КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

Класифікація текстів натуральною мовою

Виконав студент 4 курсу

Єрковіч Марко Драганович _____
(підпис)

Науковий керівник:

професор, доктор фіз.-мат. наук

Марченко Олександр Олександрович _____
(підпис)

Засвідчую, що в цій дипломної роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
«____»_____202__р.

протокол №_____

Завідувач катедри

Терещенко В. М. _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 42 сторінок, 9 джерел посилань.

НАЇВНИЙ БАЄСІВ МЕТОД, ЛОГІСТИЧНА РЕГРЕСІЯ, ПОРІВНЯННЯ МЕТОДІВ, ГЕНЕРАТИВНА ТИ ДИСКРИМІНАНТНА КЛАСИФІКАЦІЯ, МАШИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ ТЕКСТІВ НАТУРАЛЬНОЮ МОВОЮ, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, МЕТОДИ ВИМІРЮВАННЯ ЯКОСТІ ЙМОВІРНІСНИХ КЛАСИФІКАТОРІВ.

Об’єктом роботи є класифікація текстів натуральною мовою.

Предметом роботи є програмні засоби для класифікації текстів природною мовою.

Метою роботи є створення програмних засобів для класифікації текстів природною мовою на основі генеративного наївного баєсівського методу та дискримінантного методу логістичної регресії машинного навчання на мові програмування Java та їх порівняння.

Методи розроблення: методи розробки програм машинного навчання. Інструменти розроблення: інтегроване середовище розробки Intelij IDEA, мова програмування Java 11.

Результати роботи: описано формалізацію генеративного наївного баєсівського методу та дискримінантного методу логістичної регресії машинного навчання для текстової класифікації. Описано методи вимірювання якості класифікації. Показано процес розробки архітектури та коду програми. Проведене тестування розроблених класифікаторів на двох класичних задачах текстової класифікації використовуючи тренувальні та тестові вибірки даних різних розмірів. Порівняння методів. Підводження висновків.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта дослідження. Обробка тексів натуральною мовою це велика сфера досліджень. Частина задач вже мають хороші рішення, але велика частина потребують подальшого дослідження. І немало програмних реалізацій класифікаторів існує, наприклад класифікатор на Java від Стендфордського університету оснований на методі максимальної ентропії, тому для коректної роботи потребує чималу тренувальну вибірку.

Актуальність роботи та підстави для її виконання. Робота ставить задачу покрокової наглядної розробки універсальних класифікаторів на мові Java з методами машинного навчання на основі генеративного наївного баєсівського методу та дискримінативного методу логістичної регресії . У нашому випадку класифікатори повині показувати хороші результати вже на невеликих навчальних вибірках. Класифікатори мають бути легкими у використанні. Будуть порівнянні їхня робота та особливості. Робота може розглядатись як джерело навчальної інформації або використовувати у подальшій розробці програм.

Мета й завдання роботи. Метою дипломної роботи є дослідження методів та алгоритмів класифікації текстів природною мовою та розробка текстового класифікаторів для Java і їх порівняння.

- Вибрати і зібрати існуючу інформацію про методи класифікації текстів природною мовою.
- Описати методи класифікації текстів природною мовою.
- Розробити текстові класифікатори на мові програмування Java.
- Порівняти результати їхньої роботи для різних завдань

Об'єкт, методи й засоби дослідження і розробки. Об’єктом дослідження є зібрання інформації про методи класифікації текстів природною мовою,

розробка програмних класифікаторів тексту на мові програмування Java та їх порівняння.

В якості джерел використовувались дослідницькі роботи, інтернет статті, результати дослідження і досвід великих компаній практикуючих методи.

Як середовище програмування буде використовуватись Intelij IDEA.

В бібліотеці будуть використовуватись Java 11 і її стандартні бібліотеки.

Можливі сфери застосування. Результат роботи може стати корисним при високорівневій розробці програм з функціями класифікації текстів.

Взаємозв'язок з іншими роботами. Результат роботи можна використовувати у подальших дослідницьких чи розробницьких роботах.

РОЗДІЛ 1. МЕТОДИ КЛАСИФІКАЦІЇ ТЕКСТІВ ПРИРОДНОЮ МОВОЮ

1.1 Класифікація текстів природною мовою

Класифікація текстів - одне з головних завдань обробки текстів природною мовою (англ. Natural Language Processing). Метою класифікації тексту є призначення тексту однієї або кількох категорій з визначеної множини категорій. Наприклад, якщо перед нами стоїть завдання сортування листів із рекламою або спамом від звичайних, нам потрібно розподілити тексти листів до однієї з двох категорій, лист зі спамом і звичайний лист. Існують два різні шляхи класифікувати текст: ручний і автоматичний. У цій роботі ми розглянемо тільки автоматичні методи. Автоматичні методи це алгоритми здатні бути виконані на комп’ютері. Алгоритмічні методи теж бувають двох типів: вручну створенні алгоритми і алгоритми використовуючи машинне навчання. Вручну створенні алгоритми розробляються по різному для конкретної задачі і базуються на специфічній для завдання інформації, як то слова властиві до категорії. Тому далі ми зосередимося на алгоритмах які використовують машинне навчання.

1.2 Наївний байєсів метод

Наївний байєсів метод(англ. Naive Bayes) - ймовірнісний метод класифікації текстів, який базується на формулі байєса(2) і на двох спрощеннях, тому і називається наївним. Його як правило використовують у машинному навчанні, тому і ми його тут розглянемо в контексті машинного навчання. Це генеративний метод класифікації завдяки чому ми можемо приблизно згенерувати архетип документа для кожного класу після навчання.

Всі ймовірності ми будемо визначати з тренувальної множини документів із заздалегідь коректно підібраними категоріями.

Категорія документа(тексту) дорівнює категорії в якої ймовірність для цього документа найбільша(1).

$$Cdoc = argmax\, p(c|d), c \in C \ (1)$$

Cdoc – результуюча категорія документа d, c – категорія, d – документ, $p(c|d)$ – ймовірність категорії c за умови істинності документа d.

Перетворимо ймовірність з допомогою формули байєса(2).

$$p(c|d) = \frac{p(d|c)p(c)}{p(d)} \ (2)$$

p(c) - ймовірність категорії c, p(d) - ймовірність документа d, $p(d|c)$ - ймовірність документа d за умови істинності c.

Вставимо (2) в (1).

$$Cdoc = argmax\ \frac{p(d|c)p(c)}{p(d)}\ , c \in C \ (3)$$

В (3) ми визначаємо категорію для конкретного документа d, тому d і $p(d)$ константа. Константа не вплине на функцію $argmax$, тому ми можемо забрати знаменник $p(d)$ (4).

$$Cdoc = argmax\, p(d|c)p(c)\ \ , c \in C \ (4)$$

Ймовірність $p(c)$ дорівнює частці к-сті документів з категорією c до к-сті всіх документів(5).

$$p(c) = \frac{N_c}{N} \ (5)$$

N_c – к-сть документів з категорією c, N - к-сть всіх документів.

