Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**ЗАСТОСУВАННЯ НЕЙРОМЕРЕЖЕВИХ АЛГОРИТМІВ ДЛЯ АНАЛІЗУ СЕМАНТИЧНОЇ СХОЖОСТІ ТЕКСТІВ**»

Виконала: студентка 4-го року навчання,

Освітньої програми «Комп'ютерні науки» 122

Єсипова Дар'я Андріївна

Керівник Олецький О.В.

кандидат тех. наук, доцент

Рецензент _____

Кваліфікаційна робота захищена з оцінкою _____

Секретар ЕК _____

«____» _____ 20____ р.

Київ – 2023

ГРАФІК ПІДГОТОВКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Підпис наукового керівника
1	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи	жовтень	
2	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	жовтень – листопад	
3	Складання плану каліф. роботи та узгодження з науковим керівником	листопад	
4	Написання розділів роботи	листопад – березень	
5	Проміжний контроль виконання роботи	лютий	
6	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	січень – березень	
7	Розділ 1 – Розділ 3: Постановка проблеми, теоретичні основи, огляд літературних джерел	лютий	
8	Розділ 4 – Розділ 6: Аналіз реалізації використаних методів	березень	
9	Написання висновків	квітень	
10	Попередній захист кваліфікаційної роботи на засіданні кафедри	до 13 травня	
11	Подання кваліфікаційної роботи на кафедру з усіма супроводжувальними документами	до 22 травня	
12	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	до 29 травня	

ВСТУП

У сучасному інформаційному суспільстві, яке характеризується великим обсягом текстової інформації, виникає необхідність у розвитку ефективних методів обробки та аналізу текстів. Зростання обсягу текстових даних, їх різноманітність та складність поставляють перед науковцями та практиками складні завдання з виявлення та розуміння зв'язків між текстами, виявлення семантичних залежностей та розробки інтелектуальних систем для автоматичної обробки текстової інформації.

Ця робота присвячена проблемам аналізу та вимірювання подібності текстових документів. У контексті постійного зростання обсягу текстових даних у сучасному світі, такий аналіз набуває все більшої актуальності і стає ключовим для багатьох доменів, включаючи пошукові системи, рекомендаційні системи, аналітику соціальних мереж та багато інших.

Метою цієї роботи є розробка та порівняння різних методів для вимірювання схожості текстових документів та виявлення залежностей між ними. Подібність текстових документів є важливою характеристикою, яка може бути використана для класифікації, кластеризації, пошуку інформації, а також для розробки інтелектуальних систем, здатних розуміти та обробляти текстову інформацію.

Об'єктом дослідження є текстові документи, які можуть включати різні типи текстів, від коротких фрагментів до повних документів. Предметом дослідження є методи та алгоритми аналізу та порівняння текстових документів з метою вимірювання їх схожості.

У даній роботі будуть розглянуті такі програмні реалізації мовою Python методів аналізу та вимірювання подібності текстових документів як TF-IDF, косинусна міра подібності, подібність Жаккара, MLP та BERT. Ці методи вже мають визнання в галузі обробки природної мови та текстового аналізу і

показують добрі результати у вимірюванні схожості текстів та виявленні залежностей між ними. Додатково робота передбачає їх порівняння між собою з метою визначення їхньої ефективності та придатності для вимірювання схожості текстових документів. Порівняльний аналіз різних методів дозволить виявити їх переваги та недоліки, а також визначити, які методи найбільш точно та надійно вимірюють схожість між текстами.

Огляд літератури, який міститься в роботі, дасть можливість ознайомитися з попередніми дослідженнями в області текстового аналізу, методів вимірювання схожості текстових документів та їх використання в різних доменах. Цей огляд літератури дозволить розібратися з актуальними тенденціями та напрямами досліджень у даній галузі.

Методологічна основа роботи базується на теорії машинного навчання, нейромережах, статистичних методах аналізу даних та обробки природної мови. Використання цих методів дозволить розробити ефективні алгоритми для вимірювання подібності текстів та отримати практичні результати, які можуть бути застосовані у різних сферах.

Отже, дана робота має важливу актуальність у контексті розвитку методів аналізу та вимірювання схожості текстових документів. Дослідження, проведені у цій роботі, знайдуть застосування у різних галузях, де обробка та аналіз тексту є ключовим завданням.

1. ІНДЕКС ПОДІБНОСТІ ЖАККАРА

Після того, як люди почали опановувати алгоритми та використовувати їх у повсякденному житті, з'явилися надзвичайні технології, які приводять до спрощення попередніх задач та автоматизації їх вирішення. Ці технології неухильно розвиваються, забезпечуючи нові можливості та перетворюючи наш спосіб сприйняття та обробки інформації.

Ми легко можемо сприйняти на слух семантичну схожість або відмінність текстів і сьогодні машина може, як людина, сприйняти текст, обробити його і зробити виновок відносно поставленої задачі. Самі по собі слова для машини – лише набір символів, який не несе в собі жодного сенсового навантаження, і жоден електронний пристрій, яким би потужним він не був, не відповість на задане питання до тих пір поки йому не навчать тому як це зробити. Завдяки чітким правилам, які може надати машині людина, наборам цим символів будуть присвячуватись деякі значення або індекси. Певним заданим методом ці значення в подальшому будуть використані для обробки, яка в результаті своєї роботи прийде до кінцевого результату, що і буде відповіддю на питання, на яке без цих правил вона відповісти, очевидно, не могла. Проте цікаво дослідити ланцюжок дій і процесів, що відбуваються в проміжку між поставленою задачею і результатом.

1.1. Загальні положення методу

Індекс подібності Жаккара, також відомий як коефіцієнт подібності Жаккара, використовується для порівняння елементів двох наборів, з метою визначення, які елементи є спільними, а які відмінними. Це вимірювання акцентує увагу на схожості між кінцевими наборами вибірок і формально визначається як відношення розміру перетину до розміру об'єднання цих наборів. Індекс подібності Жаккара є мірою подібності для двох наборів даних,

яка варіюється від 0% до 100%. Чим вищий відсоток, тим більше схожі дві популяції. Математичне представлення індексу записується так:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Зображення 1.1.1. Математичне представлення індексу подібності Жаккара

Розбираючи формулу, розуміємо, що для того щоб обчислити індекс Жаккара треба поділити кількість спільних елементів на загальну кількість елементів у будь-якому з наборів з подальшим множенням результата на 100. Результатом буде відсоток вимірювання подібності. Відповідно, щоб знайти відстань Жаккара, потрібно просто відняти відсоткове значення від 1. Наприклад, якщо вимірювання подібності становить 35%, то відстань Жаккара (1 - 0,35) дорівнює 0,65 або 65%.

Недоліком такого підходу є те, що за умови надходження малих вибірок або вибірок із відсутніми спостереженнями він може дати з великою долею ймовірності некоректну відповідь.

Індекс Жаккара використовується в згорткових нейронних мережах, які займаються розпізнаванням зображень, для вимірювання точності виявлення об'єктів. Наприклад, якщо ми маємо алгоритм комп'ютерного зору, який намагається знайти обличчя на зображенні, індекс Жаккара допомагає оцінити, наскільки добре результати комп'ютерного виявлення облич відповідають навчальним даним. Він надає кількісну міру подібності між знайденими обличчями і очікуваними обличчями на основі об'єктних позначень.

1.2. Реалізація подібності Жаккара

Для реалізації будь-якого з методів оцінки подібності текстів використовуватиметься один спільний для всіх методів процес – процес розділення цілого тексту на список всіх слів.

Нехай на вході маємо два документа documentA та documentB для подальшого порівняння їх між собою. Після отримання списку із словами потрібно обчислити частоту входжень слів у двох документах. Така процедура виглядає наступним чином:

```
words = set([word.lower().translate(str.maketrans('', '', string.punctuation))
             for word in documentA.split() + documentB.split()
             if word.lower() not in stop_words])
counts = np.zeros((2, len(words)))
for i, doc in enumerate([documentA, documentB]):
    for j, word in enumerate(words):
        counts[i, j] = doc.split().count(word) if word != "the" else 0.1
```

Зображення 1.2.1. Код подібності Жаккара: частота входжень слів у документах

В цій частині коду створюється множина, що містить унікальні слова з обох документів. Кожне слово проходить через послідовні операції:

- Перетворення на нижній реєстр, щоб уникнути дублікатів через регістр;
- Видалення всіх знаків пунктуації;
- Перевірка на належність до списку попередньо завантажених стоп-слів.

Після цього створюється масив нулів розміром 2*N, де 2 відповідає кількості документів, а N – кількості унікальних слів у множині *words* унікальних слів. Кожен елемент масиву початково встановлений на значення 0, а після проходження циклів for заповнюється числами – кількістю входжень унікальних слів з множини *words* у кожному з двох документів.

Далі обчислюємо матрицю подібності Жаккара для набору слів:

