < oldKeyWords.size(); i++){
            if(oldKeyWords.get(i).isEmpty()){
                continue;
            }
            ArrayList<String> keyWords = new ArrayList<>();
            if(set == KeywordCount.ONE){
                //if u wanna get only first keyword
                keyWords.add(oldKeyWords.get(i).get(0));
            }else if(set == KeywordCount.THREE_FIRST){
                //if u wanna get only 3  keyword at first
                keyWords.add(oldKeyWords.get(i).get(0));
                if (oldKeyWords.get(i).size() > 1) {
                    keyWords.add(oldKeyWords.get(i).get(1));
                }
                if (oldKeyWords.get(i).size() > 2) {
                    keyWords.add(oldKeyWords.get(i).get(2));
                }
            }else if(set == KeywordCount.THREE_LAST){
                //if u wanna get only 3 keyword at last, uncomment this
                int size = oldKeyWords.get(i).size();
                keyWords.add(oldKeyWords.get(i).get(size-1));
                if (size - 2 >= 0) {
                    keyWords.add(oldKeyWords.get(i).get(size - 2));
                }
                if (size - 3 >= 0) {
                    keyWords.add(oldKeyWords.get(i).get(size - 3));
                }
            }else if(set == KeywordCount.THREE_RANDOM){
                //if u wanna get three random keywords, uncomment this
                int size = oldKeyWords.get(i).size();
                if(size == 1){
                    keyWords.add(oldKeyWords.get(i).get(0));
                }
                else if (size == 2){
                    keyWords.add(oldKeyWords.get(i).get(0));
                    keyWords.add(oldKeyWords.get(i).get(1));
                }
                else if (size == 3) {
                    keyWords.add(oldKeyWords.get(i).get(0));
```

```
                    keyWords.add(oldKeyWords.get(i).get(1));
                    keyWords.add(oldKeyWords.get(i).get(2));
                }else{
                    keyWords.addAll(pickNRandomElements(oldKeyWords.get(i), 3,
new Random()));
                }
            }
            preparedTestKeyWords.add(keyWords);
            ArrayList<String> subjectAreas = new ArrayList<>();
            if(oldSubjectAreas.get(i).contains("Engineering")){
                subjectAreas.add("Engineering");
            }else{
            }
            preparedTestSubjectAreas.add(subjectAreas);
        }
    }

    private static void prepareAdvancedTestData(ArrayList<ArrayList<String>>
oldKeyWords, ArrayList<ArrayList<String>> oldSubjectAreas,
ArrayList<ArrayList<String>> preparedTestKeyWords, ArrayList<ArrayList<String>>
preparedTestSubjectAreas, KeywordCount set) {
        if(oldKeyWords.size() != oldSubjectAreas.size()){
            throw new IllegalArgumentException("subject areas array and keywords
array sizes not match!");
        }
        for (int i = 0; i < oldKeyWords.size(); i++) {
            ArrayList<String> SingleKeywords = new ArrayList<String>();
            //here i choose not all keywords, but only needed count
            if (oldKeyWords.get(i).isEmpty()) {
                continue;
            }
            if(set == KeywordCount.ONE){
                //if u wanna get only first keyword
                SingleKeywords.add(oldKeyWords.get(i).get(0));
            }else if(set == KeywordCount.THREE_FIRST){
                //if u wanna get only 3  keyword at first, uncomment this
                SingleKeywords.add(oldKeyWords.get(i).get(0));
                if (oldKeyWords.get(i).size() > 1) {
                    SingleKeywords.add(oldKeyWords.get(i).get(1));
                }
                if (oldKeyWords.get(i).size() > 2) {
                    SingleKeywords.add(oldKeyWords.get(i).get(2));
                }
            }else if(set == KeywordCount.THREE_LAST){
                //if u wanna get only 3 keyword at last, uncomment this
```