Для розрахунку ймовірності $p(d|c)$ ми введемо два припущення:

- Порядок слів у документі не важливий.
- Слова не мають ніякого взаємозв'язку між собою.

Ці припущення є абсолютно невірними, але вони роблять розрахунки можливими, тому для нас вони є корисними спрощеннями.

Завдяки припущенням ми можемо представити документ як множину пар слів і к-сті їхніх появ у документі. Тому ми можемо представити $p(d|c)$ як добуток ймовірностей слів у категорії(6).

$$p(d|c) = \prod_{w_i \in d} p(w_i|c) \quad (6)$$

w_i – слово.

$$p(w_i|c) = \frac{count(w_i, c)}{count(w, c)} = \frac{count(w_i, c)}{\sum_{w_j \in d_c} count(w_j, c)} (7)$$

$count(w_i, c)$ – к-сть появ слова w_i в документах з категорією c, $count(w, c)$ – к-сть всіх слів в документах з категорією c.

Вставимо (6) в (4) і отримаємо остаточну формулу наївного байєсівського метода

$$Cdoc = argmax\, p(c) \prod_{w_i \in d} p(w_i|c), c \in C \ (8)$$

1.3 Параметри наївного байєсівського методу

В цьому підрозділі ми розглянемо деталі розрахунку ймовірностей для наївного байєсівського методу.

При використанні методу можлива ситуація коли слово не зустрілося в певній категорії, тоді в (7) чисельник буде нулем і вся ймовірність категорії(6) стане нулем. Розрахунки стануть не дійсними. Щоб запобігти цій ситуації додамо до (6) згладжуючу одиницю Лапласа.

$$p(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w_j \in d_c} count(w_j, c) + 1} = \frac{count(w_i, c) + 1}{|d_c| + \sum_{w_j \in d_c} count(w_j, c)} \quad (9)$$

Ще один випадок коли слово взагалі не зустрілося в тренувальній множині, це невідоме слово. Будемо з ним поводитись як із новим словом в документах.

$$p(w_u|c) = \frac{count(w_u, c) + 1}{|d_c + 1| + \sum_{w_j \in d_c} count(w_j, c)} = \frac{1}{|d_c + 1| + \sum_{w_j \in d_c} count(w_j, c)} \quad (10)$$

$p(w_u|c)$ – ймовірність невідомого слова у документах з категорією с.

1.4 Логістична регресія

Логістична регресія це статистичний метод класифікації набору ознак між двома класами. Цей метод використовує дискримінанту модель класифікації. Тому що ця напряму рахує ймовірність класу для документу. Коли наприклад наївний байєс з генеративною моделлю класифікації відповідає перше на запитання ”якщо б ми знали що цей документ класу с, які б риси він мав?”(відображається в формулі (4)).

Так як і наївний баєс, логістична регресія це ймовірнісний метод класифікації. Для цього методу наша система машинного навчання буде мати чотири компонента:

1. Числові представлення ознак вхідного документу у вигляді вектора цілих чисел [x1, x2, …, xn].

2. Функція яка класифікує даний вектор, вона буде використовувати сигмоїдну функцію, а ще потім софт макс функцію.
3. Функція для навчання. Вона буде мінімізувати помилку класифікації шляхом зміни ваг для нашого вектору ознак.
4. Функція для оптимізації ваг. Вона буде рахувати напрямок в якому потрібно рухати ваги вектору для зменшення помилки класифікації.

Ми будемо рахувати ймовірність класу напряму.

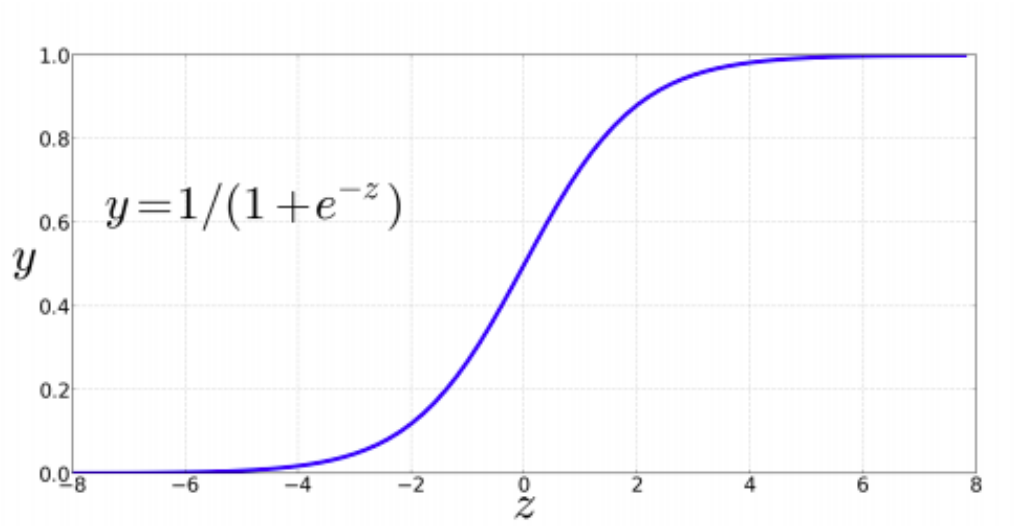
$$Cdoc = p(c|d)(11)$$

Число яке буде виражати ймовірність буде рахуватись лінійною комбінацією ознак і їх ваг плюс баяс(вага без ознаки).

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b(12)$$

Але (12) не правильна ймовірність, вона не лежить між 0 і 1. Насправді вона лежить між $-\infty$ до ∞. Тому ми використаємо сигмоїду щоб отримати бажане нам число.

$$y = \frac{1}{1 + \exp(-z)} \ (13)$$



1.4.1 Навчання логістичної регресії

Нам потрібна функція щоб відобразити помилку класифікації, тобто наскільки вихідний результат близький до правильного.

$$\mathrm{L}(\hat{y}, y) = \text{наскільки } y \text{ відрізняється від правильного } y \ (14)$$

Для цього представимо кросс ентропічну функцію втрат для двох класів (0, 1).

