НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

«До захисту допущено»
В.о. Завідувач кафедри
_____ Іван ТЕРЕЩЕНКО
«___»_____ 2025 р.

Дипломна робота

на здобуття ступеня бакалавра
зі спеціальності: 113 Прикладна математика
на тему: «Порівняння методів машинного навчання для класифікації сільськогосподарських культур за супутниковими знімками »

Виконав:
студент IV курсу, групи ФІ-11
Мельник Володимир Володимирович _____

Керівник:
док філософії, доцент кафедри ММАД,
Яйлимова Ганна Олексіївна_____

Рецензент:
Доктор філософії,
старший науковий співробітник ІКД НАНУ-ДКАУ
Лавренюк Микола Сергійович_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ - 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

Рівень вищої освіти — перший (бакалаврський)
Спеціальність: 113 Прикладна математика,
Освітня програма: «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____Іван ТЕРЕЩЕНКО
« » 2025 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Мельник Володимир Володимирович

1. Тема роботи: *«Порівняння методів машинного навчання для класифікації сільськогосподарських культур за супутниковими знімками»*,

керівник: звання доцент ММАД, д-р філософії Яйлимова Ганна Олексіївна,

затверджені наказом по університету №1761-с від «26» травня 2025 р.

2. Термін подання студентом роботи: «05» червня 2025 р.

3. Вихідні дані до роботи: супутникові знімки Sentinel-2, методи машинного навчання, маски даних сільськогосподарських культур

4. Зміст роботи: У дослідження покладено аналіз даних, що включають багатоспектральні зображення та маски даних сільськогосподарських культур. Виконується порівняння методів машинного навчання і оцінка впливу вегетаційних показників на їхню точність.

5. Перелік ілюстративного матеріалу: презентація доповіді

6. Дата видачі завдання: 19 вересня 2025 р.

Каленларний план

№	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми з науковим керівником	Вересень 2024	Виконано
2	Аналізу літературних джерел за темою дослідження	Жовтень 2024-лютий 2025	Виконано
3	Освоєння програм і обробка даних	Березень 2025	Виконано
4	Створення датасету з навчальною і тестовою вибіркою	Квітень 2025	Виконано
5	Побудова ММН для розв’язання даної задачі	1 .05.2025 – 15.05.2025	Виконано
6	Аналіз результатів роботи моделей	15.05.2025-20.05.2025	Виконано
7	Оформлення дипломної роботи	20.05.2025-5.06.2024	Виконано

Студент _____ Мельник В.В.

Керівник _____ Яйлимова Г.О.

РЕФЕРАТ

Кваліфікаційна робота містить: 88 сторінок, 19 рисунків, 18 таблиць, 30 джерел.

Сільське господарство відіграє ключову роль у забезпеченні глобальної продовольчої безпеки та підтримці економічної стабільності. Ефективний моніторинг сільськогосподарських угідь є важливим для оптимізації врожайності, управління ресурсами та прийняття обґрунтованих політичних рішень у цьому секторі. У сучасному сільському господарстві зростає важливість точної та своєчасної класифікації сільськогосподарських культур. Точна інформація про типи культур та їх просторовий розподіл є необхідною для оцінки врожайності, управління ресурсами та прийняття обґрунтованих політичних рішень в аграрному секторі.

Метою є порівняти ефективність різних моделей машинного навчання для задачі класифікації типів сільськогосподарських культур за супутниковими знімками і оцінити вплив вегетаційних показників на їхню точність. Об’єктом дослідження є процес автоматизованої класифікації типів сільськогосподарських культур за супутниковими знімками Sentinel-2. Предметом дослідження є методи машинного навчання, що застосовуються для аналізу супутникових знімків та класифікації типів сільськогосподарських культур.

У даному аналізі порівнювалася точність та продуктивність методів машинного навчання, використовуючи два датасети – з вегетаційними індексами та без них, щоб визначити їх вплив на результати. З порівняльного аналізу випливає, що найкращий результат за точністю мав SVM, але був найповільнішим і був зі зменшеною вибіркою. Найкращим по відношенні часу і точності був XGBoost (з вегетаційними індексами та без точність виявилася однаковою).

КЛЮЧОВІ СЛОВА: ВЕГЕТАЦІЙНИЙ ІНДЕКС, МАШИННЕ НАВЧАННЯ, SENTINEL-2, АНАЛІЗ ДАНИХ.

ABSTRACT

Qualification work contains: 88 pages, 19 figures, 18 tables, 30 sources.

Agriculture plays a key role in ensuring global food security and maintaining economic stability. Effective monitoring of agricultural land is essential for optimizing yields, managing resources, and making informed policy decisions in this sector. In modern agriculture, accurate and timely crop classification is increasingly important. Accurate information on crop types and their spatial distribution is essential for yield estimation, resource management, and informed policy decisions in the agricultural sector.

The aim is to compare the effectiveness of different machine learning models for the task of classifying crop types from satellite images and to assess the impact of vegetation indicators on their accuracy. The object of research is the process of automated classification of crop types from Sentinel-2 satellite images. The subject of the study is machine learning methods used to analyze satellite images and classify crop types.

This analysis compared the accuracy and performance of machine learning methods using two datasets - with and without vegetation indices - to determine their impact on the results. The comparative analysis shows that SVM had the best result in terms of accuracy, but was the slowest and had a reduced sample. The best in terms of time and accuracy was XGBoost (with and without vegetation indices, the accuracy was the same).

KEYWORDS: VEGETATION INDEX, MACHINE LEARNING, SENTINEL-2, DATA ANALYSIS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БПЛА – безпілотний літальний апарат

RF – Random Forest

LG - Logistic regression (укр. – Логістична регресія)

SWIR – shortwave infrared

ММН – Методи машинного навчання

NDMI – Normalized Difference Moisture Index

NDWI – Normalized Difference Water Index

NDVI – Normalized Difference Vegetation Index

NDRE – Normalized difference red edge index

SVM – Support Vector Machine

XGBoost – eXtreme Gradient Boosting

ВСТУП

Актуальність дослідження. Сільське господарство відіграє ключову роль у забезпеченні глобальної продовольчої безпеки та підтримці економічної стабільності. Ефективний моніторинг сільськогосподарських угідь є важливим для оптимізації врожайності, управління ресурсами та прийняття обґрунтованих політичних рішень у цьому секторі. Традиційно, моніторинг здійснювався шляхом безпосереднього фізичного огляду полів, проте розвиток технологій дистанційного зондування, зокрема супутникової зйомки вектор моніторингу трохи змінився. Початкові методи візуального спостереження еволюціонували, і сьогодні віртуальний моніторинг стає реальністю завдяки значним технологічним досягненням.

У сучасному сільському господарстві зростає важливість точної та своєчасної класифікації сільськогосподарських культур. Точна інформація про типи культур та їх просторовий розподіл є необхідною для оцінки врожайності, управління ресурсами та прийняття обґрунтованих політичних рішень в аграрному секторі. Своєчасні дані про типи культур та їх поширення є критично важливими для забезпечення продовольчої безпеки та сталого розвитку сільського господарства. Традиційні методи моніторингу врожаю часто є дорогими, повільними та обмеженими за охопленням, що робить супутникову класифікацію більш життєздатною альтернативою. Попит на точну та своєчасну класифікацію сільськогосподарських культур зростає через глобальні виклики, такі як зміна клімату, зростання населення та потреба в сталому використанні ресурсів. Ці фактори зумовлюють необхідність ефективних сільськогосподарських практик, які залежать від точної інформації про типи культур та їх стан.

Метою дослідження: є порівняти ефективність різних моделей машинного навчання для задачі класифікації типів сільськогосподарських культур за супутниковими знімками і оцінити вплив вегетаційних показників на їхню точність.

Завдання роботи:

1. Проаналізувати існуючі підходи до класифікації землекористування на основі супутникових даних.
2. Обрати підходячи ММН для даного аналізу. Також взяти підходячи вегетаційні показники.
3. Проведення аналізу супутникових зображень Sentinel-2.
4. Підготувати навчальну та тестову вибірки із супутникових даних.
5. Реалізувати класифікацію типів культур за допомогою моделей: Логістичної регресії, SVM, Random Forest та XGBoost з вегетаційними показниками і без них.
6. Оцінити якість побудованих моделей за стандартними метриками (Accuracy, F1-score, матриця помилок).
7. Порівняти отримані результати і сформулювати висновки щодо доцільності використання кожного з методів.

Об'єктом дослідження є процес автоматизованої класифікації типів сільськогосподарських культур за супутниковими знімками Sentinel-2.

Предметом дослідження є методи машинного навчання, що застосовуються для аналізу супутникових знімків та класифікації типів сільськогосподарських культур.

Наукова новизна цього полягає у порівняльному аналізі ефективності різних підходів машинного навчання на основі витягнутих спектральних даних супутникових знімків і порівняння впливу вегетаційних показників на точність цих моделей.

Практична значущість: полягає роботи у підвищення ефективності моніторингу стану сільськогосподарських угідь, що дозволить оптимізувати процеси управління аграрними ресурсами.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ У КЛАСИФІКАЦІЇ СІЛЬСЬКОГОСПОДАРСЬКИХ КУЛЬТУР ЗА СУПУТНИКОВИМИ ЗНІМКАМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі

Завантажуємо супутникові зображення Sentinel-2, у яких є інформація у 12 спектральних каналах,. Для аналізу будуть використовуватися ММН, які й будуть порівнюватися для визначення найбільш ефективного підходу для цієї задачі, та їх поєднання з спектральними індексами та вегетаційними показниками. Перевірити чи впливають показники чи ні.

1.2 Методи моніторингу сільськогосподарських угідь

Моніторинг сільськогосподарських угідь виконується різними методами в залежності від поставлених завдань, доступності ресурсів та необхідної точності.. Існує багато методів моніторингу - від більш застарілих. Таких як: наземне обстеження - традиційний метод збору даних шляхом безпосереднього огляду полів, до новіших - авіаційна зйомка, використання літаків або БПЛА для збору даних з повітря, та супутниковий моніторинг, який дає змогу покривати великі території з прийнятною роздільною здатністю. Звичайно, комбінація всіх цих методів може дати найкращий результат в дослідженні сільськогосподарського покриву.

1.2.1 Дистанційне зондування з використанням безпілотних літальних апаратів

Завдяки технологічному прогресу, сьогодні використання безпілотних літальних апаратів (БПЛА). Сільськогосподарські БПЛА мають високі

можливості, і їх використання розширилося в усіх сферах сільського господарства, включаючи розпилення пестицидів і добрив, посів насіння, оцінку росту та картографування [1]. Дистанційне зондування БПЛА відрізняється від супутникового не регулярною зйомкою, а саме моніторингом, який обмежується часом польоту у порівнянні з регулярною супутниковою зйомкою. Але такий моніторинг може проводитися тоді, коли є в цьому необхідність, в той час як супутники мають певний графік. Зондування БПЛА дозволяє отримувати зображення високої роздільності, але не на зовсім великих територіях в порівнянні із супутниками. За допомогою БПЛА краще можна відстежувати стан полів або невеликих ділянок. Також на БПЛА сильно впливають погодні умови, що робить їх не сильно ефективними у дощові періоди.

1.2.2 Наземний моніторинг

Наземний моніторинг значною мірою залежить від візуальної оцінки. Цей звичайний процес був значно вдосконалений завдяки технології. Сьогодні фермери використовують різноманітні інструменти та технології для більш ефективного моніторингу своїх ферм [2].

Наземні дані часто використовуються як "еталонні" або "ground truth" (будемо ще називати маскою чистих даних) дані для калібрування та валідації моделей, побудованих на основі даних дистанційного зондування (супутникових знімків чи даних БПЛА). Це дозволяє перевірити точність класифікації культур, оцінки стану посівів чи прогнозування врожайності, отриманих дистанційними методами [3]. Недоліками наземного моніторингу є його трудомісткість, висока вартість та обмежене просторове покриття порівняно з дистанційним зондуванням.

1.2.3. Дистанційне зондування

Супутникові знімки стали незамінним інструментом у сучасному сільському господарстві, надаючи широкий спектр можливостей для моніторингу та управління сільськогосподарськими угіддями [4]. Використання цих технологій надає аграріям можливість швидко виявляти проблемні ділянки та приймати обґрунтовані рішення для оптимізації врожайності та сталого використання ресурсів [4].