```
                int size = oldKeyWords.get(i).size();
                SingleKeywords.add(oldKeyWords.get(i).get(size-1));
                if (size - 2 >= 0) {
                    SingleKeywords.add(oldKeyWords.get(i).get(size - 2));
                }
                if (size - 3 >= 0) {
                    SingleKeywords.add(oldKeyWords.get(i).get(size - 3));
                }
            }else if(set == KeywordCount.THREE_RANDOM){
                //if u wanna get three random keywords, uncomment this
                int size = oldKeyWords.get(i).size();
                if(size == 1){
                    SingleKeywords.add(oldKeyWords.get(i).get(0));
                }
                else if (size == 2){
                    SingleKeywords.add(oldKeyWords.get(i).get(0));
                    SingleKeywords.add(oldKeyWords.get(i).get(1));
                }
                else if (size == 3) {
                    SingleKeywords.add(oldKeyWords.get(i).get(0));
                    SingleKeywords.add(oldKeyWords.get(i).get(1));
                    SingleKeywords.add(oldKeyWords.get(i).get(2));
                }else{
                  SingleKeywords.addAll(pickNRandomElements(oldKeyWords.get(i),
3, new Random()));
                }
            }
            ArrayList<String> subjectAreas = new ArrayList<>();
            if (oldSubjectAreas.get(i).contains("Engineering")) {
                subjectAreas.add("Engineering");
            }
            preparedTestSubjectAreas.add(subjectAreas);
            //if uncomment this then we get all words, not only needed
            //SingleKeywords.addAll(singleKeywords);
            SingleKeywords.replaceAll(String::trim);
            SingleKeywords.replaceAll(String::toLowerCase);
            ArrayList<String> allSingleKeyWords = new ArrayList<>();
            for (String keyWord : SingleKeywords) {
                ArrayList<String> keyWs = new ArrayList<>(asList(keyWord.split(" |-
|\\(|\\)|\\.|,|/")));
                allSingleKeyWords.addAll(keyWs);
            }
            preparedTestKeyWords.add(allSingleKeyWords);
        }
        for (int i = 0; i < oldKeyWords.size(); i++){
```

```
                if(oldKeyWords.get(i).isEmpty()){
                    continue;
                }
                ArrayList<String> keyWords = new ArrayList<>();
                keyWords.add(oldKeyWords.get(i).get(0));
                if (oldKeyWords.get(i).size() > 1){
                    keyWords.add(oldKeyWords.get(i).get(1));
                }
                if (oldKeyWords.get(i).size() > 2){
                    keyWords.add(oldKeyWords.get(i).get(2));
                }
                preparedTestKeyWords.add(keyWords);
                ArrayList<String> subjectAreas = new ArrayList<>();
                if(oldSubjectAreas.get(i).contains("Engineering")){
                    subjectAreas.add("Engineering");
                }
                preparedTestSubjectAreas.add(subjectAreas);
            }
        }

        private static void getAllFrequencies(HashMap<String, HashMap<String,
Double>> freqCountMap, ArrayList<ArrayList<String>> testKeyWords,
        ArrayList<ArrayList<String>> testSubjectAreas, ArrayList<Double> accuracy,
        ArrayList<Integer> fetchedCount, ArrayList<Integer> noFetchedCount) {
            for (int i = 0; i < testKeyWords.size(); i++) {
                ArrayList<String> keywords = testKeyWords.get(i);
                HashMap<String, Double> freqResult =
getFrequencyForKeyWords(keywords, freqCountMap);
                List<String> resultSubjectAreas = freqResult.entrySet().stream()
                        .sorted(Map.Entry.comparingByValue())
                        .filter(map -> map.getValue().doubleValue() > 0.5 /*1. /
Main.names.size()*/)
                        .map(map -> map.getKey())
                        .collect(Collectors.toList());
                double acc = calculateAccuracyForSubjectAreas(resultSubjectAreas,
testSubjectAreas.get(i));
                accuracy.add(acc);
                int fetchedCnt =
calculateFetchedCountForSubjectAreas(resultSubjectAreas, testSubjectAreas.get(i));
                fetchedCount.add(fetchedCnt);
                int noFetchedCnt =
calculateNoFetchedCountForSubjectAreas(resultSubjectAreas, testSubjectAreas.get(i));
                noFetchedCount.add(noFetchedCnt);
            }
        }
```