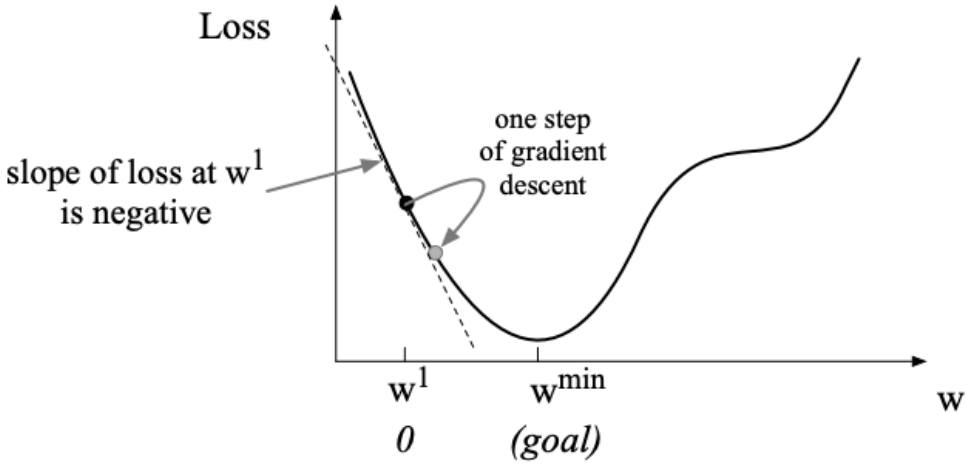
$$\mathrm{L}_{CE}(\hat{y}, y) = -\log p(y|x) \ (15)$$

$p(y|x)$ виражає ймовірність правильного класу.

$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y} (16)$$

Тепер ми можемо порахувати помилку нашої класифікації. Тепер представимо метод який допоможе нам підібрати правильні ваги щоб мінімізувати помилку. Це метод градієнтного спуску. Представимо наші ваги w і баєс b в одному векторі для оптимізації $\theta = w, b.$ $\hat{\theta}$ – новий вектор після одної ітерації оптимізації.

$$\hat{\theta} = \underset{\theta}{arg\min} \frac{1}{m} \sum_{i=1}^{m} \mathrm{L}_{CE}(f(x,\theta), y) (17)$$



Метод градієнтного спуску дає відповідь на запитання куди рухати наші ваги. Він бере градієнт і рухає їх в протилежному напрямку.

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w) (17)$$

Тут η крок градієнтного спуску. Важливо при тренуванні підібрати його правильно.

Градієнт для логістичної регресії.

$$\frac{\partial \mathrm{L}_{CE}(\hat{y}, y)}{\partial w_j} = [sigmoid(w \cdot x + b) - y] x_j (18)$$

Метод стохастичного градієнтного спуску полягає в тому що ми беремо випадковий тренувальний документ для градієнтного спуску і так для всіх. Саме його ми і будемо використовувати в розробці.

Представимо концепт тренування порціями для отримання більш оптимізованих ваг. Вони будуть більш оптимізовані тому що при одній ітерації

оптимізації будуть використовуватись декілька тренувальних елементів, а не один як при звичайній оптимізації.

Для функції помилки ми будемо використовувати середнє арифметичне функцій крос ентропічних втрат для кожного елемента.

$$Cost(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{L}_{CE}(\hat{y}, y) \quad (19)$$

Так само новий градієнт буде просто середнє арифметичне з градієнтів окремих елементів.

$$\frac{\partial \mathrm{Cost}(\hat{y}, y)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m} [sigmoid(w \cdot x + b) - y] x_j \quad (20)$$

Існує проблема при тренуванні моделі що робить її ідеальною для деяких наборів даних. Вона виникає коли деякі ознаки дуже добре поділяють на класи, тому їм приділяють великі ваги. Але на тестуванні така модель покаже значно гірше тому що вона тепер ідеально поділяє тільки тренувальні елементи. Модель повинна добре визначати загальні важливі ознаки для класу, тому ми будемо штрафувати її під час тренування, якщо вона буде збільшувати одні ваги непропорційно сильно.

$$\hat{\theta} = \underset{\theta}{argmax}\left[\sum_{i=1}^{m} \mathrm{L}_{CE}(f(x, \theta), y)\right] - \alpha \sum_{j=1}^{n} \theta_j^2 \quad (21)$$

Тут ми використали Евклідову відстань для штрафування спуску.

1.5 Мультиноміальна логістична регресія

Для цієї роботи ми ще визначимо мультиноміальну версію логістичної регресії для класифікації між трьома і більше класами. Ця версія буде використовувати софт макс фунцію замість сигмоїдної.

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp\,(z_j)} \quad (22)$$

Через неї покажемо ймовірність класа

$$p(y = c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} \exp\,(w_j \cdot x + b_j)} \quad (23)$$

Так як в нас тепер більше ніж два класи, в кожного з них будуть свої ваги.

Визначимо кросс ентропічну функцію втрат

$$\mathrm{L}_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log(\widehat{y_k})$$

$$= -\log(\widehat{y_k})\ (k \text{ правильний клас})$$

$$= -\log\left(\frac{exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} exp\left(w_j \cdot x + b_j\right)}\right) (24)$$

Градієнтний спуск для мультиномінальної моделі

$$\frac{\partial \mathrm{L}_{CE}(\hat{y}, y)}{\partial w_{k,i}} = -\left(1 - \frac{exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} exp\left(w_j \cdot x + b_j\right)}\right) x_i \ (25)$$

1.6 Вимірювання якості класифікатора

Автоматичні методи класифікації базуються на ймовірностях, тому вони не є абсолютно точними. Щоб вимірювати якість результату класифікатора тексту використовують чотири міри:

- Точність (англ. Accuracy)
- Влучність (англ. Precision)

- Охоплення (англ. Recall)
- F – міра

Щоб визначити ці міри використовують таблицю показників:

	Правильні	Не правильні
Вибрані	tp	fp
Не вибрані	tn	fn

- tp – це к-сть вибраних класифікатором правильних елементів
- fp – це к-сть вибраних класифікатором не правильних елементів
- tn – це к-сть не вибраних класифікатором правильних елементів
- fn – це к-сть не вибраних класифікатором не правильних елементів

Точність рахується за формулою:

$$A = \frac{tp + tn}{tp + fp + tn + fn} \quad (21)$$

Точність використовується, коли елементи тестування розподіленні рівномірно між класами.

Влучність рахується за формулою:

$$P = \frac{tp}{tp + fp} \quad (22)$$

Влучність показує яка частина результату правильна.

Охоплення рахується за формулою:

$$R = \frac{tp}{tp + fn} \quad (23)$$

Охоплення показує яка частина всіх правильних елементів увійшли у результат.

F – міра рахується за формулою:

$$F = \frac{1}{a\frac{1}{P} + (1-a)\frac{1}{R}} = \frac{(b^2+1)PR}{b^2P + R} \quad (24)$$

F – міра використовується для порівняння класифікаторів між собою. Параметр a(або b) підбирається під конкретну задачу, в залежності що і наскільки важливіше влучність чи охоплення.

Для спільних показників(tp, fp, tn, fn) для багатьох класів існують два способи обчислення: мікро-середнє і макро-середнє.

$$macro\ P = \frac{\sum_i^n P_i}{n} \quad (25)$$

$$macro\ R = \frac{\sum_i^n R_i}{n} \quad (26)$$

Де P_i - це влучність конкретного класа, R_i - це охоплення конкретного класа, n – кількість класів.

Макросереднє більш справедливе для класів з малою кількістю тестуючих елементів ніж мікросереднє. В цій роботі ми будемо використовувати макросереднє, щоб враховувати результати всіх класів однаково незалежно від кількості тестових елементів для кожного з них.

