НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра інформатики та програмного забезпечення

(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з Оброблення надвеликих масивів даних

(назва дисципліни)

на тему: Аналіз та детекція пропаганди у Telegram-каналах з використанням NLP та розподілених обчислень

Студента (ки) 1 курсу ІП-51мн групи

Спеціальності F2 Інженерія програмного забезпечення

Шляхтуна Д. М.

(прізвище та ініціали)

Керівник:

доцент каф. ІПІ, к.т.н., Олійник Ю.О.

ас. кафедри ІПІ, Ph/D. Зарічковий О.А.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка _____

Кількість балів: _____

Члени комісії

_____ _____

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2025 рік

Національний технічний університет України “КПІ імені Ігоря Сікорського”

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна

«Оброблення надвеликих масивів даних»

Спеціальність F2 "Інженерія програмного забезпечення"

Курс 1 Група ІП-51мн Семестр 1

ЗАВДАННЯ

на курсову роботу студента

Шляхтуна Дениса Михайловича

(прізвище, ім’я, по батькові)

1. Тема роботи Аналіз та детекція пропаганди у Telegram-каналах з використанням NLP та розподілених обчислень

2. Строк здачі студентом закінченої роботи 22.12.2025

3. Вихідні дані до роботи датасет публікацій пропагандистських Telegram-каналів з колонками «Date», «ChannelName», «PostText» на 11850 рядків; датасет публікацій Telegram-каналів з колонками «channel», «date», «text», «views» на 1 мільйон рядків

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Аналіз предметної області, розробка ETL-процесів, екстракція багатовимірних ознак, класифікація, верифікація та інтерпретація результатів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Дата видачі завдання 01.10.2025

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	01.10.2025	
2.	Визначення основних задач курсової роботи	16.10.2025	
3.	Пошук та вивчення літератури з питань курсової роботи	24.10.2025	
4.	Розробка процесу попередньої обробки даних, ETL процесів, моделі БД	21.11.2025	
5.	Розробка методів обробки даних	05.12.2025	
6.	Дослідження ефективності методів обробки даних	12.12.2025	
7.	Підготовка пояснювальної записки	17.12.2025	
8.	Здача курсової роботи на перевірку	22.12.2025	
9.	Захист курсової роботи	23.12.2025	

Студент _____ Шляхтун Денис Михайлович

(підпис) (прізвище, ім’я, по батькові)

Керівник _____ Олійник Юрій Олександрович

(підпис) (прізвище, ім’я, по батькові)

"___" _____ 2025 р.

АНОТАЦІЯ

Шляхтун Денис Михайлович. Курсова робота на тему «Аналіз та детекція пропаганди у Telegram-каналах з використанням NLP та розподілених обчислень».

Текстова частина курсової роботи складається із вступу, 4 розділів, висновків, списку використаних джерел з 12 найменувань та 1 додатку, які викладено на 45 сторінках. В тексті роботи оформлено 12 рисунків, 3 таблиці.

У курсовій роботі розроблено та досліджено масштабовану систему для автоматизованого виявлення пропагандистського контенту в месенджері Telegram.

Зміст роботи охоплює повний цикл оброблення надвеликих масивів даних: від розробки ETL-конвеєра до екстракції багатовимірних лінгвістичних ознак (TTR, суб'єктивність, частота займенників) та мережевого аналізу координованої поведінки за допомогою алгоритму MinHashLSH.

Отримані результати підтверджують ефективність обраного підходу. Побудована модель Random Forest із використанням PU Learning продемонструвала показник AUC-ROC = 0.7104. Найважливішими ознаками для детекції виявилися рівень суб’єктивності та використання специфічних займенникових груп. Агрегований аналіз дозволив успішно ідентифікувати пропагандистські канали на фоні змішаного масиву нерозмічених даних, що доводить практичну цінність системи для моніторингу інформаційної безпеки.

Ключові слова: Spark, Spark NLP, SpaCy, пропаганда, класифікація, Random Forest.

ЗМІСТ

ВСТУП

Сучасний етап розвитку глобального інформаційного суспільства характеризується переходом міждержавних протистоянь у цифрову площину, де інформаційно-психологічні операції (ІПСО) стають невід'ємною частиною гібридних конфліктів. Особливу роль у поширенні маніпулятивного контенту відіграють месенджери, серед яких Telegram посідає провідне місце завдяки високому рівню анонімності користувачів та політиці мінімальної модерації вмісту. В умовах повномасштабної збройної агресії росії проти України Telegram перетворився на стратегічний майданчик для розповсюдження дезінформації та пропаганди, що створює прямі загрози національній безпеці, соціальній стабільності та критичному сприйняттю інформації населенням.

Традиційні методи модерації та ручного експертного аналізу виявляються неефективними через експоненціальне зростання обсягів текстових даних, що генеруються щодня. Це зумовлює гостру потребу в розробці масштабованих автоматизованих систем детекції пропаганди, здатних забезпечувати високу швидкість обробки надвеликих масивів даних.

Метою роботи є розробка та імплементація масштабованої системи для автоматизованого виявлення та аналізу пропагандистського контенту в месенджері Telegram із використанням технологій розподілених обчислень та методів інтелектуального оброблення природної мови.

Методи дослідження базуються на використанні фреймворку Apache Spark для розподіленої обробки даних, бібліотек Spark NLP та SpaCy для лематизації та морфологічного аналізу, методів тематичного моделювання (LDA), аналізу тональності на основі трансформерних моделей XLM-RoBERTa та методів машинного навчання.

Практичне значення отриманих результатів полягає у створенні програмного прототипу системи, яка дозволяє не лише класифікувати окремі повідомлення, а й формувати цілісну аналітичну картину діяльності інформаційних ресурсів, що може бути використано фахівцями з інформаційної безпеки для оперативного моніторингу медіапростору.

1 ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис проблеми

Сучасний глобальний інформаційний простір характеризується інтенсифікацією інформаційних операцій, де соціальні мережі та месенджери стають основними інструментами маніпулювання суспільною думкою. Особливе місце у цій екосистемі посідає месенджер Telegram [1], який, завдяки політиці мінімальної модерації та високому рівню анонімності, перетворився на ключовий майданчик для розповсюдження дезінформації та пропаганди, особливо в контексті російсько-української війни.

Головна складність ідентифікації пропаганди в Telegram полягає у її багатогранності та здатності до мімікрії під нейтральні новинні повідомлення. Пропагандистські кампанії часто мають характер «координованої неавтентичної поведінки», коли мережі каналів синхронно поширюють ідентичні або семантично схожі тексти для створення ілюзії масової підтримки певного наративу [2].

Обсяги інформації в Telegram унеможливлюють ручну модерацію чи експертний аналіз у реальному часі. Це вимагає застосування технологій розподілених обчислень, зокрема Apache Spark, для забезпечення необхідної швидкості обробки.

Використання української та російської мов у межах одного інформаційного простору створює додаткові виклики для систем обробки природної мови, що потребує залучення складних мультимовних моделей для лематизації та аналізу тональності.

Пропагандисти постійно змінюють лінгвістичні стратегії, використовуючи «завантажену лексику», апеляцію до страху та поділ на «ми-вони», що робить класичні методи детекції на основі ключових слів неефективними.

Невирішеність проблеми автоматизованої детекції пропаганди призводить до зниження критичного сприйняття інформації населенням та створює прямі загрози національній безпеці. Таким чином, існує гостра потреба у розробці

масштабованих систем, що здатні не лише класифікувати тексти, а й виявляти приховані зв’язки та координацію між джерелами розповсюдження контенту.

1.2 Постановка задачі

Метою даної курсової роботи є розробка та імплементація масштабованої системи для автоматизованого виявлення та аналізу пропагандистського контенту в месенджері Telegram. Система має базуватися на технологіях розподілених обчислень для забезпечення обробки масивів даних у двомовному середовищі (українська та російська мови).

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Розробка ETL-процесів: реалізація процесів збору, очищення та попередньої обробки текстових даних (нормалізація, видалення нетекстових елементів, лематизація) з використанням фреймворку Apache Spark та бібліотеки Spark NLP.
2. Екстракція багатовимірних ознак:
   - Обчислення стилістичних метрик (Type-Token Ratio, частота вживання займенників «Ми/Вони») для ідентифікації маніпулятивного стилю.
   - Проведення аналізу тональності на основі мультимовних трансформерних моделей.
   - Застосування тематичного моделювання (LDA) для класифікації контексту повідомлень.
   - Ідентифікація координованих мереж за допомогою алгоритму MinHashLSH для виявлення дубльованого та майже дубльованого контенту, що є характерною ознакою пропагандистських кампаній.
3. Побудова моделі класифікації із застосуванням методики Positive-Unlabeled (PU) Learning для ефективної роботи в умовах відсутності повної розмітки негативного класу.
4. Проведення агрегованого аналізу діяльності Telegram-каналів та оцінка ефективності запропонованого підходу за метриками AUC-ROC та F1-Score.

На рисунку 1.1 зображена діаграма потоку даних, якій потрібно слідувати при виконанні курсової роботи.

Рисунок 1.1 – Діаграма потоку даних

Процес обробки даних у системі організовано за принципом послідовного конвеєра (pipeline). На вхід системи надходять два масиви даних: розмічений корпус пропаганди та змішаний потік публікацій. На першому етапі відбувається фільтрація шуму та лематизація. Другий етап відповідає за екстракцію багатовимірних ознак: психолінгвістичних метрик, результатів аналізу тональності XLM-RoBERTa та мережевих зв'язків MinHashLSH. Фінальний етап включає агрегацію ознак у єдиний вектор та запуск класифікатора Random Forest для отримання результату прогнозування.

2 ТЕОРЕТИЧНІ ЗАСАДИ ІДЕНТИФІКАЦІЇ ПРОПАГАНДИ

2.1 Поняття пропаганди та її ключові лінгвістичні ознаки

У сучасному науковому дискурсі пропаганда визначається як цілеспрямоване та систематичне поширення інформації, ідей або наративів для формування певних психологічних установок і керування поведінкою цільової аудиторії [1]. На відміну від об’єктивного інформування, пропагандистський контент характеризується ідеологічною заангажованістю та використанням маніпулятивних технік, що мають на меті обійти раціональний критичний аналіз отримувача.

Особливого значення детекція пропаганди набуває в умовах цифрових комунікацій, де месенджер Telegram виступає як середовище з мінімальним рівнем цензури, що дозволяє швидко поширювати емоційно поляризований контент [2]. Дослідники виділяють наступні ключові лінгвістичні та психолінгвістичні ознаки, що дозволяють ідентифікувати пропаганду.

Дихотомія «Ми – Вони» (поляризація): створення штучного антагонізму між групами, де «своя» група наділяється виключно позитивними рисами, а «чужа» — негативними [4]. У тексті це виражається через аномально високу частоту вживання займенників першої та третьої особи множини.

Завантажена лексика: використання слів із надмірним емоційним забарвленням, які мають на меті викликати автоматичну реакцію (страх, гнів, патріотичне піднесення) [1]. Такий підхід корелює з високим показником суб'єктивності тексту.

Навішування ярликів та дегуманізація: стратегія використання принизливих епітетів щодо опонентів, що позбавляє їх суб’єктності та виправдовує агресивні дії щодо них [5].

Апеляція до авторитету або страху: використання маніпулятивних посилань на сумнівні джерела або створення відчуття екзистенційної загрози для придушення логічного мислення [1].