```
        private static void getAllAdvFrequencies(HashMap<String, HashMap<String,
Double>> freqCountMap, ArrayList<ArrayList<String>> testKeyWords,
                                        ArrayList<ArrayList<String>> testSubjectAreas,
ArrayList<Double> accuracy, ArrayList<Integer> advFetchedCount,
                                        ArrayList<Integer> advNoFetchedCount) {
            for (int i = 0; i < testKeyWords.size(); i++) {
                ArrayList<String> keywords = testKeyWords.get(i);
                HashMap<String, Double> freqResult =
getAdvancedFrequencyForKeyWords(keywords, freqCountMap);
                List<String> resultSubjectAreas = freqResult.entrySet().stream()
                        .sorted(Map.Entry.comparingByValue())
                        .filter(map -> map.getValue().doubleValue() >  0.5 /*1. /
Main.names.size()*/)
                        .map(map -> map.getKey())
                        .collect(Collectors.toList());
                double acc = calculateAccuracyForSubjectAreas(resultSubjectAreas,
testSubjectAreas.get(i));
                accuracy.add(acc);
                int fetchedCnt =
calculateFetchedCountForSubjectAreas(resultSubjectAreas, testSubjectAreas.get(i));
                advFetchedCount.add(fetchedCnt);
                int advNoFetchedCnt =
calculateNoFetchedCountForSubjectAreas(resultSubjectAreas, testSubjectAreas.get(i));
                advNoFetchedCount.add(advNoFetchedCnt);
            }

        }
        private static int calculateNoFetchedCountForSubjectAreas(List<String>
resultSubjectAreas, ArrayList<String> realSubjectAreas) {
            if(realSubjectAreas.size() == 0){
                if(resultSubjectAreas.size() == 0){
                    return 0;
                }
                return -1;
            }
            if(resultSubjectAreas.size() == 0){
                return 0;
            }
            if(realSubjectAreas.size() > resultSubjectAreas.size()){
                return realSubjectAreas.size() - intersection(resultSubjectAreas,
realSubjectAreas).size();
            }else{
                return resultSubjectAreas.size() - intersection(resultSubjectAreas,
realSubjectAreas).size();
```

```
            }
        }

        private static double calculateAccuracyForSubjectAreas(List<String>
resultSubjectAreas, ArrayList<String> realSubjectAreas) {
            if(realSubjectAreas.size() == 0){
                if(resultSubjectAreas.size() == 0){
                    return 1;
                }
                return 0;
            }
            if(resultSubjectAreas.size() == 0){
                return 0;
            }
            //for one result
            if(resultSubjectAreas.contains(realSubjectAreas.get(0))){
                return 1;
            }else{
                return 0;
            }
            //for many results
    //        if(realSubjectAreas.size() > resultSubjectAreas.size()){
    //            return (double)intersection(resultSubjectAreas, realSubjectAreas).size() /
realSubjectAreas.size();
    //        }else{
    //            return (double)intersection(resultSubjectAreas, realSubjectAreas).size() /
resultSubjectAreas.size();
    //        }
            }

        private static int calculateFetchedCountForSubjectAreas(List<String>
resultSubjectAreas, ArrayList<String> realSubjectAreas) {
            if(realSubjectAreas.size() == 0){
                if(resultSubjectAreas.size() == 0){
                    return 0;
                }
                return -1;
            }
            if(resultSubjectAreas.size() == 0){
                return 0;
            }
            return intersection(resultSubjectAreas, realSubjectAreas).size();
        }
```