Існує багато робіт для аналізів сільськогосподарських полів за супутниковими знімками. Завдяки дистанційному зондуванню точність і площа дослідження полів сильно збільшилася. Підготування вибірки і аналіз є однією з найважливіших частин для ММН. Існують різні способи підготовки даних і способі їхнього аналізу і навчання. Автор статті [5] використовує піксельний підхід, де для навчання в кожному пікселі є великий діапазон даних і ознак з різних знімків протягом вегетаційного сезону. Це “класичний” підхід до аналізу супутникових, але він не єдиний. Наприклад автор статті [6] розглядує об’єктно-орієнтований підхід, замість піксельного. Він використовує сегментацію зображення, перед класифікацією. Для формування навчальної вибірки є не менш важливим мати дані, які саме споруди і культури знаходяться на цій місцевості. Такі дані називаються Ground Truth Data (чисті дані) - це інформація про те, яка саме культура росла на конкретних ділянках полів у певний період часу. Саме завдяки ним модель буде розуміти, яка культура знаходиться цій конкретній зоні. Про важливість володіння якісними Ground Truth Data розповідає автор [7] статті. Він розповідає про вплив невизначеності в референтних даних на оцінку точності. Для моєї роботи, я скористувався напрацюванням фахівців кафедри ММАД, які отримали карти земного покриву в межах міжнародних та національних проєктів [30] України, яка була теж отримана за допомогою ММН. Саме на цих Ground Truth Data будуть ґрунтуватися мої вибірки.

Переваги застосування супутникового моніторингу включають високу точність оцінки площі посівів, можливість відстеження стану рослин на всіх фазах росту, раннє прогнозування врожаю та ефективний контроль за проведенням агротехнічних заходів [9].

1.3 Sentinel-2

Sentinel-2 — це європейська широкозонна місія з високою роздільною здатністю, багатоспектральним зображенням. Повна специфікація місії супутників-близнюків, що летять на одній орбіті, але з фазою 180°, розроблена для забезпечення високої частоти повторного відвідування екватора 5 днів [10].

Мультиспектральний інструмент (MSI), встановлений на супутниках, фіксує 13 спектральних каналів у діапазонах від видимого та ближнього інфрачервоного (VNIR) до короткохвильового інфрачервоного (SWIR) [11]. Просторове розрізнення варіюється від 10 м (для видимих та NIR каналів) до 20 м (для каналів червоної межі та SWIR) та 60 м (для атмосферних каналів). Поєднання високої просторової та часової роздільної здатності Sentinel-2, а також конфігурація його спектральних каналів (включаючи канали червоної межі), роблять його особливо придатним для моніторингу рослинності та сільськогосподарських застосувань [12].

Дані Sentinel-2 є вільно доступними, що робить їх цінним ресурсом для наукової спільноти та різних застосувань [13].

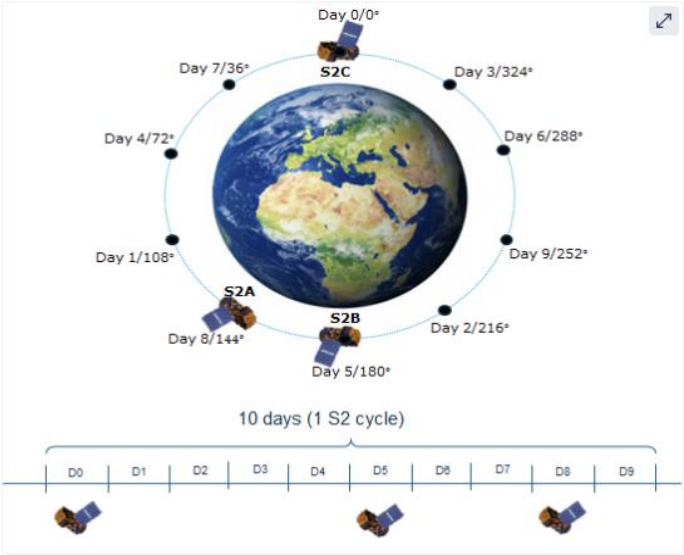


Рис 1.1. Розподіл супутників Sentinel-2 на орбіті місії [10]

1.4 Вегетаційні індекси

Основна ідея використання спектральних індексів полягає у підсиленні контрасту між об'єктом, що досліджується, та навколишнім фоном, або у визначенні кількісної міри певної властивості об'єкта, яка не є очевидною при розгляді окремих спектральних каналів [14]. Наприклад, здорова рослинність має високу відбивну здатність в ближньому інфрачервоному діапазоні та низьку в червоному, тоді як вода поглинає випромінювання в обох цих діапазонах. Проста різниця або відношення між цими каналами може ефективно виділити рослинність на тлі води, ґрунту чи інших покривів [14].

І також вплив деяких негативних факторів, таких як, наприклад, яскравість, може бути зменшеною або взагалі нівельованою, надаючи чіткішу інформацію про стан рослинного покриву залежно від математичної структури індексу [15]. У моєму аналізі використовуються такі індекси, як NDVI, NDWI, NDMI, NDRE, бо саме їх найчастіше використовують для аналізів полів. На основі них ми дізнаємося чи покращують вони точність ММН чи навпаки погіршують їхній результат.

Normalized Difference Built-up Index (NDVI) є широко використовуваним і фундаментальним показником у дистанційному зондуванні для кількісної оцінки здоров'я та щільності зеленої рослинності. Цей показник використовує відмінні характеристики спектральної відбивної здатності здорової рослинності, яка сильно поглинає червоне світло для фотосинтезу та відбиває значну частину ближнього інфрачервоного світла завдяки своїй клітинній структурі [16]. NDVI слугує простим графічним індикатором для оцінки наявності живої зеленої рослинності на спостережуваному об'єкті.

$$NDVI = \frac{B08 - B04}{B08 + B04}. \quad (1.1)$$

Normalized Difference Water Index (NDWI) є індексом дистанційного зондування, пов'язаним із вмістом рідкої води. Він використовується для виділення відкритих водних об'єктів та покращення їхньої видимості на супутникових знімках. Він також слугує для вимірювання вмісту рідкої води в рослинності та оцінки вологості рослин і ґрунту.

$$NDWI = \frac{B03 - B08}{B03 + B08}. \quad (1.2)$$

Нормалізований різницевий індекс вологості (NDMI) є вегетаційним індексом, отриманим з даних дистанційного зондування для оцінки та моніторингу вмісту вологи в рослинності [17] .

$$NDMI = \frac{B08 - B11}{B08 + B11}. \quad (1.3)$$

Normalized Difference Red Edge Index (NDRE) є індексом, що використовує червоний та ближній інфрачервоний діапазон. Ці канали особливо чутливі до вмісту хлорофілу та структурних змін у листі на пізніших стадіях розвитку рослин, коли відбиття у червоному каналі (B04) може досягати насичення. Він корисний для оцінки стану рослинності та може допомогти розрізняти культури [18] .

$$NDRE = \frac{B08 - B05}{B08 + B05}. \quad (1.4)$$

1.5. Методи машинного навчання

ММН є напрямком штучного інтелекту, що спрямований на розробку алгоритмів, які здатні виявляти закономірності у даних та робити прогнози на основі нової інформації. У задачах класифікації метою є віднесення кожного об'єкта до певного з наперед визначених класів. Алгоритми ММН продемонстрували значні можливості в класифікації сільськогосподарських культур за супутниковими знімками, часто перевершуючи традиційні параметричні методи [19]. У роботі [20] було побудова ММН на основі алгоритмів RF, отримано і навчено модель для аналізу супутникових знімків за вегетаційний період рослин (1 квітня по 30 жовтня). Ця робота показує, що можна побудувати ефективну і точну модель для аналізу полів.

1.5.1 Random Forest

Цей алгоритм швидко здобув популярність завдяки своїй здатності надавати високоточні прогнози та ефективно працювати зі складними та великорозмірними даними [21]. Random Forest (RF) успішно застосовується для широкого кола задач, включаючи класифікацію та регресію. Кожне окреме дерево тренується на випадковій підмножині (бутстреп-вибірці) з оригінального набору даних. Крім того, на кожному етапі побудови дерева для вибору оптимального розщеплення вузла розглядається лише випадково обрана підмножина ознак [21]. Ця подвійна випадковість (як у вибірці даних, так і у виборі ознак) допомагає знизити кореляцію між окремими деревами, що робить ансамбль більш стійким до перенавчання та покращує його узагальнювальну здатність порівняно з одним деревом рішень [22].

Найпоширенішими критеріями неоднорідності, що використовуються для класифікації є індекс Джині і ентропія [21]. У бібліотеці Scikit-learn, яку я використав у своєму аналізі використовується індекс Джині (за замовченням). Ця формула має вид:

$$\sum_{i=1}^{C} f(1 - f_i)\,. \tag{1.5}$$

Нижче наведена формула для ентропії:

$$\sum_{i=1}^{C} - f_i \log(f_i)\,, \tag{1.6}$$

де

f_i представляє собою частку об'єктів, що належать до певного класу i у вузлі, а C означає собою загальну кількість унікальних класів, які представлені у вузлі.

Для кожного вузла j у дереві рішень, де відбувається розщеплення, обчислюється зменшення неоднорідності (ni_j) за формулою [21]:

$$ni_j = w_j C_j - w_{L_j} C_{L_j} - w_{R_j} C_{R_j}, \tag{1.7}$$

де

– ni_j – важливість (зменшення неоднорідності) вузла j,

– w_j – зважена кількість зразків, які досягають вузла j,

– C_j – значення неоднорідності вузла j,

– L_j – лівий дочірній вузол, отриманий після розщеплення вузла j,

– R_j – правий дочірній вузол, отриманий після розщеплення вузла j.

Потім обчислюється важливість кожної ознаки i в межах одного дерева [22]:

$$f_{i_i} = \frac{\sum_{j:\text{вузол } j \text{ поділ на функцію } i} ni_j}{\sum_{k \in \text{всі вузли}} ni_k}, \tag{1.8}$$

де

– f_{i_i} – важливість ознаки i,

– ni_j – важливість вузла j.

Далі значення важливості ознак нормалізується в межах кожного дерева, щоб їх сума дорівнювала:

$$NORMfi_i = \frac{fi_i}{\sum_{j \in \text{усім функціям}} fi_j}, \quad (1.9)$$

Остаточна важливість ознаки i обчислюється для всієї моделі обчислюється як середнє значення важливості цієї ознаки у лісі [22]:

$$RFfi_i = \frac{\sum_{j \in \text{усі дерева}} NORM_{f_{ij}}}{T}, \quad (1.10)$$

де

– $RFfi_i$ – важливість ознаки i, обчислена з усіх дерев у моделі RF,

– $NORM_{f_{ij}}$ – нормалізована важливість ознаки для i у дереві j,

– T – загальна кількість дерев.

RF широко використовується у багатьох галузях, і зокрема, він є ефективним інструментом для задач моніторингу та аналізу стану лісів, що розглядається у цьому дослідженні. Його універсальність дозволяє працювати з даними різної природи – як числовими, так і категоріальними.

Однією з переваг RF є його переваг це стійкість до перенавчення порівняно з іншими моделями. Це досягається за рахунок того, що велика кількість дерев тренуються на випадково обраних підмножин даних і ознак. А фінальний прогноз є комбінацією (усереднення або голосуванням) їхніх індивідуальних прогнозів. Такий підхід допомагає покращити здатність моделі і забезпечити високу точність і надійність результатів [21].

1.5.2 Логістична регресія

Логістична регресія є класичним статистичним алгоритмом, який широко використовують для розв’язання задач бінарної класифікації [23]. На відміну від лінійної регресії, вихід якої може набувати будь-яких значень, логістична регресія обмежує свій вихід діапазоном від 0 до 1, що дозволяє

інтерпретувати його як ймовірність [23]. Основу логістичної регресії становить лінійна комбінація ознак, яка аналогічна до лінійної регресії:

$$z = b_o + b_1X_1 + b_2X_2 + \ldots + b_pX_p. \quad (1.11)$$

де

$-b_o \ldots b_p$ – коефіцієнти регресії,

$-X_o \ldots X_p$ – вхідні змінні.

Математично, регресійну модель можна визначити, як ймовірність того, що випадкова змінна Y приймає значення 1 за умови вектора вхідних змінних X. Так як передбачається ймовірність, то значення будуть належати проміжку [0,1]. Від «0» - низька ймовірність і «1» - висока ймовірність. Сигмоїдна функція відображає будь-яке дійсне число на значення в інтервалі від «0» до «1». Значення p інтерпретується як оцінка умовної ймовірності належності об'єкта до позитивного класу (У=1) за умови вхідних ознак:

$$f(x) = P(Y = 1|X) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}, \quad (1.12)$$

де

– P – ймовірність події.

Навчання логістичної регресії полягає у визначенні оптимальних значень коефіцієнтів $(b_o, \ldots b_p)$ регресії. Це досягається шляхом функції максимізації правдоподібності [24]:

$$L(b_o, \ldots b_p) = \prod_{i=1}^{n} \left(P(Y_i = 1|X_i)^{Y_i} \right) \times \left(1 - P(Y_i = 1|X_i) \right)^{1-Y_i}, \quad (1.13)$$

де

– n – кількість точок даних,

$-X_i$ – фактична мітка ознак для i-ї точки даних,

$-Y_i$ – фактична мітка класу для i-ї точки даних.

