tendency to algorithmization in solving certain practical problems, it is advisable to use schematization when studying mathematical analysis by the above students – and during mastering the theoretical material, and in solving problems. Schematization as a sign-symbolic activity in the teaching of mathematical analysis to students-programmers occupies a significant place. During the acquaintance of students with the new material and during the application of knowledge, several types of learning situations are identified in which they encounter the activities of schematization.

Conclusion. *We see further research in the development of a competency-based dark system of tasks in mathematical analysis, taking into account the specifics of training future professionals in data analysis.*

Keywords: *mathematical analysis, data mining, schematization, schemes, students-programmers.*

Одержано редакцією 03.12.2019 р.
Прийнято до публікації 24.02.2020 р.

УДК 004.85:519.6

DOI 10.31651/2076-5886-2020-1-86-100

PACS 02.70.Wz, 07.05.Kf, 07.05.Mh, 07.05.Tp

КОВАЛЕНКО Олена Сергіївна,
магістрантка спеціальності «Прикладна математика» Черкаського національного університету імені Богдана Хмельницького

СЕРДЮК Олександр Анатолійович,
кандидат економічних наук, старший викладач кафедри прикладної математики та інформатики Черкаського національного університету імені Богдана Хмельницького
e-mail: serdyuk@ukr.net
ORCID 0000-0002-3919-4661

ВИКОРИСТАННЯ КЛАСИЧНИХ МЕТОДІВ МАШИННОГО НАВЧАННЯ ДЛЯ КЛАСИФІКАЦІЇ ТЕКСТІВ У ПРОГРАМАХ ГЕНЕРАЦІЇ АВТОМАТИЧНИХ ВІДПОВІДЕЙ

У статті подано опис методів машинного навчання для розв'язання задач класифікації текстів з метою їх подальшого використання у програмах автоматичної генерації відповідей на основі аналізу контексту питань користувача. Розглянуто кроки підготовки та проведення класифікації текстів разом з фрагментами програмного коду мовою Python з використанням бібліотеки обробки природної мови. Проведено попередній огляд автоматичних генераторів відповідей та вибрано метод автоматичної генерації відповідей. Розроблено структуру програми автоматичної генерації відповідей та визначено напрямки подальшої роботи.

Ключові слова: *машинне навчання, класифікація текстів, чат-боти, програмування, Python.*

Вступ

Методи машинного навчання наразі широко використовуються для задач обробки природної, чи людської, мови (Natural Language Processing) з метою її розпізнавання, розуміння, інтерпретації та генерації комп'ютерними алгоритмами. У сучасному світі інформаційних технологій постійно збільшується спектр задач, що можуть використовувати результати інтерпретації людської мови, зокрема це автоматична класифікація текстів для класифікаторів електронних бібліотек, рубрикація новин,

розпізнавання питань користувача у автоматичних будинках та генерація відповідей тощо. Широке розповсюдження месенджерів для передачі повідомлень між користувачами разом з постійним збільшенням кола користувачів відкрило ще один напрямок застосування задач класифікації текстів: розробка автоматичних помічників для генерації повідомлень. Для розв'язання такої задачі необхідно не лише вміти обробляти природну мову, а й генерувати адекватні відповіді, чи, загалом – речення для діалогу. Саме на огляд методів класифікації текстів для їх наступного використання при генерації автоматичних відповідей спрямована стаття.

Метою статті є дослідження методі та алгоритмів, а також демонстрація програмних фрагментів роботи класичних методів машинного навчання для класифікації текстів і застосування результатів класифікації у програмах автоматичної генерації відповідей.

Подальший виклад матеріалу має наступну структуру:

1. Огляд умов відбору даних.
2. Огляд методів попередньої обробки тексту.
3. Огляд методів векторизації тексту.
4. Огляд деяких алгоритмів класифікації тексту.
5. Ознайомлення з поняттям автоматичної генерації відповідей.
6. Розробка структури програми автоматичної генерації відповідей на основі попередньої класифікації текстів.

Виклад основного матеріалу

Відбір даних та вимоги до них

Набір даних, його обсяг та якість у машинному навчанні мають найбільше значення, адже саме від цього залежить результат тренування будь-якої моделі. Потрібний обсяг зразків даних залежить, перш за все, від типу задачі, яку розв'язує дослідник: так, вважається, що прості, класичні моделі машинного навчання, наприклад, лінійної регресії, не потребують так багато даних, як штучні нейронні мережі. Однак, цього не можна сказати про якість даних, адже те, наскільки добре вони структуровані й розмічені, буде вирішальним фактором у роботі моделі незалежно від її складності. Навіть звичайна лінійна регресія може дати дуже гарні результати, що нам доводить Google зі своїми результатами у тренуванні "розумних моделей" [15].

Нами для роботи було використано стандартний набір даних під назвою 20 newsgroups text dataset з бібліотеки Scikit-learn, що включає 18000 різних текстів, розподілених за 20 темами [14].

Вибір середовища розробки та його налаштування

Для роботи з попередньою обробкою, векторизацією та класифікацією текстів було використано пакет бібліотек і програм для символьної та статичної обробки природної мови – NLTK, доступної для роботи на операційній системи Windows [10].

У якості середовища розробки було обрано інтерактивне середовище Jupyter Notebook, що працює у браузері. Цей інструмент зручний у використанні для роботи з даними, статистичного моделювання та машинного навчання [12].

Для того, щоб запустити Jupyter Notebook у операційній системі Windows, спочатку було встановлено дистрибутив з відкритим кодом Anaconda (рис. 1), що містить у собі набір вільних бібліотек та найбільш популярних модулів для мови Python, зокрема, NumPy, SciPy, Astropy та ін. [2].

Попередня обробка текстів

Роботі з кластеризацією чи класифікацією текстів зазвичай не обходиться без їх попередньої обробки, адже для адекватного сприйняття даних моделлю потрібно спочатку очистити їх від небажаних символів та слів, тегів, повернути словам їх початкову форму, тобто лематизувати, видалити небажані закінчення, перетворити всі літери до нижнього регістру, прибрати цифри, пробіли, ASCII-символи тощо.