РОЗДІЛ 2. РОЗРОБКА ТЕКСТОВИХ КЛАСИФІКАТОРІВ

2.1 Основні вимоги до класифікатора

Будуть розроблені класифікатори для біноміального та мультиноміального використовуючи методи наївного баєса та логістичної регресії. Принципи роботи всіх методів були описанні в першому розділі. Класифікатор для коректної роботи потребуватиме тренувальні вибірки даних на основі яких будуть розраховуватись параметри алгоритму(ймовірності різних елементів). Вони будуть надходити зовні, тобто будуть попередніми вхідними даними для класифікатора. Тренувальні дані це документи(тексти) з попередньо визначеними класами. Класифікатор коли натренований буде здатний класифікувати документ, тобто повернути найімовірніший клас для даного документу.

2.2 Архітектура

2.2.1 Попередні вхідні данні

Більшість вибірок даних для тренування текстових класифікаторів у відкритому доступі зберігаються в звичайних тестових файлах (.txt) поділеними на папки. Тому для зручності баєсівський класифікатор буде приймати як тренувальну вибірку окремі текстові файли і папки з багатьма файлами. Один текстовий файл буде сприйматися як один документ. При надходженні тренувальних папки чи файла потрібно буде вказати клас до якого документи будуть відноситись.

Класифікатори з логістичною регресією будуть приймати підготовленні данні у виді векторів значень ознак документа, тому що експортування ознак займає великий час і завжди проходить однаково. Один документ буде відповідати одному вектору.

2.2.2 Зберігання попередніх вхідних даних

Користуючись двома основними спрощеннями наївного байєсівського алгоритму ми можемо представити документи деякого класа як множину пар слів і кількості цих слів у документах відповідно, таке представлення ще називають мішком словами. Так само будемо зберігати кількість зчитаних документів кожного класу для розрахунку параметрів алгоритму.

Буде розроблений інструмент для визначення і потім тимчасового запису векторів ознак разом із класами для логістичної регресії. Весь тренувальний набір даних буде опрацьований цим інструментом і записаний в окремий файл.

2.2.3 Можливість донавчатись баєса

Щоб ввести цю можливість ми будемо зберігати попередні вхідні дані і розраховані параметри алгоритму окремо на весь час процесу. Так ми зможемо використовувати попередні вхідні дані і добавляти до них нові. Це дозволить класифікатору не зчитувати старі дані з файлів і перетворювати їх на мішок зі словами знову.

2.2.4 Визначення ознак документа для логістичної регресії

Для коректної роботи логістичної регресії нам знадобиться представити документ як вектор цілих чисел. Тому потрібно придумати які риси документу ми хочемо і можемо виразити для класифікації.

- Буду ознака для кожного класу які будуть рахувати слова які зустрічаються в їхніх класах. Кожне слово із входу буде рахуватися у всьому класі і ці кількості сумуються. Так ми отримаємо які будуть сильно впливати на один конкретний клас.
- В наступному типу ознак нам допоможе наївний баєсівський класифікатор. Ми розіб’ємо слова на групи по 50 наприклад слів. Всі вони будуть розподілятися ранжовано по ймовірності слова у

класі за баєсом. Тобто найбільш ймовірні 50 слів для класу будуть в одній групі, а наступні 50 в другій групі і так далі. Кількість слів які входять в цю групу з вхідного документу і буде ознакою зв’язаною з цією групою.

2.2.5 Розрахунок параметрів та їх зберігання

Для баєсівського класифікатора розрахунок параметра(ймовірності слова у класі) проводиться для кожного слова у мішку зі словами. При зміні попередніх вхідних даних(донавчання) параметри повинні бути розраховані заново. Це затратна операція при великих об’ємах інформації машинного навчання. Тому класифікатор не буде її проводити кожен раз коли змінюються попередні вхідні дані, а тоді коли параметри стануть потрібними, тобто перед першою класифікацією. Такий спосіб ініціалізації даних називають лінивим(lazy) він допомагає запобігти не потрібному обчисленню. Зберігання параметрів буде таке ж саме як і кількості слів у мішку зі словами.

Розрахунок векторів для логістичної регресії відбудеться заздалегідь

2.2.6 Класифікація, вхідні і вихідні дані

Для баєсівського класифікатора вхідними даними буде документ у виді строки. Класифікація буде проводитись по алгоритму і буде використовувати заздалегідь обчисленні параметри. Вихідними даними буде назва класу ймовірність якого найбільша для даного документа.

Для класифікатора логістичної регресії вхідними даними буде вектор ознак документа. Вихідними даними буде номер класу який буде найбільш ймовірний для цих ознак.

2.3 Розробка

2.3.1 Представлення класифікатора

Інтерфейс класифікатора у бібліотеці будуть представлений джава інтерфейсом. Всі взаємодії з класифікаторами будуть проводитись через цей інтерфейс та його методи.

```
package core;
public interface TextClassifier {
    String matchClass(String str);
}
```

2.3.2 Зберігання попередніх вхідних даних

Для баєсівського класифікатора кількість появлень слів у класі буде зберігати як пари слів і кількостей у хеш-таблиці вкладеними у ще одну хеш-таблицю.

```
        {
                "class1": {
                        "word1": 123,
                        "word2": 456
                },
                "class2": {
                        "word1":789
                }
        }
```

Функції хеш-таблиці в Java стандартно відображає HashMap.

```
    private Map<String, Map<String, Integer>> wordCountPerClass = new HashMap<>();
```

Кількість документів класів будемо зберігати як пари клас і кількість його документів в хеш-таблиці.

```
    private Map<String, Integer> documentsPerClass = new HashMap<>();
```

Для логістичної регресії її наперед визначені вектори будуть зберігатися в окремому файлі. Разом з номером класу.

```
1,2,3,4,5
6,7,8,9,0
```

Де числа 1, 2, 3, 4 це значення вектора першого документу. 6, 7, 8, 9 значення вектору для другого документа. А 5 і 0 це номера класів для першого і другого класу відповідно.

2.3.3 Читання попередніх вхідних даних

Для баєсівського класифікатора у джава класа буде два публічних метода для находження попередніх вхідних даних. Один метод буде приймати шлях до файла у виді строки(клас String у Java), а другий шлях до папки з файлами у виді строки.

```
public void addDocument(String name, String classFile) {…}

public void addDocumentDir(String name, String classDir) {…}
```

Метод addDocumentDir дістає шляхи файлів з папки і для кожного з них викликає addDocument. Метод addDocument рахує кількість слів у файлі і записує у хеш-таблицю wordCountPerClass.

Класифікатор логістичної регресії буде приймати матрицю яка складається з векторів ознак документів та вектор який складається з номерів класів відповідних документів.