Завдяки спроможності аналізувати ймовірності, логічна регресія є дуже корисною для аналізування територій сільськогосподарських полів. І її корисність не зупиняється тільки в цій сфері.

1.5.3 Support Vector Machine (SVM)

Метод опорних векторів (SVM) є потужним алгоритмом керованого навчання, який в основному використовується для задач класифікації [25]. Основна ідея алгоритму полягає у знаходженні оптимальної розділяючої гіперплощини, яка максимально віддалена від найближчих точок даних кожного класу [25] рис(1.2). SVM обирає ту гіперплощину, яка максимізує "відступ" (margin) — відстань між гіперплощиною та найближчими до неї точками даних з кожного класу. Більший відступ вважається кращим, оскільки він забезпечує кращу узагальнювану здатність моделі на нових даних [25]. Точки, що лежать на кордонах, називаються опорними векторами, а серединна лінія – розділова гіперплощина.

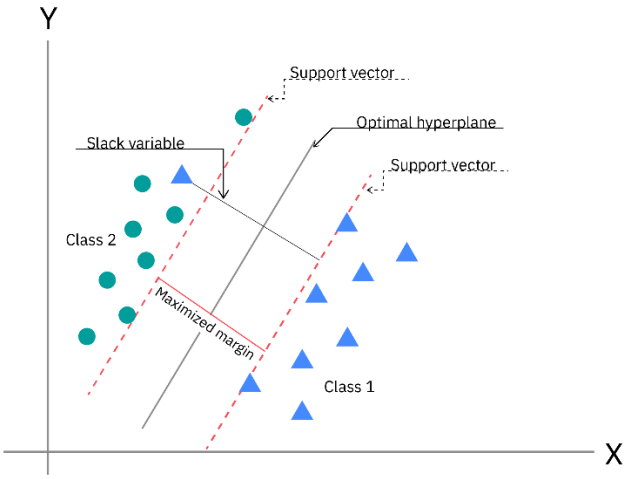


Рис 1.2 Лінійна сепарабельність [25]

У лінійному розділюваному випадку математичний вигляд SVM буде:

$$w \times x + b = 0. \quad (1.14)$$

де

– w – вектор нормалі до гіперплощини,

– b – зсув відносно початку координат,

– x – вектор вхідних ознак.

Для того, щоб визначити чи знаходиться точка по один бік чи інший від сторони гіперплощини, є рівняння:

$$y_i = (w \times x_i + b) \geq 1, \quad (1.15)$$

де

– y_i – мітка класу;

Функція ядра може бути будь-якою з наступних[26]:

– лінійний:⟨x,x′⟩;

Еквівалентно лінійному SVM у початковому просторі.

– polynomial: $(\gamma\langle x, x'\rangle + r)^d$;

Підходить до не лінійно роздільних задач або де є акцент на взаємодію ознак

– rbf (Гаусовське ядро): $\exp(-\gamma|x - x'| + r)$;

Дуже гнучке ядро, яке може відобразити дані у нескінченновимірний простір і моделювати складні, нелінійні межі рішень.

– sigmoid: $\tanh(\gamma\langle x, x'\rangle + r)$;

Використовується в задачах де необхідно перетворити у двійкове значення.

Незважаючи на деякі недоліки, (обчислювальну складність великих наборів даних і чутливості ядра) SVM залишається популярним та ефективним методом для класифікації супутникових знімків у задачах розпізнавання агрокультур, часто демонструючи високу точність класифікації.

У нашому аналізі ми будемо використовувати SVM з 2 різними ядрами (лінійним і не лінійно роздільним).

1.5.4 XGBoost

XGBoost є реалізацією алгоритму градієнтного бустингу над деревами рішень. Він поєднує ідеї бустингу (послідовного побудови слабких моделей для виправлення помилок попередніх) з градієнтним спуском та має низку

оптимізацій, які роблять його швидким та ефективним. Основу XGBoost складає цільова функція, яку алгоритм намагається мінімізувати під час навчання. Завдяки варіаціям функції втрат модель може уникнути перенавчення. Втрати, які модель прагне мінімізувати можна розрахувати за формулою [27]:

$$L_{xgb} = \sum_{i=0}^{N} L(y_i, F(x_i)) + \sum_{m=1}^{M} \Omega(h_m)\,. \quad (1.16)$$

$$\Omega(\mathrm{h}) = \gamma T + 0{,}5\lambda|w|^2, \quad (1.17)$$

де

– L_{xgb} – представляє втрати, які модель прагне мінімізувати під час навчання загалом,

– y_i – фактичне значення,

– T – кількість листя дерева,

– w – ваги на листях дерева,

– $F(x_i)$ – передбачене моделі для i навчального прикладу.

1.6 Висновки до розділу 1

У цьому ми досліділи сучасні способи моніторингу полів, які включають в себе від традиційних – наземного моніторингу до сучасних – дистанційного зондування. Розібрані відповідні ММН для майбутнього аналізу, обрані вегетаційні показники (NDVI, NDWI, NDMI, NDRE), для оцінки їхнього впливу на точність майбутніх моделей . Для порівнянь ММН у вигляді початкових даних ми будемо використовувати супутникові знімки Sentinel-2.

РОЗДІЛ 2. ОПИС ДАНИХ ТА РЕЗУЛЬТАТИ

2.1 Обранні територія та характеристика супутникових знімків

Для навчання і тестування ММН, я обрав знімки на території України. Загальну площу і області з кількістю локацій наведено у табл. 2.1.

Таблиця 2.1 Обрані локації і їх детальний опис

Назва області	Кількість локацій	Площа території
Харківська область	5	441,1 км^2
Полтавська область	2	277,3 км^2
Дніпропетровська область	1	221,5 км^2

Так як на точність моделі впливають багато різних факторів і одним і не менш важливим є тип ґрунтів і багато інших географічних факторів, то для покращення точності моделі було вирішено обрати локації, які знаходилися неподалік один одного (рис 2.1).

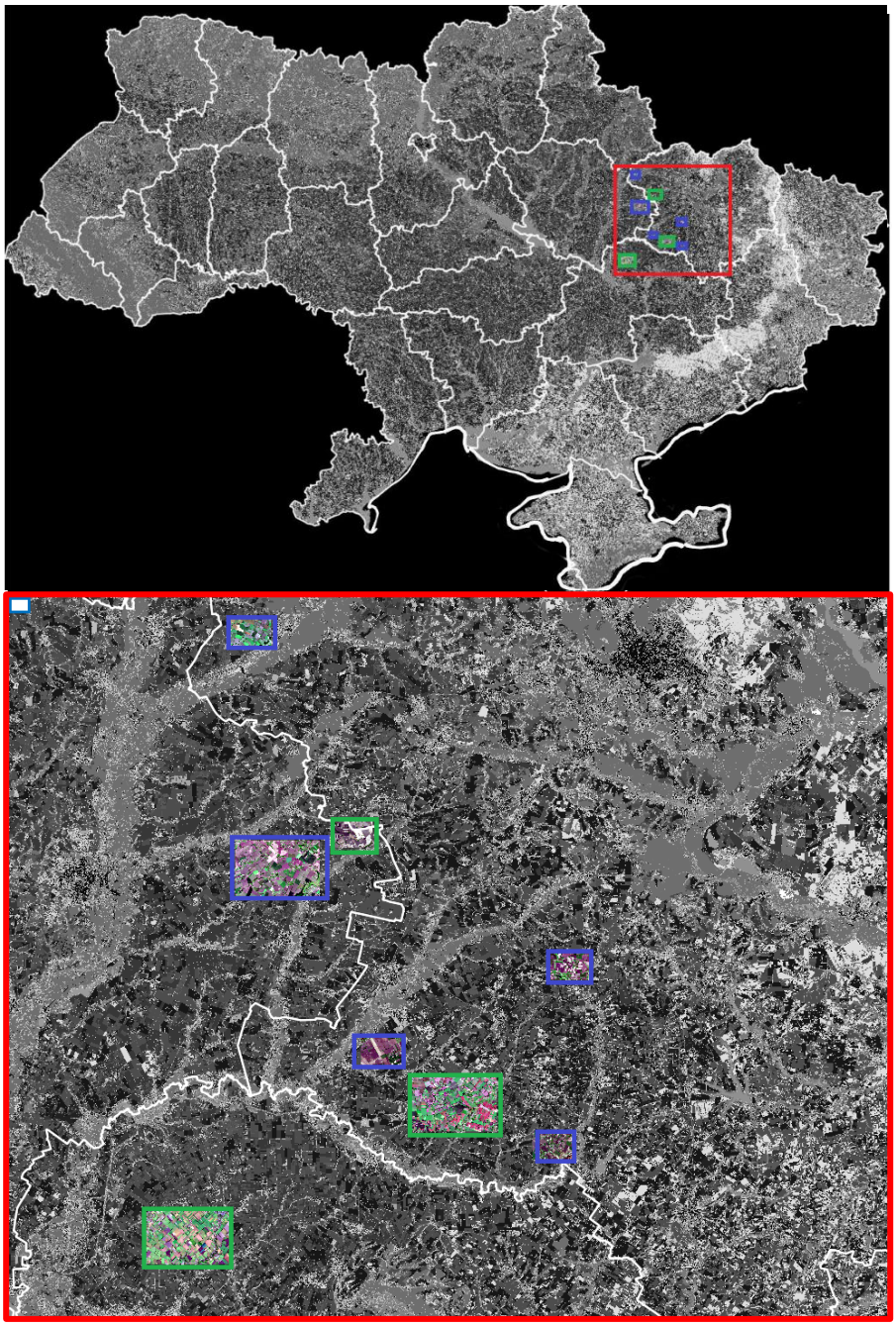


Рис 2.1 Досліджувана зона: сині області – train + test, зелені -train

У практичні роботі були розглянуті супутникові зображення з Sentinel-2 (рис 2.5) з розширення tif, які містять інформацію в 12 спектральних каналах (табл 2.1). Знімки бралися протягом вегетаційного сезону (квітень-жовтень). Було взяти по одному знімку з кожного з цих місяців, тобто по 7 знімків протягом вегетаційного сезону і їхні 12 спектральних каналів.

У якості Ground Truth Data я скористувався напрацюванням фахівців кафедри ММАД, які отримали карти земного покриву в межах міжнародних

та національних проєктів [30] (рис 2.2) і (рис. 2.3). Ця мапа є в Державному Аграрному реєстрі України [8], будемо використовувати дані за 2024 рік. Цей знімок має просторову роздільність в 10×10 метрів. Саме під такі параметри завантажував знімки для аналізу.

Рис 2.2 Мапа з Державного Аграрного реєстра

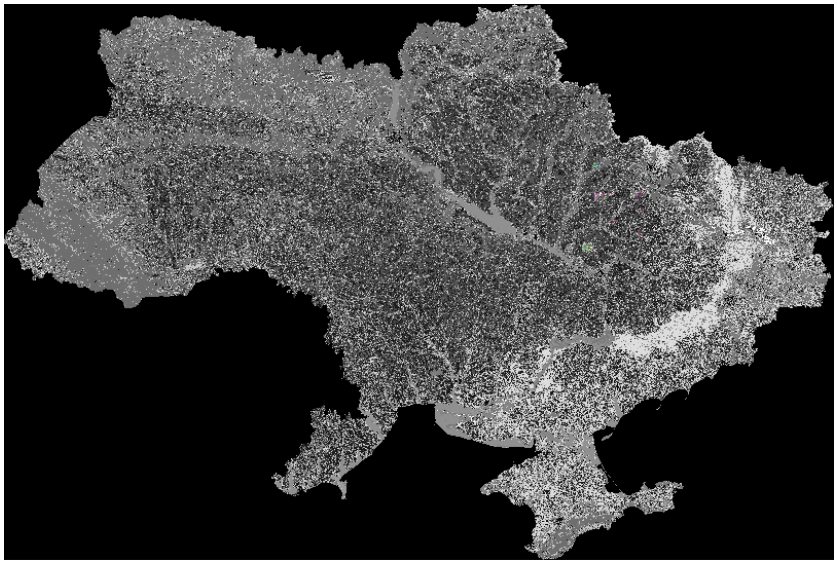


Рис 2.3 Мапа Ground Truth Data

Для співставлення завантаженого знімку і мапи маски Ground Truth Data я використав програму QGIS. Завантажував знімки через сайт Copernicus BROWSER [29]. Так знімок з Ground Truth Data мав координатну систему UTM 36N (EPSG: 32636), то завантажую знімки з такою ж системою. Хоч

знімки і мапа були з розширенням .tif, і мали вбудовані координати свого місцезнаходження (Рис 2.4) і точно стали один на одного. Залишилась розбіжність у приблизно у 0,2 пікселя у всі боки. (як було вказано). Завдяки функції align rasters ця проблема вирішита.