Повторюваність (ітеративність): багаторазове відтворення одного й того самого повідомлення або наративу для закріплення його у свідомості аудиторії

як беззаперечного факту [2]. На мережевому рівні це проявляється через координоване поширення ідентичних текстових фрагментів у різних каналах.

Спрощення причинно-наслідкових зв’язків: подання складних суспільних процесів у вигляді примітивних схем, що виключає можливість альтернативної інтерпретації подій [4].

Крім зазначених ознак, пропаганда часто супроводжується зниженням лексичного розмаїття тексту, оскільки використання сталих кліше та лозунгів є більш ефективним для масового впливу. Таким чином, аналіз пропаганди є комплексною задачею, що потребує одночасного розгляду як лексико-семантичного складу окремих повідомлень, так і динаміки їх поширення у мережі Telegram.

2.2 Огляд підходів до автоматизованого виявлення пропаганди

Автоматизація детекції пропаганди є однією з найскладніших задач обробки природної мови (NLP), оскільки вона вимагає аналізу не лише змісту, а й інтенції автора та контексту розповсюдження інформації. Сучасні підходи до вирішення цієї задачі можна класифікувати на три основні групи.

Перша група – методи на основі лінгвістичних ознак та класичного машинного навчання. Ранні дослідження фокусувалися на екстракції статистичних та стилістичних атрибутів тексту. До них належать розрахунок частотності слів (TF-IDF), аналіз частин мови (POS-tagging) та використання спеціалізованих словників (наприклад, LIWC) для виявлення емоційно забарвленої лексики [4]. Класифікація зазвичай здійснюється за допомогою алгоритмів Logistic Regression, Support Vector Machines (SVM) або Random Forest [3]. Перевагою таких методів є висока інтерпретованість результатів, проте вони демонструють низьку ефективність при аналізі складних маніпулятивних стратегій, таких як іронія чи прихований сарказм.

Друга група – підходи на основі глибокого навчання. Розвиток нейромережевих архітектур дозволив перейти до векторного представлення слів (Word Embeddings). Використання рекурентних нейронних мереж (RNN), зокрема моделей з довгою короткостроковою пам'яттю (LSTM) та Gated

Recurrent Units (GRU), дало змогу враховувати послідовність слів та контекстуальну залежність у межах речення [5]. Такі моделі краще ідентифікують пропаганду на рівні фрагментів тексту, проте потребують значних обчислювальних ресурсів та великих обсягів розмічених навчальних даних.

Третя група – сучасні трансформерні моделі та аналіз мережевої координації. На сьогодні «золотим стандартом» у детекції пропаганди є використання моделей архітектури Transformer, таких як BERT, RoBERTa та їх мультимовних варіацій (XLM-RoBERTa) [6]. Ці моделі здатні вловлювати глибокі семантичні зв'язки та контекст на рівні цілих документів. Окремим перспективним напрямом, який критично важливий для месенджера Telegram, є аналіз координованої неавтентичної поведінки [2], [5]. Замість аналізу одного повідомлення, дослідники пропонують виявляти «сітки» джерел, що синхронно поширюють ідентичний контент. Для ефективної обробки таких мережевих структур на великих масштабах використовують алгоритми локально-чутливого хешування (MinHashLSH), які дозволяють знаходити майже ідентичні тексти серед мільйонів об’єктів у розподілених середовищах обробки даних, таких як Apache Spark [3].

Вибір методології для даної роботи базується на поєднанні лінгвістичного аналізу та сучасних методів детекції координації. Використання методики Positive-Unlabeled (PU) Learning дозволяє адаптувати ці підходи до реальних умов, де наявна лише часткова розмітка пропагандистських джерел, що робить систему більш стійкою до шуму у вхідних даних.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ ОБРОБЛЕННЯ ДАНИХ

3.1 Розробка ETL-процесів та попередня обробка даних

Початковий етап передбачає завантаження даних із двох джерел: масиву повідомлень Telegram обсягом 1 млн об'єктів у форматі JSON та спеціалізованого набору даних пропагандистського спрямування у форматі CSV. Для забезпечення консистентності структури обидва набори приводяться до єдиного формату колонок: channel_name, date та text.

З метою уникнення перенавчання моделі на контенті найбільш активних каналів, реалізовано алгоритм балансування вибірки. Здійснюється відбір обмеженої кількості повідомлень на кожне джерело (до 500 для новинних та 250 для пропагандистських каналів). Це дозволяє отримати репрезентативну вибірку загальним обсягом близько 18 000 повідомлень, рівномірно розподілених за класами.

Текстовий контент Telegram містить велику кількість неінформативних елементів, які видаляються за допомогою регулярних виразів. Процедура очищення deep_clean_text виконує:

- видалення гіперпосилань та згадок користувачів за допомогою шаблонів http та @username;
- видалення числових значень, що не несуть семантичного навантаження;
- фільтрацію піктограм емодзі за допомогою спеціалізованої бібліотеки emoji [10].

Оскільки набір даних містить повідомлення українською та російською мовами, застосовується алгоритм автоматичного визначення мови langdetect [9]. На основі отриманої мітки текст спрямовується до відповідної моделі лінгвістичного аналізу бібліотеки SpaCy: uk_core_news_sm для української або ru_core_news_sm для російської мови [8].

Процес лінгвістичної підготовки реалізовано через конвеєр nlp.pipe, що дозволяє виконувати лематизацію в пакетному режимі, значно прискорюючи обробку [8]. Під час лематизації здійснюється фільтрація стоп-слів та токенів, що

не є алфавітними символами (token.is_alpha), а також розрахунок початкового коефіцієнта суб'єктивності на основі частин мови (ADJ, ADV). Результатом цього етапу є сформований корпус лем, який зберігається у форматі Parquet для подальшої обробки в Apache Spark.

3.2 Вибір програмних засобів для визначення ознак

Після завершення етапу попередньої обробки та формування корпусу лем, дані переходять до стадії еквівалентного представлення у вигляді багатовимірних векторів ознак. Враховуючи обсяг вхідної інформації та необхідність виконання складних обчислень у двомовному середовищі, було обрано стек технологій на базі Apache Spark [3] та бібліотеки Spark NLP [11]. Це дозволяє забезпечити масштабованість обчислень та мінімізувати час обробки шляхом паралелізації завдань на рівні вузлів кластера.

Для ідентифікації емоційного забарвлення було обрано мультимовну модель XLM-RoBERTa (xlmroberta_classifier_verdict), інтегровану через Spark NLP [12]. Вибір цієї архітектури обумовлений її здатністю ефективно обробляти контекстуальні зв'язки в українській та російській мовах одночасно, що є критично важливим для Telegram-простору. Модель генерує ймовірності належності тексту до категорій «misinformation», «factual» та «neutral», що слугує ключовими ознаками для подальшої класифікації.

Для визначення спрямованості контенту каналів використано алгоритм LDA із бібліотеки Spark MLlib [3]. Побудова моделі з параметром k=20 тем дозволяє формалізувати контекст повідомлень, виділяючи специфічні для пропаганди теми (наприклад, військові зведення, ППО, міжнародні конфлікти). Хоча LDA було імплементовано для аналізу тем, у фінальну модель класифікації ці ознаки не увійшли з причин, описаних у розділі 4.

Для формалізації лінгвістичних маркерів, визначених у теоретичній частині, реалізовано наступні програмні механізми:

- Type-Token Ratio (TTR): розраховується як відношення кількості унікальних лем до загальної кількості слів у повідомленні, що дозволяє кількісно оцінити лексичну складність тексту [4];

- Аналіз займенників («Ми/Вони»): реалізовано через механізм Broadcast-змінних у Spark [3]. Це дозволяє ефективно поширювати словники маркерних слів на всі вузли кластера, розраховуючи частоту вживання «колективних» займенників («ми», «наші») порівняно з «ворожими» («вони», «їх»), що є ознакою поляризації суспільства [1], [5];
- Коефіцієнт суб'єктивності: обчислюється на основі нормалізованої щільності прикметників та прислівників у повідомленні, отриманих під час POS-тегування бібліотекою SpaCy [8].

Детекція координованої поведінки здійснюється за допомогою алгоритму MinHashLSH із пакету Spark ML [3]. На відміну від прямого порівняння текстів, цей метод дозволяє виконувати «м’який» пошук дублікатів серед мільйонів повідомлень. Це є технічним рішенням задачі ідентифікації «координованої неавтентичної поведінки» [5], оскільки дозволяє автоматично групувати канали, що синхронно розповсюджують ідентичні або злегка змінені пропагандистські матеріали.

Вибраний стек програмних засобів дозволяє не лише отримати глибоку аналітику за кожним окремим повідомленням, а й сформувати цілісну картину інформаційного впливу на рівні цілих каналів, об'єднуючи семантичні, тональні та мережеві ознаки в єдину аналітичну модель.

3.3 Побудова моделі класифікації

Заключним етапом розробки системи є побудова прогностичної моделі, здатної класифікувати повідомлення за ознакою наявності пропаганди. Враховуючи високу розмірність отриманих ознак та складність зв’язків між лінгвістичними показниками, реалізація була виконана за допомогою компонентів бібліотеки Spark MLlib [3].

Для забезпечення сумісності даних з алгоритмами машинного навчання Spark, було проведено агрегацію всіх обчислених на попередніх етапах ознак у єдиний вектор за допомогою трансформера VectorAssembler. Фінальний вектор all_features об’єднав:

- семантичні ознаки: вектор розподілу тем LDA та індекс домінуючої теми;
- тональні ознаки: результати класифікації моделі XLM-RoBERTa [12];
- стиліметричні показники: лексичне розмаїття (TTR) та мережевий показник дублікатів (MinHashLSH);
- нормалізовані психолінгвістичні метрики: частота вживання займенників «Ми/Вони» та рівень суб'єктивності, розраховані із застосуванням коефіцієнта згладжування SMOOTHING=10.0 для мінімізації впливу аномалій у коротких текстах.

Враховуючи різницю в обсягах між розміченим набором пропаганди та змішаним масивом даних, було сформовано збалансовану вибірку, що запобігає зміщенню моделі в бік мажоритарного класу [3]. Оскільки один із датасетів містить нерозмічену інформацію (unlabeled), завдання було сформульовано як ідентифікація специфічних «маркерів» пропаганди на фоні загального новинного потоку.

Як базовий класифікатор було обрано Random Forest Classifier. Вибір ансамблевого методу на основі дерев рішень обґрунтований його стійкістю до перенавчання та здатністю ефективно працювати з нелінійними залежностями в текстових ознаках.

Для пошуку найкращих гіперпараметрів було застосовано метод крос-валідації CrossValidator на 3 фолдах. Процес оптимізації включав перебір параметрів глибини дерев та стратегій вибору підмножин ознак.

Оцінка ефективності моделі здійснювалася за допомогою метрик AUC-ROC, F1 та Accuracy.

Для проведення порівняльного аналізу та вибору найбільш ефективного класифікатора було імплементовано алгоритм Gradient Boosted Trees (GBT). Вибір GBT обумовлений його здатністю до послідовної мінімізації помилок прогнозування через градієнтний спуск, що часто дозволяє досягти вищої точності на складних наборах даних.

4 ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ

4.1 Оцінка результатів

У ході експериментального дослідження було проведено оцінку розробленої системи на базі алгоритму Random Forest. Отримані метрики якості та аналіз значущості ознак дозволяють зробити висновки про ефективність обраного підходу для детекції пропаганди в умовах специфічного контенту месенджера Telegram.