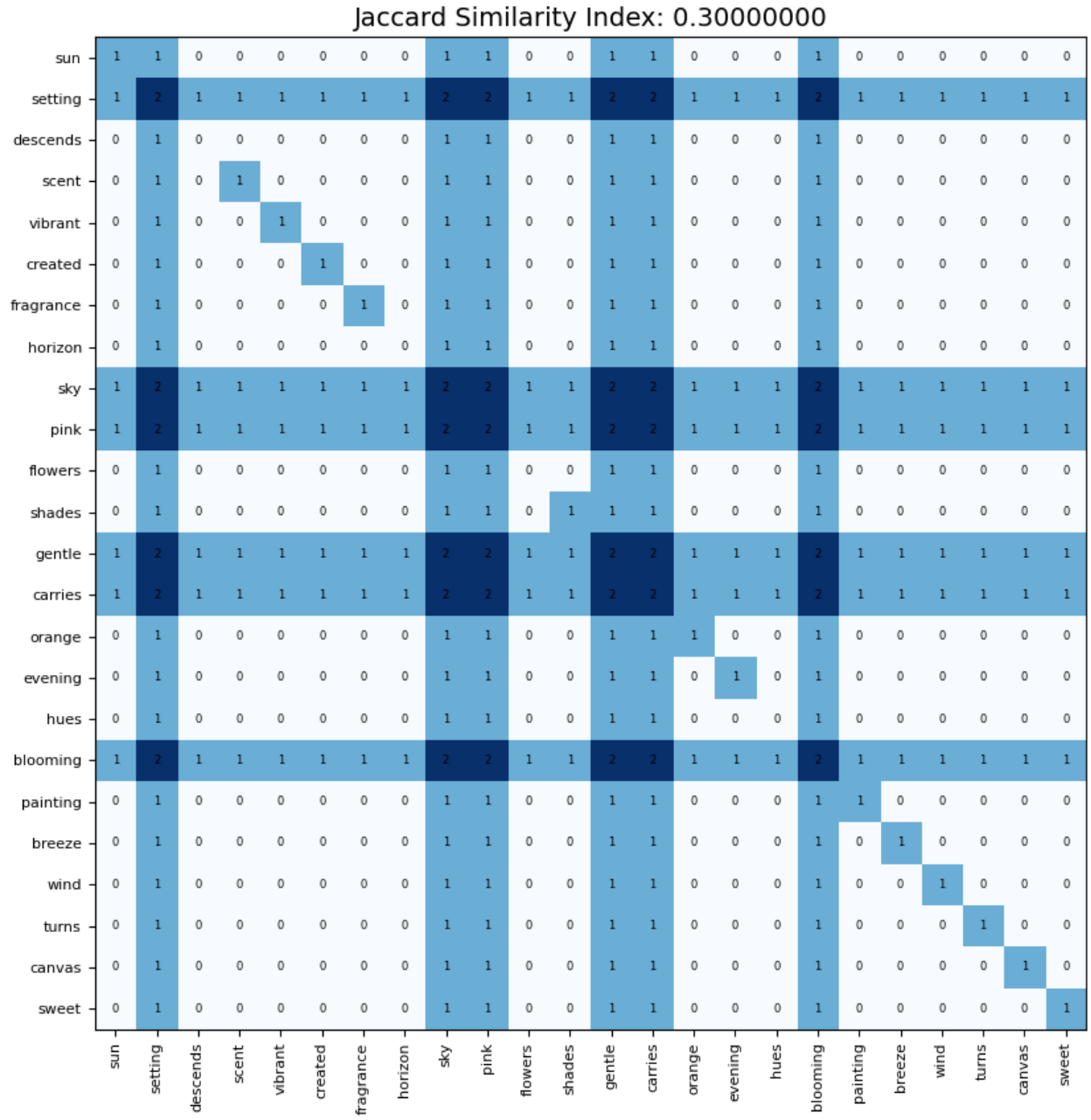
```
intersection = np.sum(np.min(counts, axis=0))
union = np.sum(np.max(counts, axis=0))
jaccard_similarity = intersection / union
n = len(words)
grid = np.zeros((n, n))
for i in range(n):
    for j in range(n):
        if i == j:
            grid[i, j] = counts[:, i].sum()
        else:
            grid[i, j] = counts[:, i].min() + counts[:, j].min()
```

Зображення 1.2.2. Код подібності Жаккара: матриця подібності Жаккара для набору слів

Послідовно знаходимо перетин *intersection* (кількість спільних слів) та об’єднання *union* (загальну кількість слів). Для цього використовуємо функції np.min та np.max відповідно для знаходження мінімального та максимального значення для кожного слова, а потім ці значення сумуються. В значенні *jaccard_similarity* міститься коефіцієнт подібності Жаккара як відношення перетину до об’єднання. Це число відображає ступінь схожості між документами - чим вищі значення, тим більша схожість.

Заповнення матриці відбувається в наступних циклах. Діагоналі матриці міститимуть суми кількостей входжень слів відповідно до кожного документа, інші позиції будуть заповнені сумою мінімальних значень кількостей входжень слів – це відображає міру подібності між парами слів у двох документах, враховуючи їх мінімальні кількості входжень.

1.3. Демонстрація роботи методу подібності Жаккара



Зображення 1.3.1. Демонстрація роботи подібності Жаккара: матриця подібності Жаккара для набору слів

1.4. Висковки за главою 1

Обмежена точність: Метод Жаккара використовує простий підхід до вимірювання схожості між документами шляхом порівняння кількості спільних елементів з загальної кількості елементів. Однак, цей підхід може бути

недостатнім для визначення семантичної схожості, оскільки він не враховує смисловий зміст та структуру тексту.

Недостатня розрізнюваність: Метод Жаккара може не виявити семантичну схожість між документами, якщо вони містять різні формулювання, синоніми або деяку варіацію у використанні слів. Він спроможний виявляти лише повністю співпадаючі елементи, ігноруючи нюанси та контекстуальні відтінки.

Потребує додаткових методів: Щоб отримати більш точні результати визначення семантичної схожості між документами, можна розглянути використання інших методів аналізу тексту, таких як векторні моделі, нейронні мережі або методи, що базуються на природній мові. Ці методи можуть враховувати контекстуальні залежності та семантичні зв'язки для кращої оцінки схожості.

2. TERM FREQUENCY – INVERSE DOCUMENT FREQUENCY

2.1. TF-IDF як ключовий алгоритм у сфері обробки природньої мови

TF-IDF (Term Frequency - Inverse Document Frequency) - це числова статистика, що відображає релевантність ключових слів для певних документів або, як можна сказати, надає ті самі ключові слова, які можуть ідентифікувати або класифікувати певні документи. Наприклад, уявімо блогера, який підтримує блог зі сотнею співавторів. Щойно він взяв на роботу стажиста, чия головна відповідальність полягає в щоденному додаванні нових публікацій до блогу. Виявлено, що стажисти часто не звертають уваги на теги, які використовуються для класифікації багатьох публікацій у блозі. Це створює ідеальні умови для використання алгоритму TF-IDF, який може автоматично ідентифікувати теги для блогерів. Це заощаджує багато часу як для блогерів, так і для стажистів, оскільки їм більше не потрібно турбуватися про теги.

TF-IDF є одним з ключових алгоритмів у сфері обробки природної мови та інформаційного пошуку. Цей алгоритм використовується для вимірювання важливості термів (слів) у документі, що залежить від їх частоти в цьому документі та інших документах в колекції.

2.2. Компоненти алгоритму

2.2.1. TF - Term Frequency

Алгоритм TF-IDF складається з двох компонентів: Term Frequency (TF) і Inverse Document Frequency (IDF).

Term Frequency (TF) відображає частоту вживання термів у документі. Вона вимірює, наскільки часто конкретний термін зустрічається в документі.Відомо, що загальна довжина документів може варіюватися від

маленької до великої, тому існує ймовірність того, що будь-який термін може зустрічатися частіше у великих документах порівняно з маленькими. Отже, щоб вирішити цю проблему, кількість термінів у документі ділиться на загальну кількість термінів у цьому документі, щоб знайти частоту термінів. Вираховується за формулою TF(t, d) = (кількість разів, коли термін t зустрічається у документі d) / (загальна кількість термінів у документі d).

2.2.2. IDF - Inverse Document Frequency

Inverse Document Frequency (IDF) є мірою важливості слова в усьому корпусі (сукупності всіх проаналізованих документів). Ідея полягає в тому, що якщо слово зустрічається в багатьох документах, воно не додає багато інформації. Під час обчислення частоти термінів у документі можна помітити, що алгоритм обробляє всі ключові слова однаково, не має значення, якщо це стоп-слово, наприклад «of», що є помилкою. Усі ключові слова мають різну важливість. Припустимо, стоп-слово «of» присутнє в документі 2000 разів, але воно не приносить користі або має дуже менше значення, саме для цього і призначений IDF. Зворотна частота документа призначає меншу вагу частим словам і призначає більшу вагу словам, які є рідкісними.

Обчислюється шляхом ділення кількості документів (N) на загальну кількість документів, у яких зустрічається термін, і взяття цього логарифмічного значення. Якщо термін зустрічається 10 разів у 10 000 документах, IDF дорівнює 3. Термін, що зустрічається 100 разів у тому самому наборі документів, матиме значення IDF 2. Якщо термін зустрічається в усіх документах, значення IDF дорівнює 0,0. Логарифмічне значення використовується для зменшення широкого діапазону значень, які може мати IDF. Вираховується за формулою IDF(t) = log(N / df(t)), де N - загальна кількість документів у колекції, а df(t) - кількість документів, в яких зустрічається термін t.

2.2.3. TF-IDF

Більша кількість випадків появи слова в документах дасть вищу частоту термінів, а менша кількість появ слова в документах дасть вищу важливість (IDF) для цього ключового слова, яке шукається в конкретному документі. TF-IDF — це множення TermFrequency і зворотної частоти документа (IDF). TF-IDF обчислюється як добуток TF і IDF: TF-IDF(t, d) = TF(t, d) * IDF(t).

В результаті, терміни, які часто зустрічаються в конкретному документі, але рідко зустрічаються в інших документах колекції, матимуть високе значення TF-IDF. Це означає, що такі терміни є важливими для цього документа і можуть використовуватись для класифікації, ідентифікації теми або пошуку релевантних документів.

2.3. Обмеження алгоритму та способи їх уникнення

Алгоритм TF-IDF знаходить широке застосування в багатьох областях, включаючи пошукові системи, автоматичну категоризацію документів, рекомендації, аналіз текстів та багато іншого. Його ефективність полягає в здатності виділити важливі терміни та знизити вагу загальних та менш значущих термінів у документах.

Існують деякі обмеження алгоритму TF-IDF, які необхідно розглянути. Основним обмеженням TF-IDF є те, що алгоритм не може ідентифікувати слова навіть із незначною зміною часу, наприклад, алгоритм розглядатиме «go» та «goes» як два різні незалежні слова, так само він розглядатиме « грати» і «грати», «маркувати» і «маркувати», «рік» і «роки» як різні слова. Через це обмеження при застосуванні алгоритму TF-IDF інколи він дає деякі несподівані результати. Іншим обмеженням TF-IDF є те, що він не може перевіряти семантику тексту в документах, і через це він корисний лише на лексичному рівні. Він також не може перевірити співпадіння слів. Існує багато методів, які

можна використовувати для покращення продуктивності та точності, таких як дерева рішень, класифікатори на основі шаблонів або правил, класифікатори SVM, класифікатори нейронної мережі та класифікатори Байєса.

З прогресом часу народжуються нові алгоритми, які допомагають подолати обмеження старих. Один із таких алгоритмів — процес стемінгу, який дозволяє вирішити проблему, пов'язану з тим, що TF-IDF не розрізняє, що "play" та "plays" в основному представляють одне і те ж саме слово. Стемінг - це процес в обробці природної мови, який використовується для зведення різних форм одного слова до його базової форми, відомої як стем. Наприклад, "play" і "plays", або "played", можна звести в одне загальне представлення, наприклад, "play". Стемінг здійснюється за допомогою правилових або евристичних алгоритмів, які враховують мовні правила та закономірності. Ці алгоритми використовуються для відкидання суфіксів та закінчень, зберігаючи лише основу слова.

Використання стемінгу у аналізі текстів дозволяє зменшити розмірність словникового простору, а також покращує точність і релевантність при роботі з пошуковими системами, категоризацією текстів, машинним перекладом та іншими завданнями обробки природної мови.

Другим покращенням є використання стоп-слів, які виключаються з обробки даних, щоб фільтрувати та видаляти слова, які не несуть значення, наприклад, "the" або "a". Це гарантує, що отримані результати містять лише корисні слова.