Потім була вирізка маски під розмір супутникового знімка для дослідження. Це було зроблено завдяки функції “обхват растру по охопленню”. Після чого маска з Ground Truth Data були вже готові для подальшого аналізу (табл. 2.2).

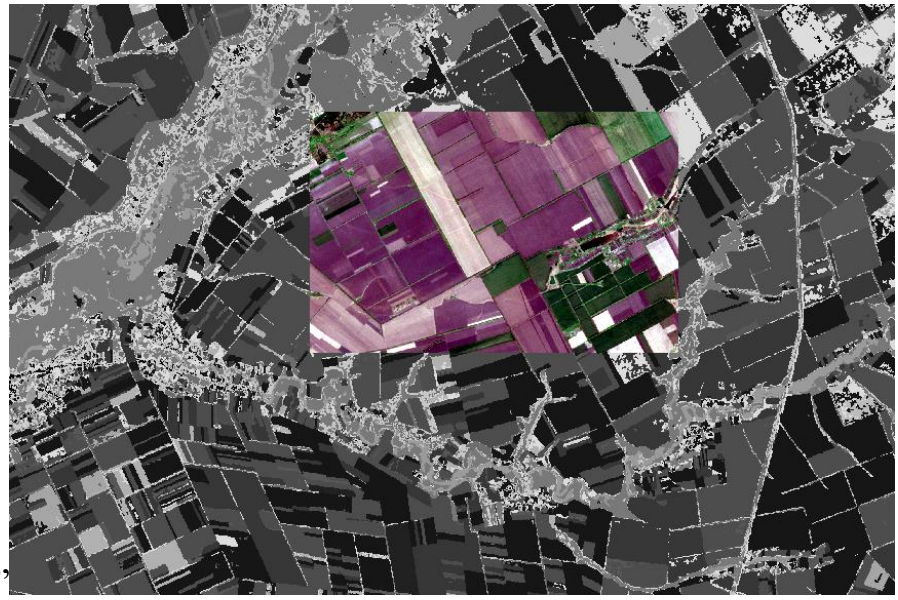
” 

Рис 2.4. Область дослідження на мапі з Ground Truth Data

Рис 2.5. Супутниковий знімок для дослідження

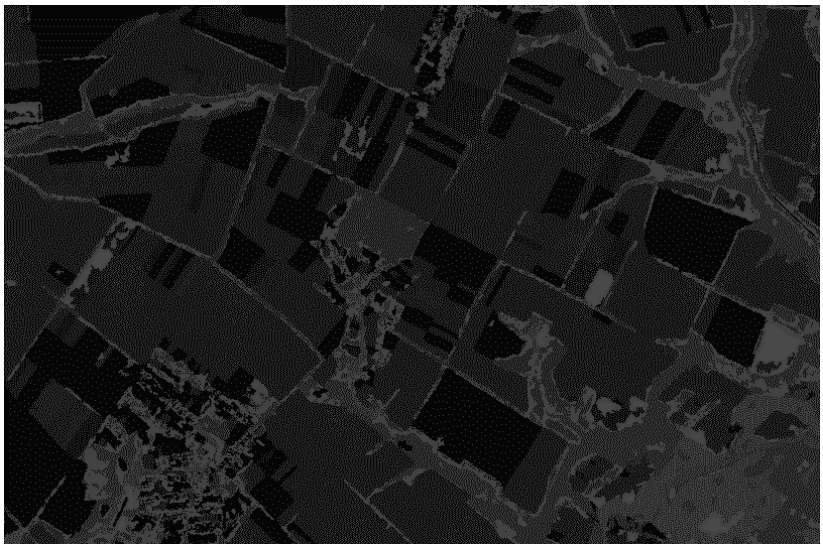


Рис 2.6. Вирізана маска з Ground Truth Data (чорно-білий фільтр)

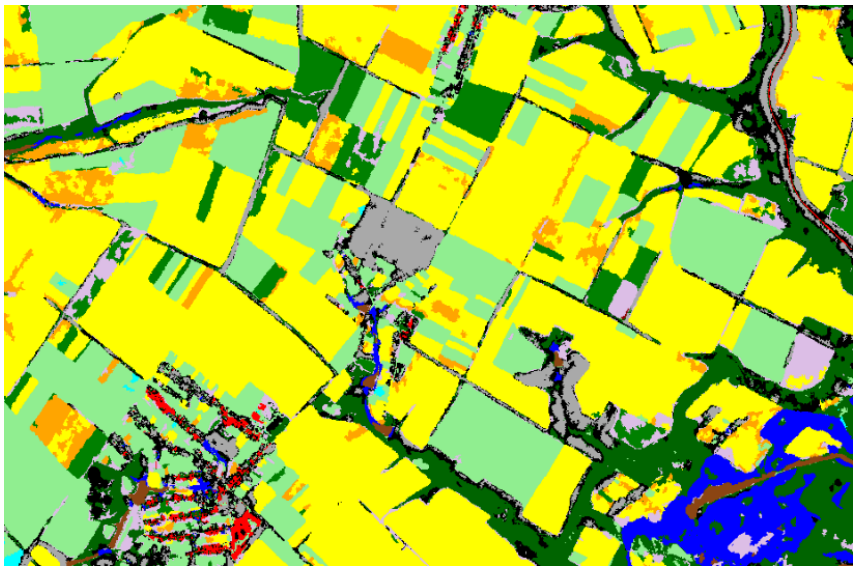
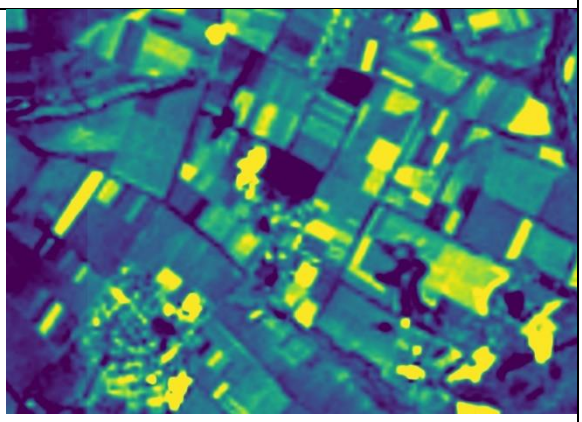
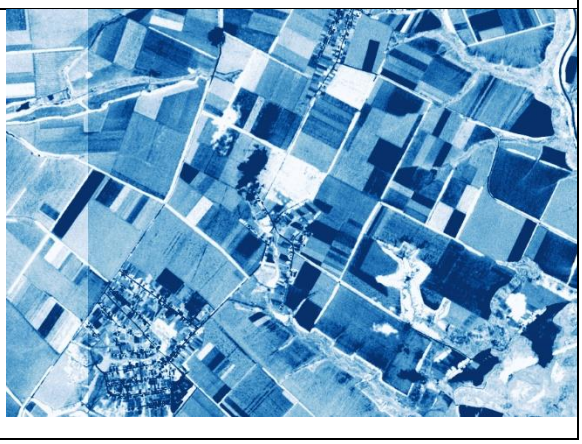
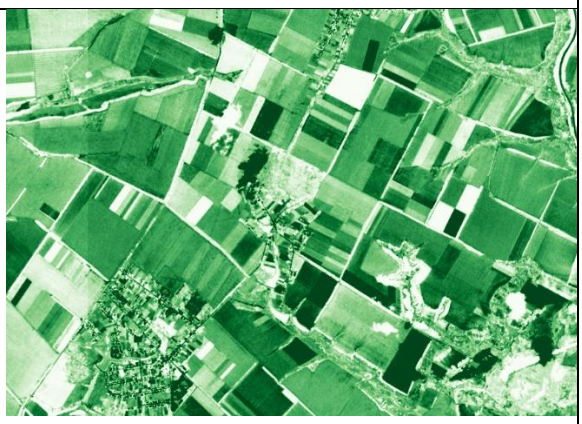
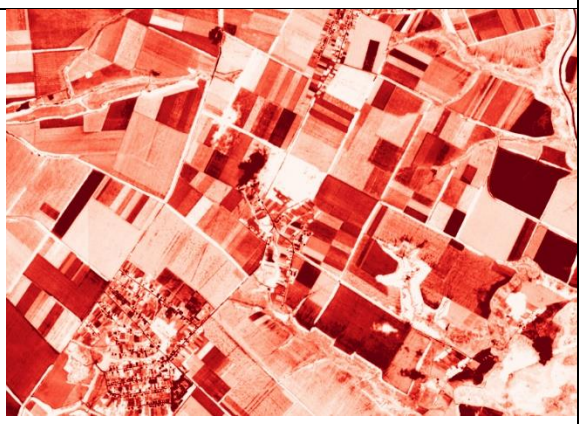
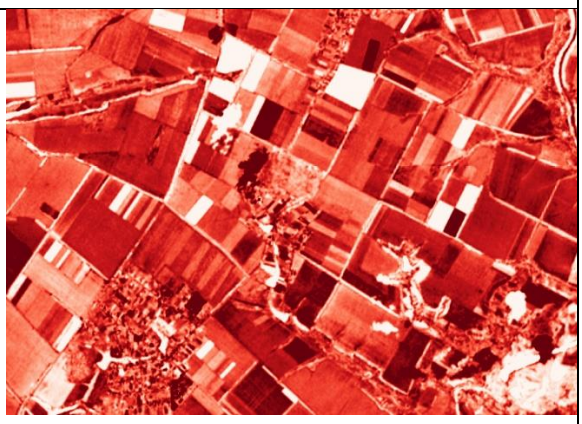
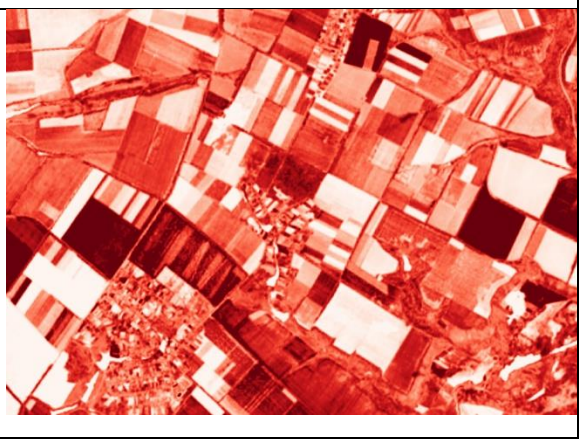


Рис 2.7. Вирізана маска з Ground Truth Data (кольоровий фільтр)

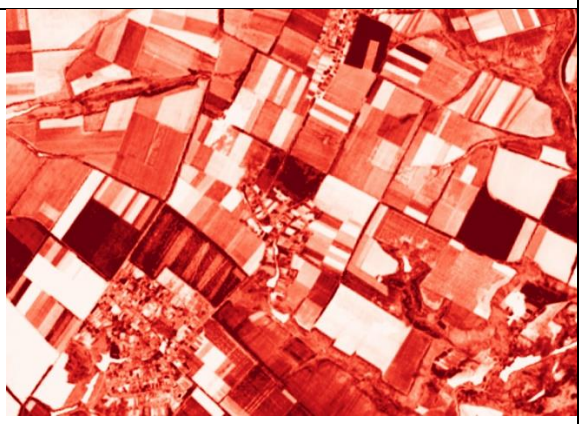
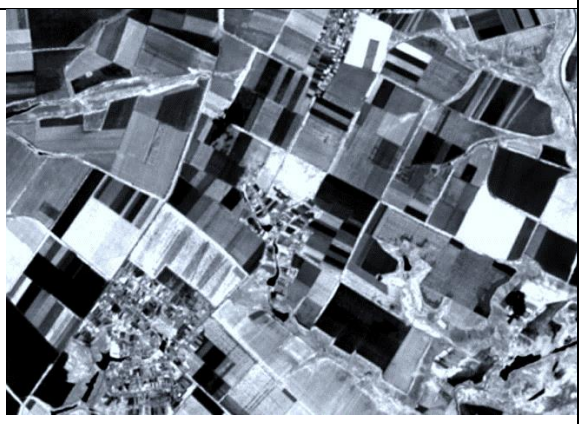
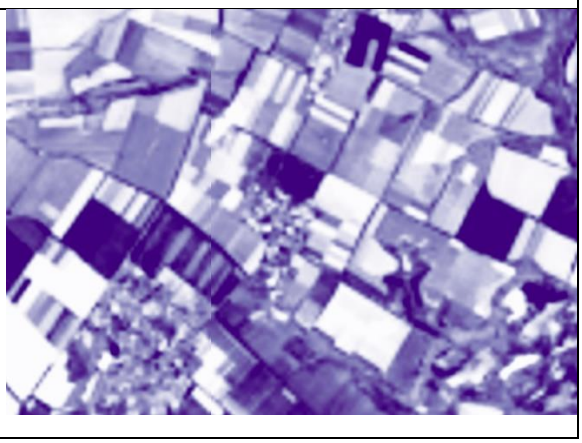
Таблиця 2.2 Оригінальне зображення і його спектральні канали з Sentinel-2 з прикладом досліджуваної ділянки test1