У процесі розробки було прийнято рішення виключити результати тематичного моделювання (LDA) з фінального вектора ознак. Попередні тести показали, що LDA-теми часто базуються на конкретних наборах слів, які прямо корелюють з мовою публікації. Це призводило до «мовного перекосу», де модель ідеально розділяла дані за лінгвістичною ознакою, а не за наявністю пропагандистських технік. Поточний набір ознак дозволив інтегрувати російськомовні канали з нейтральним контентом у загальний масив, забезпечивши об’єктивність аналізу.

Для оцінки моделей класифікації на рівні окремих повідомлень було використано стандартний набір метрик бінарної класифікації (таблиця 4.1).

Таблиця 4.1 – Значення метрик класифікації

Модель	**AUC-ROC**	**Accuracy**	**F1-Score**
RandomForestClassifier	0.7104	0.6453	0.6365
GBTClassifier	0.6450	0.6017	0.5947

Модель Random Forest продемонструвала вищу ефективність за всіма показниками. Отримане значення AUC-ROC свідчить про стабільну здатність моделі розрізняти класи. На відміну від класичних досліджень пропаганди у довгих статтях чи політичних промовах, повідомлення в месенджерах характеризуються екстремальною стислістю, відсутністю складних синтаксичних конструкцій та великою кількістю неформальної лексики. Це суттєво обмежує кількість ознак, доступних для аналізу в межах одного об’єкта,

що робить результат AUC > 0.7 вагомим показником для даної предметної області.

Нижчий результат GBT пояснюється специфікою набору даних та методикою PU Learning. Оскільки нерозмічений масив даних містить певний відсоток прихованої пропаганди, алгоритм GBT, схильний до більш агресивного навчання на помилках, почав підлаштовуватися під цей шум. У свою чергу, Random Forest, завдяки механізму усереднення, виявився більш стійким до зашумленої розмітки негативного класу, що зробило його оптимальним вибором для даної системи.

Також отримано матрицю помилок Random Forest (рисунок 4.1).

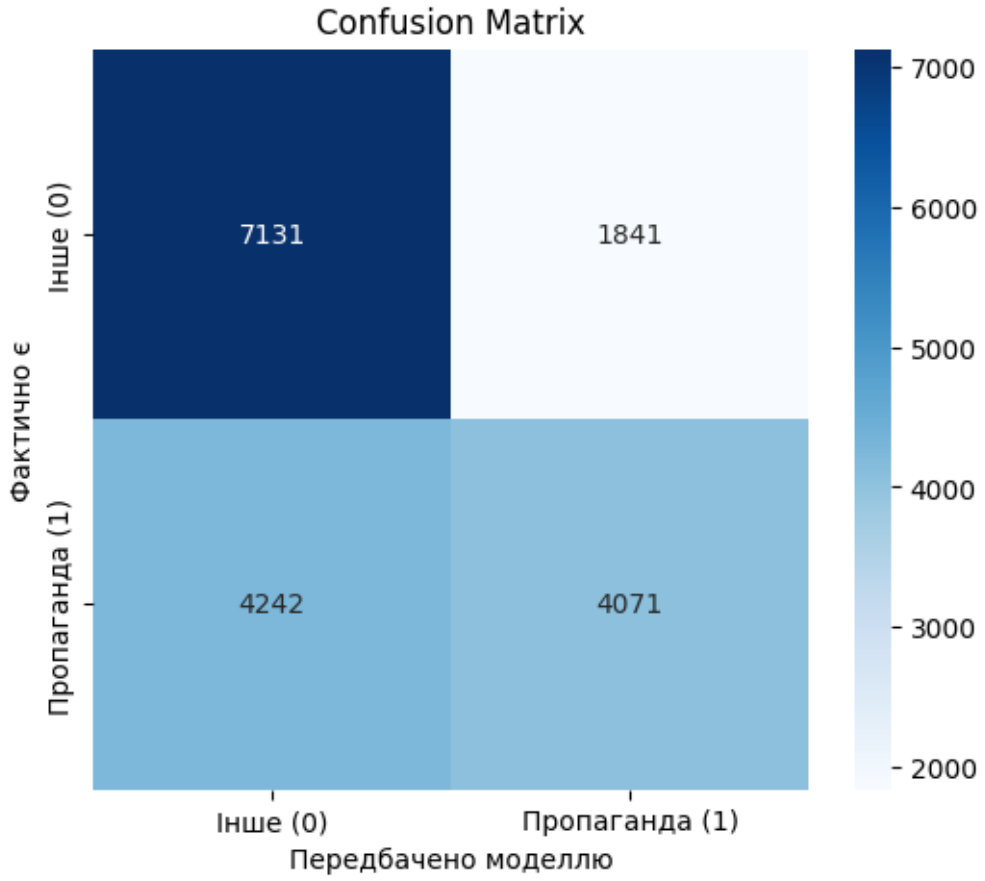

Рисунок 4.1 – Матриця помилок для Random Forest

Модель пропускає значну частину пропагандистських повідомлень, але кількість помилкових звинувачень є малою. Для систем моніторингу це є позитивним фактором, оскільки знижує ризик хибної цензури нейтральних новинних ресурсів.

Матриця помилок GBT відображена на рисунку 4.2. Загально вона відображає тенденцію, подібну до Random Forest, але з гіршими результатами.

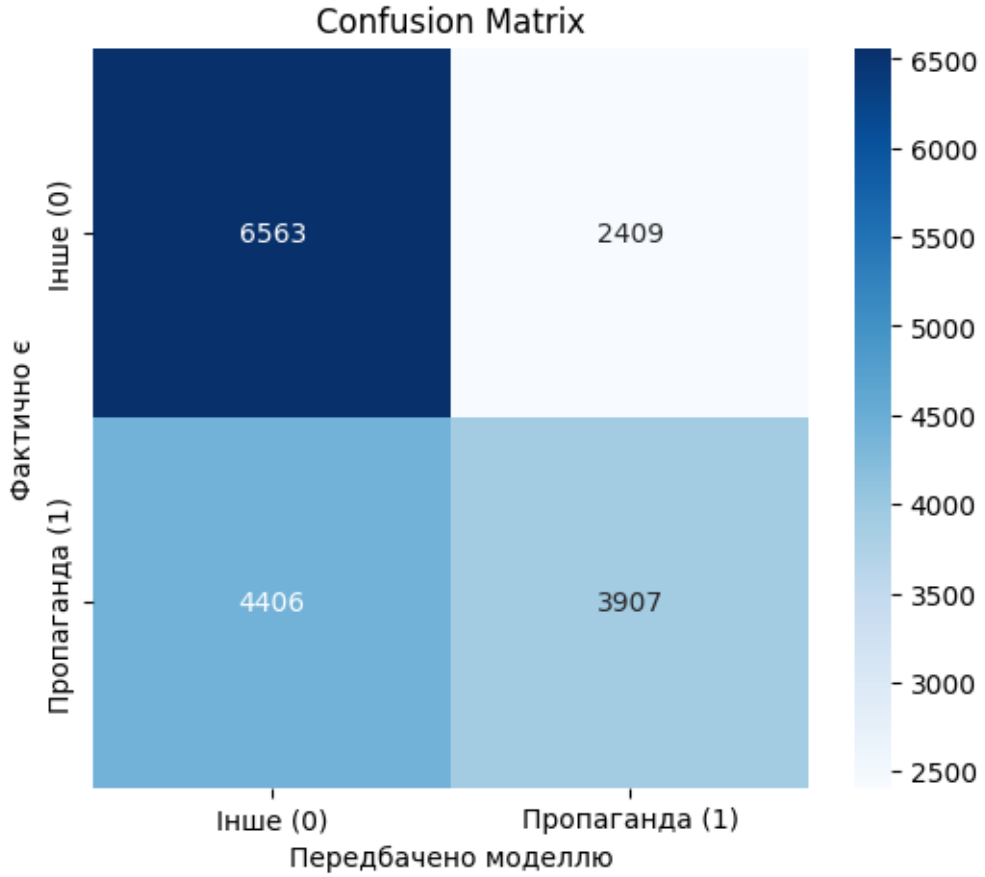

Рисунок 4.2 – Матриця помилок для GBT

Результати аналізу важливості ознак моделі Random Forest зазначено у таблиці 4.2.

Таблиця 4.2 – Важливість ознак

Назва ознаки	**% Впливу**
Type-Token Ratio	21.4%
Частота вживання займенника «ми»	19.1%
Частота вживання займенника «вони»	19.7%
Суб’єктивність	**25.7%**
Кількість дублікатів	7.5%
Результат аналізу тональності	6.6%

Найвищий вплив ознаки суб’єктивності (25.7%) та показників «Ми/Вони» (разом ~39%) доводить, що модель успішно навчилася ідентифікувати психолінгвістичні патерни, характерні для інформаційних операцій. Високе значення важливості Type-Token Ratio вказує на схильність пропагандистських ресурсів до використання повторюваних кліше та лозунгів, що призводить до збіднення лексичного складу повідомлень.

На рисунках 4.3-4.8 відображені розподіли знайдених ознак у формі гістограм для даних з двох датасетів.