Ці нові підходи сприяють покращенню точності та релевантності аналізу даних. Вони дозволяють зберегти сутність тексту, зменшуючи варіацію через морфологічні форми та виключення незначущих слів. Застосування цих методів допомагає збільшити якість та ефективність алгоритму TF-IDF в процесі обробки текстових даних.

2.4. Реалізація TF-IDF

2.4.1. Попередня обробка вхідних даних

Нехай дано два документи для подальшого порівняння між собою. Аналогічно до попередньої реалізації, попередньо оброблюємо вхідні тексти: розбиваємо на окремі слова, видаляємо знаки пунктуації та зводимо до нижнього регістру. Отримані слова зберігаються в *bagOfWordsA* та *bagOfWordsB*. Підраховані кількості входжень слів з кожного документа зберігаються в словнику numOfWordsA та numOfWordsB відповідно.

```
punctuations = string.punctuation

bagOfWordsA = [word.lower().translate(str.maketrans('', '', punctuations)) for word in documentA.split()]
bagOfWordsB = [word.lower().translate(str.maketrans('', '', punctuations)) for word in documentB.split()]

uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))

numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1
numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1
```

Зображення 2.4.1.1. Код TF-IDF: попередня обробка тексту

2.4.2. Реалізація TF

Далі нам потрібні окремі функції для обчислення компонентів алгоритму з подальшим множенням між собою отриманих результатів.

Реалізація TF функцією *computeTF*:

```
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict
```

Зображення 2.4.2.1. Код TF-IDF: реалізація TF

Маємо список слів *bagOfWords* та словник *wordDict*, де ключі – це слова, а значення – кількість входжень цього слова у документі. Функція перебирає

слова у *wordDict* та обчислює TF шляхом ділення кількості входжень кожного слова на загальну кількість слів у *bagOfWords*. Результати обчислень зберігаються в словнику *tfDict*, де ключі – це слова, а значення – відповідне значення TF. На виході функція повертає словник *tfDict*.

2.4.3. Реалізація IDF

Реалізуємо наступну частину алгоритму IDF функцією *computeIDF*:

```
def computeIDF(documents):
    N = len(documents)
    print("\nIDF of: ")

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
        print(f'{word:>15}: {idfDict[word]:>10}')
    return idfDict
```

Зображення 2.4.3.1. Код TF-IDF: реалізація IDF

Ця функція обчислює значення IDF для кожного слова, використовуючи список *documents*, що містить словники з кількостями входжень слів у відповідних документах.

Функція спочатку визначає загальну кількість документів N, а потім створює словник *idfDict* зі всіма словами з першого документа, де значення початково встановлені на 0. Далі функція обчислює кількість документів, в яких кожне слово має значення більше 0. За допомогою цих значень обчислюється IDF для кожного слова, що передбачає взяття натурального логарифму відношення загальної кількості документів N до кількості документів, в яких воно зустрічається. На виході функція повертає словник *idfDict*, де ключі - це слова, а значення - відповідне значення IDF.

2.4.4. Реалізація алгоритму TF-IDF

Код для фінального кроку алгоритму:

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

Зображення 2.4.4.1. Код TF-IDF: реалізація TF-IDF

Функція обчислює значення TF-IDF для кожного слова з *tfBagOfWords* використовуючи IDF значення з *idfs*. Словники *tfBagOfWord* та *idfs* містять в собі ключі, що є словами, і значення, що є відповідними значеннями TF та IDF.

Функція перебирає слова у *tfBagOfWords* і обчислює TF-IDF, множачи значення TF кожного слова на відповідне значення IDF. Результати обчислень зберігаються у словнику *tfidf*, де ключі - це слова, а значення - відповідне значення TF-IDF. На виході функція повертає словник *tfidf*.

2.5. Демонстрація роботи TF-IDF

	and	as	blooming	breeze	by	canvas	carries	\
0	0.131558	0.000000	0.131558	0.184901	0.000000	0.000000	0.131558	
1	0.125290	0.176091	0.125290	0.000000	0.176091	0.176091	0.125290	

	created	descends	evening	flowers	fragrance	gentle	horizon	\
0	0.000000	0.000000	0.000000	0.131558	0.000000	0.131558	0.184901	
1	0.176091	0.176091	0.176091	0.125290	0.176091	0.125290	0.000000	

	hues	into	is	it	of	orange	over	\
0	0.000000	0.000000	0.184901	0.000000	0.263117	0.131558	0.184901	
1	0.176091	0.176091	0.000000	0.176091	0.250580	0.125290	0.000000	

	painting	pink	scent	setting	shades	sky	sun	\
0	0.184901	0.131558	0.184901	0.131558	0.184901	0.131558	0.131558	
1	0.000000	0.125290	0.000000	0.125290	0.000000	0.125290	0.125290	

	sweet	the	turns	vibrant	wind	with
0	0.184901	0.657792	0.000000	0.184901	0.000000	0.131558
1	0.000000	0.626450	0.176091	0.000000	0.176091	0.125290

Зображення 2.5.1. Демонстрація TF-IDF

2.6. Висновки за главою 2

Документ може бути представлений тисячами атрибутів, кожен з яких записує частоту певного слова (наприклад, ключового слова) або фрази в документі. Таким чином, кожен документ є об'єктом, представленим term-frequency вектором.

Term-frequency вектори зазвичай дуже довгі та розріджені (тобто вони мають багато нульових значень). Програми, що використовують такі структури, включають в себе пошук інформації, кластеризацію текстових документів, біологічну таксономію та відображення ознак генів. Класичні підходи до вимірювання відстані мають обмежену ефективність у випадках, коли маємо

справу з такими розрідженими числовими даними. Наприклад, два term-frequency вектори можуть мати багато спільних значень 0, що означатиме, що відповідні документи мають небагато спільних слів, але це не робить їх схожими. Потрібен такий метод вимірювання відстані, який буде головним чином зосереджуватись на словах, які є спільними у двох документах, і на частоті появи таких слів. Іншими словами, потрібна така міра для числових даних, яка ігнорує нульові збіги.

Тобто тепер, коли значення TF-IDF обчислено, відстань між двома документами можна визначити шляхом обчислення косинусної подібності між цими значеннями. Це робиться за допомогою значень TF-IDF по кожному із слів в документах.

3. КОСИНУСНА МІРА ПОДІБНОСТІ

3.1. Індекси подібності

У контексті теми подібності, індекс відноситься до числового значення або міри, яка використовується для вимірювання ступеня схожості або відмінності між двома об'єктами, наприклад, текстовими документами. Індекс дозволяє кількісно визначити, наскільки два об'єкти близькі один до одного з певного погляду.

Принципова різниця між індексами подібності та всіма іншими індексами полягає в тому, що отримані значення порівнюються не з деякою еталонною шкалою, а визначають взаємну упорядкованість об’єктів відносно один одного. Спочатку можна розрізнити показники:

- Основані на якісних даних (присутність або відсутність видів на пробній площі)
- Основані на кількісних даних (велика кількість видів або їх численність)
- Міри асоціації, що виражають різні відношення кількості ознак що співпадають до їх спільної кількості і близькі до них коефіцієнти сполученості.
- Вибіркові коефіцієнти зв’язку типу кореляції (нормовані «косинусні» міри)
- Показники відстані в метричному просторі.

Існує велика кількість способів для вираження мір подібності або відстані між об’єктами. З плином часу і розвитком галузі дослідники зменшили цю кількість типів до двох, подаючи асоціативні міри *природнім поширенням косинусної міри* на номінальній шкалі. Також виділялися ймовірнісні міри, інформаційні міри та перетворені показники. Однак оскільки оцінюється ймовірність того, чи будуть об’єкти ідентичними, всі міри в деякому ступені є ймовірнісними і являють собою деякі алгебраїчні вирази.

3.2. Косинусна відстань

Косинусна відстань — це різниця кутів між лініями від початку координат до двох точок у N-вимірному просторі.

Різницю косинусів можна обчислити за формулою евклідового скалярного добутку. Нехай x і y — два вектори для порівняння. Використовуючи міру косинуса як функцію подібності, ми маємо:

$$\text{подібність} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Зображення 3.2.1. Косинусна міра подібності

Це скалярний добуток двох векторів, поділений на множення довжин векторів. Значення косинуса 0 означає, що два вектори розташовані під кутом 90 градусів один до одного (ортогональні) і не збігаються, а коли два вектори паралельні, різниця косинусів дорівнює 1,0.Цю функцію потрібно застосовувати до всіх комбінацій документів у корпусі. Різниця від A до B дорівнює різниці від B до A, а різниця від A до A дорівнює 1,0 - чим ближче значення косинуса до 1, тим менший кут і тим більша збіг між векторами.

3.3. Реалізація косинусної міри подібності

Реалізація функції, що обчислить косинусну міру подібності між двома векторами *vector1* та *vector2*:

```python
def cosine_similarity(vector1, vector2):
    dot_product = sum(p*q for p,q in zip(vector1, vector2))
    magnitude1 = math.sqrt(sum([val**2 for val in vector1]))
    magnitude2 = math.sqrt(sum([val**2 for val in vector2]))
    return dot_product/(magnitude1*magnitude2)
```