На поточний момент існує багато бібліотек для природньої обробки мови (Natural Language Processing), які вже мають у собі весь вищеперерахований набір функцій, наприклад SpaCy, gensim, NLTK. У якості альтернативного варіанту обробку можна виконати самостійно, звертаючись до певних функцій, класів, методів бібліотеки Scikit-learn чи інших популярних бібліотек для мови Python.

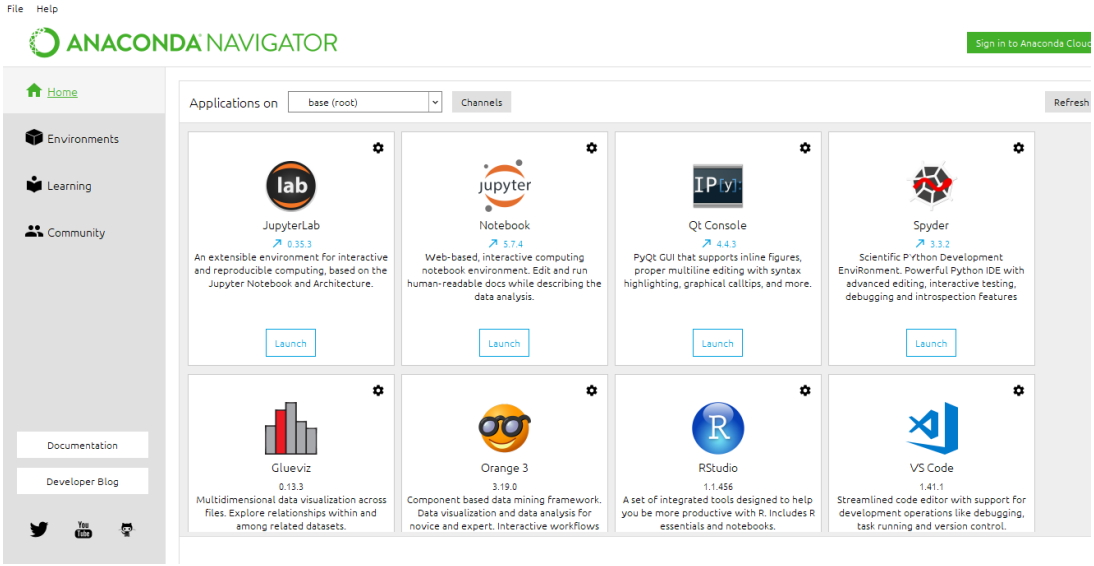

Рис. 1. Початкова сторінка інструментального середовища Anaconda

Підготовка корпусу

Корпус тексту – це великі структуровані набори текстів, подані у електронному вигляді. Корпус тексту може бути одномовним або багатомовним. Існує кілька найрозповсюдженіших видів структури корпусу текстів: найпростіша з них – це ізольовані тексти, що не мають якоїсь визначеної організації, а представляють собою лише набір текстів. Наступним видом корпусу текстів є тексти, зібрані по категоріях за жанрами, джерелом, авторами, мовою і т. ін. Ще одним видом структури текстів є тексти, де присвоєні їм категорії можуть перемежовуватись: наприклад, у випадку з визначенням теми тексту один і той же текст може належати до кількох тем. Іноді структура текстів може змінюватись з часом; зразком такого виду текстів може бути набір новин. Останній вид текстів належить до категорії за вступною адресою (з англ. Inaugural Address). Вище перераховані види текстів проілюстровано на рис. 2.

Для роботи з текстами не обо'язково мати власний корпус: його можна завантажити з бібліотеки NLTK, яка налічує більше, аніж 25000 електронних книг, що знаходяться у вільному доступі з ресурсу Gutenberg [13]. Щоб завантажити їх, потрібно спочатку імпортувати пакет NLTK, а потім скористатися однією з функцій, що дає доступ до корпусу текстів, наприклад `nltk.download('gutenberg')`:

```
import nltk
nltk.download('gutenberg')
```

У разі, якщо необхідно скористатись власним набором текстів, можна зробити це за допомогою `PlaintextCorpusReader` з бібліотеки NLTK, присвоюючи змінній

`corpus_root` адресу розташування у мережі відповідного набору текстів:

```
from nltk.corpus import PlaintextCorpusReader
corpus_root = '/usr/share/dict'
```

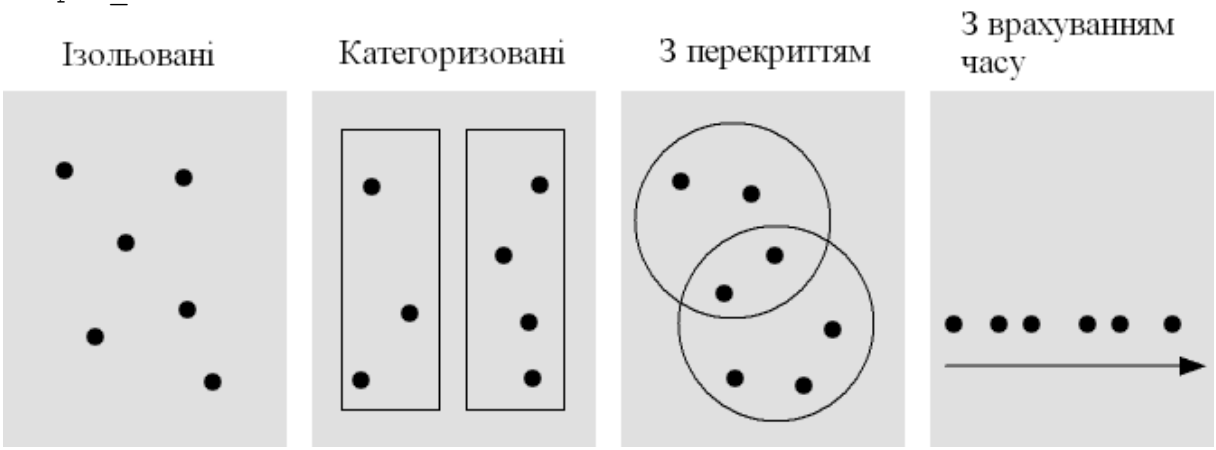

Рис. 2. Види структури корпусу текстів

Видалення пунктуації та тегів

Після завантаження набору текстів проводиться їх попередня обробка з метою підготовки їх до векторизації. Обробка починається з видалення пунктуації, тегів, ASCII-символів:

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
tokenizer.tokenize()
```

Також на даному етапі усі слова у наборі тексту перетворюються до нижнього регістру, що можна зробити за допомогою наступного виразу:

```
sorted([w for w in set(sent7) if not w.islower()])
```

Видалення стоп-слів

Видалення стоп-слів є досить важливим етапом очищення тексту перед векторизацією, адже більшість займенників, прийменників та артиклів у англійській мові на кшталт “it, his, for, the” не несуть вагомого значення для класифікації текстів за темами, натомість лише заплутуючи алгоритм. Видалити ці слова можна за допомогою наступних рядків:

```
nltk.download(stopwords)
from nltk.tokenize import word_tokenize
example_sent = str([twenty_train])
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
filtered_sentence = []
for w in word_tokens:
   if w not in stop_words:
      filtered_sentence.append(w)
```

Лематизація

Лематизація означає перетворення слова в його словникову форму, тобто – лему. Цей етап застосовується для того, щоб спростити роботу алгоритму й відкинути видозмінені форми слів, що заважатимуть класифікації і не несуть вагомого значення. Лематизація виконується за наступним прикладом:

```
from nltk.stem import WordNetLemmatizer
```

```
lemmatizer = WordNetLemmatizer(twenty_train)
```

Стематизація

Кінцевим етапом обробки тексту у багатьох випадках є стематизація слова, але існує ще багато способів, якими можна підготувати текст до тренування моделі, що, звісно, залежать від мети наукового дослідження чи практичного завдання. До них, зокрема, належать Part-of-Speech tagging, Named-entity recognition та інші.

Під стематизацією розуміють процес знаходження основи кожного слова, тобто, відсікання закінчень та суфіксів слова, залишаючи лише його корінь:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
ps = PorterStemmer()
words = ["program", "programs", "programer",
        "programing", "programers"]
```

Векторизація текстів

Необхідною умовою того, щоб комп'ютер міг працювати з текстами, що складаються зі слів природної мови, є їх векторизація. Векторизація – процес перетворення слів у цифрові вектори. Для цього використовуються спеціальні моделі, найбільш популярними з яких є: Bag of Words, TF/IDF, Word2Vec, Sent2Vec та інші. Нижче подано два типи векторизації текстів, кожен з яких має свої переваги та недоліки.

Bag of words

У перекладі з англійської Bag of words означає «Мішок слів», що грунтується на головній ознаці цього методу: порядок слів у цій моделі векторизації не береться до уваги, і головною ознакою, що визначатиме вагу слова, є кількість появ слова у корпусі текстів. На основі корпусу текстів складається його словник, що містить усі унікальні слова, наявні в усіх текстах.

Наприклад, корпус нараховує загалом 24 слова, а після вилучення повторюваних слів його словник матиме 10 унікальних слів. Оскільки відома кількість унікальних слів, що входять до словника, вектор кожного речення у наборі текстів матиме 10 позицій, де кожна позиція буде нулем або одиницею, що відповідатиме наявності або відсутності кожного слова зі словника у реченні:

1 "it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
2 "it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
3 "it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]

Якщо корпус текстів складається з десятків книг художньої літератури, то словник такого корпусу може містити мільйони слів і вектор кожного речення, таким чином, може мати до мільйона координат з нулями та одиницями, де нулі, очевидно, займатимуть майже всі позиції.

Для того, щоб застосувати метод Bag of Words до певного корпусу попередньо обробленого тексту, можна використовувати наступний фрагмент коду:

```
for sentence in allsentences:
   words = word_extraction(sentence)
   bag_vector = numpy.zeros(len(vocab))
   for w in words:
      for i,word in enumerate(vocab):
```

```
if word == w:
   bag_vector[i] += 1
```

TF/IDF-векторизація

Метод TF/IDF-векторизації складається з двох термінів, які визначають її роботу: TF (Term Frequency) та IDF (Inverse Document Frequency), що допомагає визначати вагу слова у всьому наборі даних.

Term Frequency визначає те, наскільки часто слово з'являється у документі чи корпусі текстів. Оскільки кожне речення має різну довжину, це є ймовірністю появи слова у довгих реченнях порівняно з короткими.

Формула розрахунку частоти слів наступна:

$$TF = \frac{кількість\ появ\ слова\ у\ документі}{загальна\ кількість\ слів\ у\ документі}$$

Inverse Term Frequency використовується для розрахунку вагомості слова. Поняття базується на думці про те, що слова, які рідше зустрічаються, є більш важливими. Формула для визначення IDF має вигляд:

$$IDF = log_{10} \frac{кількість\ документів}{кількість\ документів\ з\ заданим\ словом}$$

Таким чином, якщо загалом є, наприклад, 10 документів, а якесь слово зустрічається у даному наборі документів лише 2 рази, вагомість такого слова розраховується як $^{10}\!/_{2} = 5$. Такий розрахунок дозволяє додавати ваги словам, які зустрічаються рідко, на противагу тим, що наявні майже у кожному документі [8].

Отримані значення TF та IDF перемножуються для кожного слова і результат використовується у подальшій роботі.