2.3.4 Зберігання параметрів

Ми будемо зберігати три види ймовірностей: ймовірність слова у класі, ймовірність класу і ймовірність невідомого слова у класі. Ймовірність слова у класі будемо зберігати так само як кількість появлень слів у класі.

```
private Map<String, Map<String, Double>> wordPosPerClass = new HashMap<>();
```

Ймовірності невідомого слова у класі і ймовірність класу будемо зберігати у хештаблиці.

```
private Map<String, Double> classPos = new HashMap<>();
private Map<String, Double> unknownWordPosPerClass = new HashMap<>();
```

Для логістичної регресії ми будемо зберігати данні у тих самих векторах, які були вхідними даними для навчання.

2.3.5 Розрахунок параметрів

Метод calculate буде розраховувати параметри і буде викликатись перед початком класифікації

```
private void calculate() {…}
```

Щоб не обчислювати параметри які вже були обчислені введемо булеву зміну, яка буде показувати чи актуальні параметри обчислені.

```
private boolean isParamsCalculated = false;
```

2.3.6 Класифікація

Метод класифікації приймає строку для класифікації, а як результат повертає назву найімовірнішого класу для даної строки.

```
public String matchClass(String str) {…}
```

Для баєсівського класифікатору при обчислені використовуються заздалегідь обчислені ймовірності. В цьому класі відбувається множення багатьох раціональних чисел значущих цифри яких знаходяться далеко від коми. Стандартні типи даних Java не можуть зберігати таку велику кількість цифр після коми. Тому ми розробимо власний презентацію таких чисел з потрібними для нас операціями порівняння і множення. Цим представлення буде внутрішній клас Possibility.

```
private class Possibility {
        int num;
        int exp;
        Possibility(double number) {…}
        Possibility multiply(Possibility possibility) {…}
        boolean more(Possibility p) {…}
}
```

Для логістичної регресії використовується цей же метод. Він розділяє текст на вектор, а вектор вже класифікує так як описано в теоретичній частині.

2.3.7 Точність обчислення

Кожне раціональне число із значущими цифрами можна розбити на множення значущих цифр і десяти в степені. Клас Possibility зберігає числа у виді двох цілих чисел. Перше число це значущі цифри, а друге це степінь десяти.

$$0.00x = x * 10^{-3}$$

```
int num;
int exp;
```

По замовчуванню класифікатор зберігає три перші значущих цифр, але їхню кількість можна змінити.

```
private int accuracy = 3;

public int getAccuracy() {…}
public void setAccuracy(int accuracy) {…}
```

РОЗДІЛ 3. ТЕСТУВАННЯ ТЕКСТОВИХ КЛАСИФІКАТОРІВ

3.1 Інструмент для тестування

Для перевірки якості розробленого класифікатора нам знадобляться інструменти для обчислення показників якості описаних у першому розділі. Java клас TestUtils містись метод який буде перевіряти у даному класифікаторі дані документи з попередньо визначеними класами і на основі перевірки покаже показники якості класифікатора: влучність та охоплення для кожного класу, також точність, влучність, охоплення, F-міру для цілого класифікатора. Повний код класа TestUtils доступний у додатку Б. Код класифікаторів можна буде знайти у додатку А.

3.2 Вибір тренувальних і тестових даних

Ми випробуємо класифікатори для двох різних завдань: біноміальній класифікації (аналіз сентиментів) і мультиноміальна класифікація (класична текстова класифікація). Класична текстова класифікація це призначення тексту найбільш підходящу для нього тему. Аналіз сентиментів відповідає на запитання: який настрій несе текст позитивний чи негативний.

Для аналізу сентиментів ми будемо використовувати вибір даних сайту imdb, який збирає рейтинги і рецензії фільмів. Вибірка поділена на позитивні і негативні рецензії для фільму. В цій ситуації класифікатор буде повертати найімовірніший клас для даної йому рецензії. Тренувальні дані містять 25000 рецензій, по 12500 на кожен клас. Тестові дані містять теж саме число рецензій і поділені вони як і тренувальні.

Для класична класифікація ми використаємо AG’s News вибірку новин поділенні на чотири категорії: світ, спорт, бізнес та технології. Вибірка містить

120000 тренувальних статей, по 30000 на кожен клас. Тестові дані містять 7600 статей по 1900 на кожен клас.

3.3 Тестування

3.3.1 Біноміальна баєсівська класифікація (Аналіз сентиментів)

Запустимо баєсівський класифікатор з вибіркою позитивних та негативних рецензій на фільми.

```
private static void imdbBayes(){
    TextClassifier classifier = new TextClassifier();
    classifier.addDocumentDir("negative", "imdb/train/neg");
    classifier.addDocumentDir("positive", "imdb/train/pos");

    Map<String, String> testDirs = Map.of(
            "negative", "imdb/test/neg",
            "positive", "imdb/test/pos");
    TestUtils.testClassifier(classifier, testDirs);
}
```

Результати:

```
Test result:
negative:  10951 true,   1549 false,  12500 total
positive:   9621 true,   2879 false,  12500 total

Class classification quality:
negative precision: 0,876
negative recall:    0,792

positive precision: 0,770
positive recall:    0,861

Overall classification quality:
accuracy:  0,823
precision: 0,827
recall:    0,823
F-measure: 0,825
```

3.3.2 Біноміальна класифікація з логістичною регресією (Аналіз сентиментів)

Спершу запустимо генерацію наших ознак з допомогою наївного баєсівського класифікатора.

```
private static void generateFiles(){
    NaiveBayes bayesClassifier = new NaiveBayes();
    bayesClassifier.addClassDir("neg", "src/main/resources/train/neg");
    bayesClassifier.addClassDir("pos", "src/main/resources/train/pos");
    bayesClassifier.calculate();
    bayesClassifier.writeMostPosWordInFiles("src/main/resources/prepared", 50);
}
```

Запустимо класифікатор з логістичною регресією з вибіркою позитивних та негативних рецензій на фільми.

```
private static void imdbLogistic() {
    ArrayList<Function<String, Integer>> functions = new ArrayList<>();
    functions.add(
        FeatureExtractor.vocabularyFeatureDir("src/main/resources/train/neg"));
    functions.add(
        FeatureExtractor.vocabularyFeatureDir("src/main/resources/train/pos"));
    functions.addAll(
        FeatureExtractor.vocabularyFeatureForEach("src/main/resources/prepared"));

    double[][] neg = FeatureExtractor.inputData(functions,
        "src/main/resources/train/neg");
    double[][] pos = FeatureExtractor.inputData(functions,
        "src/main/resources/train/pos");

    double[][] combined = new double[neg.length + pos.length][functions.size()];
    int[] labels = new int[neg.length + pos.length];
    for (int i = 0; i < neg.length; i++) {
        labels[i] = 0;
        for (int j = 0; j < combined[i].length; j++) {
            combined[i][j] = neg[i][j];
```

```
        }
    }
    for (int i = neg.length; i < neg.length + pos.length; i++) {
        labels[i] = 1;
        for (int j = 0; j < combined[i].length; j++) {
            combined[i][j] = pos[i - neg.length][j];
        }
    }
    LogisticRegression logisticRegression =
        LogisticRegression.binomial(combined, labels);
    TestUtils.testClassifier(logisticRegression, Map.of(
            "0", "src/main/resources/test/neg",
            "1", "src/main/resources/test/pos"
    ));
}
```