Зображення 3.3.1. Код косинусної міри подібності: реалізація методу

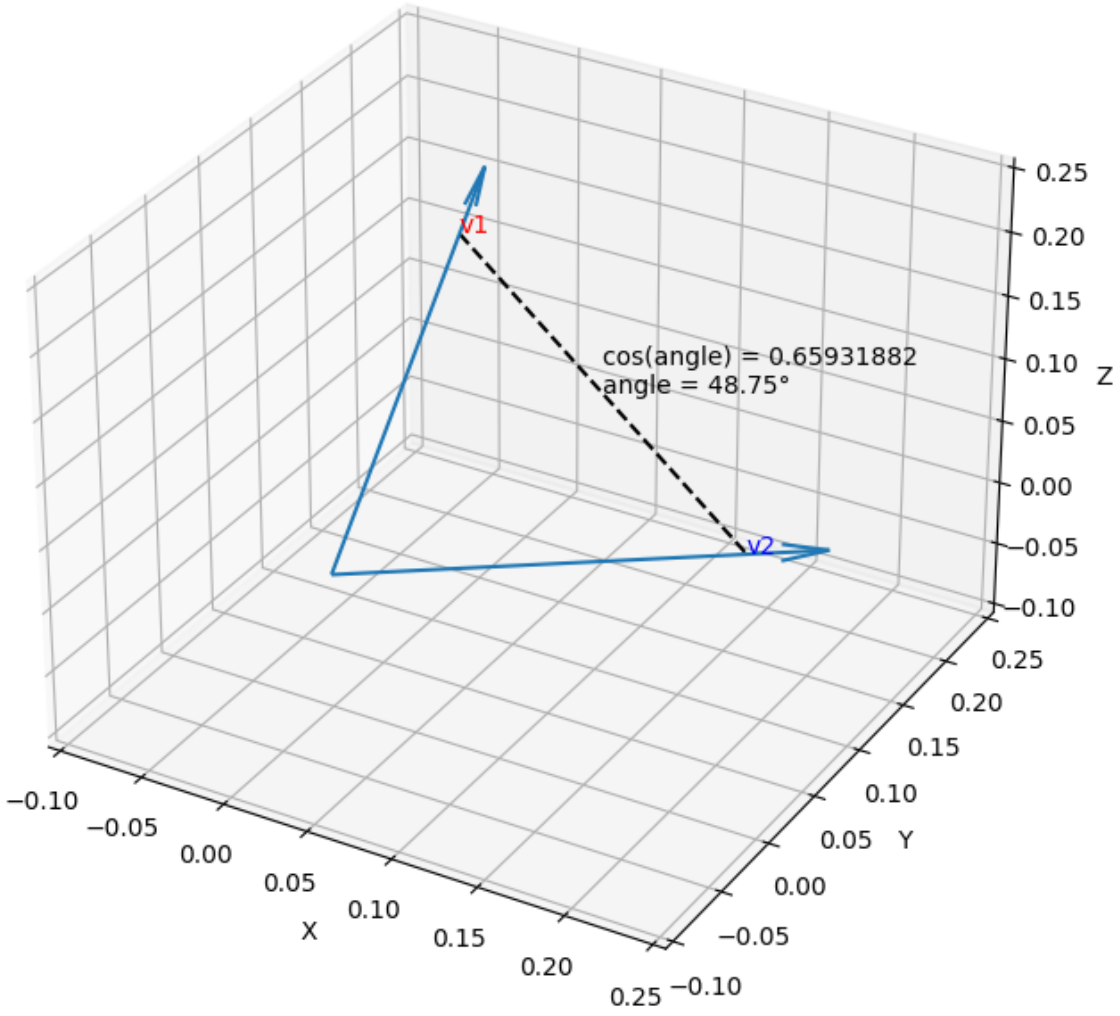
Тут ми визначаємо функцію *cosine_similarity()*, яка приймає на вхід два вектори представлені у вигляді списків чисел і обчислює косинусну міру подібності між ними.

Алгоритм полягає в наступному:

1. Обчислюємо скалярний добуток векторів *vector1* і *vector2*, множачи відповідні елементи кожного вектора та складаючи результати. Використовується функція *zip()*, яка об'єднує елементи двох списків *vector1* і *vector2* в пари, та оператор *, який виконує поелементне множення. Функція *sum()* сумує всі отримані добутки.
2. Обчислюємо магнітуду (евклідову норму) кожного вектора, обчислюючи квадратний корень з суми квадратів елементів кожного вектора. Використовується генератор списку *[val**2 for val in vector1]*, що обчислює квадрат кожного елемента *val* у *vector1*. Функція *sum()* сумує всі отримані квадрати, а потім за допомогою *math.sqrt()* обчислюється квадратний корінь з суми. Аналогічні дії виконуємо з другим вектором.
3. Обчислюємо міру подібності шляхом ділення скалярного добутку до добутку магнітуд векторів. Результат повертається як значення функції *return dot_product/(magnitude1*magnitude2)*.

В результаті отримуємо число від 0 до 1, де 0 означає повну розбіжність між векторами, а 1 – повну подібність.

3.4. Демонстрація роботи косинусної міри подібності



Зображення 3.4.1. Демонстрація косинусної міри подібності

3.5. Висновки за главою 3

В порівнянні з подібністю Жаккара, косинусна міра подібності є значно ефективнішим методом, якщо мова йде про порівняння текстів на семантичну схожість, оскільки такий метод враховує семантичний зміст тексту, має гнучкість у вимірюванні схожості, надає числові результати та використовує векторне представлення тексту.

4. НЕЙРОННІ МЕРЕЖІ

4.1. Перцептрон

Один з найбільш корисних та цікавих типів нейронних мереж знаходиться в сфері дослідження багатошарових перцептронів. Нейронна мережа, яка використовує штучний інтелект для відстеження особливостей у вхідних даних, відома як *перцептрон*. Перцептрон складається зі штучних нейронів, які взаємодіють за допомогою простих логічних елементів з двійковими виходами. Кожен штучний нейрон має вузол, вхід, ваги і вихід, які відповідають клітинному ядру, дендритам, синапсу і аксону, відповідно, біологічного нейрона.

Ця наукова область досліджує, як прості моделі біологічного мозку можуть бути використані для розв'язання складних обчислювальних завдань, зокрема в машинному навчанні. Основна мета полягає не в створенні реалістичних моделей мозку, а в розробці надійних алгоритмів та структур даних, які здатні моделювати складні проблеми.

4.2. Прогнозувальна потужність нейронних мереж

Нейронні мережі відрізняються потужністю в їхній здатності вивчати представлення навчальних даних та ефективно пов'язувати їх зі змінною вихідного значення, що потребує прогнозування. Вони вчаться відображенням - з математичної точки зору, вони здатні вивчити будь-яку функцію відображення, і було доведено, що вони є універсальними алгоритмами наближення.

Прогнозувальна потужність нейронних мереж залежить від ієрархічної або багаторівневої структури. Структура даних може відображати об'єкти на різних масштабах або з різною роздільною здатністю, а потім об'єднувати їх у більш складні об'єкти, наприклад, від простих ліній до набору ліній, а потім до форм. Така ієрархічна побудова дозволяє нейронним мережам моделювати та розпізнавати складні зразки та структури в даних.

4.3. Ваги

Вагові коефіцієнти моделі — це всі параметри (включно з такими, які можна навчити, і які не можна навчити) моделі, які, у свою чергу, є всіма параметрами, що використовуються в шарах моделі.

Подібно до лінійної регресії, в кожному нейроні присутнє зміщення, яке можна розглядати як вхідні дані з фіксованим значенням 1,0. Зміщення також піддається вагуванню разом з іншими вхідними даними нейрона, що дозволяє нейронній мережі враховувати вплив зміщення при обчисленнях і прийнятті рішень.

Наприклад, нейрон може мати два вхідні сигнали, для яких потрібно три вагові коефіцієнти: по одному для кожного вхідного сигналу та один для зміщення. Ці вагові коефіцієнти часто ініціалізуються малими випадковими значеннями, наприклад, в діапазоні від 0 до 0,3. Звісно, також можна використовувати більш складні методи ініціалізації ваг.

4.4. Регуляризація

Аналогічно до лінійної регресії, ваги в нейронній мережі можуть вказувати на складність та крихкість моделі. Значення вагових коефіцієнтів можуть впливати на ефективність та точність мережі. Щоб контролювати значення ваг, рекомендується використовувати методи *регуляризації*. Ці методи дозволяють управляти значеннями ваг та уникати перенавчання мережі, що може виникнути при занадто складних або надмірно специфікованих моделях. Регуляризація допомагає зберегти розумні значення ваг та підвищує стійкість та універсальність мережі.

Регуляризація - це метод у машинному навчанні, який використовується для контролю над вагами моделі з метою уникнення перенавчання та поліпшення її загальної універсальності. Вона впроваджує додаткові обмеження в модель з метою зменшення її складності. Ці обмеження, як

правило, виражаються як штрафи у функції втрати, що стягуються за великі значення ваг. Це допомагає утримувати значення ваг на адекватному рівні та уникнути їх надмірного зростання.

Поширені методи регуляризації - це L1-регуляризація (також відома як Lasso) і L2-регуляризація (або Ridge). L1-регуляризація додає штраф до функції втрати, пропорційний сумі абсолютних значень ваг, тоді як L2-регуляризація додає штраф, пропорційний сумі квадратів ваг. Обидва методи допомагають контролювати значення ваг та зменшують ризик перенавчання моделі, що дозволяє їй краще узагальнювати на нові дані.

4.5. Функція активації

Зважені вхідні дані нейрона надсилаються через *функцію активації*, яку часто називають функцією передачі.

Функція активації виконує просте відображення зваженого сумарного входу на вихід нейрона. Вона отримала назву "функція активації" через свою роль у керуванні порогом, за яким нейрон активується, а також у визначенні сили вихідного сигналу.

У минулому поширеною практикою було використання простих крокових функцій активації. Наприклад, коли сума зважених вхідних даних перевищує порогове значення 0,5, нейрон видає вихідне значення 1,0; у протилежному випадку виводиться значення 0,0.

Зазвичай використовуються нелінійні функції активації. Це дозволяє мережі поєднувати вхідні дані більш складними способами та розширює її можливості для моделювання різноманітних функцій. Нелінійні функції, такі як логістична функція, відома також як сигмоїдна функція, використовуються для отримання значень в діапазоні від 0 до 1 з характерним s-подібним розподілом. Функція гіперболічного тангенса, також відома як tanh, виводить значення у діапазоні від -1 до +1, зберігаючи подібний розподіл.

Нещодавні дослідження показали, що функція активації ReLU (випрямляча) забезпечує кращі результати у багатьох випадках. Ця функція дозволяє передавати лише додатні значення та ефективно вирішує проблему зникнення градієнту у глибоких нейронних мережах.

4.6. Архітектура

Нейрони, що взаємодіють у мережах нейронів, створюють захоплюючу систему, в якій кожен нейрон відіграє важливу роль. Вони поєднуються, утворюючи велику мережу, яка здатна до складних обчислень та інформаційної обробки. Вони організовані у групи, відомі як шари.

Шар - це колекція нейронів, розташованих у послідовному порядку. Це структурна одиниця, яка групує нейрони за їхніми функціями та ролями у мережі. Мережа може містити один або більше шарів, залежно від її складності та завдань, які вона виконує.

Архітектура нейронів у мережі, їхні зв'язки та взаємодія визначаються топологією мережі. Це описує геометричну структуру мережі, включаючи розташування та зв'язки між шарами та нейронами. Топологія визначає, як інформація потоком розповсюджується у мережі та як виконуються обчислення, роблячи її однією з ключових характеристик нейронних мереж.

Нижній рівень, який отримує вхідні дані з нашого набору даних, називається *видимим шаром*, оскільки це доступна для спостереження частина мережі. Зазвичай нейронна мережа зображується з видимим шаром, що містить по одному нейрону для кожного вхідного значення або стовпця в вашому наборі даних. Ці нейрони не описуються вище, а просто передають вхідне значення на наступний рівень.

Шари, розташовані після вхідного шару, називаються *прихованими,* оскільки вони не отримують безпосереднього впливу від вхідних даних.

Найпростіша структура мережі передбачає наявність одного нейрона у прихованому шарі, який безпосередньо виводить значення.

Завдяки зростанню обчислювальної потужності та ефективних бібліотек, можна побудувати дуже глибокі нейронні мережі. Глибоке навчання означає наявність багатьох прихованих шарів у вашій нейронній мережі. Вони є глибокими, оскільки історично їх навчання було надзвичайно повільним, але за сучасних методів і обладнання вони можуть навчатися за декілька секунд або хвилин.