Завантажити TF/IDF-векторизатор з бібліотеки Scikit-learn можна наступним чином:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer()
text_tf = tf.fit_transform(data['Phrase'])
```

Класифікація текстів з учителем

Класифікація текстів – це процес визначення відповідного класу для тексту, який передається класифікатору. Якщо розглядати базові задачі класифікації, тобто, класифікацію з учителем, то тут кожен вихідний текст сприймається класифікатором незалежно від усіх інших текстів, адже ці тексти попередньо були розмічені [11]. Серед деяких задач класифікації можна виділити наступні:

– визначити тип повідомлення у електронному листі: спам, чи ні;
– визначити тему статті з новинами, вибравши її з попередньо визначеного набору тем, наприклад: "спорт", "технології", "політика";
– визначити, яке з існуючих значень певного слова, наприклад, слова "коса", мається на увазі у реченні: як назва жіночої зачіски, як сільськогосподарське знаряддя праці чи як місце суші, оточене водою.

Класифікація за допомогою Decision Trees

Дерево рішень – це проста блок-схема, яка вибирає мітки для вхідних значень. Ця блок-схема складається з вузлів рішення, які перевіряють важливість ознак, та вузлів-листків, які присвоюють мітки. Щоб у блок-схемі вибрати мітку для вхідного значення, робота починається зі стартового вузла розв'язку, відомого як його кореневий вузол.

Цей вузол містить умову, яка перевіряє одну з ознак введеної одиниці даних і вибирає гілку на основі значення цієї ознаки. Слідом за гілкою, яка описує вхідне значення, відбувається перехід до нового вузла розв'язку з новою умовою для перевірки значення вхідної ознаки. Такі дії переходу до нових гілок, вибраних згідно умов кожного вузла, продовжуються, поки не буде досягнуто вузол-лист, що містить мітку для введеного значення. На рис. 3 продемонстровано приклад моделі дерева рішень для задачі з розпізнавання статі за її назвою [13].

Побудоване дерево рішень далі використовується для розмітки нових вхідних значень. Однак, при цьому залишається відкритим питання щодо того, як побудувати дерево рішень, яке змоделює існуючий набір даних. Перед тим, як сконцентруватись на природі алгоритму побудови дерева рішень, потрібно розглянути одну простішу задачу – вибір найкращого остова рішень (з англ. decision stump) для корпусу текстів.

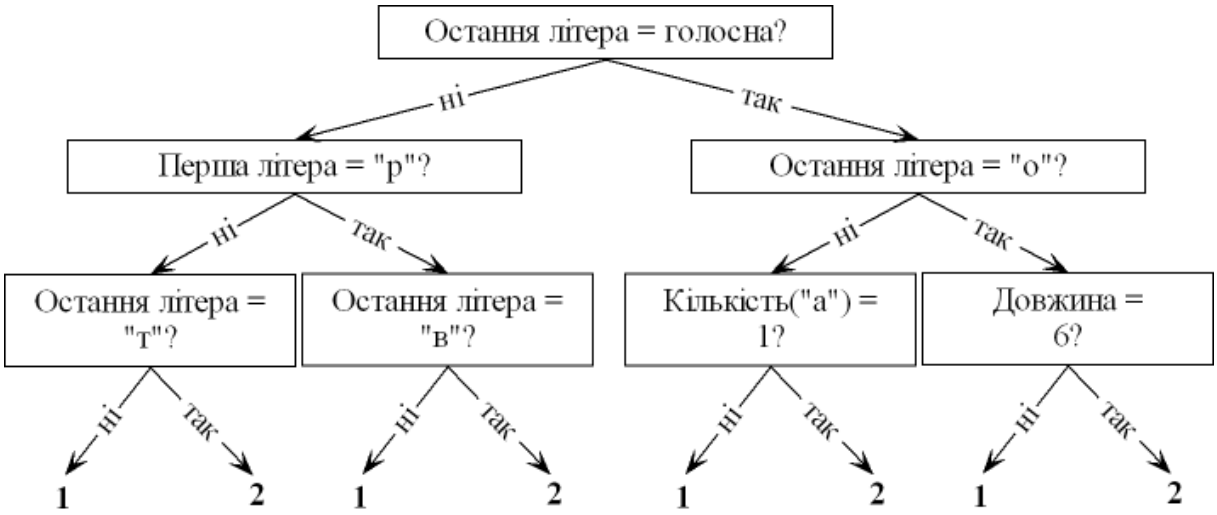

Рис. 3. Приклад дерева рішень

Остів рішень – це дерево рішень, яке має лише один вузол-листок, що розмічає вхідні дані на основі лише однієї ознаки. Загалом остів рішень має один листок для кожного можливого значення ознаки, визначаючи клас, який має бути присвоєно вхідним даних, зважаючи на цю ознаку. Щоб побудувати остів рішень, спершу необхідно визначити, яка ознака має бути використана. Найпростіший шлях – побудувати остів рішень для усіх можливих ознак і простежити, яка з них матиме найбільшу точність на тренувальному наборі даних, після чого можна побудувати остів рішень, присвоюючи мітку кожному листку і опираючись при цьому на те, яка з ознак превалює для вибраних зразків з тренувального набору даних [7].

Після отримання алгоритму для вибору остова рішень, будується алгоритм для дерева рішень. Його робота починається з вибору найкращого остова рішень для задачі класифікації. Потім перевіряється кожен листок дерева на тренувальному наборі даних. Листки, що не досягають достатньої точності, замінюються новими остовами рішень, тренованими на корпусі текстів, що обирається шляхом до листка. Наприклад, можна побудувати дерево рішень з рис. 3, замінивши листки, що розташовуються зліва, новим остовом рішень, тренованим на підмножині набору даних, що не починаються з "р" та не закінчуються на голосну.

Програмний коду для класифікації за допомогою Decision Trees після попередньої обробки має вигляд:

```
from sklearn import tree
unique_words = []
```

```
for sentence in tagged_tokenized_comments_corpus:
for word in sentence[0]:
   unique_words.append(word)
unique_words = set(unique_words)

dictionary = {}
i = 0
for dict_word in unique_words:
   dictionary.update({i, dict_word})
   i = i + 1
   train_target = []
   train_data = []
   for sentence in tagged_tokenized_comments_corpus:
      train_target.append(sentence[0])
      train_data.append(sentence[1])

clf = tree.DecisionTreeClassifier()
clf.fit(train_data, train_target)

test_data = """Beautiful Keep it up..
               this far is the most usable app editor..
               it makes my photos more beautiful and alive.. """

test_words = tokenizer.tokenize(test_data)
test_tokenized_sentence = []
for test_word in test_words:
   if test_word not in stop_words:
   test_tokenized_sentence.append(
      lemmatizer.lemmatize(test_word.lower()))