Спектральні канали	Просторове розділення	Спектральний діапазон	Супутникове зображення для досліджуваної області
Оригінальний знімок	10		
B1	60	443	
B2	10	490	

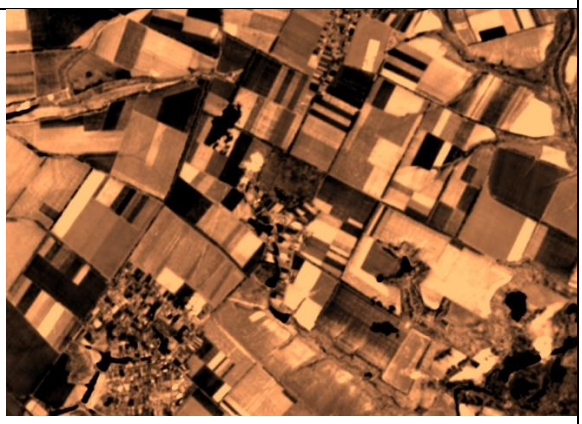
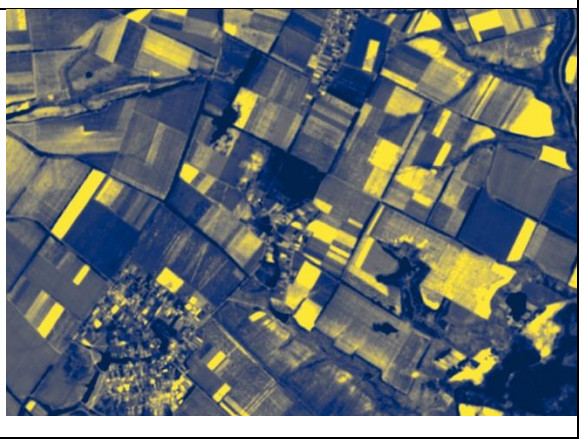
Продовження таблиці 2.2

B3	10	560	
B4	10	665	
B5	20	705	
B6	20	740	

Продовження таблиці 2.2

B7	20	783	
B8	10	842	
B8A	20	865	
B9	60	940	

Продовження таблиці 2.2

B11	20	1610	
B12	20	2190	

2.2 Підготовка датасету і даних

Для обробки даних і побудові ММП була використана мова “Python” на “Python Notebook”. Завантажувалися локації окремо у кожному файлі.

Першим кроком було завантаження усіх знімків з 1 локації, яка була в папці “test 1”. Вийшов датасет з 84 стовпцями, до кожного з значень з рядка цього датасету буде 1 значення з маски, на якому і ми будемо дізнаватися, що за культура у нас росте (табл 2.3), індекси (label) і їхнє значення візьмемо з сайту Державного Аграрного Реєстру [8]. Маску з Ground Truth Data ми перетворюємо на датасет, який потім будемо зіставляти з базою даних, яку робили вище. Для покращення точності і не отримання «Перенавчення » моделі було вирішено замість того, щоб робити навчальну вибірку на 1-2 повних локацій. Було взяти 5 локацій і розділення їх на навчальні і тестові

вибірки з відношення 20% на 80%. Завдяки такому рішення наші ММН отримали більше даних з різних локацій і стали більш точними. З іншими 3 локаціями такі дії не проводилися.

Таблиця 2.3 Назва культури і її індекс у базі даних

Назва культури	**Індекс**
Штучні об’єкти	1
Пшениця	2
Ріпак	3
Гречка	4
Кукурудза	5
Культури за 2022	6
Соняшник	7
Соєві боби	8
Цукровий буряк	9
Інші культури	10
Ліс	11
Пасовища	12
Гола земля	13
Вода	14
Водно-болотна місцевість	15
Ячмінь	16
Горох	17
Люцерна	18
Сади, парки	19
Виноград	20
Картопля	21
Не культивується	22
Пошкоджений ліс	23

Перші п’ять рядків створеного датасету можна побачити у табл 2.4

Таблиця 2.4 Перші 5 рядків датасету (скорочений варіант)

Номер	B1_1	B2_1	B3_1	B4_1	B5_1	B6_1	B7_1
0	3028.0	2779.0	3296.0	3932.0	4515.0	4784.0	5210.0
1	3028.0	2838.0	3277.0	3912.0	4463.0	4830.0	5217.0
2	3021.0	2844.0	3323.0	3971.0	4502.0	4850.0	5203.0
3	3015.0	2877.0	3270.0	4057.0	4568.0	4856.0	5184.0
4	3008.0	2844.0	3329.0	3952.0	4535.0	4823.0	5164.0

Продовження таблиці 2.4

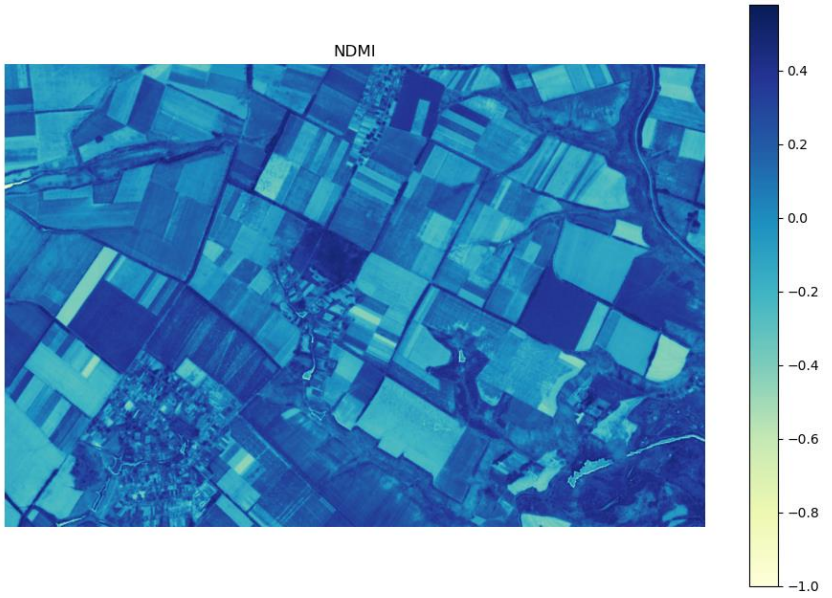
B8_1	B8A_1	B9_1	…	B6_7	B8A_7	B9_7
5721.0	6200.0	6259.0	…	8146.0	8546.0	8664.0
5826.0	6239.0	6245.0	…	8218.0	8683.0	8657.0
5715.0	6265.0	6239.0	…	8343.0	8742.0	8644.0
5675.0	6285.0	6239.0	…	8513.0	8729.0	8638.0
5721.0	6285.0	6232.0	…	8388.0	8552.0	8624.0

Продовження таблиці 2.4

B11_7	B12_7	NDVI_7	NDWI_7	NDMI_7	NDRE_7	label
18579.0	18841.0	0.196885	-0.32585	-0.39038	0.117451	8
18618.0	18861.0	0.201111	-0.32559	-0.38775	0.126381	8
18533.0	18809.0	0.203202	-0.33019	-0.37914	0.195671	8
18396.0	18691.0	0.216838	-0.33852	-0.36727	0.121568	8
18258.0	18527.0	0.212664	-0.33685	-0.37041	0.098642	8

Після цього ми до цього датасету додали вегетаційні показники. Приклад мапи вегетаційних показників для ділянки, яку ми досліджували вище можна також побачити у таблиці 2.5, а значення для даних у таблиці 2.4.

Таблиця 2.5 Приклад мап вегетаційних показників

Вегетаційний показник	Відображення вегетаційного показнику
NDVI	
NDMI	

Продовження таблиці 2.5

NDRE	
NDWI	

2.3 Використані метрики

Для порівняння методів були використані такі метрики як F1-score, Accuracy, confusion matrix (матриця помилок). Ці метрики дають важливу інформацію про продуктивність моделі і дають хорошу інформацію для порівняння.

Confusion matrix (матриця помилок) для нашої вибірки (табл 2.6).

Таблиця 2.6 Матриця помилок

Істині класи / Передбачені класи	Клас0	Клас1	Клас2	…	Клас21	Клас22	Клас23
Клас 0	$C_{0,0}$	$C_{0,1}$	$C_{0,2}$	…	$C_{0,21}$	$C_{0,22}$	$C_{0,23}$
Клас1	$C_{1,0}$	$C_{1,1}$	$C_{1,2}$	…	$C_{1,21}$	$C_{1,22}$	$C_{1,23}$
Клас2	$C_{2,0}$	$C_{2,1}$	$C_{2,2}$	…	$C_{2,21}$	$C_{2,22}$	$C_{2,23}$
…	…	…	…	…	…	…	…
Клас21	$C_{21,0}$	$C_{21,1}$	$C_{21,2}$	…	$C_{21,21}$	$C_{21,22}$	$C_{21,23}$
Клас22	$C_{22,0}$	$C_{21,1}$	$C_{22,2}$	…	$C_{22,21}$	$C_{22,22}$	$C_{22,23}$
Клас23	$C_{23,0}$	$C_{23,1}$	$C_{23,2}$	…	$C_{23,21}$	$C_{23,22}$	$C_{23,23}$

де

– $C_{i,i}$ – кількість прикладів, які були правильно класифіковані для кожного класу,

– $C_{i,j}$ – кількість прикладів, які показують як часто приклади класу i були класифіковані як інші класи,

– $C_{j,i}$– кількість прикладів, які показують як часто приклади інших класів були класифіковані як клас j.

Формула Accuracy показує частку правильно класифікованих результатів зразків до їхньої загальної кількості:

$$\text{Accuracy} = \frac{\sum_{i=0}^{23} C_{i,i}}{\sum_{i=0}^{23}\sum_{j=0}^{23} C_{i,j}}. \quad (2.1)$$

Precision визначає частку позитивних передбачень, які дійсно є позитивними: Чим вище значення Precision , то тим меншим є кількість хибних позитивних відповідей. Формула Precision для кожного класу окремо:

$$\text{Precision}_i = \frac{C_{i,i}}{\sum_{k=0,\text{k}\neq\text{i}}^{23} C_{k,i} + C_{i,i}}. \quad (2.2)$$

Формула Precision для всієї вибірки:

$$\textit{P}\text{recision} = \frac{1}{23}\sum_{i=0.}^{23} \text{Precision}_i \quad (2.3)$$

Recall визначає частку дійсних позитивних передбачень, які були правильно визначені моделлю серед усіх дійсно позитивних прикладів. Recall показує частку дійсно позитивних результатів, серед усіх позитивних прикладів. Формула Recall для кожного класу окремо:

$$\text{Recall}_{\ i} = \frac{C_{i,i}}{\sum_{k=0,,\text{k}\neq\text{i}}^{23} C_{i,k}+C_{i,i}}. \quad (2.3)$$

Формула Recall для загальної вибірки:

$$\text{Recall} = \frac{1}{23}\sum_{i=0.}^{23} \text{Recall}_i \quad (2.4)$$

F1-score – це метрика, що використовується для оцінювання балансу між точністю та повнотою (тобто саме precision та recall):

$$\mathrm{F1-score} = 2 * \frac{\mathrm{Recall} \times \mathrm{Precision}}{\mathrm{Recall} + \mathrm{Precision}} \quad (2.5)$$

2.4 Стандартизація даних

Так як деякі з алгоритмів машинного навчання, я обрав особливо чутливі до величин ознак (SVM), то цій у роботі було проведена стандартизація даних. Для більш чесних результатів, її було проведено для всіх з алгоритмів машинного навчання, які були досліджені.

Стандартизація (Standardization): Стандартизує ознаки, видаливши середнє значення та масштабуючи до одиничної дисперсії [28]:

$$X_{scaled} = \frac{X - \mu}{\sigma}. \quad (2.6)$$

де

– σ – стандартне відхилення ознаки,

– X – вхідне значення,

– μ – середнє значення вибірки,

– X_{scaled} – масштабоване значення.

Стандартизація була проведена завдяки методу StandardScaler у бібліотеці scikit-learn. Вона забезпечує точніший аналіз моєї моделі.