```
        public static void createFrequencyMapAndTrainForCsv(Reader reader,
ArrayList<ArrayList<String>> list, HashMap<String, HashMap<String, Double>>
freqCountMap, ArrayList<ArrayList<String>> testKeyWords,
ArrayList<ArrayList<String>> testSubjectAreas, int trainPercent) throws Exception {
            CSVReader csvReader = new CSVReader(reader);
            String[] line;
            boolean isFirst = true;
            while ((line = csvReader.readNext()) != null) {
               ArrayList<String> strings = new ArrayList<String>(asList(line));
               if(isFirst){
                  isFirst = false;
                  continue;
               }
               list.add(strings);
            }
            csvReader.close();
            System.out.println("Strings in csv = " + list.size());
            ArrayList<String> keyWords = new ArrayList<>();
            int stringIndex = 0;
            for (ArrayList<String> csvLine: list) {
               if(stringIndex % trainPercent == 0){
                  //read to test list
                  ArrayList<String> testAuthorKeywords = new
ArrayList<String>(asList(csvLine.get(3).split(";")));
                  testAuthorKeywords.replaceAll(String::trim);
                  testAuthorKeywords.replaceAll(String::toLowerCase);
                  testAuthorKeywords.removeAll(Arrays.asList("", null));
                  ArrayList<String> testIndexKeywords = new
ArrayList<String>(asList(csvLine.get(4).split(";")));
                  testIndexKeywords.replaceAll(String::trim);
                  testIndexKeywords.replaceAll(String::toLowerCase);
                  testIndexKeywords.removeAll(Arrays.asList("", null));
                  ArrayList<String> testSubjectAreasForLine = new
ArrayList<String>(asList(csvLine.get(5).split(";")));
                  testSubjectAreasForLine.replaceAll(String::trim);
                  ArrayList<String> testResultKeyWords = new ArrayList<>();
                  testResultKeyWords.addAll(testIndexKeywords);
                  testResultKeyWords.addAll(testAuthorKeywords);
                  testKeyWords.add(testResultKeyWords);
                  testSubjectAreas.add(testSubjectAreasForLine);
               }else{
                  ArrayList<String> authorKeywords = new
ArrayList<String>(asList(csvLine.get(3).split(";")));
                  authorKeywords.replaceAll(String::trim);
                  authorKeywords.replaceAll(String::toLowerCase);
```

```
                ArrayList<String> indexKeywords = new
ArrayList<String>(asList(csvLine.get(4).split(";")));
                indexKeywords.replaceAll(String::trim);
                indexKeywords.replaceAll(String::toLowerCase);

                keyWords.addAll(authorKeywords);
                keyWords.addAll(indexKeywords);
            }

            stringIndex++;
        }
        //remove duplicates
        ArrayList<String> uniqueKeywords = new ArrayList<String>(new
HashSet<>(keyWords));
        int stringIndex2 = 0;
        for (ArrayList<String> csvLine: list) {
            if(stringIndex2 % trainPercent == 0){
                //passing test values
            }else{
                ArrayList<String> keyWordsForLine = new ArrayList<>();
                ArrayList<String> authorKeywords = new
ArrayList<String>(asList(csvLine.get(3).split(";")));
                authorKeywords.replaceAll(String::trim);
                authorKeywords.replaceAll(String::toLowerCase);
                ArrayList<String> indexKeywords = new
ArrayList<String>(asList(csvLine.get(4).split(";")));
                indexKeywords.replaceAll(String::trim);
                indexKeywords.replaceAll(String::toLowerCase);
                keyWordsForLine.addAll(authorKeywords);
                keyWordsForLine.addAll(indexKeywords);
                ArrayList<String> subjectAreas = new
ArrayList<String>(asList(csvLine.get(5).split(";")));
                subjectAreas.replaceAll(String::trim);
                for (String key: keyWordsForLine) {
                    if(freqCountMap.get(key) == null){
                        HashMap<String, Double> emptyMap = new HashMap<>();
                        for (String subjectArea: Main.names
                        ) {
                            emptyMap.put(subjectArea, new Double(0));
                        }
                        freqCountMap.put(key, emptyMap);
                        for (String localSubjectArea: subjectAreas
                        ) {
                            HashMap<String, Double> localMap = freqCountMap.get(key);
```

```
                        localMap.put(localSubjectArea,localMap.get(localSubjectArea)
+ 1);
                        freqCountMap.put(key, localMap);
                    }
                }else{
                    for (String localSubjectArea: subjectAreas
                    ) {
                        HashMap<String, Double> localMap = freqCountMap.get(key);
                        localMap.put(localSubjectArea,localMap.get(localSubjectArea)
+ 1);
                        freqCountMap.put(key, localMap);
                    }
                }
            }
        }
        stringIndex2++;
    }
    System.out.println("Strings in testKeyWords = " + testKeyWords.size());
    System.out.println("Strings in testSubjectAreas = " +
testSubjectAreas.size());
}