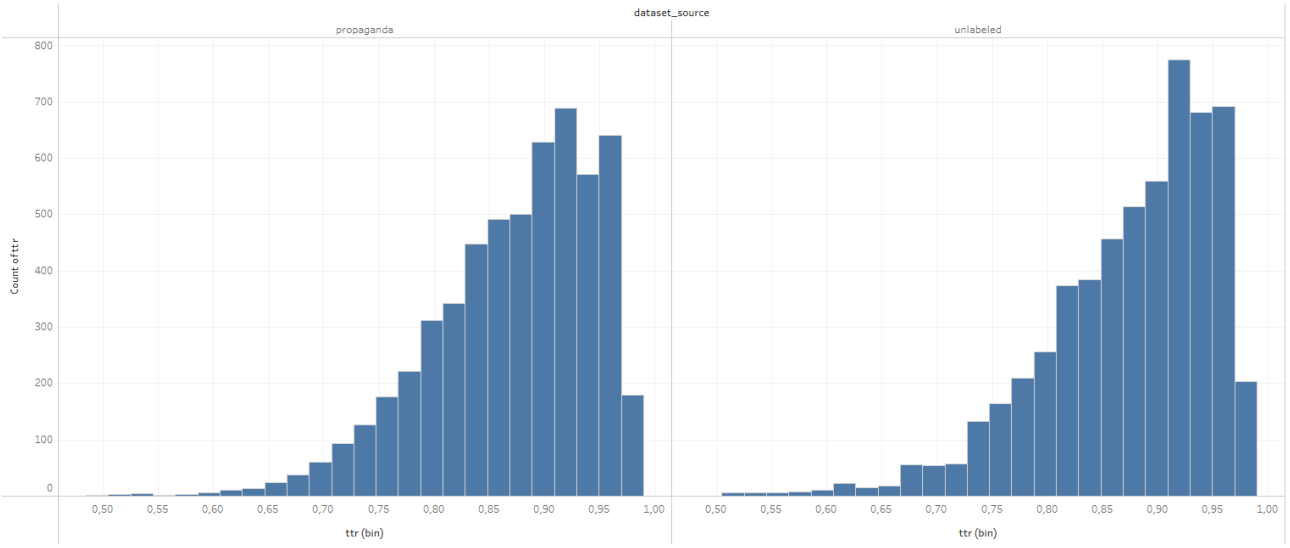

Рисунок 4.3 – Розподіл TTR

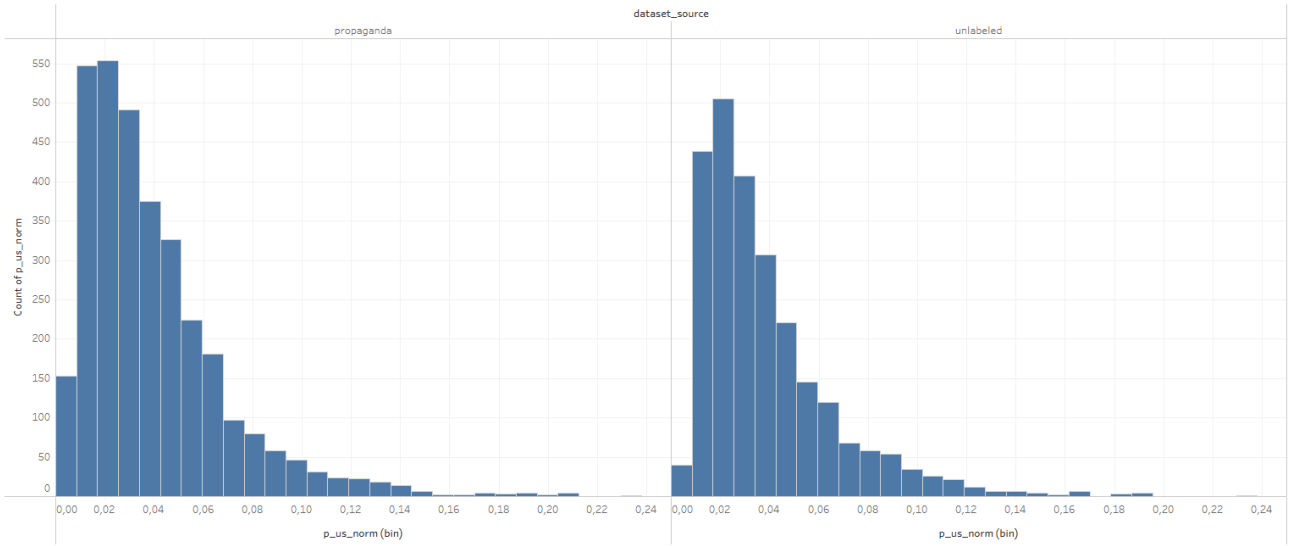

Рисунок 4.4 – Розподіл частоти вживання займенника «ми»

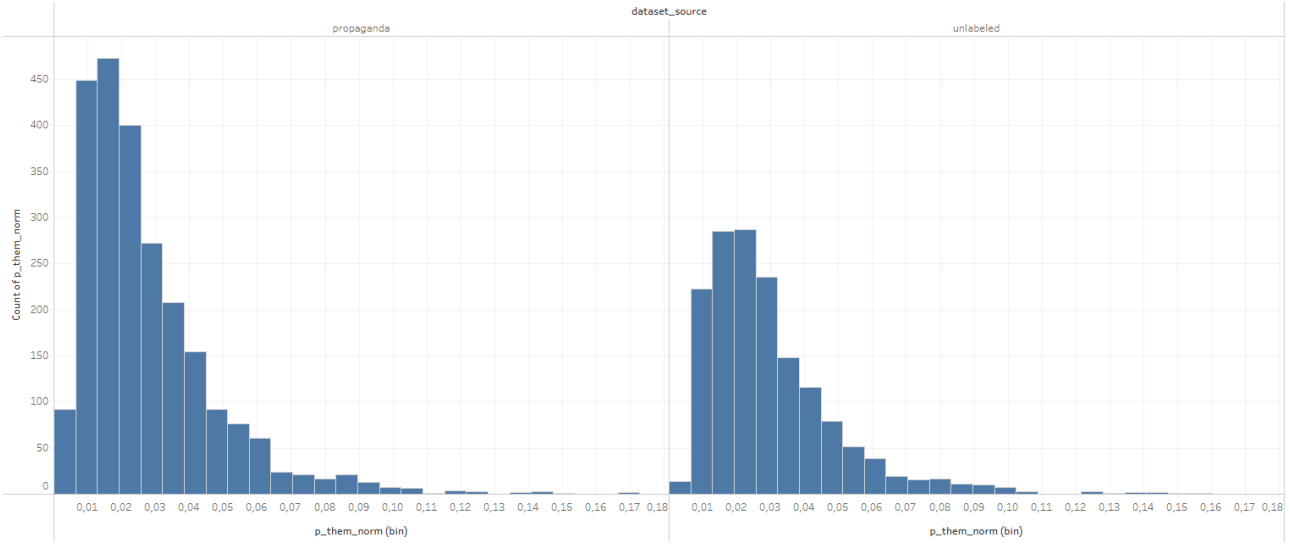

Рисунок 4.5 – Розподіл частоти вживання займенника «вони»

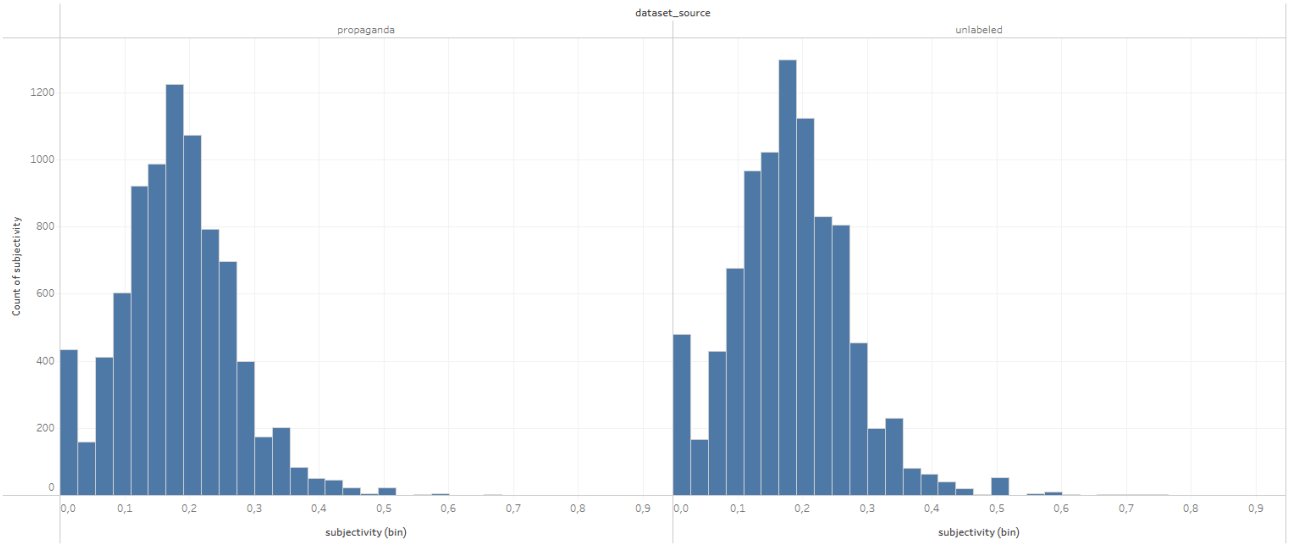

Рисунок 4.6 – Розподіл суб’єктивності

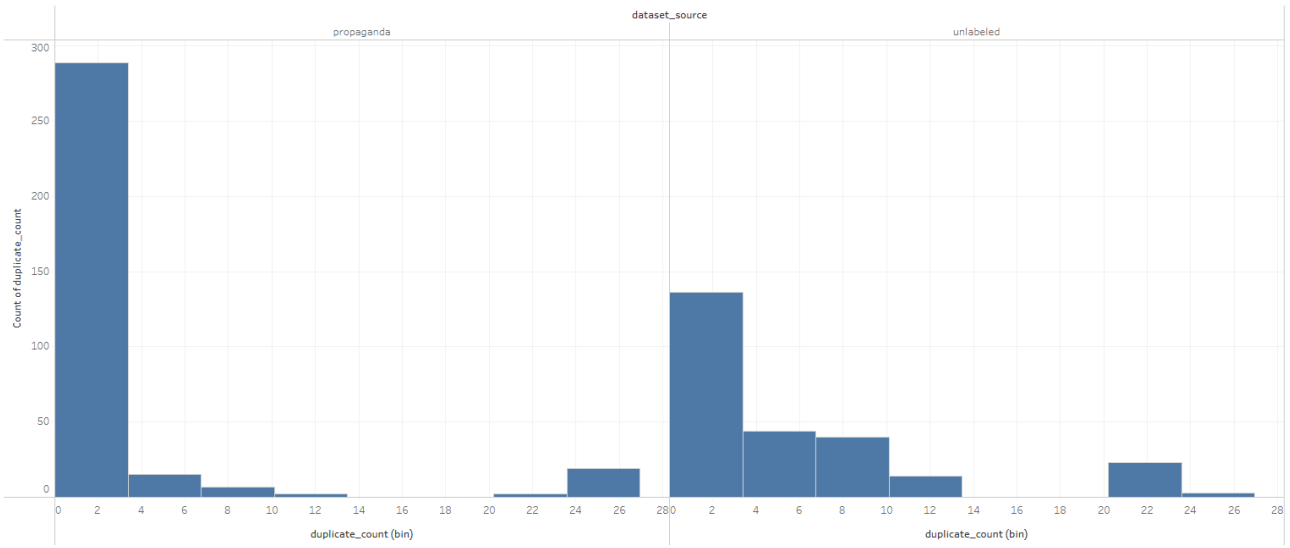

Рисунок 4.7 – Розподіл кількості дублікатів

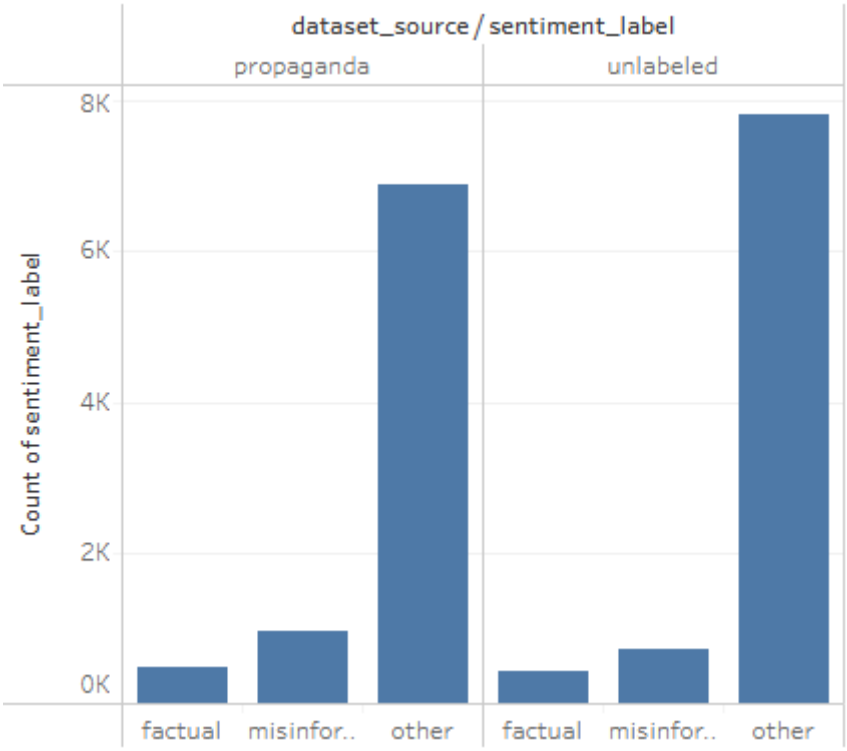

Рисунок 4.8 – Розподіл тональності

Найпомітніша різниця на гістограмах вживання займенників та значенню TTR, що підтверджує отримані важливості ознак.