print("predicting the labels: ")
print(clf.predict(test_tokenized_sentence))
```

Класифікація за допомогою Naïve Bayes

У класифікації за допомогою Naïve Bayes для кожної ознаки розраховується прогноз того, яку мітку має отримати дане вхідне значення. Щоб вибрати мітку для вхідного значення, класифікатор Naïve Bayes спочатку розраховує попередню ймовірність того, що цьому зразку буде присвоєна певна мітка, що визначається перевіркою того? як часто така мітка зустрічається у тренувальному наборі даних. Потім розраховується вплив кожної ознаки у поєднанні з попередньою (апріорною) ймовірністю, щоб визначити ймовірність для кожної мітки. Мітка, що отримує найбільшу ймовірність, присвоюється вхідному значенню (рис. 4).

Рис. 4 ілюструє процес, використовуваний класифікатором Naïve Bayes, щоб обрати тему для вхідного документа. У тренувальному наборі даних більшість документів стосуються теми “наука”, тому класифікатор визначає свою першу точку розрахунку у місці, наближеному до відповідного кута трикутника. Але потім він починає зважати на значення усіх наявних ознак. Конкретно у поданому прикладі вхідний документ містить слово “райтинг”, що є слабким індикатором наближеності до тем “політика” та “спорт”, але в ньому також наявне і слово “рахунок”, що має велику вагу ознаки, яка відповідає темі “спорт”. Після того як для кожної наявної ознаки розраховане значення, класифікатор перевіряє, для якої ознаки це значення найбільш вагоме, і присвоює цю ознаку вхідному документу.

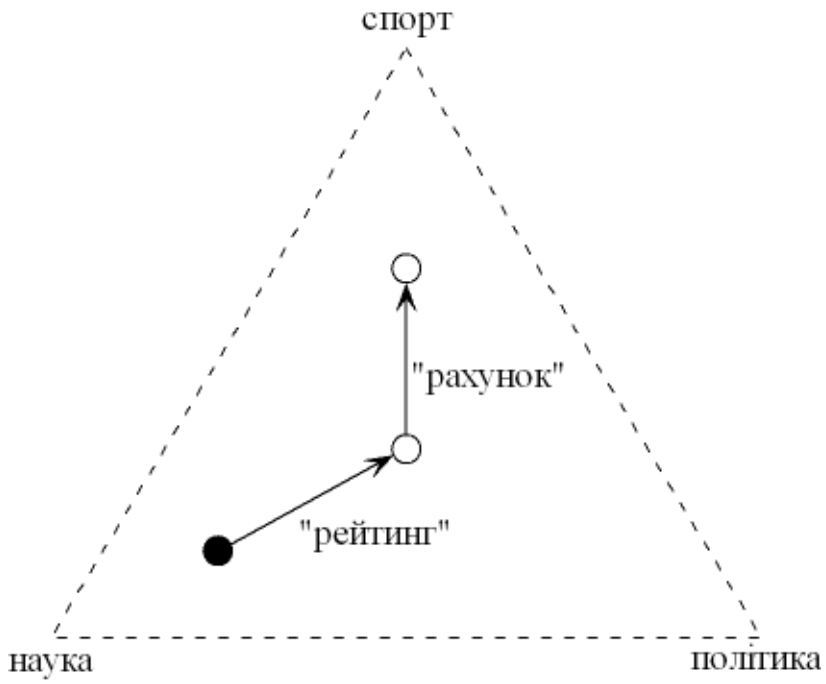

Рис. 4. Приклад оцінки ймовірності мітки

Окремі ознаки впливають на розраховане значення завдяки тому, що “голосують проти” міток, які не дуже часто присвоюються якійсь ознаці у наборі тренувальних даних. Зокрема, якщо слово “рейтинг” зустрічається у 12% документів про спорт, у 10% документів про політику, та лише у 2% документів про науку, то ймовірність того, що документ належатиме до теми “спорт”, множиться на 0.12, до теми про політику – на 0.1, а до теми про науку – лише на 0.02. У результаті ймовірність присвоєння вхідній ознаці теми про спорт буде трохи вищою, аніж для теми про політику, і значно меншою для мітки про науку порівняно з двома попередніми. Кожен етап розрахунку ймовірностей для міток можна прослідкувати на рис. 5.

Програмний код для класифікації за допомогою Naïve Bayes на стандартному наборі текстів з бібліотеки NLTK “Movie reviews” має наступний вигляд:

```
from nltk.corpus import movie_reviews

documents = [(list(movie_reviews.words(fileid)), category)
for category in movie_reviews.categories():
   for fileid in movie_reviews.fileids(category)]:
      random.shuffle(documents)
all_words = nltk.FreqDist(w.lower())
for w in movie_reviews.words()):
   word_features = all_words.keys()[:2000]

def document_features(document):
   document_words = set(document)
   features = {}
   for word in word_features:
      features['contains(%s)' % word] = (word in document_words)
   return features

featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print 'Accuracy: %4.2f'  \
```