Результати:

```
Test result:
negative:  9831 true,   2669 false,  12500 total
positive:  9325 true,   3175 false,  12500 total

Class classification quality:
negative precision: 0,786
negative recall:    0,756

positive precision: 0,746
positive recall:    0,777

Overall classification quality:
accuracy:  0,766
precision: 0,766
recall:    0,767
F-measure: 0,766
```

3.3.3 Мультиноміальна баєсівська класифікація (Класична класифікація)

Запустимо баєсівський класифікатор з вибіркою новиних статей на одну з чотирьох тем.

```
private static void agNewsBayes(){
    NaiveBayes bayesClassifier = new NaiveBayes ();

    bayesClassifier.addDocumentDir("business", "ag_news/train/business");
    bayesClassifier.addDocumentDir("sport", "ag_news/train/sport");
    bayesClassifier.addDocumentDir("tech", "ag_news/train/tech");
    bayesClassifier.addDocumentDir("world", "ag_news/train/world");

    Map<String, String> testDirs = Map.of(
            "business", "ag_news/test/business",
            "sport", "ag_news/test/sport",
            "tech", "ag_news/test/tech",
            "world", "ag_news/test/world");
    TestUtils.testClassifier(bayesClassifier, testDirs);
}
```

Результати:

```
Test result:
   world:   1708 true,    192 false,   1900 total
   sport:   1848 true,     52 false,   1900 total
    tech:   1661 true,    239 false,   1900 total
business:   1609 true,    291 false,   1900 total

Class classification quality:
    tech precision: 0,874
    tech recall:    0,868

   world precision: 0,899
   world recall:    0,908

business precision: 0,847
business recall:    0,866

   sport precision: 0,973
   sport recall:    0,950

Overall classification quality:
accuracy:  0,898
precision: 0,898
```

```
recall:    0,898
F-measure: 0,898
```

3.3.4 Мультиномінальна логістична класифікація (Класична класифікація)

Спершу запустимо генерацію наших ознак з допомогою наївного баєсівського класифікатора.

```
private static void generateFiles(){
    NaiveBayes bayesClassifier = new NaiveBayes();
    bayesClassifier.addDocumentDir("business", "ag_news/train/business");
    bayesClassifier.addDocumentDir("sport", "ag_news/train/sport");
    bayesClassifier.addDocumentDir("tech", "ag_news/train/tech");
    bayesClassifier.addDocumentDir("world", "ag_news/train/world");

    bayesClassifier.calculate();
    bayesClassifier.writeMostPosWordInFiles("src/main/resources/prepared", 50);
}
```

Запустимо класифікатор з мультиномінальною логістичною регресією з вибіркою новиних статей на одну з чотирьох тем.

```
private static void agNewsLogistic() {
    ArrayList<Function<String, Integer>> functions = new ArrayList<>();
    functions.add(
        FeatureExtractor.vocabularyFeatureDir("ag_news/train/business"));
    functions.add(
        FeatureExtractor.vocabularyFeatureDir("ag_news/train/sport"));
    functions.add(
        FeatureExtractor.vocabularyFeatureDir("ag_news/train/tech"));
    functions.add(
        FeatureExtractor.vocabularyFeatureDir("ag_news/train/world"));

    functions.addAll(
        FeatureExtractor.vocabularyFeatureForEach("src/main/resources/prepared"));
```

```
    double[][] combined =  …
    LogisticRegression logisticRegression =
        LogisticRegression.multinominal(combined, labels);
    TestUtils.testClassifier(logisticRegression, Map.of(
            "0", "ag_news/test/business ",
            "1", "ag_news/test/sport",
            "2", "ag_news/test/tech",
            "3", "ag_news/test/world"
    ));
}
```

Результати:

```
Test result:
   world:   1634 true,    266 false,   1900 total
   sport:   1705 true,    195 false,   1900 total
    tech:   1551 true,    349 false,   1900 total
business:   1491 true,    409 false,   1900 total

Class classification quality:
    tech precision: 0,816
    tech recall:    0,812

   world precision: 0,860
   world recall:    0,869

business precision: 0,785
business recall:    0,782

   sport precision: 0,897
   sport recall:    0,891

Overall classification quality:
accuracy:  0,839
precision: 0,839
recall:    0,839
F-measure: 0,839
```

3.4 Висновки тестування

F-міра баєсівського класифікатора для аналізу сентиментів рецензій imdb дорівнює 0,825. Це непоганий результат для порівняно малою вибіркою в 12500 документів на клас. F-міра біномінальної логістичної регресії на цих же даних була 0,766 це значно менше ніж у баєса. Це може бути зумовлено типами класифікаторів. Так як логістична регресія дискримінантна важливою частиною розробки цієї моделі грають дизайн рис які будуть які будуть представляти документ.

F-міра баєсівської класифікації статей новин дорівнює 0,898. Це хороший результат (90% вважається хорошим результатом). Вибірка теж була порівняно невелика по 30000 документів на клас. І знову логістична справилась гірше. Її F-міра 0,839, що теж не погано, але гірше ніж у баєса. Припускаю що причиною стала той же дизайн рис документу.

Різниця в результатах виникла, тому що вибірка статей була більша і більш презентативна за вибірку рецензій. Щоб підняти якість класифікатора можна збільшити кількість документів у вибірці і профільтрувати її щоб видалити документи які неоднозначно підходять під свій клас. Щоб збільшити ефективність логістичної регресії слід придумати кращий дизайн рис документа.

ВИСНОВКИ

Описано формалізація наївного генеративного баєсівського метода класифікації текстів, метода логістичної регресії(як біномінального та і мультиномінального) і методи вимірювання якості класифікаторів. Розроблено текстові класифікатори на мові Java, на основі наївного баєсівського метода та методу логістичної регресії машинного навчання для класифікації текстів натуральною мовою. Порівняно генеративний(наївний баєс) і дискримінантний(логістична регресія) класифікатори при біномінальній(два класи) і мультиномінальній(більше двох класів) випадках. Класифікатори випробуваний на двох класичних завданнях класифікації текстів і показав хороший результат на малих кількостях документів у вибірці. Класифікатори може бути використаним як частина високорівневих програм.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

Книги:

1. D. Jurafsky Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2018.– 558 с.

Інтернет джерела:

2. http://ai.stanford.edu/~amaas/data/sentiment/
3. https://github.com/mhjabreel/CharCnn_Keras/tree/master/data/ag_news_csv
4. https://lionbridge.ai/datasets/14-best-text-classification-datasets-for-machine-learning/
5. https://nlp.stanford.edu/static/software/classifier.shtml