Також для частоти вживання займенників наочною є діаграма середнього значення на рисунку 4.9.

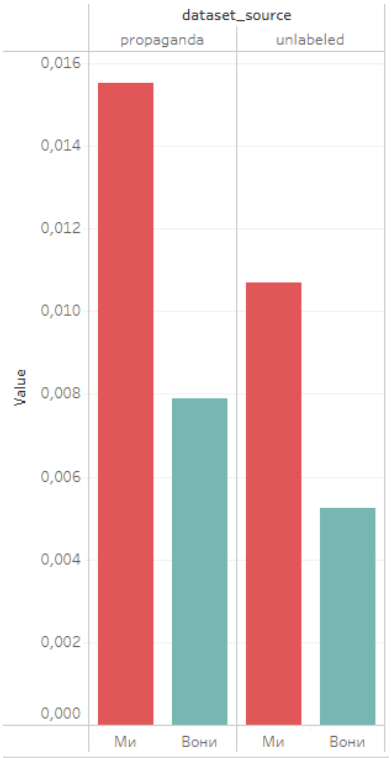

Рисунок 4.9 – Середнє значення частки використання займенників

Незважаючи на помірні метрики на рівні окремих постів, агрегований аналіз діяльності каналів продемонстрував високу ефективність системи. За результатами розрахунку відсотку публікацій з пропагандою, пропагандистські джерела та нейтральні канали були розділені майже ідеально. Лише в одному випадку показник нерозміченого каналу перевищив значення пропагандистського, що можна пояснити або специфікою контенту конкретного медіа, або наявністю в ньому прихованих маніпулятивних технік, які модель ідентифікувала як позитивні. Фрагмент агрегованих даних наведений у таблиці 4.3.

Таблиця 4.3 – Фрагмент результату агрегації по каналам

Канал	Тип	Частка пропаганди	Мова
tgnews_ua	unlabeled	0.1915	uk
truexanewsua	unlabeled	0.2341	ru
vanek_nikolaev	unlabeled	0.2384	ru
verkhovnaradaukrainy	unlabeled	0.2619	uk
censor_net	unlabeled	0.2692	uk
Военный обозреватель	propaganda	0.2697	ru
uniannet	unlabeled	0.2847	ru
Заря | Новости, Россия, Украина, Мир	propaganda	0.2869	ru
Directorate 4	propaganda	0.2910	ru
КОНФЛИКТ | ХАРЬКОВ ВОЛЧАНСК	propaganda	0.3187	ru
Вестник Дамаска	propaganda	0.3194	ru
Avia.pro	propaganda	0.3260	ru
Известия ЛНР/ДНР, Луганск/Донецк	propaganda	0.3333	ru

Хоча результати LDA не були використані для класифікації, розподіл тем відображений на рисунку 4.10.

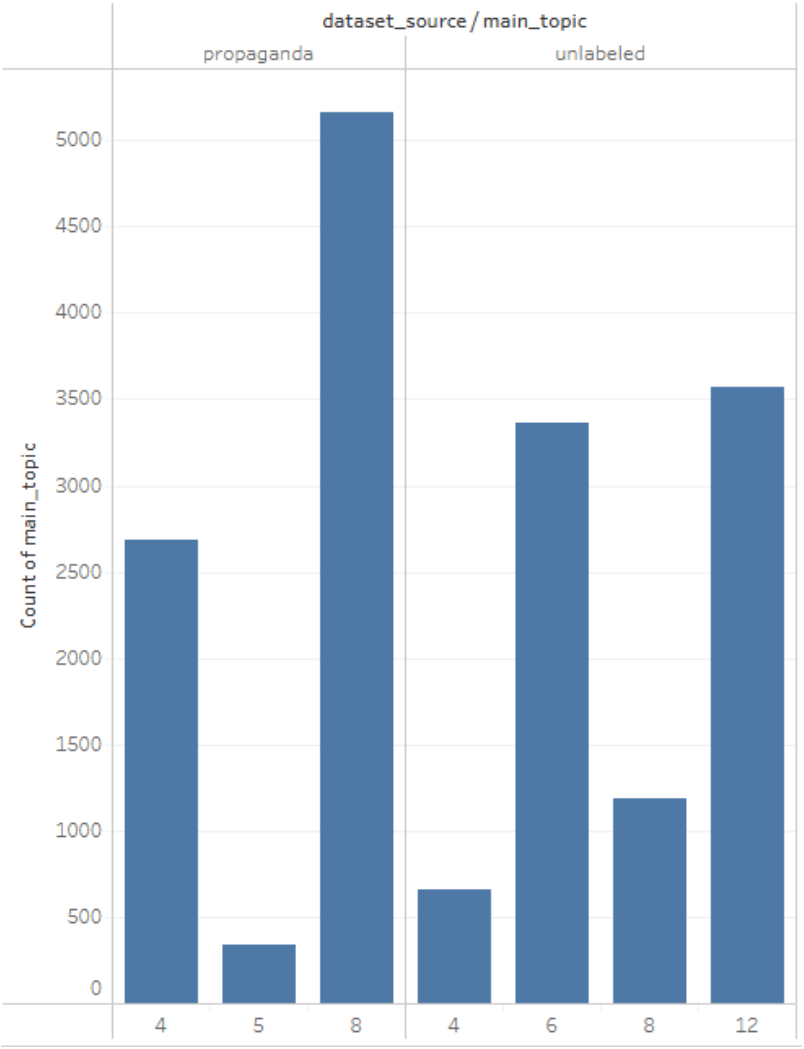

Рисунок 4.10 – Розподіл тем LDA

Набори слів для найбільш популярних тем:

4 – "всу", "область", "район", "противник", "удар", "направление", "ракета", "пункт", "войско", "дрон".

5 – "израиль", "украина", "военный", "иран", "заявить", "израильский", "учение", "газа", "сша", "министр".

8 – "россия", "украина", "год","человек","страна", "подписаться", "новый", "военный", "день", "российский".

6 – "україна", "росія", "facebook", "український", "країна", "президент", "сша", "російський", "підписатися", "війна".

12 – "україна", "російський", "область", "підписатися", "сила", "військовий", "український", "людина", "повідомити", "окупант".

4.2 Рекомендації щодо вдосконалення системи

На основі отриманих результатів та виявлених обмежень під час класифікації коротких повідомлень Telegram, можна сформувати перелік рекомендацій для подальшого вдосконалення системи. Ці пропозиції охоплюють як етап підготовки даних, так і перехід до більш складних архітектур машинного навчання.

Ключовим фактором, що обмежує точність моделі, є якість вхідного датасету. Для покращення результатів рекомендується залучення фахівців з моніторингу інформаційних операцій для верифікації навчальних даних, що дозволить зменшити рівень шуму, характерний для автоматично зібраних масивів. Рекомендується формування вибірки таким чином, щоб українська та російська мови були рівномірно репрезентовані як у класі пропаганди, так і в класі нейтральних новин, адже зараз розподіл мови не є оптимальним (рисунок 4.11, де жовтим кольором – українська мова, синім – російська). Це дозволить моделі ще ефективніше ігнорувати лінгвістичну приналежність тексту, фокусуючись виключно на маніпулятивних патернах. Також потрібно здійснити жорстку фіксацію кількості постів від кожного каналу для запобігання ситуації, коли специфічний стиль одного дуже активного джерела диктує ознаки для всього класу.

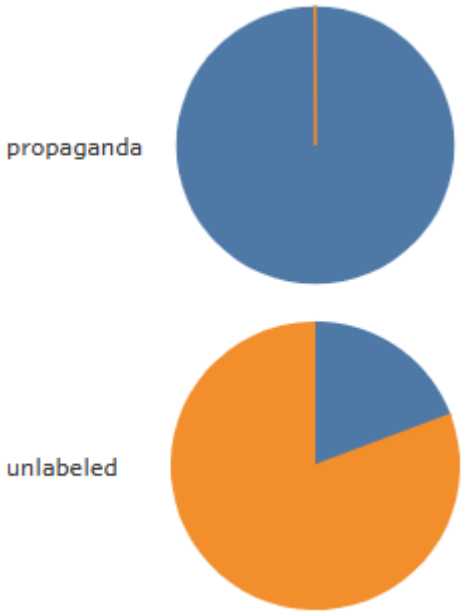

Рисунок 4.11 – Розподіл мов у вибірці даних

Для вдосконалення системи пропонується перехід до SOTA-архітектур – Fine-tuning Transformers. Сучасним стандартом у задачах NLP є відмова від ручної екстракції ознак на користь наскрізного навчання. Замість використання результатів трансформера як однієї з ознак, доцільно застосувати метод донавчання ваг всієї моделі (наприклад, XLM-RoBERTa або mBERT) безпосередньо на специфічному корпусі Telegram-пропаганди. Такий підхід дозволить моделі самостійно виділяти приховані семантичні зв'язки, які важко формалізувати, що особливо актуально для коротких повідомлень, де кожен токен має критичне значення.

Враховуючи швидку еволюцію пропагандистських наративів, система потребує механізмів адаптації: автоматизоване поповнення словників «завантаженої лексики» на основі аналізу нових виявлених аномалій, використання Spark Streaming для ідентифікації координованої неавтентичної поведінки у реальному часі, що дозволить блокувати поширення маніпуляцій на етапі їх зародження.

Реалізація цих рекомендацій дозволить трансформувати розроблене рішення з дослідницького прототипу у високонадійну систему моніторингу інформаційної безпеки, здатну працювати в динамічному середовищі сучасних соціальних медіа.

ВИСНОВОК

У ході виконання курсової роботи було розроблено масштабовану систему для автоматизованого виявлення пропагандистського контенту в месенджері Telegram. Результати роботи дозволяють зробити наступні висновки:

На аналітичному етапі було підтверджено актуальність проблеми ідентифікації інформаційних операцій у Telegram-сегменті. Встановлено, що ключовими труднощами є екстремальна стислість повідомлень, їх двомовність (UA/RU) та координація мереж каналів. Було визначено основні лінгвістичні маркери пропаганди: дихотомія «Ми/Вони», надмірна суб'єктивність та знижене лексичне розмаїття.

На етапі проєктування та реалізації було побудовано ETL-конвеєр, використання бібліотек SpaCy та langdetect дозволило реалізувати точну лінгвістичну обробку для української та російської мов, а формат Parquet забезпечив високу швидкість доступу до даних на етапі екстракції ознак.

На етапі експериментального дослідження було розроблено систему багатовимірного аналізу, що включає: розрахунок Type-Token Ratio та коефіцієнта суб'єктивності, підрахунок частоти вживання колективних та «ворожих» займенників за допомогою розширених словників, виявлення координованої поведінки через алгоритм MinHashLSH, визначення тональності за допомогою моделі XLM-RoBERTa.

У процесі класифікації було застосовано модель Random Forest із методикою Positive-Unlabeled Learning. Модель продемонструвала якісні кількісні показники на рівні окремих постів: AUC-ROC = 0.7104, Accuracy = 0.6453 та F1-Score = 0.6365. Встановлено, що для специфіки коротких повідомлень Telegram та умов відсутності повної розмітки, модель Random Forest є більш надійною порівняно з Gradient Boosted Trees, оскільки демонструє вищу здатність до генералізації на зашумлених даних. Найбільш значущими ознаками виявилися суб’єктивність (25.7%) та використання займенників (разом ~39%), що підтверджує раціональність обраного набору параметрів.