//here used keyWords not keyPhrases
public static void createAdvancedFrequencyAndTrainMapForCsv(Reader
reader, ArrayList<ArrayList<String>> list,
HashMap<String, HashMap<String, Double>> freqCountMap,
ArrayList<ArrayList<String>> testAdvKeyWords,
ArrayList<ArrayList<String>> testAdvSubjectAreas,
int trainPercent) throws Exception {
    CSVReader csvReader = new CSVReader(reader);
    String[] line;
    boolean isFirst = true;
    while ((line = csvReader.readNext()) != null) {
        ArrayList<String> strings = new ArrayList<String>(asList(line));
        if(isFirst){
            isFirst = false;
            continue;
        }
        list.add(strings);
    }
    csvReader.close();
    ArrayList<String> keyWords = new ArrayList<>();
    int stringIndex = 0;
    for (ArrayList<String> csvLine: list) {
        if(stringIndex % trainPercent == 0){
```

```
                //read to test list
                ArrayList<String> authKeywords = new
ArrayList<String>(asList(csvLine.get(3).split(";")));
                authKeywords.replaceAll(String::trim);
                authKeywords.replaceAll(String::toLowerCase);
                ArrayList<String> testAuthorKeywords = new ArrayList<String>();
                for (String keyPhrase:authKeywords) {
                   ArrayList<String> keyWs = new
ArrayList<>(asList(keyPhrase.split(" |-|\\(|\\)|\\.|,|/")));
                   testAuthorKeywords.addAll(keyWs);
                }
                ArrayList<String> indKeywords = new
ArrayList<String>(asList(csvLine.get(4).split(";")));
                indKeywords.replaceAll(String::trim);
                indKeywords.replaceAll(String::toLowerCase);
                ArrayList<String> testIndexKeywords = new ArrayList<String>();
                for (String keyPhrase:indKeywords) {
                   ArrayList<String> keyWs = new
ArrayList<>(asList(keyPhrase.split(" |-|\\(|\\)|\\.|,|/")));
                   testIndexKeywords.addAll(keyWs);
                }
                ArrayList<String> testSubjectAreasForLine = new
ArrayList<String>(asList(csvLine.get(5).split(";")));
                testSubjectAreasForLine.replaceAll(String::trim);
                ArrayList<String> testResultKeyWords = new ArrayList<>();
                testResultKeyWords.addAll(testIndexKeywords);
                testResultKeyWords.addAll(testAuthorKeywords);
                testAdvKeyWords.add(testResultKeyWords);
                testAdvSubjectAreas.add(testSubjectAreasForLine);
             }else{
                ArrayList<String> authKeywords = new
ArrayList<String>(asList(csvLine.get(3).split(";")));
                authKeywords.replaceAll(String::trim);
                authKeywords.replaceAll(String::toLowerCase);
                ArrayList<String> authorKeywords = new ArrayList<String>();
                for (String keyPhrase:authKeywords) {
                   ArrayList<String> keyWs = new
ArrayList<>(asList(keyPhrase.split(" |-|\\(|\\)|\\.|,|/")));
                   authorKeywords.addAll(keyWs);
                }
                ArrayList<String> indKeywords = new
ArrayList<String>(asList(csvLine.get(4).split(";")));
                indKeywords.replaceAll(String::trim);
                indKeywords.replaceAll(String::toLowerCase);
                ArrayList<String> indexKeywords = new ArrayList<String>();
```

```
                for (String keyPhrase:indKeywords) {
                    ArrayList<String> keyWs = new
ArrayList<>(asList(keyPhrase.split(" |-|\\(|\\)|\\.|,|/")));
                    indexKeywords.addAll(keyWs);
                }
                keyWords.addAll(authorKeywords);
                keyWords.addAll(indexKeywords);
            }
            stringIndex++;
        }
        //remove duplicates
        ArrayList<String> uniqueKeywords = new ArrayList<String>(new
HashSet<>(keyWords));
        int stringIndex2 = 0;
        for (ArrayList<String> csvLine: list) {
            if(stringIndex2 % trainPercent == 0){
                //passing test value
            }else{
                ArrayList<String> keyWordsForLine = new ArrayList<>();
                ArrayList<String> authKeywords = new
ArrayList<String>(asList(csvLine.get(3).split(";")));
                authKeywords.replaceAll(String::trim);
                authKeywords.replaceAll(String::toLowerCase);
                ArrayList<String> authorKeywords = new ArrayList<String>();
                for (String keyPhrase:authKeywords) {
                    ArrayList<String> keyWs = new
ArrayList<>(asList(keyPhrase.split(" |-|\\(|\\)|\\.|,|/")));
                    authorKeywords.addAll(keyWs);
                }