```
% nltk.classify.accuracy(classifier, test_set)
```

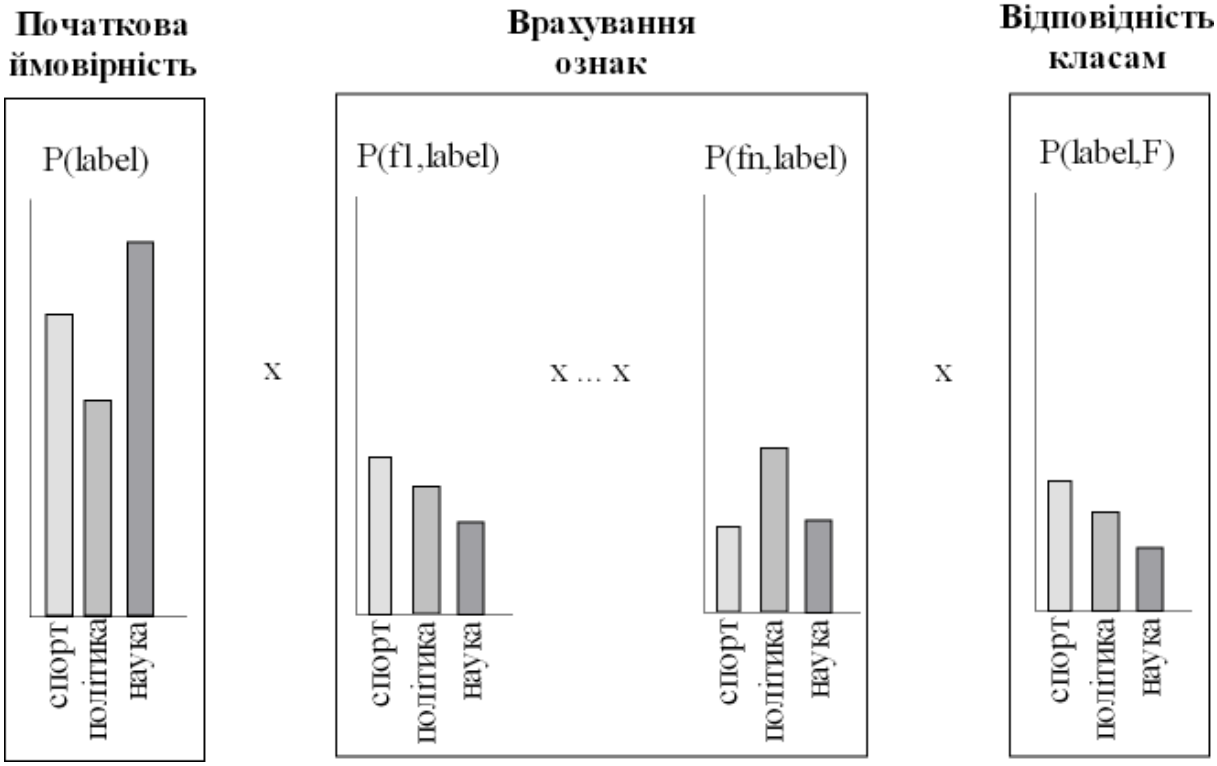

Рис. 5. Розрахунок ймовірностей для кожної мітки

Поняття автоматичного відповідача (чат-бота)

Чат-боти – це такий вид програмного забезпечення, що використовує алгоритми штучного інтелекту та обробки природньої мови, щоб зрозуміти, яку команду хоче дати йому людина, і надає їй бажаний результат з найменшою кількістю витраченого часу з боку користувача. Цей вид програмного забезпечення почав набувати популярності з 2016 року, коли Фейсбук відкрив свою розробницьку платформу [5] і показав світу, якого результату можна досягти за допомогою їхнього Messenger App. Трохи пізніше Гугл створив свого голосового асистента. З того часу кількість чат-ботів почала надзвичайно стрімко зростати, вони вбудовувалися у веб-сайти, додатки, соціальні мережі, використовувалися для служби підтримки клієнтів та у багатьох інших застосуваннях.

Чат-боти можна побудувати різними способами. Технологія, що лежить у їх основі, може дещо відрізнятися залежно від використовуваного методу. Загалом, типи чат-ботів, що існують на поточний момент, можна розділити на дві категорії:

– чат-боти на основі правил, тобто ті, що у своїй роботі керуються лише певними правилами і не мають сценарію поведінки, який міг би відрізнятись від наявного набору правил, або на який може впливати контекст розмови з користувачем.
– чат-боти на основі штучного інтелекту, тобто ті, що автоматично донавчаються у процесі роботи після проходження попереднього етапу навчання розробником на основі певного набору даних.

Чат-боти на основі правил працюють за допомогою встановлених для них правил, що прописані розробником, опираючись на певні ключові слова. Такі чат-боти не запрограмовані на зміну повідомлень у відповідності зі змінами у мові та мають

визначену структуру діалогу, у якій користувач отримує відповіді на потрібні йому питання, співвідносячи ввід користувача з підготовленими заранні відповідями.

Існують різни види архітектури чат-ботів, що обирається залежно від основної мети розробки. Є два способи того, як чат-бот може надати відповідь: він може або створити відповідь за допомогою моделей машинного навчання, або використати евристики, щоб обрати потрібний варіант з бібліотеки зарані визначених відповідей.

Генеративна модель архітектури

Генеративна архітектура чат-боту (рис. 6) використовується для розробки розумних ботів, що можуть самі спілкуватися з користувачем, не використовуючи правил. Цей вид чат-ботів дуже рідко використовується, адже потребує застосування комплексу алгоритмів. Генеративні моделі відносно важко побудувати та розробити. Тренування цього типу ботів потребує дуже багато часу та зусиль на збір мільйонів тренувальних даних, адже саме за допомогою даних модель глибокого навчання може ефективно спілкуватись з користувачем. Однак, незважаючи на складність використовуваних алгори тмів, достовірність відповідей, які генерує модель, не є стовідсотковою [3].

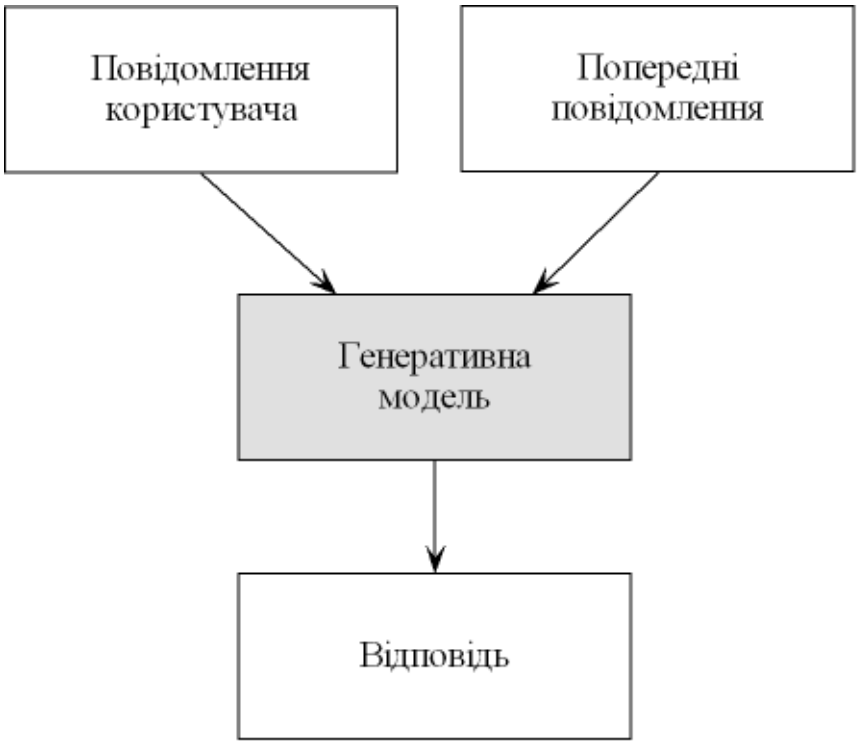

Рис. 6. Генеративна архітектура чат-боту

Моделі на основі пошуку (Retrieval-based)

Цей тип архітектури моделі чат-боту легше побудувати, ніж генеративний, а також він набагато надійніший. Хоча не можна бути стовідсотково впевненим у достовірності відповідей такого чат-боту, можливі типи його відповідей зарані відомі, за рахунок чого можна забезпечити відповідь чат-боту за потрібною темою [4].

Моделі на основі пошуку (рис. 7) є найбільш популярними на даний момент. Наразі розробникам доступні кілька алгоритмів та API, щоб побудувати чат-бота на основі цієї моделі архітектури. Такий чат-бот розпізнає повідомлення та його контекст, щоб вибрати найкращу відповідь з-поміж списку зарані визначених повідомлень.

Застосування попередньої обробки та класифікації текстів для побудови автоматичних відповідачів

Найпростіший варіант генерації відповіді чат-бота використовує if-else умови чи навчання машинних класифікаторів для визначення набору правил із заздалегідь визначеними шаблонами, які виступають умовою для оформлених правил. Найбільшою популярністю для формування таких шаблонів та відповіді у процесі розробки чат-боту використовується мова розмітки Artificial Intelligence Markup Language (AIML) [1, 9].

З правильно налаштованою обробкою тексту за допомогою NLP та попередньо визначеними добре продуманими шаблонами, AIML можна використати для побудови розумних чат-ботів. Такі боти роблять аналіз повідомлення користувача, знаходять у ньому синоніми та теми, розмічають його по членам речення та визначають, який з множини попередньо заданих шаблонів підходить для запиту користувача. Однак, ці боти не використовують алгоритми машинного навчання або будь-які інші API за замовчуванням [5].

Рис. 7. Архітектура чат-боту на основі пошуку

Хоча евристика на основі зразків приносить досить непогані результати, проблема полягає в тому, що всі шаблони спілкування з користувачем потрібно прописувати вручну. Це досить кропітке завдання, особливо якщо чат-бот повинен розпізнавати сотні намірів користувачів, що можуть мати різні сценарії діалогів.

Класифікація намірів – завдання, побудоване цілком на технології машинного навчання, що дозволяє тренувати ботів. За допомогою набору тренувальних даних, що

містять тисячі прикладів комунікації, які найімовірніше знадобляться у спілкуванні з користувачем, чат-бот навчиться виділяти ознаки у даних і вдосконалюватись.