На етапі аналізу результатів встановлено, що агрегований аналіз на рівні Telegram-каналів є ефективнішим за класифікацію поодиноких повідомлень. Застосування накопичувальної статистики дозволило майже ідеально розділити апріорно пропагандистські джерела від нейтральних новинних ресурсів. Виявлені «аномалії» вказують на потенційно приховані маніпулятивні кампанії в нерозміченому сегменті, що потребує подальшого вивчення.

Мета роботи досягнута. Створена система демонструє стійкість до мовного зміщення та здатна ефективно працювати в умовах реальних масивів даних Big Data. Основними напрямками подальшого розвитку є розширення навчальних датасетів для кращого балансування мов та перехід до методів fine-tuning сучасних трансформерних архітектур (SOTA).

ПЕРЕЛІК ПОСИЛАНЬ

1. Telegram as a tool for information operations / NATO Strategic Communications Centre of Excellence. Riga, 2023. 45 p.
2. Digital Forensic Research Lab / Atlantic Council. URL: https://www.atlanticcouncil.org/programs/digital-forensic-research-lab/ (дата звернення: 19.12.2025).
3. Apache Spark Documentation / Apache Software Foundation. URL: https://spark.apache.org/docs/latest/ (дата звернення: 19.12.2025).
4. Vanetik N., Litvak M., Reviakin E. Propaganda Detection in Russian Telegram Posts in the Scope of the Russian Invasion of Ukraine. *RANLP Proceedings*. 2023. P. 1162–1170.
5. Kireev K., Mykhno Y., Overdorf R. Characterizing and Detecting Propaganda-Spreading Accounts on Telegram. *USENIX Security Symposium*. 2025.
6. XLM-RoBERTa / Facebook AI Community. URL: https://huggingface.co/docs/transformers/model_doc/xlm-roberta (дата звернення: 19.12.2025).
7. Pandas documentation / Pandas Development Team. URL: https://pandas.pydata.org/docs/ (дата звернення: 19.12.2025).
8. SpaCy: Industrial-strength Natural Language Processing in Python / Explosion AI. URL: https://spacy.io/usage (дата звернення: 19.12.2025).
9. Langdetect library / Mimino666. URL: https://pypi.org/project/langdetect/ (дата звернення: 19.12.2025).
10. Emoji library documentation / Carpedm20. URL: https://pypi.org/project/emoji/ (дата звернення: 19.12.2025).
11. Spark NLP / John Snow Labs. URL: https://sparknlp.org/ (дата звернення: 19.12.2025).
12. Multilingual XlmRobertaForSequenceClassification Cased model / John Snow Labs. URL:

https://sparknlp.org/2024/09/03/xlmroberta_classifier_verdict_xx.html (дата звернення: 19.12.2025).