Останній прихований рівень називається *вихідним рівнем*, і його завданням є виведення значення або вектора значень у відповідному форматі, необхідному для конкретної задачі.

Вибір функції активації на вихідному рівні сильно залежить від типу проблеми, яку ви моделюєте. Наприклад:

- У задачах регресії може бути один вихідний нейрон, який не має функції активації.
- У задачах бінарної класифікації може бути один вихідний нейрон, а для виведення значення від 0 до 1 та представлення ймовірності прогнозування класу 1 може використовуватися функція активації сигмоїди. Порогове значення 0,5 можна використовувати для перетворення цих ймовірностей на чіткі класифікаційні значення: значення менше 0,5 відповідають класу 0, а значення більше або дорівнює 0,5 відповідають класу 1.
- У задачах багатокласової класифікації у вихідному шарі може бути декілька нейронів, по одному для кожного класу (наприклад, три нейрони для трьох класів у відомій задачі класифікації квітів ірису). В такому випадку може використовуватися функція активації softmax для виведення ймовірностей, що мережа прогнозує кожен з класів. Чітке значення класифікації може бути визначене шляхом вибору результату з найвищою ймовірністю.

4.7. Навчання, підготовка вхідних даних

Після налаштування нейронну мережу потрібно навчити на нашому набору даних. Для успішного навчання нейронної мережі необхідно підготувати дані належним чином.

Перш за все, дані повинні бути числовими, такими як реальні значення. Якщо у нас є категоріальні дані, наприклад, атрибут "стать" зі значеннями "чоловічий" і "жіночий", ми можемо перетворити його на числове представлення за допомогою методу, що називається one-hot encoding. Для цього додається нова колонка для кожного значення категорії (дві колонки у випадку статі - "чоловічий" та "жіночий"), а потім для кожного рядка встановлюється значення 0 або 1 залежно від наявності відповідного значення категорії в цьому рядку.

Таке саме кодування з one-hot encoding можна застосовувати до вихідної змінної у задачах класифікації з більш ніж одним класом. Це призведе до створення бінарного вектора з однієї колонки, який можна прямо порівнювати з виводом нейрона на вихідному рівні мережі. Це, як описано вище, дозволить отримати одне значення для кожного класу.

Нейронні мережі потребують однорідного *масштабування* вхідних даних. Ви можете масштабувати дані до діапазону від 0 до 1, що називається *нормалізацією*. Іншим поширеним методом є *стандартизація*, при використанні якої кожна колонка має середнє значення 0 і стандартне відхилення 1.

Цей процес масштабування також застосовується до піксельних даних зображень. Для текстових даних, таких як слова, є різні методи кодування, наприклад, рейтинг популярності слова у наборі даних або інші методи кодування.

Вагові коефіцієнти в мережі можуть бути оновлені на основі помилок, обчислених для кожного навчального прикладу. Цей процес називається *онлайн-навчанням* і може призвести до швидких, але хаотичних змін у мережі.

Крім того, помилки можуть бути накопичені для всіх навчальних прикладів, і оновлення мережі може відбутися в кінці. Цей метод називається *пакетним навчанням* і зазвичай є більш стабільним.

Зазвичай, через великий обсяг даних та обчислювальну складність, розмір пакету, тобто кількість прикладів, на яких мережа робить прогноз перед оновленням, зазвичай зменшується до невеликого числа, наприклад, десятків або сотень прикладів.

Кількість оновлених ваг контролюється параметром конфігурації, який називається *швидкістю навчання*. Цей параметр також відомий як *крок* або *розмір кроку*, і він визначає, як швидко змінюються ваги мережі відносно помилки. Зазвичай використовуються невеликі значення швидкості навчання, такі як 0,1 або 0,01 або навіть менші.

Рівняння для оновлення ваг може бути розширене додатковими термінами конфігурації, які ви можете налаштувати.

Поняття "*імпульс*" включає в себе властивості попереднього оновлення ваги, щоб дозволити вагам продовжувати змінюватися в тому самому напрямку, навіть якщо помилка менша. Це допомагає уникнути застрягання в локальних мінімумах.

Зменшення швидкості навчання (Learning Rate Decay) використовується для поступового зменшення швидкості навчання протягом епох, що дозволяє мережі вносити значні зміни до ваг на початку навчання і менші зміни для тонкої настройки пізніше у процесі навчання.

Після навчання нейронної мережі, ми можемо використовувати її для прогнозування. Для оцінки точності моделі на невидимих даних можна зробити

прогнози на основі даних тестування або перевірки. Також, ви можете розгорнути мережу в робочому стані і використовувати її для постійного прогнозування.

Топологія мережі та остаточний набір ваг - це все, що вам потрібно зберегти від моделі. Для створення прогнозів ви вводите вхідні дані в мережу та виконуєте прямий прохід, що дозволяє отримати вихідні дані, які можна використовувати як прогнози.

4.8. Класифікація навчальних проблем

Корисно класифікувати навчальні проблеми залежно від типу використовуваних даних. Це допомагає зустрічати нові проблеми, оскільки часто подібні проблеми можна вирішити за допомогою схожих методів. Наприклад, обробка природної мови і біоінформатика використовують подібні інструменти для роботи з текстовими рядками та послідовностями ДНК. Вектори є основним елементом, з яким ми стикаємося у своїй роботі. Наприклад, страхові компанії можуть бути зацікавлені отримати вектор змінних (артеріальний тиск, частота серцевих скорочень, зріст, вага, рівень холестерину, куріння, стать) для прогнозування тривалості життя потенційного клієнта. Фермери можуть бути зацікавлені у визначенні стиглості фруктів на основі їх розміру, ваги та спектральних даних. Інженери можуть бажати знайти залежності між напругою та струмом. Також можна представити документи у вигляді векторів підрахунків, які описують вживання слів. Цей підхід часто називають "мішком слів" (Bag of Words).

Одна з проблем у роботі з векторами полягає в тому, що масштаби та одиниці різних координат можуть сильно відрізнятися. Наприклад, ми можемо вимірювати зріст у кілограмах, фунтах, грамах, тоннах, стоунах, і все це будуть множники зміни. Так само, коли ми представляємо температуру, ми маємо повний клас афінних перетворень, залежно від того, чи представляємо ми її у

градусах Цельсія, Кельвіна або Фаренгейта. Один із способів автоматичного вирішення цих проблем - нормалізація даних. Ми обговоримо методи, як це можна зробити автоматично.

4.9. Представлення зв’язків

Списки: у деяких випадках вектори, які ми отримуємо, можуть містити різну кількість ознак. Наприклад, лікар може необов'язково проводити повний комплекс діагностичних тестів, якщо пацієнт виглядає здоровим.

Множини можуть з'являтися в навчальних задачах, коли є велика кількість потенційних причин ефекту, які недостатньо визначені. Наприклад, легко отримати дані про токсичність грибів. Ці дані можна використовувати для висновку про токсичність нового гриба, враховуючи інформацію про його хімічні сполуки. Але гриби містять комбінацію сполук, одна або кілька з яких можуть бути токсичними. Тому потрібно робити висновки про властивості об'єкта з урахуванням набору ознак, склад та кількість яких можуть значно відрізнятися.

Матриці є зручним засобом представлення парних зв'язків. Наприклад, у програмах спільної фільтрації рядки матриці можуть представляти користувачів, а стовпці - продукти. У деяких випадках ми можемо мати інформацію лише про певну комбінацію (користувач, продукт), наприклад, оцінку користувача продукту. Схожа ситуація виникає, коли у нас є лише інформація про подібність між спостереженнями, що вимірюється напівемпіричним вимірюванням відстані. Деякі пошукові алгоритми в біоінформатиці, наприклад, варіанти BLAST, повертають лише оцінку подібності, яка не обов'язково задовольняє вимоги метрики.

Зображення можна розглядати як двовимірні масиви чисел, тобто матриці. Однак це представлення є досить загальним, оскільки вони

відображають просторову когерентність (лінії, форми), а природні зображення мають різну роздільну здатність. Тобто зменшення дискретизації зображення призводить до об'єкта, який має статистику, дуже схожу на вихідне зображення. Комп'ютерний зір і психооптика розробили ряд інструментів для опису цих явищ.

Відео додає зображенням часовий вимір. Знову ж таки, ми можемо представити його у вигляді тривимірного масиву. Проте ефективні алгоритми обробки відео також враховують часову когерентність послідовності зображень, що означає, що вони враховують залежності і взаємозв'язки між зображеннями в різні моменти часу.

Дерева та графи часто використовуються для опису зв'язків між колекціями об'єктів. Наприклад, соціальні мережі. У соціальних мережах, таких як Facebook або LinkedIn, користувачі та їх зв'язки можуть бути представлені у вигляді графа, де кожен користувач - це вузол, а зв'язки між користувачами - це ребра графа. Цей граф може мати спрямовані ребра, що вказують напрямок взаємозв'язку (наприклад, один користувач дружить з іншим, але не навпаки). Такий граф соціальних зв'язків може бути використаний для різних аналітичних завдань, таких як виявлення спільнот, аналіз впливу, рекомендації або прогнозування поведінки користувачів. Використання графових структур дозволяє враховувати взаємозв'язки між користувачами та здійснювати складні аналізи на основі цих зв'язків.

Обидва наведені вище приклади описують проблеми оцінки, де наші спостереження є вершинами дерева або графа. Однак самі графіки можуть бути спостереженнями. Наприклад, DOM-дерево веб-сторінки або граф викликів комп'ютерної програми можуть сформувати основу, за якою можна буде робити висновки відповідно поставленої задачі.

Рядки зустрічаються часто, головним чином у сфері біоінформатики та обробки природної мови. Вони можуть бути вхідними даними для наших

проблем оцінювання, наприклад, при класифікації електронного листа як спаму, при спробі знайти всі імена осіб і організацій у тексті або при моделюванні тематичної структури документа. Так само вони можуть становити вихід системи. Наприклад, ми можемо захотіти виконати узагальнення документа, автоматичний переклад або спробувати відповісти на запити природною мовою.

Складені конструкції є найбільш поширеним об'єктом, тобто в більшості ситуацій ми матимемо структуроване поєднання різних типів даних. Наприклад, веб-сторінка може містити зображення, текст, таблиці, які, у свою чергу, містять числа і списки, які можуть являти собою вузли на графіку веб-сторінок, пов'язаних між собою. Якісне статистичне моделювання враховує такі залежності та структури, щоб адаптувати достатньо гнучкі моделі.