                ArrayList<String> indKeywords = new
ArrayList<String>(asList(csvLine.get(4).split(";")));
                indKeywords.replaceAll(String::trim);
                indKeywords.replaceAll(String::toLowerCase);
                ArrayList<String> indexKeywords = new ArrayList<String>();
                for (String keyPhrase:indKeywords) {
                    ArrayList<String> keyWs = new
ArrayList<>(asList(keyPhrase.split(" |-|\\(|\\)|\\.|,|/")));
                    indexKeywords.addAll(keyWs);
                }
                keyWordsForLine.addAll(authorKeywords);
                keyWordsForLine.addAll(indexKeywords);
                ArrayList<String> subjectAreas = new
ArrayList<String>(asList(csvLine.get(5).split(";")));
                subjectAreas.replaceAll(String::trim);
```

```
                for (String key: keyWordsForLine) {
                   if(freqCountMap.get(key) == null){
                      HashMap<String, Double> emptyMap = new HashMap<>();
                      for (String subjectArea: Main.names
                      ) {
                         emptyMap.put(subjectArea, new Double(0));
                      }
                      freqCountMap.put(key, emptyMap);
                      for (String localSubjectArea: subjectAreas
                      ) {
                         HashMap<String, Double> localMap = freqCountMap.get(key);
                         localMap.put(localSubjectArea,localMap.get(localSubjectArea)
+ 1);
                         freqCountMap.put(key, localMap);
                      }
                   }else{
                      for (String localSubjectArea: subjectAreas
                      ) {
                         HashMap<String, Double> localMap = freqCountMap.get(key);
                         localMap.put(localSubjectArea,localMap.get(localSubjectArea)
+ 1);
                         freqCountMap.put(key, localMap);
                      }
                   }
                }
             }
             stringIndex2++;
          }
          System.out.println("Strings in testAdvKeyWords = " +
testAdvKeyWords.size());
          System.out.println("Strings in testAdvSubjectAreas = " +
testAdvSubjectAreas.size());
       }

       public static HashMap<String, Double>
getFrequencyForKeyWords(ArrayList<String> kWords, HashMap<String,
HashMap<String, Double>> freqCountMap){
          ArrayList<HashMap<String, Double>> hashMaps = new ArrayList<>();
          HashMap<String, Double> resultMap = new HashMap<>();
          ArrayList<String> aboba;
          //initializing Out by zeros
          for (String subjectArea: Main.names
          ) {
             resultMap.put(subjectArea, new Double(0));
          }
```

```
            //lowercase keywords
            ArrayList<String> keyWords = new ArrayList<String>(kWords
                  .stream()
                  .map(String::toLowerCase)
                  .collect(Collectors.toList()));
            //keyWords.stream().forEach(System.out::println);
            //return empty map if no keywords
            if(keyWords.size() == 0){
               HashMap<String, Double> emptyMap = new HashMap<>();
               for (String subjectArea: Main.names
               ) {
                  emptyMap.put(subjectArea, new Double(0));
               }
               return emptyMap;
            }
            //Get weights of each subject area by keywords. Get List of (List of (Subject
area weights)))
            for (String keyWord: keyWords) {
               if(freqCountMap.get(keyWord) != null){
                  hashMaps.add(freqCountMap.get(keyWord));
               }
            }
            //summing all counts in one hashMap
            for (HashMap<String, Double> map: hashMaps) {
               for (Map.Entry<String, Double> entry: map.entrySet()) {
                  resultMap.put(entry.getKey(),
                        resultMap.get(entry.getKey()) + entry.getValue());
               }
            }
            //convert to fuzzySet (0..1)
            //get sum of points for all
            double power = 0;
            for (Map.Entry<String, Double> entry: resultMap.entrySet()) {
               power+=entry.getValue();
            }
            if(power == 0){
               power = 1;
            }
            HashMap<String, Double> fuzzyMap = new
HashMap<>(resultMap);//converted weights to [0..1]
            for (Map.Entry<String, Double> entry: fuzzyMap.entrySet()) {
               fuzzyMap.put(entry.getKey(), entry.getValue()/power);
            }
            //create multiplier to normalize fuzzy set
            Optional<Map.Entry<String, Double>> max = fuzzyMap.entrySet()
```