2.5 Отримані результати без вегетаційних індексів

Для більш глибокого і точного вимірювання якості побудованих ММН, ми будемо обрахувати декілька F1-score і Accuracy, а саме: загальний (для всіх test), для знімків, з яких бралися навчальні дані (test1, test2, test3, test4, test6. Для спрощення будемо називати її навчальною групою) і для знімків, з яких ці навчальні дані не бралися (test5, test7, test9. Будемо назвати її тестовою

групою). Їх значення теж можна буде побачити у таблицях для відповідних алгоритмів Також хочу зауважити, що test8 не було, одразу був test9. Це вийшло через деякі технічні проблеми. Також у порівнянні буде враховуватися час роботи моделі.

Для наглядності матрицю помилок ми будемо показувати з того ж знімку, по якому були приклади вегетаційних індексів (тобто test1).

Нижче у табл 2.7 можете побачити результати точності всіх моделей:

Таблиця 2.7 Результати для порівняння моделей (без вегетаційних показників)

Модель	**Кількість даних для навчання**	**F1-score**	**Accuracy**	**Час на Тренування, с**
RF	874385	0,8301	0,8361	389
LG		0.7879	0,7919	329
XGBoost		0,8292	0,8331	311
SVM (лін ядро)		0,7830	0,7984	542
SVM (нелін ядро)	200000	0,8365	0,8411	1223

Як ми бачимо найкращу точність показала SVM модель з нелінійним ядром з зменшеною кількістю навчальних даних, але вона і була найповільнішою. По відношенню якість-швидкість найкращими моделями стали RF XGBoost. Саме б ці дві моделі я б рекомендував би з усіх тих, які проводилися у дослідженні для подібних робіт. Хочу зауважити, що так як кожна з моделей повинна розглянути велику вибірку з різними культурами, то точність їхнього аналізу може сильно відрізнятися один від одного. Майже

в кожній моделі і для кожного тесту точність мажоритарних агрокультур (пшениця, соняшник) була на 5-10 пунктів вища за точність у своїй вибірці, бо і навчальних даних було набагато більше саме в цих культур в порівнянні з іншими через їхню високу популярність у наших широтах.

Модель RF показала дуже високу точність у тестовій вибірці і сильно гіршу у практичній (меншу на 10,61%). Швидкість навчання моделі була високою і навантаження на систему незначним. Результати RF для кожного тесту можна побачити у табл. 2.8 :

Таблиця 2.8 Результати Random Forest для кожного тесту (без вегетаційних індексів)

Номер тесту	**F1-score**	**Accuracy**	**Час роботи, с**	**Кількість значень**	**F1-score груп**	**Accuracy**
Test 1	0,8795	0,8855	5,89	450424	0,8969	0,8985
Test 2	0,8801	0,8809	5,92	444810		
Test 3	0,9347	0,9387	5,22	425242		
Test 4	0,9211	0,9281	5,15	442589		
Test 6	0,8869	0,8889	34,74	1734480		
Test 5	0,8203	0,8428	4,94	542956	0,7778	0,7924
Test 7	0,7657	0,7907	29,19	2216844		
Test 9	0,7616	0,7819	24,38	2231873		

З матриці помилок RF для test1 (рис 2.8). Можна побачити, що модель крім мажоритарних культур добре розрізняє воду, ліс і кукурудзу. Для інших тесті ситуація є схожою.

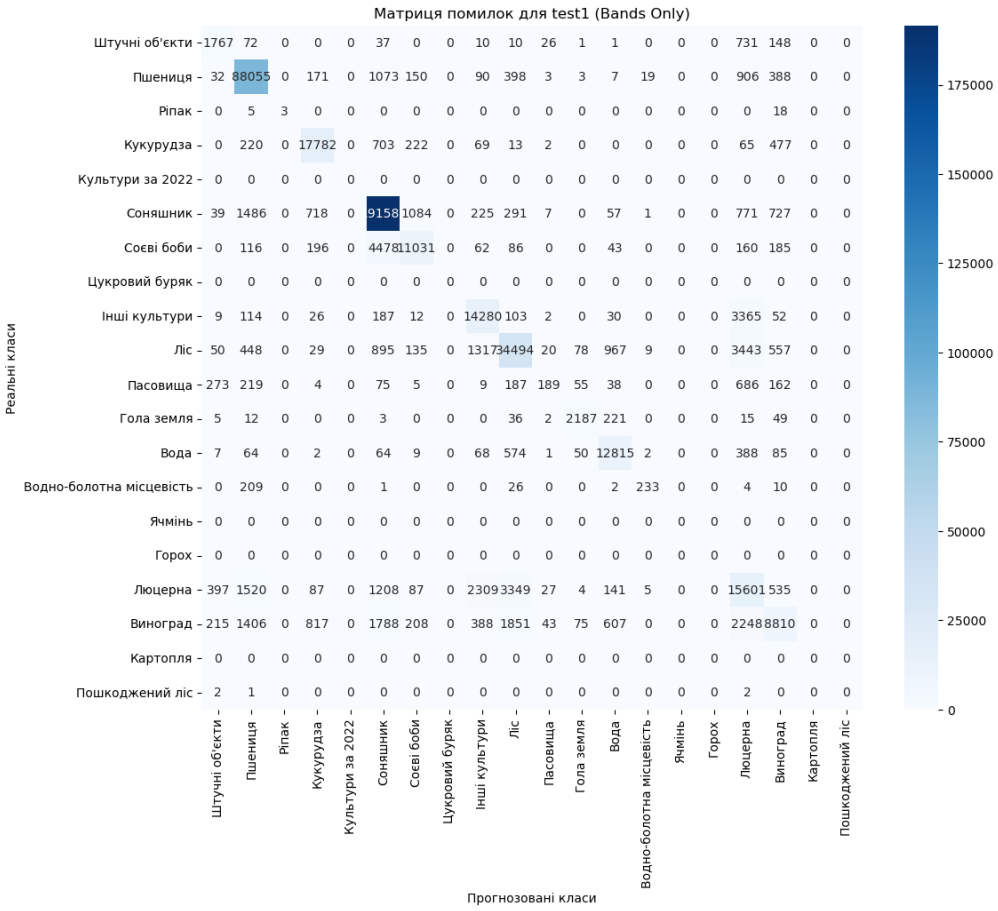


Рис 2.8 Матриця помилок test1 для RF

Логістична регресія (табл. 2.9) показала високу швидкість, але мала найгіршу точність. Це пов’язано з пошуком лінійної залежності, яку, як правило, складно знайти на супутникових знімках з великою кількість даних для обробки.

Таблиця 2.9 Результати Logistic regression для кожного тесту (без вегетаційних індексів)

Номер тесту	**F1-score**	**Accuracy**	**Час роботи, с**	**Кількість значень**	**F1-score груп**	**Accuracy**
Test 1	0,8223	0,8253	0,28	450424	0,8251	0,8308
Test 2	0,7968	0,7995	0,24	444810		
Test 3	0,8826	0,8896	0,29	425242		
Test 4	0,8659	0,8700	0,25	442589		
Test 6	0,8102	0,8118	0,96	1734480		

Продовження таблиці 2.9						
Test 5	0,8321	0,8348	0,31	542956	0,7608	0,7647
Test 7	0,7689	0,7712	1,22	2216844		
Test 9	0,7401	0,7410	1,32	2231873		

Логістична регресія (Рис 2.9) має ту ж саму ситуацію як і RF.

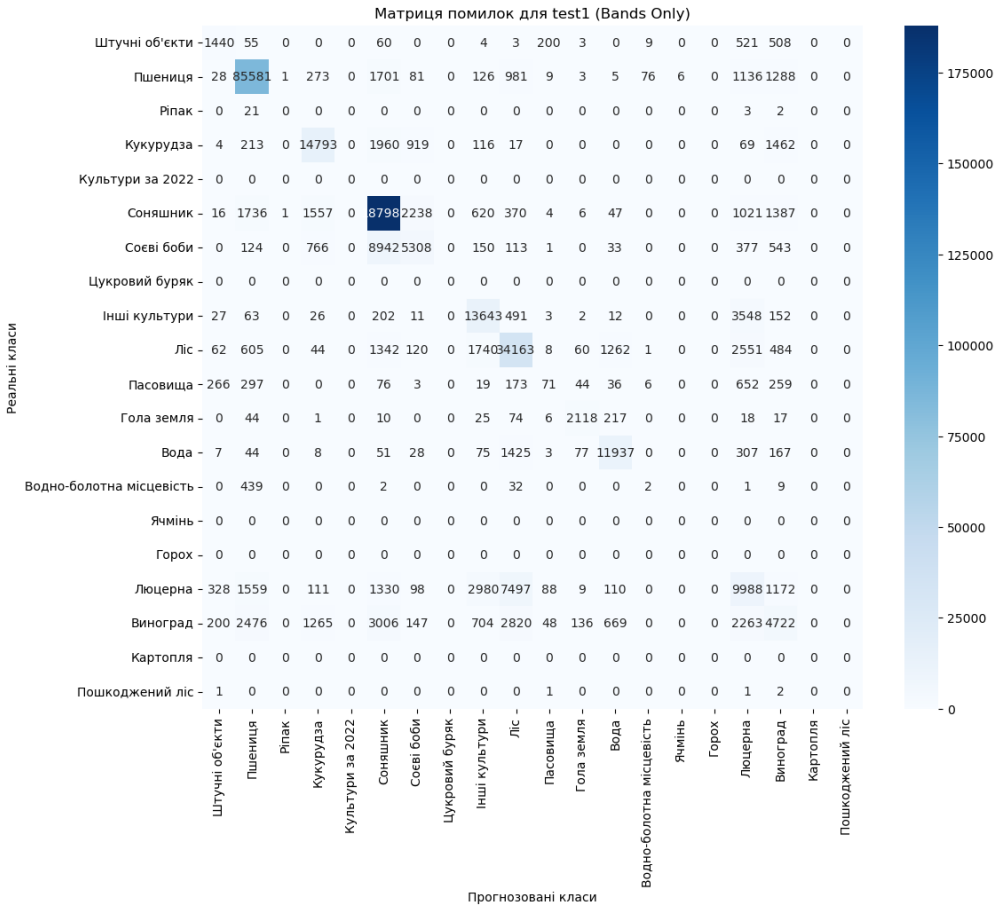


Рис 2.9 Матриця помилок test1 для LG

XGBoost (табл. 2.10) показала найкращу швидкість навчання в порівнянні з іншими досліджуваними ММН і високою швидкістю роботи, хоча різниця в точності між тестовими і навчальними вибірками була менша ніж в RF (на 8,79%).

Таблиця 2.10 Результати XGBoost для кожного тесту (без вегетаційних індексів)

Номер тесту	F1-score	Accuracy	Час роботи, с	Кількість значень	F1-score груп	Accuracy
Test 1	0,8715	0,8725	11,27	450424	0,8838	0,8848
Test 2	0,8587	0,8593	11,14	444810		
Test 3	0,9271	0,9298	10,41	425242		
Test 4	0,9209	0,9213	11,39	442589		
Test 6	0,8602	0,8740	45,10	1734480		
Test 5	0,8486	0,8489	11,39	542956	0,7871	0,7969
Test 7	0,7610	0,7911	45,87	2216844		
Test 9	0,7810	0,7899	46,13	2231873		

Як і всіх інших ММН найбільше інших культур в XGBoost (рис. 2.10) попадає в люцерну і виноград .

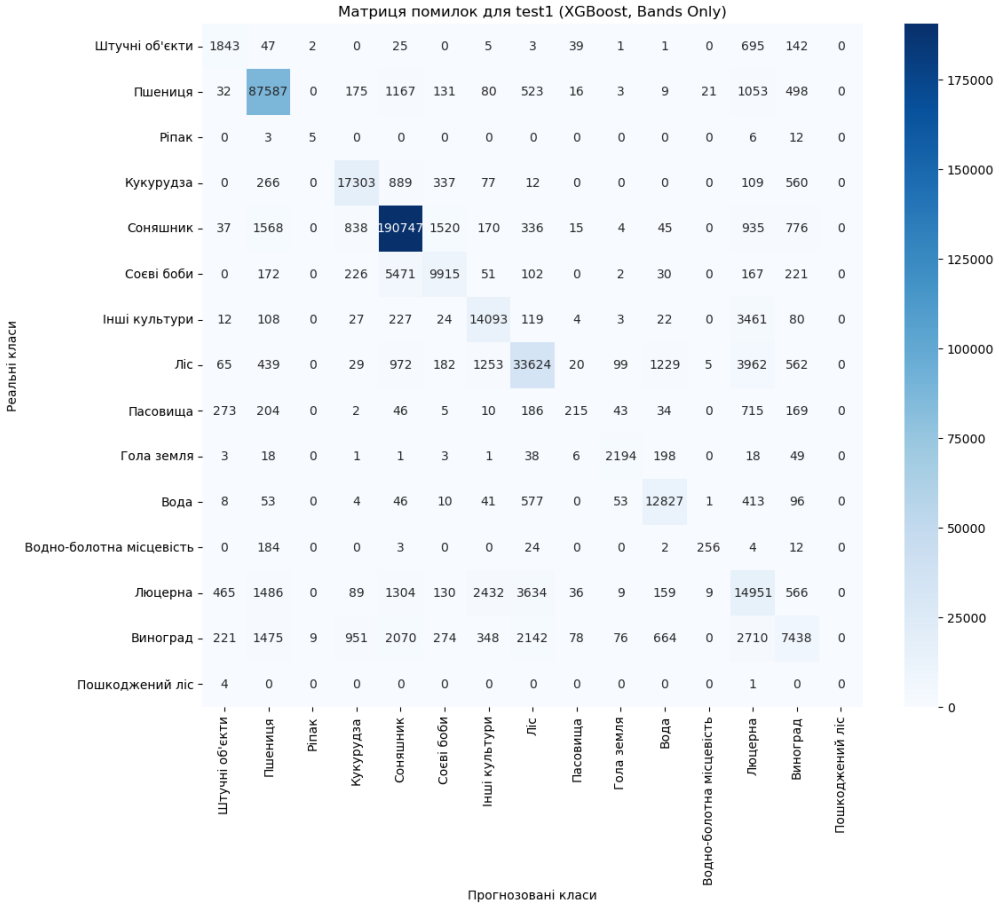


Рис 2.10 Матриця помилок test1 для XGBoost

SVM з лінійним ядром (табл. 2.11) має результати схожі на логістичну регресію. Низька точність, хоча трохи вища за LG , але більш повільна швидкість навчання і роботи. Різниця у Accuracy між навчальною і тестовою групою порівняно невелика (на 4,1%).