5. MLP

5.1. Особливості

MLP (Multilayer Perceptron) - це тип штучної нейронної мережі, яка складається з одного або більше шарів нейронів, включаючи вхідний шар, приховані шари і вихідний шар. Між вхідним та вихідним шарами у мережі знаходиться набір прихованих шарів, які функціонують як обчислювальний механізм у MLP. Подібно до прямого потоку даних у MLP, інформація проходить у прямому напрямку від вхідного до вихідного рівня. Кожен шар містить набір нейронів, які пов'язані з нейронами з попереднього шару за допомогою вагових коефіцієнтів.

MLP є одним з найпоширеніших типів нейронних мереж і використовується для багатьох завдань машинного навчання, включаючи класифікацію, регресію, впізнавання образів та рекомендації.

У MLP кожен нейрон у шарі отримує ваговану суму вхідних сигналів, яку обробляє за допомогою нелінійної функції активації, такої як сигмоїда або ReLU (Rectified Linear Unit). Інформація просувається через мережу в напрямку від вхідного шару до вихідного шару за допомогою передачі сигналів через нейрони та вагових коефіцієнтів. Процес навчання полягає у встановленні оптимальних значень вагових коефіцієнтів шляхом зменшення помилки прогнозування за допомогою методів оптимізації, таких як градієнтний спуск.

MLP розроблені для наближення будь-якої безперервної функції та можуть вирішувати проблеми, які не є лінійно роздільними. Може мати різну кількість прихованих шарів та різну кількість нейронів у кожному шарі, що дозволяє моделі адаптуватись до складності задачі та виявляти складні залежності в даних.

5.2. Релізація MLP

5.2.1. Бібліотеки

Для реалізації MLP були використані такі бібліотеки:

tensorflow.keras.models - для визначення моделей нейронних мереж.

tensorflow.keras.layers - для визначення шарів нейронних мереж.

tensorflow.keras.preprocessing.text - для обробки текстових даних.

tensorflow.keras.preprocessing.sequence - для роботи з послідовностями.

tensorflow.keras.utils - для роботи з утилітами.

tensorflow.keras.callbacks - для виклику зворотних викликів під час навчання моделі.

tensorflow.keras.layers - для визначення шарів нейронних мереж.

re - модуль для роботи з регулярними виразами (використовується для очищення тексту).

nltk - бібліотека для обробки природньої мови.

nltk.corpus - модуль NLTK для доступу до корпусів текстів.

nltk.stem - для використання алгоритмів стемінгу та лематизації.

nltk.tokenize - для токенізації тексту.

gensim.models - модуль для роботи з моделями векторних представлень слів.

numpy - бібліотека для роботи з масивами і математичними операціями.

5.2.2. Попередня обробка навчального тексту

Для початку задаємо максимальну кількість слів та довжину тексту. Задання встановлення довжини слова *max_len* та максимальної кількості слів *max_words* має на меті обмежити розмір вхідних даних для моделі.

max_len визначає максимальну довжину тексту (в кількості слів), яку буде приймати модель. Якщо текст буде коротший, його буде доповнено до цієї довжини, а якщо довший - обріже до заданої довжини. Це дозволяє уніфікувати розмір текстових даних і забезпечити їхню сумісність при виконанні обчислень моделлю.

max_words визначає максимальну кількість унікальних слів, які будуть використовуватись при побудові словника. Це дозволяє зменшити розмір словника і скоротити обчислювальну складність моделі.

Далі генеруємо тестові дані *texts* та встановлюємо мітки класів *labels* для текстових даних. Попередньо обробляємо тексти за допомогою бібліотеки NLTK (Natural Language Toolkit):

```
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
texts = [remove_stop_words(text) for text in texts]

nltk.download('wordnet')
nltk.download('punkt')
texts = [lemmatize_text(text) for text in texts]
```

Зображення 5.2.2.1. Код MLP: попередня обробка тексту

nltk.download('stopwords'): Цей виклик завантажує набір стоп-слів для англійської мови з NLTK. Стоп-слова - це слова, які часто зустрічаються в тексті, але мають незначуще значення і можуть бути виключені з аналізу. Після завантаження стоп-слів, вони будуть використовуватись для фільтрації текстів.

stop_words = set(stopwords.words('english')): Цей рядок визначає змінну *stop_words*, яка міститиме набір стоп-слів для англійської мови. Функція

stopwords.words('english') повертає список стоп-слів для англійської мови, який потім перетворюється на множину *set* для швидшого доступу.

texts = [remove_stop_words(text) for text in texts]: Цей рядок застосовує функцію *remove_stop_words* до кожного тексту у списку texts. Функція *remove_stop_words* видаляє стоп-слова з тексту, приводить його до нижнього регістру, видаляє спеціальні символи та розділові знаки. Після обробки всіх текстів, вони зберігаються знову в змінну texts.

nltk.download('wordnet'): Цей виклик завантажує ресурс WordNet для використання в лематизації текстів. WordNet - це лексична база даних, яка містить семантичні та лінгвістичні інформації про англійські слова.

nltk.download('punkt'): Цей виклик завантажує ресурс Punkt для використання в токенізації текстів. Punkt - це модуль NLTK для токенізації речень та слів в англійських текстах.

texts = [lemmatize_text(text) for text in texts]: Цей рядок застосовує функцію *lemmatize_text* до кожного тексту у списку *texts*. Функція *lemmatize_text* використовує лематизацію для перетворення слів у їхні базові форми.

5.2.3. Tokenizer для обробки текстових даних

```
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
print("sequences: ", sequences)

data = pad_sequences(sequences, maxlen=max_len)
print("data: ", data)

labels = to_categorical(labels)
print("Labels: ", labels)
```

Зображення 5.2.3.1. Код MLP: Tokenizer

У цьому коді використовується Tokenizer для обробки текстових даних і підготовки їх для використання в моделі машинного навчання. Основні кроки:

tokenizer = Tokenizer(num_words=max_words): Створюється об'єкт Tokenizer з параметром *num_words*, який вказує максимальну кількість слів, які будуть використовуватися для побудови словника.

tokenizer.fit_on_texts(texts): Передаються текстові дані texts для побудови словника Tokenizer. Кожне слово в текстах отримує унікальний індекс у словнику.

sequences = tokenizer.texts_to_sequences(texts): Текстові дані texts перетворюються на послідовності *sequences*, де кожне слово замінено відповідним індексом з побудованого словника.

data = pad_sequences(sequences, maxlen=max_len): Послідовності доповнюються до максимальної довжини max_len за допомогою функції *pad_sequences*. Це необхідно для забезпечення однакової довжини всіх текстових послідовностей, які будуть використовуватися в моделі.

Далі перетворюємо мітки *labels* у вектори категорій (one-hot вектори) за допомогою функції *to_categorical* з бібліотеки Keras. У вихідних даних, *labels* є масивом цілих чисел, які представляють категорії або класи. Функція *to_categorical* перетворює ці цілі числа у вектори категорій з довжиною, що відповідає кількості унікальних класів. Кожен вектор має значення 0, за винятком однієї позиції, яка відповідає класу і має значення 1.

5.2.4. Архітектура моделі

Визначаємо архітектуру нейронної мережі за допомогою класу Sequential з бібліотеки Keras та встановлюємо параметри компіляції моделі:

```
model = Sequential()
model.add(Embedding(max_words, 32, input_length=max_len))
model.add(Conv1D(64, 5, activation='relu'))
model.add(MaxPooling1D(pool_size=4))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))
print("model: ", model)

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Зображення 5.2.4.1. Код MLP: архітектура

Sequential є одним з варіантів моделей в Keras і дозволяє створювати послідовні стеки шарів. У даному випадку модель має наступну структуру:

Embedding шар: Використовується для векторного представлення слів (embeddings). Вхідний розмір словника (кількість унікальних слів) задається параметром max_words, а вихідна довжина векторів у вкладенні - 32. Параметр input_length вказує на максимальну довжину вхідних послідовностей.

Conv1D шар: 1D згортковий шар з 64 фільтрами і ядром розміром 5. Активаційна функція relu застосовується після згортки.

MaxPooling1D шар: Застосовується пулінг за максимумом з розміром пулінгу 4.

Flatten шар: Використовується для розгортання (приведення до однієї вимірності) вихідних даних перед передачею до наступного шару.

Dropout шар: Використовується для регуляризації моделі, де деякі нейрони випадковим чином вимикаються з ймовірністю 0.5.

Dense шар: Повнозв'язаний шар з 3 нейронами (вихідними класами) і активаційною функцією softmax, що генерує вихідний розподіл ймовірностей для класів.

Отже, ця модель має загальну структуру "Embedding - Conv1D - MaxPooling1D - Flatten - Dropout - Dense" і призначена для класифікації на 3 класи.

Параметри компіляції моделі перед її навчанням:

loss='categorical_crossentropy': Вказує функцію втрати, яка використовується для оцінки різниці між прогнозованими значеннями моделі та справжніми мітками даних. У цьому випадку використовується категоріальна перехресна ентропія, оскільки дані представлені у формі one-hot векторів.

optimizer='adam': Вказує оптимізаційний алгоритм, який використовується для навчання моделі. В даному випадку використовується алгоритм оптимізації Adam, який ефективно пристосовується до змінюваних градієнтів.

metrics=['accuracy']: Вказує метрики, які використовуються для оцінки продуктивності моделі під час навчання. У цьому випадку використовується метрика точності (accuracy), яка вимірює відсоток правильно класифікованих зразків.

Компіляція моделі підготовлює її до навчання шляхом встановлення відповідних параметрів для оптимізації і втрати.

5.2.5. Прогнозування

```
tensorboard_callback = TensorBoard(log_dir='./logs')

model.fit(data, labels, epochs=20, batch_size=32, callbacks=[tensorboard_callback])

new_texts = ['As the evening descends, the sky turns into a canvas
             'of pink and orange hues, created by the setting sun.']
new_texts = [remove_stop_words(text) for text in new_texts]
new_data = pad_sequences(new_sequences, maxlen=max_len)
pred = model.predict(new_data)
print('Predictions:', pred)
```

Зображення 5.2.4.1. Код MLP: прогнозування

У цьому коді використовується TensorBoard для візуалізації метрик та статистики під час тренування моделі. Створюється об'єкт TensorBoard, де log_dir вказує шлях до каталогу, де будуть зберігатися дані журналу TensorBoard.

model.fit(data, labels, epochs=20, batch_size=32, callbacks=[tensorboard_callback]): Модель навчається на даних data та labels. За допомогою параметра callbacks передається список зворотних викликів (callbacks), включаючи об'єкт TensorBoard, який додається для відстеження прогресу під час навчання. Визначається новий текст для передбачення.

new_sequences = tokenizer.texts_to_sequences(new_texts): Тексти перетворюються на послідовності *sequences* за допомогою токенізатора *tokenizer*.

new_data = pad_sequences(new_sequences, maxlen=max_len): Послідовності доповнюються до максимальної довжини *max_len* за допомогою функції *pad_sequences*.

pred = model.predict(new_data): Виконується передбачення за допомогою навченої моделі на нових даних *new_data*.

5.3. Демонстрація роботи MLP

Для перевірки роботи MLP обрано три семантично різні текстові документи. На вхід подамо текст, за семантикою близький до третього документа з навчальних текстів.