6. https://github.com/haifengl/smile
8. https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html
9. https://web.stanford.edu/~jurafsky/slp3/5.pdf

ДОДАТОК А. Код програми

NaïveBayes.java

```
public class TextClassifier {
    private int accuracy = 3;

    private Map<String, Integer> documentsPerClass = new HashMap<>();
    private Map<String, Map<String, Integer>> wordCountPerClass = new HashMap<>();

    private boolean isParamsCalculated = false;
    private Map<String, Map<String, Double>> wordPosPerClass = new HashMap<>();
    private Map<String, Double> classPos = new HashMap<>();
    private Map<String, Double> unknownWordPosPerClass = new HashMap<>();

    public TextClassifier() {
    }

    public void addDocument(String name, String classFile) {
        Map<String, Integer> newWordCount = countWords(classFile);
        Map<String, Integer> wordCount = wordCountPerClass.get(name);
        if (Objects.isNull(wordCount)) {
            wordCountPerClass.put(name, newWordCount);
            documentsPerClass.put(name, 1);
        } else {
            newWordCount.forEach((word, count) ->
                    wordCount.merge(word, count, Integer::sum));
            documentsPerClass.compute(name, (s, c) -> c + 1);
            isParamsCalculated = false;
        }
    }

    public void addDocumentDir(String name, String classDir) {
        IOUtils.getFilesInDir(classDir)
                .forEach(file -> addDocument(name, file));
    }

    public String matchClass(String str) {
        calculate();
        List<String> words = IOUtils.getWords(str);
        Pair<String, Possibility> biggestPos = wordCountPerClass.keySet().stream()
                .map(clas -> new Pair<>(clas, getPos(clas, words)))
                .reduce((pair, pair2) -> pair.rigth.more(pair2.rigth) ? pair : pair2)
                .orElseThrow();

        return biggestPos.left;
    }

    private void calculate() {
        if (isParamsCalculated) {
            return;
        }
```

```
        calculateClassPos();
        Set<String> vocab = getVocabulary();
        calculateWordPosPerClass(vocab);
        calculateUnknownWordPosPerClass(vocab);
        isParamsCalculated = true;
    }

    private void calculateUnknownWordPosPerClass(Set<String> vocab) {
        unknownWordPosPerClass = wordCountPerClass.entrySet().stream()
                .map(en -> new Pair<>(en.getKey(),
                        calculateUnknownWordPos(en.getValue(), vocab)))
                .collect(Collectors.toMap(pair -> pair.left, pair -> pair.rigth));
    }

    private Double calculateUnknownWordPos(Map<String, Integer> wordCount,
                                           Set<String> vocab) {
        int numOfAllWordsInClass = wordCount.values().stream().mapToInt(c -> c).sum();
        return 1d / (numOfAllWordsInClass + vocab.size() + 1);
    }

    private void calculateWordPosPerClass(Set<String> vocab) {
        wordPosPerClass = wordCountPerClass.entrySet().stream()
                .map(en -> new Pair<>(en.getKey(),
                        calculateWordPos(en.getValue(), vocab)))
                .collect(Collectors.toMap(pair -> pair.left, pair -> pair.rigth));
    }

    private Map<String, Double> calculateWordPos(Map<String, Integer> wordCount,
                                                 Set<String> vocab) {
        int numOfAllWordsInClass = wordCount.values().stream().mapToInt(i -> i).sum();
        int vocabSize = vocab.size();

        return vocab.stream()
                .map(word -> new Pair<>(
                        word,
                        Objects.isNull(wordCount.get(word)) ? 0 : wordCount.get(word)))
                .map(pair -> new Pair<>(
                        pair.left,
                        ((double) pair.rigth + 1) /
                                (numOfAllWordsInClass + vocabSize)))
                .collect(Collectors.toMap(pair -> pair.left, pair -> pair.rigth));

    }

    private Set<String> getVocabulary() {
        return wordCountPerClass.values().stream()
                .flatMap(map -> map.keySet().stream())
                .collect(Collectors.toSet());
    }

    private void calculateClassPos() {
        int numOfAllDocs = documentsPerClass.values().stream()
                .mapToInt(c -> c)
```

```
                .sum();

        classPos = documentsPerClass.keySet().stream()
                .collect(Collectors.toMap(
                        c -> c,
                        c -> ((double) documentsPerClass.get(c)) / numOfAllDocs));
    }

    private Possibility getPos(String clas, List<String> words) {
        Map<String, Double> wordPos = wordPosPerClass.get(clas);

        Possibility posWithoutUnknown = words.stream()
                .map(wordPos::get)
                .filter(Objects::nonNull)
                .map(Possibility::new)
                .reduce(new Possibility(classPos.get(clas)), Possibility::multiply);

        return words.stream()
                .filter(w -> !wordPos.containsKey(w))
                .map(w -> unknownWordPosPerClass.get(clas))
                .map(Possibility::new)
                .reduce(posWithoutUnknown, Possibility::multiply);
    }

    private static Map<String, Integer> countWords(String filePath) {
        return IOUtils.getWordsFromFile(filePath).stream()
                .collect(Collectors.toMap(s -> s, s -> 1, Integer::sum));
    }

    public int getAccuracy() {
        return accuracy;
    }

    public void setAccuracy(int accuracy) {
        this.accuracy = accuracy;
    }

    private class Possibility {
        int num;
        int exp;

        Possibility() {
        }

        Possibility(double number) {
            double pow = Math.pow(10, accuracy - 1);
            int d = 0;
            while (number < pow) {
                number *= 10;
                d++;
            }
            num = (int) Math.round(number);
            exp = d;
```

```
        }

        Possibility multiply(Possibility possibility) {
            int number = num * possibility.num;
            double pow = Math.pow(10, 3);
            int d = 0;
            for (; number >= pow; d++) {
                number /= 10;
            }
            Possibility result = new Possibility();
            result.num = number;
            result.exp = exp + possibility.exp - d;
            return result;
        }

        boolean more(Possibility p) {
            if (exp < p.exp) {
                return true;
            } else if (exp == p.exp && num > p.num) {
                return true;
            }
            return false;
        }
    }
}
```