Таблиця 2.11 Результати SVM для кожного тесту з лінійним ядром (без вегетаційних індексів)

Номер тесту	**F1-score**	**Accuracy**	**Час роботи, с**	**Кількість значень**	**F1-score груп**	**Accuracy**
Test 1	0,7975	0,8117	0,25	450424	0,8061	0,8225
Test 2	0,7787	0,7909	0,23	444810		
Test 3	0,8771	0,8919	0,24	425242		
Test 4	0,8442	0,8562	0,25	442589		
Test 6	0,7899	0,8061	0,94	1734480		
Test 5	0,8156	0,8279	0,29	542956	0,7646	0,7815
Test 7	0,7697	0,7855	1,34	2216844		
Test 9	0,7490	0,7663	1,25	2231873		

SVM з лініним ядром (Рис 2.11) дуже сильно плутав ліс і люцерну й взагалі показав найгіршу accuracy і F1-score для цієї області з усіх ММН.

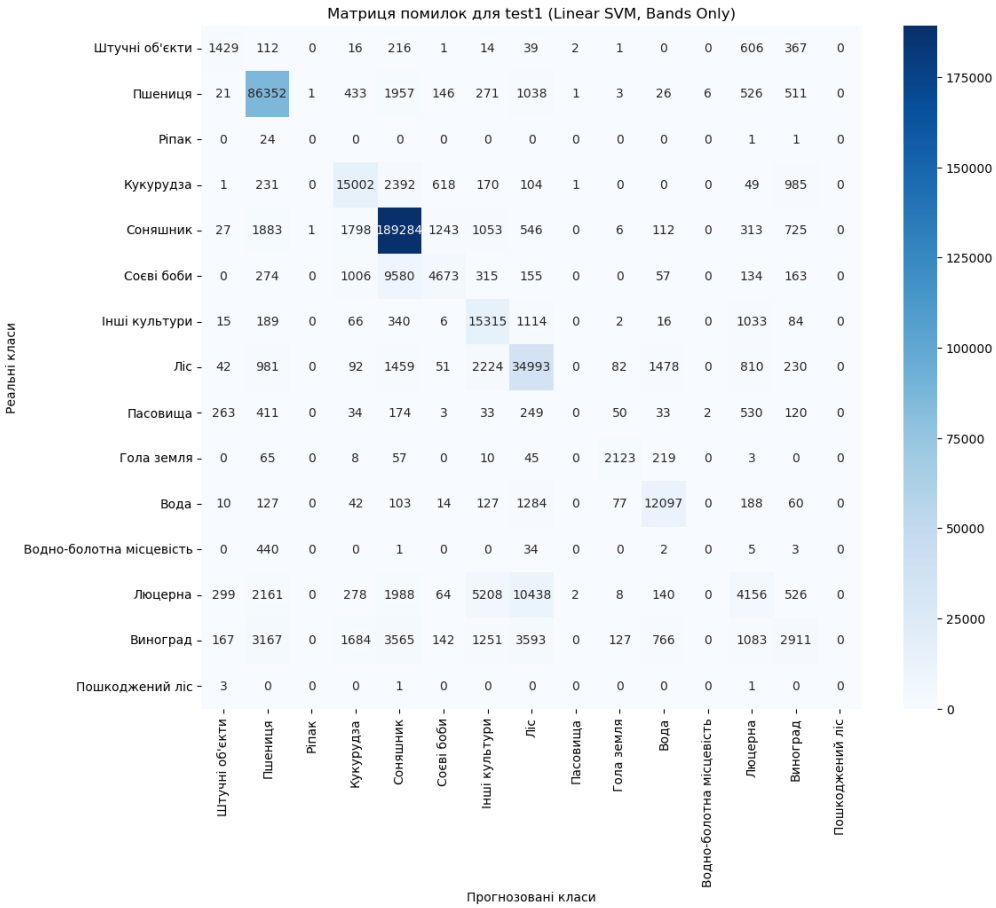


Рис 2.11 Матриця помилок test1 для SVM з лінійним ядром без вегетативних індексів

Через дуже високу обчислювальну складність SVM з не лінійним ядром (табл 2.12) було вирішено зменшити навчальну вибірку до 80000 значень до кожного знімку. Алгоритм показав найкращу точність з усіх. Хоча точність і F1-score в навчальні вибірці була менша на 2,9% з RF, то точність і F1-score у тестовій вибірці навпаки вже були кращі цій моделі на 0,5% і 2%.

Таблиця 2.12 Результати SVM для кожного тесту з нелінійним ядром (без вегетаційних індексів)

Номер тесту	**F1-score**	**Accuracy**	**Час роботи, с**	**Кількість значень**	**F1-score груп**	**Accuracy**
Test 1	0,8498	0,8530	691,48	80000	0,8656	0,87002
Test 2	0,8302	0,8349	697,34	80000		
Test 3	0,9136	0,9178	693,12	80000		

Продовження таблиці 2.12						
Test 4	0,8992	0,9040	692,39	80000		
Test 6	0,8337	0,8404	694,76	80000		
Test 5	0,7906	0,7970	694,41	80000	0,7881	0,7938
Test 7	0,7775	0,7824	695,64	80000		
Test 9	0,8001	0,8020	698,19	80000		

Результати SVM неліним ядром (рис. 2.12) для test1 були трохи гірші ніж в RF.

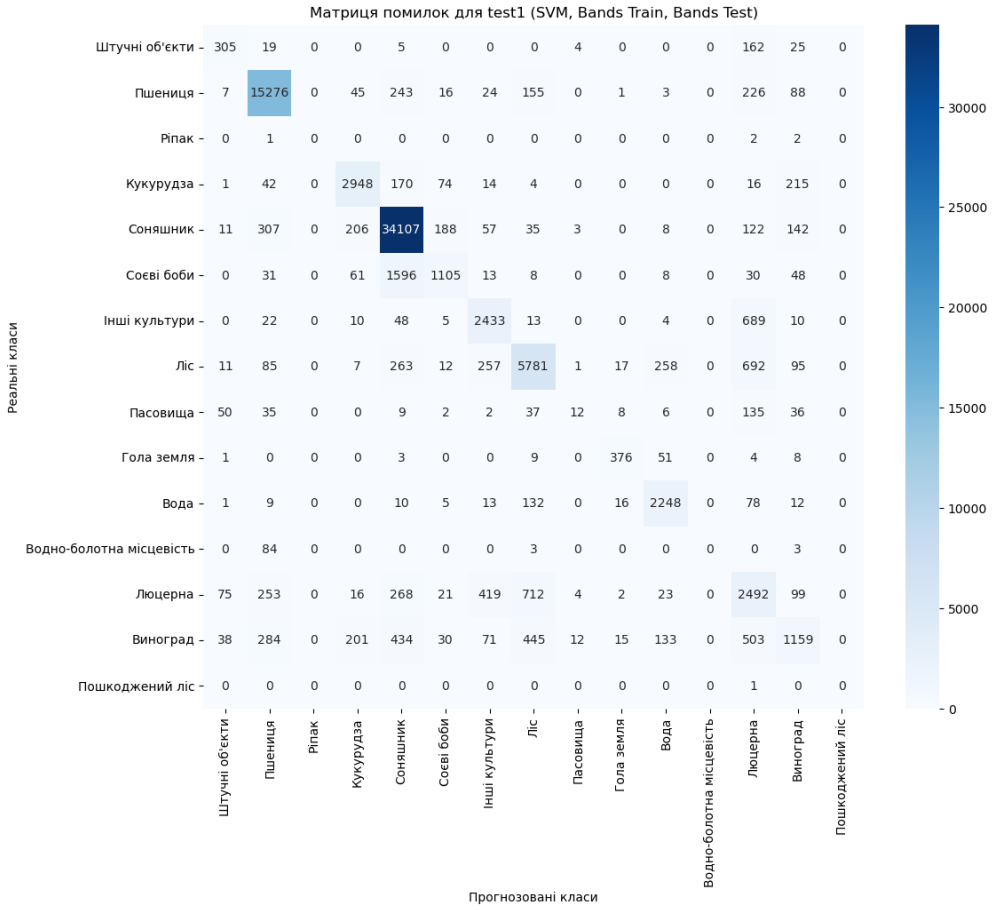


Рис 2.12 Матриця помилок test1 для SVM з нелінійним ядром

2.6 Отримані результати з вегетаційними індексами

Для порівняння результатів ми будемо використовувати ті ж самі метрики, що й для даних без вегетаційних показників. Також оцінимо їхній вплив на точність і порівняємо з попередніми результатами ММН.

Нижче у табл 2.13 можете побачити результати точності всіх моделей з вегетаційними індексами:

Таблиця 2.13 Результати для порівняння моделей (з вегетаційними індексами)

Модель	**Кількість даних для навчання**	**F1-score**	**Accuracy**	**Час на Тренування, с**
RF	874385	0,8299	0,8358	559
LG		0.7927	0,7966	512
XGBoost		0,8314	0,8352	351
SVM (лін ядро)		0,7912	0,8035	935
SVM (нелін ядро)	200000	0,8385	0,8432	1331

Як ми бачимо (табл. 2.14) загальна точність RF зменшилась на 0,03%, а F1-score на 0,02%. Притому, що час роботи суттєво збільшився. Через більшу кількість даних, завдяки вегетаційним, індексам на кожний піксель. Модель почала демонструвати більшу різницю в точності і F1-score ніж в попередньому RF на 0,11% і 0,12% при збільшенні часу роботи на декілька секунд до кожного знімку.

Таблиця 2.14 Результати Random Forest для кожного тесту (з вегетаційними індексами)

Номер тесту	F1-score	Accuracy	Час роботи, с	Кількість значень	F1-score груп	Accuracy
Test 1	0,8775	0,8821	7,01	450424	0,8935	0,8951
Test 2	0,8751	0,8762	6,81	444810		
Test 3	0,9347	0,9365	6,27	425242		
Test 4	0,9206	0,9256	5,73	442589		
Test 6	0,8829	0,8854	33,19	1734480		
Test 5	0,8301	0,8426	5,19	542956	0,7800	0,7943
Test 7	0,7767	0,7926	28,86	2216844		
Test 9	0,7659	0,7819	27,86	2231873		

Точність аналіз різних сільськогосподарських культур до test1 (рис 2.13) сильно не змінилася.