```
1/1 [==============================] - 0s 50ms/step - loss: 0.9776 - accuracy: 1.0000
Epoch 10/20
1/1 [==============================] - 0s 58ms/step - loss: 1.0409 - accuracy: 0.6667
Epoch 11/20
1/1 [==============================] - 0s 20ms/step - loss: 0.8604 - accuracy: 1.0000
Epoch 12/20
1/1 [==============================] - 0s 53ms/step - loss: 0.9883 - accuracy: 0.6667
Epoch 13/20
1/1 [==============================] - 0s 22ms/step - loss: 0.9287 - accuracy: 0.6667
Epoch 14/20
1/1 [==============================] - 0s 24ms/step - loss: 0.8026 - accuracy: 1.0000
Epoch 15/20
1/1 [==============================] - 0s 24ms/step - loss: 1.0352 - accuracy: 0.3333
Epoch 16/20
1/1 [==============================] - 0s 23ms/step - loss: 0.8401 - accuracy: 0.6667
Epoch 17/20
1/1 [==============================] - 0s 31ms/step - loss: 0.9049 - accuracy: 0.6667
Epoch 18/20
1/1 [==============================] - 0s 24ms/step - loss: 0.8544 - accuracy: 1.0000
Epoch 19/20
1/1 [==============================] - 0s 21ms/step - loss: 0.8465 - accuracy: 1.0000
Epoch 20/20
1/1 [==============================] - 0s 26ms/step - loss: 0.6682 - accuracy: 1.0000
1/1 [==============================] - 0s 153ms/step
Predictions: [[0.32124928 0.3010152  0.3777355 ]]
```

Зображення 5.3.1. Демонстрація роботи MLP

5.4. Висновки за главою 5

На основі результатів належності тексту до трьох класів можна зробити наступні висновки:

Класифікація тексту: MLP використовується для класифікації тексту на певні категорії або класи. У даному випадку ми маємо тривалість для трьох різних класів. Результати показують, що текст має найвищу належність до третього класу (0.3777355), що може вказувати на те, що текст може бути пов'язаним з цим класом.

Розподіл ймовірностей: MLP дає розподіл ймовірностей для кожного класу. За результатами, найвища імовірність спостерігається для третього класу (0.3777355), далі йде перший клас (0.32124928), а другий клас має найменшу імовірність (0.3010152).

Неоднозначність класифікації: Враховуючи результати, можна зробити висновок, що класифікація тексту може бути неоднозначною. Існує певна перекритість між другим і третім класами, де обидва мають значення імовірності, близькі до 0.3. Це може вказувати на те, що текст може мати певні риси обох класів або неоднозначний зміст.

Загалом, MLP використовується для класифікації тексту і здатний надати розподіл ймовірностей для кожного класу. Проте, важливо враховувати можливу неоднозначність та перекриття між класами при аналізі результатів.

6. BERT

6.1. Удосконалений підхід

Існує ствердження, що поточні методи стандартних мовних моделей обмежують потужність попередньо навчених уявлень, особливо для підходів тонкого налаштування. Основним обмеженням є те, що стандартні мовні моделі є односпрямованими, і це обмежує вибір архітектур, які можна використовувати під час попереднього навчання. Наприклад, у моделі OpenAI GPT автори використовують архітектуру left-to-right, де кожен маркер може звертатися лише до попередніх маркерів у рівнях самоконтролю Transformer. Такі обмеження можуть негативно позначитися на результативності при застосуванні попередньо навчених моделей до завдань, де важливий контекст з обох напрямків, наприклад, для відповідей на питання.

BERT пропонує удосконалення підходу на основі тонкого налаштування: представлення двонаправленого кодувальника від Transformers. BERT пом'якшує згадане раніше обмеження односпрямованості, використовуючи переднавчальну мету "masked language model" (MLM). Модель замаскованої мови випадковим чином маскує деякі з вхідних даних, а мета полягає в тому, щоб передбачити оригінальний ідентифікатор словника замаскованого слова на основі лише його контексту. На відміну від попереднього навчання мовної моделі left-to-right, MLM дозволяє представленню злити лівий і правий контекст, що дозволяє нам попередньо навчити глибокий двонаправлений трансформатор.

На додаток до замаскованої мовної моделі ми також використовуємо завдання "передбачення наступного речення", яке спільно попередньо тренує представлення текстових пар.

6.2. Архітектура BERT

В структурі виконуються два кроки: попереднє навчання (Pretraining) та тонке налаштування (Fine-tuning). Під час попереднього навчання модель навчається на немаркованих даних у різних завданнях попереднього навчання. Для точного налаштування моделі BERT спочатку ініціалізується попередньо навченими параметрами, а параметри піддаються налаштуванню з використанням позначених даних із завдань, що надходять за потоком. Кожне подальше завдання має окремі точно налаштовані моделі, навіть якщо вони ініціалізуються з тими самими попередньо навченими параметрами.

BERT відрізняється своєю універсальною архітектурою, яка може використовуватися для різних завдань. Між попередньо навченою архітектурою та фінальною нижчою архітектурою існує мінімальна різниця.

Архітектура моделі BERT - це багатошаровий двонаправлений кодер Transformer.

L – кількість шарів (тобто трансформерних блоків)

H – прихований розмір

A – кількість attentionheads

BERTBASE: L=12, H=768, A=12, загальні параметри=110M.

BERTLARGE: L=24, H=1024, A=16, загальні параметри=340M.

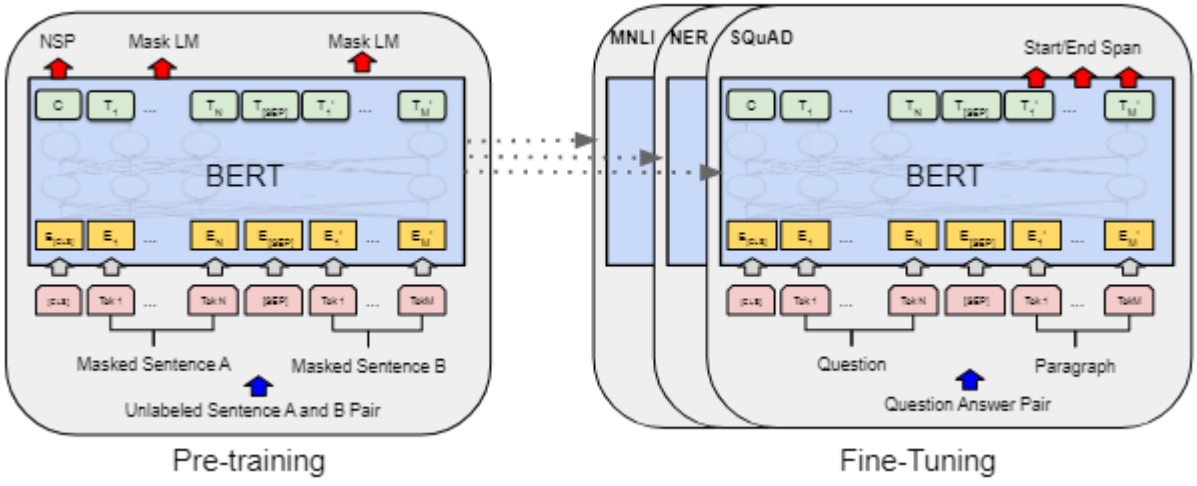
Для порівняння розмір моделі BERTBASE такий самий, як в OpenAI GPT. Важливо, однак, що BERT Transformer використовує двонаправлену самоувагу, тоді як GPT Transformer використовує обмежену самоувагу, де кожен маркер може звертати увагу лише на контекст ліворуч від нього.

У більшості конкурентоспроможних моделей для трансдукції нейронних послідовностей використовується структура, що складається з кодера і декодера.

6.3. Представлення вводу та виводу

Наше представлення вводу в BERT може гнучко обробляти різноманітні завдання шляхом однозначного включення як одного речення, так і пари речень в одну послідовність токенів. У цій роботі термін "речення" використовується для будь-якого суміжного текстового проміжку, не обов'язково для речення лінгвістичного. Послідовність в BERT відноситься до вхідної послідовності маркерів, яка може складатися з одного або двох речень, об'єднаних разом. Кожна послідовність починається з маркера спеціальної класифікації [CLS]. Остаточний прихований стан, пов'язаний з цим маркером, використовується як узагальнююче представлення послідовності для завдань класифікації. Пари речень також об'єднуються в одну послідовність для подальшої обробки.

Речення розрізнюються двома способами. Перший етап: вони розділяються за допомогою спеціального маркера [SEP]. Другий етап: додавання вивчених вбудовувань до кожного токена, щоб відображати його приналежність до речення A або речення B.



Зображення 6.3.1. Код BERT: Pre-training, Fine-tuning

Вхідне вбудовування позначається як E, остаточний прихований вектор спеціального токена [CLS] позначається як C ∈ RH, а кінцевий прихований вектор для i-го вхідного маркера позначається як Ti ∈ RH. Для кожного токена його вхідне представлення будується шляхом підсумовування відповідних ембедінгів лексем, сегментів і позицій.

6.4. Попереднє навчання

6.4.1. Завдання 1: Masked Model Language

Інтуїтивно зрозуміло, що глибока двонаправлена модель є потужнішою, ніж лише модель “left-to-right” або конкатенація неглибоких моделей “left-to-right” та “right-to-left”. На жаль, стандартні умовні мовні моделі (standard conditional language models) можна навчити лише зліва направо або справа наліво, в той час як двонаправлене кондиціонування дозволяє кожному слову опосередковано "бачити себе", і модель може легко передбачити цільове слово в багатошаровому контексті.

Для досягнення глибокого двонаправленого представлення ми випадковим чином маскуємо певну кількість вхідних токенів, а потім передбачаємо значення цих прихованих токенів. Ми називаємо цю процедуру "masked LM" (MLM), хоча в літературі часто використовується термін "завдання Клоза". У цьому випадку остаточні приховані вектори, що відповідають маскованим маркерам, піддаються softmax-функції відносно словника, як у стандартній LM. На відміну від автокодувальників з усуненням шуму, прогнозуються лише масковані слова, а не відновлюється весь вхідний сигнал.

6.4.2. Завдання 2: Прогнозування наступного речення (NSP)

Багато важливих завдань, таких як питання та відповідь (QA) і виведення природної мови (NLI), базуються на розумінні взаємозв'язку між двома реченнями, який не захоплюється безпосередньо моделюванням мови. Щоб навчити модель розуміти відношення між реченнями, ми переднавчаємо її на завданні бінаризованого прогнозування наступного речення, яке може бути легко згенероване з будь-якого монолінгвального корпусу.

Модель отримує речення і метою задачі є визначення того, чи є ці речення безпосередньо послідовними або були випадковим чином взяті з корпусу тексту.

6.5. Pre-training data

Аналіз або обробка корпусу текстів проводиться з урахуванням цілого документа як єдиного блоку інформації. Замість того, щоб розділяти текст на окремі речення та обробляти їх незалежно, увага приділяється структурі та контексту всього документа.

Такий підхід – з використанням єдиного блоку інформації - відрізняється від підходу з використанням корпусу перетасованого на рівні речень, де речення з різних документів можуть бути переставлені або перемішані, втрачаючи таким чином оригінальний контекст кожного документа, і є корисним в ситуаціях, коли речення в документі мають тісний взаємозв'язок або залежність між собою, і цей контекст може бути важливим для коректного розуміння тексту або досягнення певної мети аналізу.

Використання корпусу на рівні документа дозволяє зберігати оригінальну структуру та послідовність речень в документі, зберігаючи контекстуальну інформацію та відносини між ними.

6.6. Fine-tuning BERT

У кожного навченого BERT наявні його ваги, які були отримані в результаті навчання на обширному корпусі текстів таких як wiki-статті, які були написані науковою або літературною, структурованою мовою, в той час як задача може полягати у класифікації текстів написаних з використанням молодіжного сленгу, з помилками і т.д. (наприклад, якщо задача полягає в тому, щоб з’ясувати, чи є коментар жалобою). Тому щоб досягти кращої обробки семантики вихідних даних, можна навчити BERT на власних текстових даних, що призведе до зміни ваг моделі BERT.

Проте слід зазначити недоліки такого донавчання BERT: воно вимагає значних часових та обчислювальних витрат.

6.7. Реалізація BERT

6.7.1. Бібліотеки

Під час реалізації було використано бібліотеки для завантаження, навчання та використання моделі BERT для класифікації тексту, а також для створення датасету, керування тренуванням та запису логів.

torch: Це бібліотека для обчислення наукових обчислювань з використанням графів потоку даних. Вона надає функціональність для створення та навчання нейронних мереж, оптимізації та виконання обчислень на графічних процесорах.

transformers: Це бібліотека, розроблена Hugging Face, яка надає інтерфейс до популярних моделей глибокого навчання для обробки мови, зокрема моделей BERT. Вона містить реалізації різних архітектур моделей та набори даних, а також інструменти для попередньої навчання, налаштування та використання цих моделей.

BertTokenizer: Це клас з бібліотеки transformers, який використовується для токенізації тексту, тобто розбиття тексту на окремі токени. У випадку BERT він також виконує спеціальну обробку для вставки маркерів і роздільників, необхідних для моделі BERT.

BertForSequenceClassification: Це клас з бібліотеки transformers, який представляє модель BERT, налаштовану для задачі класифікації послідовностей. Він має попередньо навчені вбудовування слів та глибоку архітектуру для виконання класифікації.

TensorDataset: Це клас з бібліотеки torch, який дозволяє створювати датасети з вхідними даними та відповідними мітками. В цьому випадку використовується для створення датасету з вхідними тензорами тексту та мітками класів.

DataLoader: Це клас з бібліотеки torch, який дозволяє зручно завантажувати дані пакетами для тренування моделі. Він автоматично розподіляє дані на пакети, перемішує їх (у випадку параметра shuffle=True) та надає зручний ітератор для доступу до даних під час тренування.

tqdm: Це бібліотека, яка надає прогрес-бари для циклів обробки даних. Вона дозволяє візуалізувати прогрес виконання циклу та відстежувати, скільки часу залишилося до завершення циклу.

SummaryWriter: Це клас з бібліотеки torch.utils.tensorboard, який дозволяє записувати логи та візуалізації для TensorBoard. В даному випадку використовується для записування значень втрат під час навчання моделі.

datetime: Це модуль вбудованої бібліотеки Python, який надає функціональність для роботи з датою та часом. В цьому конкретному випадку використовується для отримання поточного часу для створення унікального ідентифікатора для TensorBoard logs.

Основні кроки, які виконуються для навчання та використання моделі BERT для класифікації тексту, можна розбити на декілька етапів.

6.7.2. Завантаження попередньо навченої моделі BERT

Ми завантажуємо попередньо навчену модель BERT та токенізатор, який використовується для перетворення тексту на послідовності токенів.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3)
```