LogisticRegresion.java

```
public class LogistiRegresion implements TextClassifier {

    private int chunkSize = 2;
    private double decentStep = 0.00001;

    private boolean trained;

    private final Set<String> classes = new HashSet<>();
    private final Map<String, List<String>> trainingSet = new HashMap<>();
    private final List<Function<String, Integer>> featuresF = new ArrayList<>();
    private final Map<String, List<BigDecimal>> weights = new HashMap<>();

    public void addClassDir(String name, String classDir) {
        List<String> trainingData = IOUtils.getFilesInDir(classDir).stream()
                .map(IOUtils::getStringFromFile)
                .collect(Collectors.toList());
        trainingSet.put(name, trainingData);
        classes.add(name);
        trained = false;
    }

    @Override
    public String matchClass(String str) {
        trainIfNot();
        return getMostProbable(str);
    }

    public void addFeature(Function<String, Integer> feature) {
```

```
        featuresF.add(feature);
    }

    private String getMostProbable(String str) {
        List<Integer> features = features(str);
        return classes.stream()
                .map(cls -> new Pair<>(cls, probabilityOf(cls, features)))
                .max(Comparator.comparing(Pair::getRigth))
                .orElseThrow()
                .getLeft();
    }

    private double probabilityOf(String cls, List<Integer> features) {
        try {
            BigDecimal rawExp = Exp.exp(rawProbabilityOf(cls, features));
            BigDecimal rawExpSum = classes.stream().map(inCls ->
Exp.exp(rawProbabilityOf(inCls, features))).reduce(BigDecimal.ZERO,
BigDecimal::add);
            return rawExp.divide(rawExpSum, RoundingMode.HALF_UP).doubleValue();
        } catch (Exception e) {
            throw e;
        }
    }

    private BigDecimal rawProbabilityOf(String cls, List<Integer> features) {
        List<BigDecimal> weightsForCls = weights.get(cls);

        BigDecimal sum = BigDecimal.ZERO;

        for (int i = 0; i < features.size(); i++) {
            sum =
sum.add(BigDecimal.valueOf(features.get(i)).multiply(weightsForCls.get(i)));
        }
        return sum.add(weightsForCls.get(weightsForCls.size() - 1));
    }

    private List<Integer> features(String str) {
        return featuresF.stream()
                .map(f -> f.apply(str))
                .filter(Objects::nonNull)
                .collect(Collectors.toList());
    }

    private void trainIfNot() {
        if (!trained) {
            train();
            trained = true;
        }
    }

    private void train() {
        initWeights();
        List<List<Pair<String, String>>> chunks = miniChunks();
        for (List<Pair<String, String>> chunk : chunks) {
            double prevLoss = Double.MAX_VALUE;
            double loss = 0;
            while (loss < prevLoss) {
                prevLoss = loss;
                loss = 0;
                for (Pair<String, String> pair : chunk) {
                    List<Integer> features = features(pair.getRigth());
                    double probability = probabilityOf(pair.getLeft(),
features);
```

```
                    gradientDecent(features, probability, pair.getLeft());
                    loss += Math.log(probability);
                }
                loss = loss * -1;
            }
        }
    }

    private void gradientDecent(List<Integer> features, double probability,
String cls) {
        List<BigDecimal> w = weights.get(cls);
        ArrayList<BigDecimal> newWeights = new ArrayList<>();

        for (int i = 0; i < w.size(); i++) {
            double feature;
            if (i == w.size() - 1) {
                feature = 1;
            } else {
                feature = features.get(i);
            }
            BigDecimal newWeight = w.get(i).add(
                    BigDecimal.ONE.subtract(BigDecimal.valueOf(probability))
                            .multiply(BigDecimal.valueOf(feature))
                            .multiply(BigDecimal.valueOf(-1))
                            .multiply(BigDecimal.valueOf(decentStep))
            );
            newWeights.add(newWeight);
        }
        weights.put(cls, newWeights);
    }

    private List<List<Pair<String, String>>> miniChunks() {
        List<Pair<String, String>> collect = trainingSet.entrySet().stream()
                .flatMap(e -> e.getValue().stream().map(str -> new
Pair<>(e.getKey(), str)))
                .collect(Collectors.toList());
        List<List<Pair<String, String>>> result = new ArrayList<>();
        for (int i = 0; i < collect.size() / chunkSize; i++) {
            result.add(new ArrayList<>());
        }
        Random random = new Random();
        collect.forEach(pair ->
result.get(random.nextInt(result.size())).add(pair));
        return result;
    }

    private void initWeights() {
        weights.putAll(classes.stream().collect(
                Collectors.toMap(
                        cls -> cls,
                        cls -> featuresF.stream().map(ignore ->
BigDecimal.ZERO).collect(Collectors.toList())
                )));
    }

    public void setChunkSize(int chunkSize) {
        this.chunkSize = chunkSize;
    }

    public void setDecentStep(double decentStep) {
        this.decentStep = decentStep;
    }
```

ДОДАТОК Б. Код для тестування програми

TestUtils.java

```
public class TestUtils {
    public static void testClassifier(TextClassifier classifier,
                                      Map<String, String> classDirs) {
        HashMap<String, Map<String, Integer>> confusionTable = new HashMap<>();
        System.out.println("Test result:");
        for (Map.Entry<String, String> entry : classDirs.entrySet()) {
            List<String> res = IOUtils.getFilesInDir(entry.getValue())
                    .stream()
                    .map(IOUtils::getStringFromFile)
                    .map(classifier::matchClass)
                    .collect(Collectors.toList());

            Map<String, Integer> collect = res.stream()
                    .collect(Collectors.toMap(s -> s, s -> 1, Integer::sum));
            confusionTable.put(entry.getKey(), collect);

            long right = res.stream()
                    .filter(s -> s.equals(entry.getKey()))
                    .count();
            long wrong = res.size() - right;

            System.out.println(String.format("%10s: %6d true, %6d false, %6d total",
                    entry.getKey(), right, wrong, res.size()));
        }

        ArrayList<Double> precisions = new ArrayList<>();
        ArrayList<Double> recalls = new ArrayList<>();
        int diagonal = 0;

        System.out.println();
        System.out.println("Class classification quality:");

        for (Map.Entry<String, Map<String, Integer>> entry
                : confusionTable.entrySet()) {
            Integer tp = entry.getValue().get(entry.getKey());
            diagonal += tp;

            int rowForRecall =  entry.getValue().values().stream()
                    .mapToInt(i -> i)
                    .sum();

            int columnForPrecision = confusionTable.entrySet().stream()
                    .map(Map.Entry::getValue)
                    .mapToInt(map -> map.get(entry.getKey()))
                    .sum();

            double recall = ((double) tp) / rowForRecall;
            double precision = ((double) tp) / columnForPrecision;
```

```
            System.out.println(String.format("%10s precision: %.3f",
                    entry.getKey(), recall));
            System.out.println(String.format("%10s recall:    %.3f",
                    entry.getKey(), precision));
            System.out.println();

            precisions.add(precision);
            recalls.add(recall);
        }

        int size = classDirs.size();

        int all = confusionTable.values().stream()
                .flatMapToInt(map -> map.values().stream().mapToInt(i -> i))
                .sum();

        double A = ((double) diagonal) / all;
        double P = precisions.stream().mapToDouble(i -> i).sum() / size;
        double R = recalls.stream().mapToDouble(i -> i).sum() / size;

        double a = 0.5d;
        double F = 1d / ((a / P) + (1 - a) / R);

        System.out.println("Overall classification quality:");
        System.out.println(String.format("accuracy:  %.3f", A));
        System.out.println(String.format("precision: %.3f", P));
        System.out.println(String.format("recall:    %.3f", R));
        System.out.println(String.format("F-measure: %.3f", F));
    }
}
```