ДОДАТКИ

Додаток А. Лістинг програмного коду

```
"""### Лематизація за допомогою SpaCy"""
!pip install -q spacy langdetect emoji
!python -m spacy download uk_core_news_sm
!python -m spacy download ru_core_news_sm

import spacy
import re
from langdetect import detect, LangDetectException
import emoji
import pandas as pd

nlp_uk = spacy.load('uk_core_news_sm')
nlp_ru = spacy.load('ru_core_news_sm')
stop_words = nlp_uk.Defaults.stop_words.union(nlp_ru.Defaults.stop_words)

"""Завантаження та імпорт файлів "telegram_messages_1million_november.json" та
"ru_dataset (for propaganda analysis).csv"
"""

!gdown 1NkgwCtfRs0Cawcrgnpw5gVCw9Zn1t24L

!gdown 1m1sKz6PJ-aTofB9YxzGJpaoOyZyIpob3

df_large_pd = pd.read_json("telegram_messages_1million_november.json")
df_large_pd = df_large_pd[['channel', 'date',
'text']].rename(columns={'channel': 'channel_name'})
df_large_pd['dataset_source'] = 'unlabeled'

df_propaganda_pd = pd.read_csv("ru_dataset (for propaganda analysis).csv")
df_propaganda_pd = df_propaganda_pd[['ChannelName', 'Date', 'PostText']].rename(
    columns={'ChannelName': 'channel_name', 'Date': 'date', 'PostText': 'text'}
)
df_propaganda_pd['dataset_source'] = 'propaganda'

"""Формування збалансованої підвибірки для рівномірної репрезентації усіх
каналів з обох файлів"""

def get_balanced_subset(df, posts_per_channel=500):
    return df.groupby('channel_name', group_keys=False).apply(
        lambda x: x.sample(min(len(x), posts_per_channel), random_state=42)
    )

df_prop_sample = get_balanced_subset(df_propaganda_pd, posts_per_channel=250)
df_news_sample = get_balanced_subset(df_large_pd, posts_per_channel=800)
df_train_balanced = pd.concat([df_prop_sample, df_news_sample],
ignore_index=True)

df_train_balanced = df_train_balanced.sample(frac=1,
random_state=42).reset_index(drop=True)

"""Очищення та лематизація тексту, визначення показника суб'єктивності"""

re_url = re.compile(r'https?://\S+|www\.\S+')
re_username = re.compile(r'@\w+')
re_numbers = re.compile(r'\d+')

def deep_clean_text(text):
    if not isinstance(text, str) or len(text) < 5:
        return ''
```

```
    text = re_url.sub(' ', text)
    text = re_username.sub(' ', text)
    text = emoji.replace_emoji(text, replace=' ')
    text = re_numbers.sub(' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

def process_doc(doc, target_nlp):
    lemmas = []
    subj_count = 0
    total_tokens = 0
    for token in doc:
        if (token.is_alpha and
            not token.is_stop and
            len(token.text) > 2):
            total_tokens += 1
            if token.pos_ in ["ADJ", "ADV"]:
                subj_count += 1
            lemmas.append(token.lemma_.lower())

    subjectivity_score = subj_count / total_tokens if total_tokens > 0 else 0.0
    return lemmas, subjectivity_score

def identify_language(text):
    try:
        return detect(text)
    except LangDetectException:
        return 'unknown'

df_train_balanced['clean_text'] =
df_train_balanced['text'].astype(str).apply(deep_clean_text)

df_train_balanced['lang'] =
df_train_balanced['clean_text'].apply(identify_language)

df_uk = df_train_balanced[df_train_balanced['lang'] == 'uk'].copy()
df_ru = df_train_balanced[df_train_balanced['lang'] == 'ru'].copy()
df_other = df_train_balanced[~df_train_balanced['lang'].isin(['uk',
'ru'])].copy()

uk_results = []
ru_results = []

for doc in nlp_uk.pipe(df_uk['clean_text'], batch_size=2048, disable=["ner",
"parser"]):
    uk_results.append(process_doc(doc, nlp_uk))
df_uk['lemmas'] = [r[0] for r in uk_results]
df_uk['subjectivity'] = [r[1] for r in uk_results]

for doc in nlp_ru.pipe(df_ru['clean_text'], batch_size=2048, disable=["ner",
"parser"]):
    ru_results.append(process_doc(doc, nlp_ru))
df_ru['lemmas'] = [r[0] for r in ru_results]
df_ru['subjectivity'] = [r[1] for r in ru_results]

df_final = pd.concat([df_uk, df_ru], ignore_index=True)
df_final = df_final[df_final['lemmas'].apply(len) > 3]

df_final

"""### Формування Bag of Words для застосування LDA та MinHashLSH"""
```

```
df_final['corpus'] = df_final['lemmas'].apply(lambda x: ' '.join(x))

MAX_FEATURES = 15000
MIN_DF = 5

from sklearn.feature_extraction.text import CountVectorizer
vectorizer_bow = CountVectorizer(max_features=MAX_FEATURES, min_df=MIN_DF)
X_bow_sparse = vectorizer_bow.fit_transform(df_final['corpus'])
feature_names = vectorizer_bow.get_feature_names_out()

X_coo = X_bow_sparse.tocoo()
df_sparse_components = pd.DataFrame({
    'doc_index': X_coo.row,
    'feature_index': X_coo.col,
    'value': X_coo.data
})
df_final = df_final.reset_index().rename(columns={'index': 'doc_index'})
df_bow_final = df_sparse_components.merge(
    df_final[['doc_index', 'channel_name', 'dataset_source']],
    on='doc_index',
    how='left'
)

"""### Збереження на Google Drive"""

from google.colab import drive
drive.mount('/content/gdrive')

DRIVE_PATH = '/content/gdrive/MyDrive/Coursework_Data/'

import os
os.makedirs(DRIVE_PATH, exist_ok=True)

df_final['date_string'] = df_final['date'].astype(str)

FILE_NAME = "lemmatized_data_100k.parquet"
FULL_SAVE_PATH = DRIVE_PATH + FILE_NAME
df_final[['doc_index', 'channel_name', 'dataset_source', 'date_string', 'text',
'lemmas', 'corpus', "subjectivity", "lang"]].to_parquet(FULL_SAVE_PATH)

BOW_COMPONENTS_FILE = "bow_components.parquet"
FULL_BOW_PATH = DRIVE_PATH + BOW_COMPONENTS_FILE
df_bow_final.to_parquet(FULL_BOW_PATH, index=False)

FEATURE_NAMES_FILE = "lda_feature_names.txt"
FULL_NAMES_PATH = DRIVE_PATH + FEATURE_NAMES_FILE
with open(FULL_NAMES_PATH, 'w', encoding='utf-8') as f:
    f.write('\n'.join(feature_names))

"""### Встановлення Spark та Spark NLP"""

!wget http://setup.johnsnowlabs.com/colab.sh -O - | bash

"""### Запуск Spark у режимі CPU"""

import sparknlp
spark = sparknlp.start(params={"spark.ui.port": "4050"})

"""### Запуск Spark у режимі GPU"""

from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("Spark NLP GPU") \
```

```
    .master("local[*]") \
    .config("spark.driver.memory", "16G") \
    .config("spark.driver.maxResultSize", "0") \
    .config("spark.kryoserializer.buffer.max", "2000M") \
    .config("spark.jars.packages", f"com.johnsnowlabs.nlp:spark-nlp-
gpu_2.12:6.2.3") \
    .config("spark.ui.port", "4050") \
    .getOrCreate()

"""### Запуск ngrok для Spark UI та Google Drive"""

!pip install pyngrok

from pyngrok import ngrok, conf
conf.get_default().auth_token =
"36t9RZigTxla0o8Hpib79iHsrgS_oj4FrcHZnBggrvMFKm4k"
public_url = ngrok.connect(4050).public_url
print(f"Spark UI is accessible at: {public_url}")

from sparknlp.base import *
from sparknlp.annotator import *
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_timestamp, lit, when
import pyspark.sql.functions as F
from pyspark.ml import Pipeline
from pyspark.ml.clustering import LDA, KMeans
from pyspark.sql.types import StructType, StructField, StringType, ArrayType,
IntegerType, DoubleType
import json
from sparknlp.pretrained import PretrainedPipeline
from pyspark.ml.feature import VectorAssembler, StandardScaler, MinHashLSH,
StringIndexer, VectorIndexer
from pyspark.ml.evaluation import ClusteringEvaluator,
BinaryClassificationEvaluator, MulticlassClassificationEvaluator
from pyspark.sql.window import Window
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.classification import RandomForestClassifier, GBTClassifier

from google.colab import drive
drive.mount('/content/gdrive')

DRIVE_PATH = '/content/gdrive/MyDrive/Coursework_Data/'

"""### Сентиментальний аналіз за допомогою xlmroberta_classifier_verdict"""

FILE_NAME = "lemmatized_data_100k.parquet"
FULL_SAVE_PATH = DRIVE_PATH + FILE_NAME

df_clean = spark.read.parquet(FULL_SAVE_PATH)

documentAssembler = DocumentAssembler() \
    .setInputCol("text") \
    .setOutputCol("document")

tokenizer = Tokenizer() \
    .setInputCols("document") \
    .setOutputCol("token")

seq_classifier =
XlmRoBertaForSequenceClassification.pretrained("xlmroberta_classifier_verdict","
xx") \
    .setInputCols(["document", "token"]) \
    .setOutputCol("class")
```

```
finisher = Finisher() \
    .setInputCols(["class"]) \
    .setOutputCols(["sentiment_result"])

sentimental_pipeline = Pipeline(stages=[documentAssembler, tokenizer,
seq_classifier, finisher])

model_sentimental = sentimental_pipeline.fit(df_clean)
df_sentimental = model_sentimental.transform(df_clean)
df_sentimental = df_sentimental.withColumn("sentiment_label",
F.element_at(col("sentiment_result"), 1))

df_final = df_sentimental.withColumn("sentiment_label",
F.element_at(col("sentiment_result"), 1))
cols_to_save = ['doc_index', 'channel_name', 'dataset_source', 'date_string',
'text', 'lemmas', 'corpus', "subjectivity", "lang", 'sentiment_label',
'sentiment_result']
SENTIMENT_RESULTS_FILE = DRIVE_PATH + "sentiment_results_100k.parquet"
df_final.select(*cols_to_save).write.mode("overwrite").parquet(SENTIMENT_RESULTS
_FILE)

"""### Формування тем за допомогою LDA"""

DRIVE_PATH = '/content/gdrive/MyDrive/Coursework_Data/'
SENTIMENT_RESULTS_FILE = DRIVE_PATH + "sentiment_results_100k.parquet"
BOW_COMPONENTS_FILE = DRIVE_PATH + "bow_components.parquet"
FEATURE_NAMES_FILE = DRIVE_PATH + "lda_feature_names.txt"
LDA_TOPICS_FILE = DRIVE_PATH + "lda_topic_keywords.json"
LDA_RESULTS_FILE = DRIVE_PATH + "lda_results_100k.parquet"

with open(FEATURE_NAMES_FILE, 'r', encoding='utf-8') as f:
    feature_names = [line.strip() for line in f]

vector_size = len(feature_names)

"""Розпакування векторів Bag of Words"""

df_bow_components = spark.read.parquet(BOW_COMPONENTS_FILE)

df_paired = df_bow_components.withColumn(
    "pair",
    F.struct(F.col("feature_index"), F.col("value"))
)

df_grouped_pairs = df_paired.groupBy("doc_index").agg(
    F.collect_list("pair").alias("pairs")
)

df_sorted_pairs = df_grouped_pairs.withColumn(
    "sorted_pairs",
    F.array_sort(F.col("pairs"))
)

df_grouped_final = df_sorted_pairs.withColumn(
    "indices",
    F.col("sorted_pairs.feature_index")
).withColumn(
    "values",
    F.col("sorted_pairs.value")
).select("doc_index", "indices", "values")

from pyspark.ml.linalg import VectorUDT, Vectors
```

```
@F.udf(VectorUDT())
def create_sparse_vector(indices, values):
    if not indices:
        return Vectors.sparse(vector_size, [], [])

    return Vectors.sparse(vector_size, indices, values)

df_features_only = df_grouped_final.withColumn(
    "features",
    create_sparse_vector(F.col("indices"), F.col("values"))
).select("doc_index", "features")

df_sentiment = spark.read.parquet(SENTIMENT_RESULTS_FILE)

df_analysis = df_features_only.join(
    df_sentiment,
    on="doc_index",
    how="inner"
)

"""Запуск алгоритму LDA"""

num_topics = 20
lda = LDA(k=num_topics, maxIter=20, seed=42, featuresCol="features")
lda_model = lda.fit(df_analysis)
df_with_topics = lda_model.transform(df_analysis)

get_max_topic = F.udf(lambda v: int(v.argmax()), IntegerType())
df_final_analysis = df_with_topics.withColumn("main_topic",
get_max_topic(F.col("topicDistribution")))

"""Збереження результатів запуску алгоритму"""

topics = lda_model.describeTopics(10)
topics_data = []

for row in topics.collect():
    topic_id = row.topic
    terms = [feature_names[idx] for idx in row.termIndices]

    topics_data.append({
        "topic_id": topic_id,
        "keywords": terms,
        "weights": row.termWeights
    })
    print(f"Topic {topic_id}: {terms}")

with open(LDA_TOPICS_FILE, 'w', encoding='utf-8') as f:
    json.dump(topics_data, f, ensure_ascii=False, indent=4)

df_final_analysis.write.mode("overwrite").parquet(LDA_RESULTS_FILE)

"""### Обчислення значень TTR та кількості займенників"""

LDA_RESULTS_FILE = DRIVE_PATH + "lda_results_100k.parquet"
df_analysis = spark.read.parquet(LDA_RESULTS_FILE)

"""TTR"""

df_analysis = df_analysis.withColumn(
    "ttr",
    F.when(F.size(F.col("lemmas")) > 0,
           F.size(F.array_distinct(F.col("lemmas"))) / F.size(F.col("lemmas"))
```

```
          ).otherwise(0.0)
)

"""We/They count"""

group_us_lemmas = {
    # Українська (UA)
    'я', 'мене', 'мені', 'мною', 'мій', 'мого', 'моєму', 'моїм', 'моя', 'моє',
'мої', 'моїх',
    'ми', 'нас', 'нам', 'нами', 'наш', 'нашого', 'нашому', 'нашим', 'наша',
'наше', 'наші', 'наших',
    'свій', 'свого', 'своєму', 'своїм', 'своя', 'своє', 'свої', 'своїх',

    # російська (RU)
    'я', 'меня', 'мне', 'мной', 'мною', 'мой', 'моего', 'моему', 'моим', 'моя',
'мое', 'моё', 'мои', 'моих',
    'мы', 'нас', 'нам', 'нами', 'наш', 'нашего', 'нашему', 'нашим', 'наша',
'наше', 'наши', 'наших',
    'свой', 'своего', 'своему', 'своим', 'своя', 'свое', 'своё', 'свои', 'своих'
}

group_them_lemmas = {
    # Українська (UA)
    'ти', 'тебе', 'тобі', 'тобою', 'твій', 'твого', 'твоєму', 'твоїм', 'твоя',
'твоє', 'твої', 'твоїх',
    'ви', 'вас', 'вам', 'вами', 'ваш', 'вашого', 'вашому', 'вашим', 'ваша',
'ваше', 'ваші', 'ваших',
    'вони', 'них', 'їх', 'їм', 'ними', 'їхній', 'їхнього', 'їхньому', 'їхнім',
'їхня', 'їхнє', 'їхні',

    # російська (RU)
    'ты', 'тебя', 'тебе', 'тобой', 'тобою', 'твой', 'твоего', 'твоему', 'твоим',
'твоя', 'твое', 'твоё', 'твои', 'твоих',
    'вы', 'вас', 'вам', 'вами', 'ваш', 'вашего', 'вашему', 'вашим', 'шаша',
'ваше', 'ваши', 'ваших',
    'они', 'них', 'их', 'им', 'ими', 'ихний', 'ихнего', 'ихнему', 'ихним',
'ихняя', 'ихнее'
}

bc_us = spark.sparkContext.broadcast(group_us_lemmas)
bc_them = spark.sparkContext.broadcast(group_them_lemmas)

def calc_pronouns(text):
    if not text: return (0, 0)

    words = re.findall(r'\w+', text.lower())
    total_words = len(words)

    if total_words == 0: return (0, 0)

    us_count = 0
    them_count = 0

    us_set = bc_us.value
    them_set = bc_them.value

    for w in words:
        if w in us_set:
            us_count += 1
        elif w in them_set:
            them_count += 1

    return (float(us_count), float(them_count))