Якщо чат-бот зміг зрозуміти запит користувача, наступним кроком є генерація відповіді. Одним зі шляхів є генерація простої статичної відповіді. Іншим способом є отримання шаблону на основі наміру користувача та додавання до нього деяких змінних [6]. Наприклад, чат-бот для прогнозування погоди, що використовує API, щоб отримати інформацію про погоду за певною геолокацією, може мати кілька варіантів відповіді про одну й ту ж подію “сьогодні швидше за все дощитиме”, “сьогодні буде дощовий день” або “ймовірність дощу 80%, тому захопіть з собою парасольки”.

Стиль відповіді варіюється залежно від користувача. У цьому випадку бот може дослідити та вивчити ознаки попередніх діалогів, щоб створити персоналізовані відповіді для користувача.

Таким чином, на основі поданого вище матеріалу нами пропонується наступна структура алгоритму роботи автоматичного генератора відповідей з використанням корпусу повідомлень користувача та попередньо виконаної обробки текстів повідомлень:

1. Аналіз повідомлень користувача за допомогою методів обробки природньої мови. Метою аналізу є створення словника користувача, а також бази знань, що містить відповіді користувача на різні запити. База у подальшому використовуватиметься як з метою оцінки питання користувача, так і для генерації відповіді.
2. Робота системи у діалговому режимі, тобто: питання – відповідь.
2.1. Після отримання запитання проводиться його аналіз для визначення теми запитання на основі бази знань.
2.2. Вибір множини можливих відповідей на основі бази знань та оцінка їх апріорної ймовірності.
2.3. Генерація остаточної відповіді користувачу на основі вибраної множини відповідей з використанням власних евристик.

Зважаючи на концептуальність поданої структури алгоритму, у кожному з його етапів можна виділити окремі підзадачі, що самі по собі можуть бути напрямками для подальшої роботи. Зокрема, нами визначено напрямки можливих подальших досліджень.

1. Задача попередньої обробки текстів досить добре розв’язана для англійської мови та споріднених з нею. У той же час окремі етапи попередньої обробки для української мови ставлять нові задачі у зв’язку з іншим способом словотворення та побудовою речень, що є менш формалізованою порівняно з англійською мовою.
2. Задача побудови бази знань на основі повідомлень користувача вимагає попередньої оцінки як структури програмних одиниць, так і структури синтаксичних та лексичних одиниць природньої мови.
3. Вимагають додаткового дослідження евристики вибору можливих варіантів відповідей та генерації остаточної відповіді користувачу. Для цього можуть використовуватись як зокрема методи машинного навчання, так і загалом методи штучного інтелекту, такі як обробка баз знань, автоматична генерація суджень тощо.

Висновки

Отже, у ході проведеного дослідження розглянуто основні вимоги до вибору даних та методи попередньої обробки тексту, такі як: видалення пунктуації, тегів, стоп-слів, приведення тексту до нижнього регістру. Розглянуто методи, за допомогою яких

векторизується природна мова, зокрема: Bag of Words, TF/IDF. Досліджено методи класифікації за допомогою Naïve Bayes та Decision Trees, розглянуто окремі приклади коду, що дозволяють реалізувати вказані класифікатори й отримати точність розподілення попередньо розмічених текстів за групами. Подано опис поняття автоматичного відповідача та наведено дві найпоширеніші моделі архітектури таких відповідачів: генеративну модель та модель на основі пошуку. Побудовано концептуальний алгоритм роботи автоматичного відповідача з використанням методів машинного навчання для попередньої класифікації повідомлень користувача.

Список використаних джерел

1. Allen J.F. Analyzing intention in utterances / J.F. Allen, C.R. Perrault // Artificial Intelligence. – 15(3). 1980. Pp. 143-178.
2. Anaconda | The World's Most Popular Data Science Platform / [Електронний ресурс] – Режим доступу: https://www.anaconda.com/
3. Britz D. Deep learning for chatbots / [Електронний ресурс] – Режим доступу: http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/
4. Cassell J. Human conversation as a system framework: designing embodied conversational agents, Embodied conversational agents / J . Cassell, T. Bickmore, L. Campbell, H. Vilhjálmsson. MIT Press, Boston, 2000. – P. 29-63.
5. Facebook for developers / [Електронний ресурс] – Режим доступу: https://developers.facebook.com/
6. Galitsky B.A. Chatbot with a discourse structure-driven dialogue management / B.A. Galitsky, D. Ilvovsky // EACL Demo E17-3022. Valencia, Spain. – 2017 / [Електронний ресурс] – Режим доступу: https://developers.facebook.com/
7. Jurafsky D. Speech and Language Processing / D. Jurafsky, J. H. Martin. – Stanford: Pearson Prentice Hall, 3nd edition, 2019. – 605 p.
8. Kurdi M.Z. Natural Language Processing and Computational Linguistics: speech, morphology, and syntax / M. Z. Kurdi. – London: Wiley ISTE, 2016. – 296 p.
9. Lee C. Example-based dialog modeling for practical multi-domain dialog system / C. Lee, S. Jung, S. Kim, G.G. Lee // Speech Comm 51:466, 2009.
10. Natural Language Toolkit / [Електронний ресурс] – Режим доступу: https://www.nltk.org/
11. Powers D. Machine Learning of Natural Language / D. Powers, C. Turk. – New York: Springer Verlag, 1989. – 358 p.