Зображення 6.7.2.1. Код BERT: завантаження попередньо навченої моделі

Імпортується клас *BertTokenizer* з бібліотеки transformers. BertTokenizer використовується для токенізації тексту з використанням попередньо навченої моделі BERT.

Імпортується клас *BertForSequenceClassification* з бібліотеки transformers. BertForSequenceClassification є реалізацією моделі BERT, налаштованою для класифікації послідовностей. У даному випадку модель налаштовується для класифікації з трьома можливими класами.

Ініціалізується об'єкт *tokenizer* - попередньо навчена модель BERT з базовою конфігурацією, яка навчалась на нечутливих до регістру даних.

Ініціалізується об'єкт *model* - попередньо навчена модель BERT для класифікації послідовностей з базовою конфігурацією та трьома класами.

6.7.3. Підготовка даних

Тексти для тренування представлені у вигляді списку, а їх відповідні мітки класів зберігаються у вигляді чисел. Дані підготовлюються шляхом токенізації текстів, додавання падінгу та створення тензорів для вхідних даних та міток. Після цього створюється *DataLoader* для зручного завантаження даних пакетами.

6.7.4. Навчання

В цьому кроці ми визначаємо оптимізатор, обираємо кількість епох для навчання та створюємо об'єкт SummaryWriter для запису логів у TensorBoard. Потім ми виконуємо цикл навчання, де кожен пакет даних проходить через модель BERT. Обчислюється втрата, здійснюється зворотне поширення помилки та оновлення ваг моделі. Прогрес навчання відображається у прогрес-барі, а значення втрати записуються в TensorBoard.

Ініціалізується оптимізатор Adam - використовується для оновлення ваг моделі під час навчання.

Викликається метод *model.train()* для переведення моделі у режим навчання. Це активує деякі шари моделі, які впливають на процес навчання, наприклад, регуляризацію та випадкове вимкнення *dropout*.

Встановлюється кількість епох *num_epochs*, яка визначає, скільки разів будуть пройдені всі навчальні дані під час навчання.

Запускається зовнішній цикл по епохах навчання моделі.

Внутрішній цикл виконується для кожного кроку *batch* навчального набору даних. Для кожного кроку отримуються вхідні тензори *input_ids*, *attention_mask* та *labels*.

Виконується передача вхідних тензорів до моделі, під час якої модель обчислює прогнози та втрату *loss* для даного пакету даних.

Обчислюється загальна втрата для епохи шляхом додавання *loss.item()* до *epoch_loss*, і виконується зворотне поширення помилки *backpropagation* за допомогою *loss.backward()*.

Оновлюються ваги моделі за допомогою *optimizer.step()*. Обнуляються градієнти з оптимізатора. Оновлюється прогресна панель *progress bar* з виведенням поточної інформації про епоху, крок та значення втрати.

6.7.5. Отримання передбачень

Після навчання ми можемо використовувати навчену модель для отримання передбачень для нових текстів. У цьому коді ми використовуємо один текст для демонстрації. Текст токенізується та перетворюється на вхідні тензори. Потім з використанням навченої моделі отримуємо передбачені класи та ймовірності для кожного класу. Результати ви водяться на екран, включаючи передбачені класи та відповідні ймовірності.

6.8. Демонстрація роботи BERT

Тестування роботи відбуватиметься на тих самих прикладах, що і під час реалізації MLP з метою порівняння їх результатів.

Зображення 6.8.2. Демонстрація роботи BERT

6.9. Висновки за главою 6

Порівнюючи результати BERT і MLP, помітно, що BERT показує вищі значення імовірностей для третього класу, що вказує на його вищу впевненість у приналежності тексту до цього класу. У MLP було спостережено найвищу імовірність для третього класу, але значення було значно меншим (0.3777 проти 0.5950). Це свідчить про більшу точність і ефективність BERT у класифікації тексту.

Загалом, BERT показує кращі результати в порівнянні з MLP, забезпечуючи більшу точність та впевненість у класифікації тексту. Його здатність аналізувати контекст та використовувати широкий контекстуальний

контекст допомагають покращити якість класифікації і отримати більш точні результати.

ВИСНОВКИ

Порівнюючи методи косинусної міри подібності, подібності Жаккара, TF-IDF, MLP та BERT, можна зробити такі висновки:

Косинусна міра подібності: Цей метод вимірює кут між векторами для оцінки подібності текстових документів. Він працює добре для векторних подібностей, але не враховує семантичні зв'язки між словами.

Подібність Жаккара: Цей метод вимірює подібність між двома множинами шляхом порівняння їхніх елементів. Він показує, які елементи належать обом множинам, але не враховує контексту і семантики тексту.

TF-IDF: Цей метод використовується для оцінки важливості термінів у документі на підставі їхньої частоти та інформаційної ваги. Він добре працює для виявлення ключових слів, але не здатний розуміти синтаксичні та семантичні зв'язки.

MLP (Multilayer Perceptron): Це класична модель нейронної мережі, яка має приховані шари для обробки вхідних даних. Вона здатна виявляти складні зв'язки між даними, але потребує великої кількості навчальних даних та налагодження гіперпараметрів.

BERT (Bidirectional Encoder Representations from Transformers): Цей метод заснований на трансформерних мережах і має універсальну архітектуру для різних завдань обробки природної мови. Він здатний враховувати контекст, синтаксичні та семантичні зв'язки між словами, що робить його потужним і ефективним для багатьох завдань.

Отже, метод BERT видається найбільш просунутим, оскільки він враховує контекстуальну інформацію та семантику тексту, що дозволяє досягати кращих результатів у завданнях обробки природної мови порівняно з іншими методами.

Список використаних джерел

1. Bianconi G. Multilayer network models. Oxford University Press, 2018.
2. Burkov A. The hundred-page machine learning book. Andriy Burkov, 2019. 160 p.
3. Eid D. Applying coding concepts: encoder workbook. CENGAGE Delmar Learning, 2007. 360 p.
4. Goldstone R. L., Son J. Y. Similarity. Oxford University Press, 2012.
5. Harrington P. Machine learning in action. Shelter Island, N.Y : Manning Publications Co., 2012. 354 p.
6. Kanevski M. Machine learning algorithms for environmental spatial data: theory and case studies. Chapman & Hall/CRC, 2007. 224 p.
7. Kelly B. Similarity. Alsager : Crewe & Alsager College of Higher Education, Flexible Learning Centre, 1989.
8. Semi-Supervised learning (adaptive computation and machine learning). The MIT Press, 2006. 498 p.