```

```
pronouns_schema = StructType([
    StructField("us_rate", DoubleType(), False),
    StructField("them_rate", DoubleType(), False)
])
pronouns_udf = F.udf(calc_pronouns, pronouns_schema)

df_analysis = df_analysis.withColumn("lemma_count", F.size(F.col("lemmas")))

df_analysis = df_analysis.withColumn("pronouns_struct",
pronouns_udf(F.col("text"))) \
    .withColumn("pronoun_us", F.col("pronouns_struct.us_rate") /
F.col("lemma_count")) \
    .withColumn("pronoun_them", F.col("pronouns_struct.them_rate") /
F.col("lemma_count")) \
    .drop("pronouns_struct")

TTR_RESULTS_FILE = DRIVE_PATH + "ttr_results_100k.parquet"
df_analysis.write.mode("overwrite").parquet(TTR_RESULTS_FILE)

"""### (OUT OF RAM) Визначення частин мови за допомогою BERT моделей"""

pipeline_lang = PretrainedPipeline("detect_language_220", lang="xx")
df_analysis = pipeline_lang.transform(df_analysis)

df_analysis = df_analysis.withColumn("lang_code",
F.element_at(F.col("language.result"), 1))

df_uk = df_analysis.filter(F.col("lang_code") == "uk")
df_ru = df_analysis.filter(F.col("lang_code") == "ru")

def get_pos_pipeline(model_name, lang_code):
    document = DocumentAssembler().setInputCol("text").setOutputCol("document")

    # BERT вимагає токенів
    tokenizer = Tokenizer().setInputCols(["document"]).setOutputCol("token")

    # BERT POS Model
    # Використовуємо BertForTokenClassification для POS тегів
    bert_pos = BertForTokenClassification.pretrained(model_name, lang_code) \
        .setInputCols(["document", "token"]) \
        .setOutputCol("pos")

    finisher = Finisher().setInputCols(["pos"]).setOutputCols(["pos_tags"])

    return Pipeline(stages=[document, tokenizer, bert_pos, finisher])

pipeline_uk = get_pos_pipeline("bert_pos_bert_base_slavic_cyrillic_upos", "uk")
model_uk = pipeline_uk.fit(df_uk)
df_uk_tagged = model_uk.transform(df_uk)

pipeline_ru = get_pos_pipeline("bert_pos_bert_base_russian_upos", "ru")
model_ru = pipeline_ru.fit(df_ru)
df_ru_tagged = model_ru.transform(df_ru)

df_analysis = df_uk_tagged.unionByName(df_ru_tagged)

df_analysis.select("pos_tags").show(5, truncate=False)

def calc_subjectivity(tags):
    if not tags: return 0.0

    count = 0
```

```
    for tag in tags:
        if "ADJ" in tag or "ADV" in tag:
            count += 1

    return count / len(tags)

subjectivity_udf = F.udf(calc_subjectivity, DoubleType())

df_analysis = df_analysis.withColumn("subjectivity",
subjectivity_udf(F.col("pos_tags")))

ANALYSIS_RESULTS_FILE = DRIVE_PATH + "analysis_results_100k.parquet"
df_final_analysis.write.mode("overwrite").parquet(ANALYSIS_RESULTS_FILE)

"""### Пошук кількості дублікатів за допомогою MinHashLSH"""

TTR_RESULTS_FILE = DRIVE_PATH + "ttr_results_100k.parquet"
df_analysis = spark.read.parquet(TTR_RESULTS_FILE)

mh = MinHashLSH(inputCol="features", outputCol="hashes", numHashTables=5)

model_lsh = mh.fit(df_analysis)
df_hashed = model_lsh.transform(df_analysis)

threshold = 0.1
df_matches = model_lsh.approxSimilarityJoin(
    df_hashed, df_hashed, threshold, distCol="JaccardDistance"
)

duplicate_features =
df_matches.groupBy(col("datasetA.doc_index").alias("doc_index")) \
    .agg(F.count("*").alias("duplicate_count"))

df_with_cp_feature = df_analysis.join(duplicate_features, on="doc_index",
how="left")

df_with_cp_feature = df_with_cp_feature.fillna({"duplicate_count": 1})

DC_RESULTS_FILE = DRIVE_PATH + "duplicate_results_100k.parquet"
df_with_cp_feature.write.mode("overwrite").parquet(DC_RESULTS_FILE)

"""### Застосування моделі Random Forest для класифікації"""

DC_RESULTS_FILE = DRIVE_PATH + "duplicate_results_100k.parquet"

df_analysis = spark.read.parquet(DC_RESULTS_FILE)

SMOOTHING = 10.0

df_labeled = df_analysis.withColumn(
    "label",
    F.when(F.col("dataset_source") == "propaganda", 1.0).otherwise(0.0)
).withColumn(
    # Відновлюємо кількість (Rate * Count) і ділимо на згладжений знаменник
    "p_us_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("pronoun_us") * F.col("lemma_count")) / (F.col("lemma_count")
+ SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_them_norm",
    F.when(F.col("lemma_count") > 0,
```

```
            (F.col("pronoun_them") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_subj_norm",
    F.when(F.col("lemma_count") > 0,
            (F.col("subjectivity") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
)

prop_count = df_labeled.filter(F.col("label") == 1.0).count()
non_prop_count = df_labeled.filter(F.col("label") == 0.0).count()
fraction = prop_count / non_prop_count if non_prop_count > prop_count else 1.0
df_balanced = df_labeled.sampleBy("label", fractions={1.0: 1.0, 0.0: fraction},
seed=42)

"""Результати LDA було виключено для зменшення кореляції з мовою публікації"""

sentiment_indexer = StringIndexer(inputCol="sentiment_label",
outputCol="sentiment_index").setHandleInvalid("keep")
#topic_indexer = StringIndexer(inputCol="main_topic",
outputCol="topic_index").setHandleInvalid("keep")

assembler = VectorAssembler(
    inputCols=[
        "ttr",
        "p_us_norm", "p_them_norm", "p_subj_norm", "duplicate_count",
        "sentiment_index"#, "topic_index", "topicDistribution", "features"
    ],
    outputCol="all_features"
)

rf = RandomForestClassifier(
    labelCol="label",
    featuresCol="all_features",
    seed=42,
    numTrees=200,
    maxBins=64,
    subsamplingRate=0.8
)

pipeline = Pipeline(stages=[sentiment_indexer, assembler, rf]) # topic_indexer,

evaluator = BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="rawPrediction", metricName="areaUnderROC")

paramGrid = ParamGridBuilder() \
    .addGrid(rf.maxDepth, [5, 10]) \
    .addGrid(rf.featureSubsetStrategy, ["sqrt", "log2"]) \
    .build()

cv = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator,
    numFolds=3,
    seed=42
)

cvModel = cv.fit(df_balanced)

predictions = cvModel.transform(df_balanced)
```

```
auc = evaluator.evaluate(predictions)
print(f"AUC-ROC: {auc:.4f}")

multi_evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction")
accuracy = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName:
"accuracy"})
f1 = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: "f1"})

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")

best_rf = cvModel.bestModel.stages[-1]
importances = best_rf.featureImportances

df_full_prepared = df_analysis.withColumn(
    # Відновлюємо кількість (Rate * Count) і ділимо на згладжений знаменник
    "p_us_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("pronoun_us") * F.col("lemma_count")) / (F.col("lemma_count")
+ SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_them_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("pronoun_them") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_subj_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("subjectivity") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
)

df_scored = cvModel.transform(df_full_prepared)

OUTPUT_SCORED_FILE = DRIVE_PATH + "scored_results_100k.parquet"

df_scored.write.mode("overwrite").parquet(OUTPUT_SCORED_FILE)

"""### Застосування моделі GBTClassifier для класифікації"""

DC_RESULTS_FILE = DRIVE_PATH + "duplicate_results_100k.parquet"

df_analysis = spark.read.parquet(DC_RESULTS_FILE)

SMOOTHING = 10.0

df_labeled = df_analysis.withColumn(
    "label",
    F.when(F.col("dataset_source") == "propaganda", 1.0).otherwise(0.0)
).withColumn(
    # Відновлюємо кількість (Rate * Count) і ділимо на згладжений знаменник
    "p_us_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("pronoun_us") * F.col("lemma_count")) / (F.col("lemma_count")
+ SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_them_norm",
```

```
    F.when(F.col("lemma_count") > 0,
           (F.col("pronoun_them") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_subj_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("subjectivity") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
)

prop_count = df_labeled.filter(F.col("label") == 1.0).count()
non_prop_count = df_labeled.filter(F.col("label") == 0.0).count()
fraction = prop_count / non_prop_count if non_prop_count > prop_count else 1.0
df_balanced = df_labeled.sampleBy("label", fractions={1.0: 1.0, 0.0: fraction},
seed=42)

"""Результати LDA було виключено для зменшення кореляції з мовою публікації"""

sentiment_indexer = StringIndexer(inputCol="sentiment_label",
outputCol="sentiment_index").setHandleInvalid("keep")
#topic_indexer = StringIndexer(inputCol="main_topic",
outputCol="topic_index").setHandleInvalid("keep")

assembler = VectorAssembler(
    inputCols=[
        "ttr",
        "p_us_norm", "p_them_norm", "p_subj_norm", "duplicate_count",
        "sentiment_index"#, "topic_index", "topicDistribution", "features"
    ],
    outputCol="all_features"
)

gbt = GBTClassifier(
    labelCol="label",
    featuresCol="all_features",
    seed=42,
    subsamplingRate=0.7,
    maxBins=128
)

pipeline = Pipeline(stages=[sentiment_indexer, assembler, gbt]) # topic_indexer,

evaluator = BinaryClassificationEvaluator(labelCol="label",
rawPredictionCol="rawPrediction", metricName="areaUnderROC")

paramGrid = ParamGridBuilder() \
    .addGrid(gbt.maxDepth, [3, 5]) \
    .addGrid(gbt.maxIter, [50, 100]) \
    .addGrid(gbt.stepSize, [0.05, 0.1]) \
    .build()

cv = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator,
    numFolds=3,
    seed=42
)

cvModel = cv.fit(df_balanced)
```

```
predictions = cvModel.transform(df_balanced)

auc = evaluator.evaluate(predictions)
print(f"AUC-ROC: {auc:.4f}")

multi_evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction")
accuracy = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName:
"accuracy"})
f1 = multi_evaluator.evaluate(predictions, {multi_evaluator.metricName: "f1"})

print(f"Accuracy: {accuracy:.4f}")
print(f"F1-Score: {f1:.4f}")

best_gbt = cvModel.bestModel.stages[-1]
importances = best_gbt.featureImportances

importances

df_full_prepared = df_analysis.withColumn(
    # Відновлюємо кількість (Rate * Count) і ділимо на згладжений знаменник
    "p_us_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("pronoun_us") * F.col("lemma_count")) / (F.col("lemma_count")
+ SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_them_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("pronoun_them") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
).withColumn(
    "p_subj_norm",
    F.when(F.col("lemma_count") > 0,
           (F.col("subjectivity") * F.col("lemma_count")) /
(F.col("lemma_count") + SMOOTHING)
          ).otherwise(0.0)
)

df_scored = cvModel.transform(df_full_prepared)

OUTPUT_SCORED_FILE = DRIVE_PATH + "scored_results_100k.parquet"

df_scored.write.mode("overwrite").parquet(OUTPUT_SCORED_FILE)

"""### Збереження загальних та агрегованих результатів у CSV файл"""

extract_prob_udf = F.udf(lambda v: float(v[1]) if v else 0.0, DoubleType())

df_csv_detailed = df_scored.withColumn("prop_probability",
extract_prob_udf(F.col("probability"))) \
    .select(
        "doc_index",
        "channel_name",
        "dataset_source",
        "lang",
        "sentiment_label",
        "main_topic",
        "ttr",
        "pronoun_us",
        "pronoun_them",
        "subjectivity",
```

```
        "p_us_norm",
        "p_them_norm",
        "p_subj_norm",
        "lemma_count",
        "duplicate_count",
        F.col("prop_probability").alias("propaganda_probability"),
        "prediction"
    )

pdf_detailed = df_csv_detailed.toPandas()
pdf_detailed.to_csv(DRIVE_PATH + "detailed_posts_analytics.csv", index=False)

topic_window =
Window.partitionBy("channel_name").orderBy(F.col("topic_count").desc())
df_top_topics = df_scored.groupBy("channel_name", "main_topic") \
    .count().withColumnRenamed("count", "topic_count") \
    .withColumn("rank", F.row_number().over(topic_window)) \
    .filter(F.col("rank") == 1) \
    .select("channel_name", F.col("main_topic").alias("most_frequent_topic"))

lang_window =
Window.partitionBy("channel_name").orderBy(F.col("lang_count").desc())
df_top_lang = df_scored.groupBy("channel_name", "lang") \
    .count().withColumnRenamed("count", "lang_count") \
    .withColumn("rank", F.row_number().over(lang_window)) \
    .filter(F.col("rank") == 1) \
    .select("channel_name", F.col("lang").alias("dominant_language"))

df_channel_agg = df_scored.withColumn(
    "is_misinfo", F.when(F.col("sentiment_label") == "misinformation",
1).otherwise(0)
).groupBy("channel_name").agg(
    F.first("dataset_source").alias("source_type"),
    F.avg("is_misinfo").alias("misinformation_rate"),
    F.avg("ttr").alias("avg_ttr"),
    F.avg("p_us_norm").alias("avg_balanced_pronoun_us"),
    F.avg("p_them_norm").alias("avg_balanced_pronoun_them"),
    F.avg("p_subj_norm").alias("avg_balanced_subjectivity"),
    F.count("*").alias("post_count"),
    F.sum("duplicate_count").alias("total_duplicates_sum"),
    F.avg("prediction").alias("propaganda_percentage")
)

df_final_agg = df_channel_agg.join(df_top_topics, on="channel_name",
how="left").join(df_top_lang, on="channel_name", how="left")

pdf_agg = df_final_agg.toPandas()
pdf_agg.to_csv(DRIVE_PATH + "channel_summary_analytics.csv", index=False)

"""### Матриця помилок"""

from google.colab import drive
drive.mount('/content/gdrive')

DRIVE_PATH = '/content/gdrive/MyDrive/Coursework_Data/'

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

df = pd.read_csv(DRIVE_PATH + 'detailed_posts_analytics.csv')
```

```
df['y_true'] = df['dataset_source'].apply(lambda x: 1.0 if x == 'propaganda'
else 0.0)

y_true = df['y_true']
y_pred = df['prediction']

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Інше (0)', 'Пропаганда (1)'],
            yticklabels=['Інше (0)', 'Пропаганда (1)'])
plt.xlabel('Передбачено моделлю')
plt.ylabel('Фактично є')
plt.title('Confusion Matrix')
plt.show()
```