```
                .stream()
                .max((Map.Entry<String, Double> e1, Map.Entry<String, Double> e2)
-> e1.getValue()
                        .compareTo(e2.getValue())
                );
        double mult = 1. / max.get().getValue();
        HashMap<String, Double> normalizedFuzzyMap =
fuzzyMap.entrySet().stream().map(map -> {
            map.setValue(map.getValue()*mult);
            return map;
        }).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
(prev, next) -> next, HashMap::new));
        return normalizedFuzzyMap;

    }

    public static HashMap<String, Double>
getAdvancedFrequencyForKeyWords(ArrayList<String> kWords, HashMap<String,
HashMap<String, Double>> freqCountMap){
        ArrayList<HashMap<String, Double>> hashMaps = new ArrayList<>();
        HashMap<String, Double> resultMap = new HashMap<>();
        for (String subjectArea: Main.names
        ) {
            resultMap.put(subjectArea, new Double(0));
        }
        ArrayList<String> keyWs = new ArrayList<String>(kWords
                .stream()
                .map(String::toLowerCase)
                .collect(Collectors.toList()));
        ArrayList<String> keyWords = new ArrayList<String>();
        for (String keyPhrase:keyWs) {
            ArrayList<String> keys = new ArrayList<>(asList(keyPhrase.split(" |-
|\\(|\\)|\\.|,|/")));
            keyWords.addAll(keys);
        }
        //keyWords.stream().forEach(System.out::println);
        if(keyWords.size() == 0){
            HashMap<String, Double> emptyMap = new HashMap<>();
            for (String subjectArea: Main.names
            ) {
                emptyMap.put(subjectArea, new Double(0));
            }
            return emptyMap;
        }
        for (String keyWord: keyWords) {
```

```
            if(freqCountMap.get(keyWord) != null){
                hashMaps.add(freqCountMap.get(keyWord));
            }
        }
        //summing all counts in one hashMap
        for (HashMap<String, Double> map: hashMaps) {
            for (Map.Entry<String, Double> entry: map.entrySet()) {
                resultMap.put(entry.getKey(), resultMap.get(entry.getKey()) +
entry.getValue());
            }
        }
        //convert to fuzzySet (0..1)
        double power = 0;
        for (Map.Entry<String, Double> entry: resultMap.entrySet()) {
            power+=entry.getValue();
        }
        if(power == 0){
            power = 1;
        }
        HashMap<String, Double> fuzzyMap = new HashMap<>(resultMap);
        for (Map.Entry<String, Double> entry: fuzzyMap.entrySet()) {
            fuzzyMap.put(entry.getKey(), entry.getValue()/power);
        }
        Optional<Map.Entry<String, Double>> max = fuzzyMap.entrySet()
                .stream()
                .max((Map.Entry<String, Double> e1, Map.Entry<String, Double> e2)
-> e1.getValue()
                        .compareTo(e2.getValue())
                );
        double mult = 1. / max.get().getValue();
        HashMap<String, Double> normalizedFuzzyMap =
fuzzyMap.entrySet().stream().map(map -> {
            map.setValue(map.getValue()*mult);
            return map;
        }).collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
(prev, next) -> next, HashMap::new));
        return normalizedFuzzyMap;
    }

    public static List<String> intersection(List<String> list1, ArrayList<String>
list2) {
        List<String> list = new ArrayList<String>();
        for (String t : list1) {
            if(list2.contains(t)) {
                list.add(t);
```

```
            }
        }
        return list;
    }

    public static <E> List<E> pickNRandomElements(List<E> list, int n, Random
r) {
        int length = list.size();
        if (length < n) return null;
        //We don't need to shuffle the whole list
        for (int i = length - 1; i >= length - n; --i)
        {
            Collections.swap(list, i , r.nextInt(i + 1));
        }
        return list.subList(length - n, length);
    }
    public enum KeywordCount{
        ONE, THREE_FIRST, THREE_LAST, THREE_RANDOM, ALL
    }
    public enum AlgorythmType{
        DEFAULT, ADVANCED
    }
}
```