12. Project Jupyter | Home / [Електронний ресурс] – Режим доступу: https://jupyter.org/
13. Steven B. Natural Language Processing with Python / B. Steven, E. Klein, E. Loper. – Sebastopol, CA: O'Reilly Media, 2009. – 504 p.
14. The 20 newsgroups text dataset / [Електронний ресурс] – Режим доступу: https://scikit-learn.org/
15. The Size and Quality of a Data Set / [Електронний ресурс] – Режим доступу: https://developers.google.com/machine-learning/data-prep/construct/collect/data-size-quality

Bibliography:

1. Allen, J.F., Perrault, C.R. (1980). Analyzing intention in utterances. Artificial Intelligence, 15(3):143-178.
2. Anaconda | The World's Most Popular Data Science Platform. Retrieved from: https://www.anaconda.com/
3. Britz D. Deep learning for chatbots. Retrieved from: http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/
4. Cassell, J., Bickmore, T., Campbell, L., Vilhjálmsson, H. (2000). Human conversation as a system framework: designing embodied conversational agents, Embodied conversational agents. MIT Press, Boston, pp 29–63.
5. Facebook for developers. Retrieved from: https://developers.facebook.com/
6. Galitsky, B.A., Ilvovsky, D. (2017). Chatbot with a discourse structure-driven dialogue management. EACL Demo E17-3022. Valencia, Spain. Retrieved from: https://developers.facebook.com/
7. Jurafsky, D., Martin, J.H. (2019). Speech and Language Processing. Stanford, Pearson Prentice Hall, 3nd edition.
8. Kurdi, M.Z. (2016). Natural Language Processing and Computational Linguistics: speech, morphology, and syntax. London: Wiley ISTE.
9. Lee, C., Jung, S., Kim, S., Lee, G.G. (2009). Example-based dialog modeling for practical multi-domain dialog system. Speech Comm 51:466.
10. Natural Language Toolkit. Retrieved from: https://www.nltk.org/
11. Powers, D., Turk, C. (1989). Machine Learning of Natural Language. New York: Springer Verlag.

12. Project Jupyter | Home. Retrieved from: https://jupyter.org/
13. Steven, B., Klein, E., Loper, E. Natural Language Processing with Python. Sebastopol, CA: O'Reilly Media.
14. The 20 newsgroups text dataset. Retrieved from: https://scikit-learn.org/
15. The Size and Quality of a Data Set. Retrieved from: https://developers.google.com/machine-learning/data-prep/construct/collect/data-size-quality

KOVALENKO Olena,
Student, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine
SERDIUK Oleksandr,
Candidate of Economic Sciences, Department of Informatics and Applied Mathematics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine

USING THE CLASSICAL METHODS OF MACHINE LEARNING FOR TEXT CLASSIFICATION IN THE AUTOMATIC ANSWERS GENERATION PROGRAMS

Summary. Introduction. *Machine learning methods are now widely used for natural language processing to recognize, understand, interpret, and generate natural language by computer algorithms. In the modern world of information technology, the range of tasks that can use the results of human language interpretation is constantly increasing, including automatic texts classification, news rubrication, recognition of user questions in automatic houses and generation of answers. The widespread use of messengers to transmit messages between users, along with the increasing number of users, has opened up another area of application for text classification tasks: the development of automatic assistants for generating messages. To solve this task, it is necessary not only to be able to process natural language, but also to generate adequate answers, or, in general, sentences for dialogue. The article is aimed at reviewing the methods of text classification for their using in generating automatic answers.*

The ***purpose*** *of the article is to study methods and algorithms and demonstrate program fragments of classical machine learning methods for text classification and application of classification results in automatic answer generation programs.*

To achieve the goal, the following issues are considered in the article: review of data selection conditions; review of word processing methods; review of text vectorization methods; review of some text classification algorithms; introducing the concept of automatic answers generation; developing the structure of the program for automatic answers generation using the corpus of procesed texts.

Based on the study, the following ***conclusions*** *were obtained. The problem of pre-processing texts is quite well solved for English and related languages. At the same time, some stages of pre-processing for the Ukrainian language pose new challenges due to a different way of word formation and sentence construction, which is less formalized compared to English. The task of building a knowledge base based on user messages requires a preliminary assessment of both the structure of program units and the structure of syntactic and lexical units of natural language. Required also additional research on the heuristics of the choice of possible answers and generate a final answer to the user. For this purpose, both machine learning methods and artificial intelligence methods in general, such as processing knowledge bases, automatic generation of judgments, etc. can be used.*

Keywords: *machine learning, text classification, chat-bots, programming, Python.*

Одержано редакцією 09.01.2020 р.
Прийнято до публікації 30.03.2020 р.