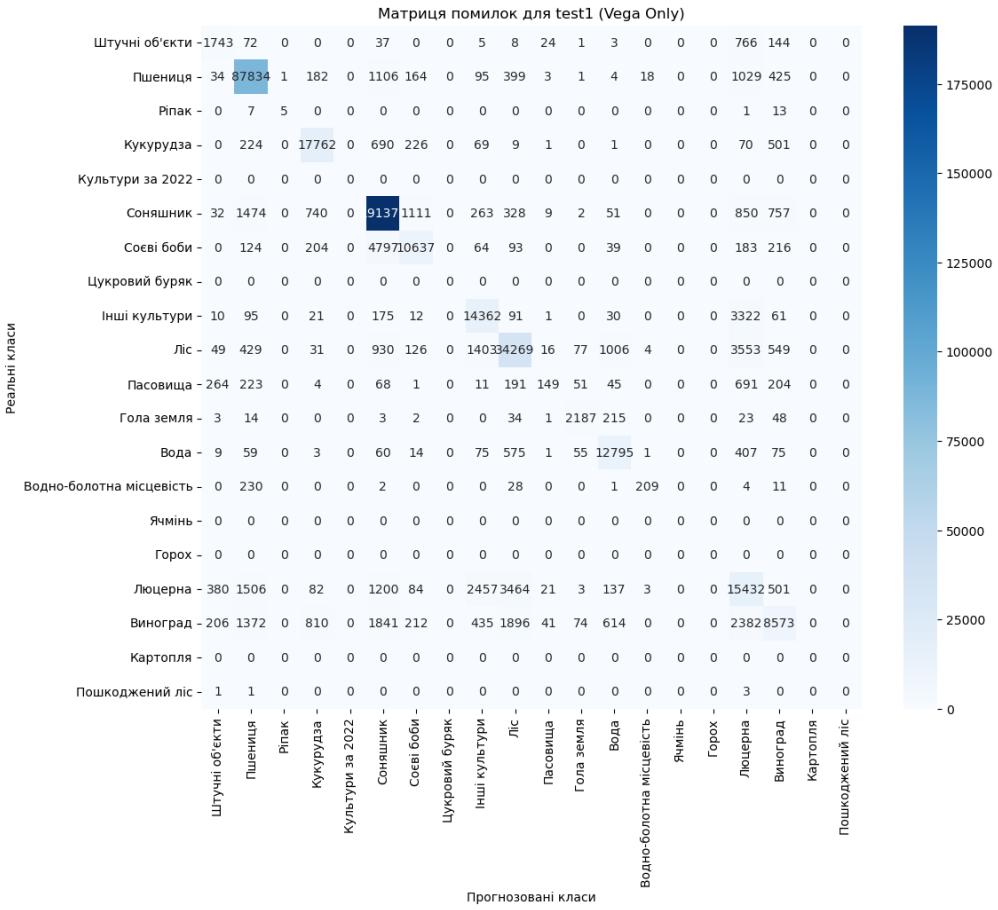


Рис 2.13 Матриця помилок test1 для RF з вегетаційними індексами

При сильно зросту часу LG загальна точність і F1-score збільшилась на 0,84% і 0,48%. Така ситуація і тестовій і навчальній вибірці (табл 2.15). Час роботи даної моделі залишається високим. Різниця F1-score і Accuracy в порівнянні з іншими моделями є невеликою і навчальній групі і тестовій групі.

Таблиця 2.15 Результати Logistic regression для кожного тесту (з вегетаційними індексами)

Номер тесту	**F1-score**	**Accuracy**	**Час роботи, с**	**Кількість значень**	**F1-score груп**	**Accuracy**
Test 1	0,8243	0,8293	0,41	450424	0,8320	0,8371
Test 2	0,7997	0,8003	0,32	444810		
Test 3	0,8978	0,8998	0,34	425242		
Test 4	0,8699	0,8743	0,30	442589		
Test 6	0,8204	0,8226	1,37	1734480		
Test 5	0,8332	0,8361	0,41	542956	0,7645	0,7681
Test 7	0,7702	0,7743	1,47	2216844		
Test 9	0,7423	0,7454	1,53	2231873		

Ситуація з матрицею помилок LG (рис 2.14) для test 1 суттєво не змінилася.

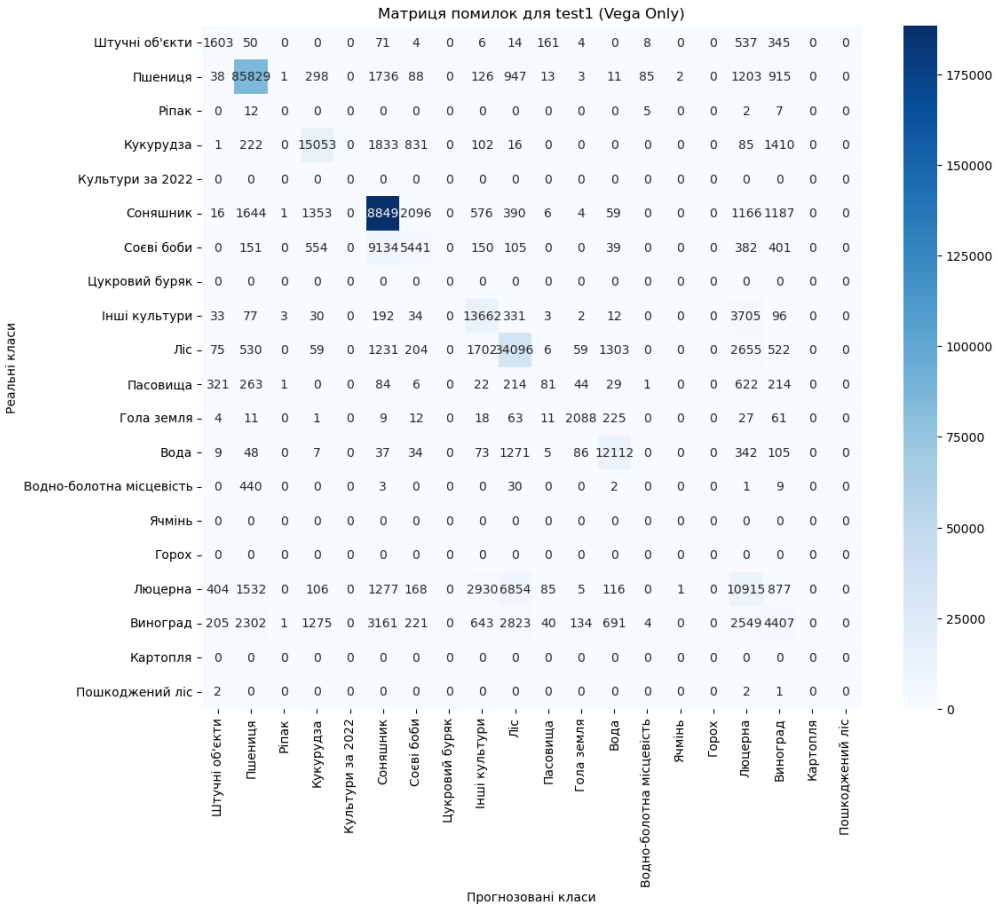


Рис 2.14 Матриця помилок test1 для LG з вегетаційними індексами

Час роботи XGBoost суттєво збільшився, що показує, що ця модель добре працює з великими даними. Точність і F1-score для навчальної вибірки (табл 2.16) зріз на 0,03% і 0,02%, а тестової вибірки зріст трохи більше на 0,36% і 0,33%.

Таблиця 2.16 Результати XGBoost для кожного тесту (з вегетаційними індексами)

Номер тесту	**F1-score**	**Accuracy**	**Час роботи, с**	**Кількість значень**	**F1-score груп**	**Accuracy**
Test 1	0,8718	0,8726	11,61	450424	0,8841	0,8850
Test 2	0,8589	0,8603	12,12	444810		
Test 3	0,9271	0,9300	10,73	425242		
Test 4	0,9211	0,9221	11,47	442589		
Test 6	0,8714	0,8745	46,69	1734480		

Продовження таблиці 2.16						
Test 5	0,8426	0,8457	11,89	542956	0,7908	0,8002
Test 7	0,7714	0,7988	49,78	2216844		
Test 9	0,7810	0,7900	51,46	2231873		

Матриця помилок XGBoost (рис 2.15) суттєво не змінилася, хоча модель стала трохи більше плутати водно-болотну місцевість і ліс.

Рис 2.15 Матриця помилок test1 для XGBoost з вегетаційними індексами

Точність і F1-score SVM моделі з лінійним ядром для навчальної вибірки (табл. 2.17) виросла на 0,77% і 1,06%, тоді як F1-score тестової вибірки зріс на 0,73%, а точність зросла на 0,34%. При значно довшій швидкості навчання моделі.

Таблиця 2.17 Результати SVM для кожного тесту з лінійним ядром(з вегетаційними індексами)

Номер тесту	F1-score	Accuracy	Час роботи, с	Кількість значень	F1-score груп	Accuracy
Test 1	0,8175	0,8248	0,40	450424	0,8167	0,8302
Test 2	0,7987	0,7958	0,30	444810		
Test 3	0,8884	0,8958	0,28	425242		
Test 4	0,8532	0,8673	0,29	442589		
Test 6	0,8076	0,8146	1,16	1734480		
Test 5	0,8191	0,83336	0,39	542956	0,7719	0,7849
Test 7	0,7673	0,7889	1,68	2216844		
Test 9	0,7518	0,7688	1,53	2231873		

Матриця помилок для цієї SVM моделі суттєво не змінилася (рис 2.16).

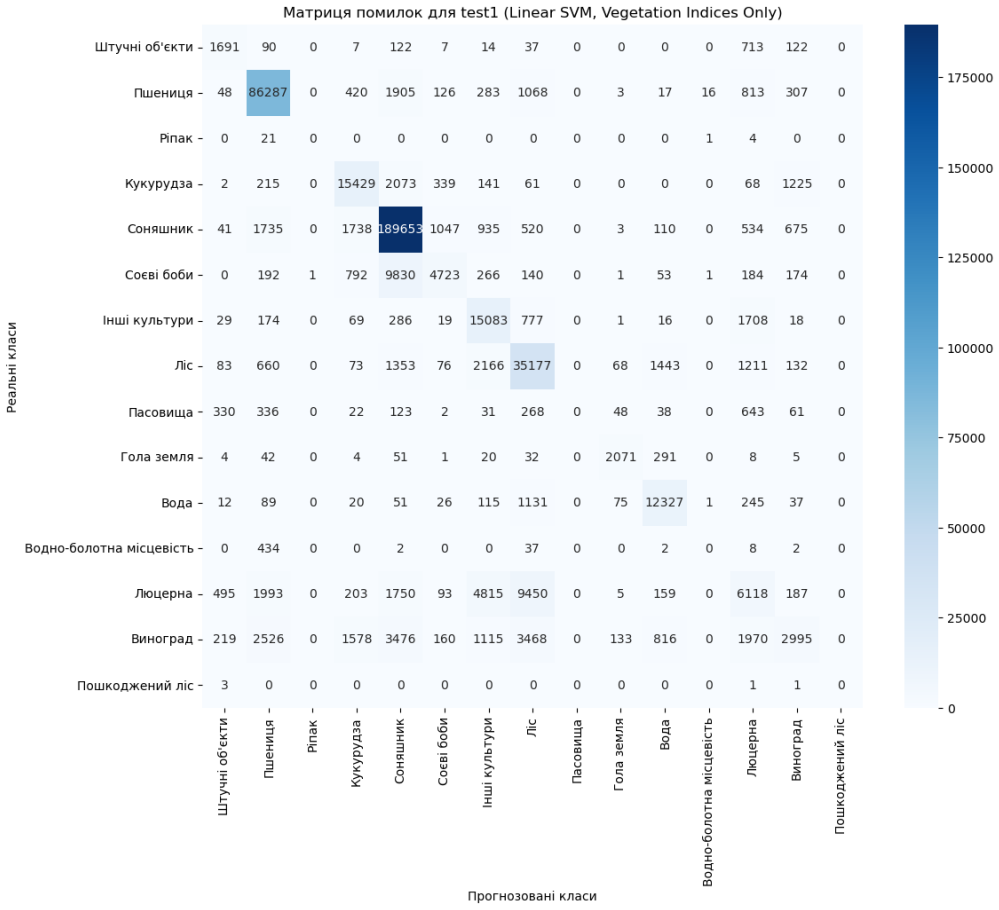


Рис 2.16 Матриця помилок test1 для SVM з лінійним ядром і з вегетаційними індексами

Точність SVM моделі з нелінійним ядром (табл. 2.18) для навчальної вибірки не змінилась, а F1-score збільшився на 0,14%/, а для тестової вибірки F1-score і точність зросли на 0,09% і 0,43%. Ця модель і зараз показує найкращу точність хоч і є найповільнішою з усіх, які ми брали.

Таблиця 2.18 Результати SVM для кожного тесту з нелінійним ядром(з вегетаційними індексами)

Номер тесту	**F1-score**	**Accuracy**	**Час роботи, с**	**Кількість значень**	**F1-score груп**	**Accuracy**
Test 1	0,8501	0,8532	792,31	80000	0,8670	0,8702
Test 2	0,8314	0,8352	798,81	80000		
Test 3	0,9139	0,9195	796,27	80000		
Test 4	0,9005	0,9040	785,57	80000		
Test 6	0,8369	0,8394	784,93	80000		
Test 5	0,7903	0,8018	787,06	80000	0,7889	0,7981
Test 7	0,7857	0,7919	786,21	80000		
Test 9	0,7903	0,8007	786,86	80000		

Результи SVM неліним ядром (рис. 2.17) для test1 можна побачити нижче.

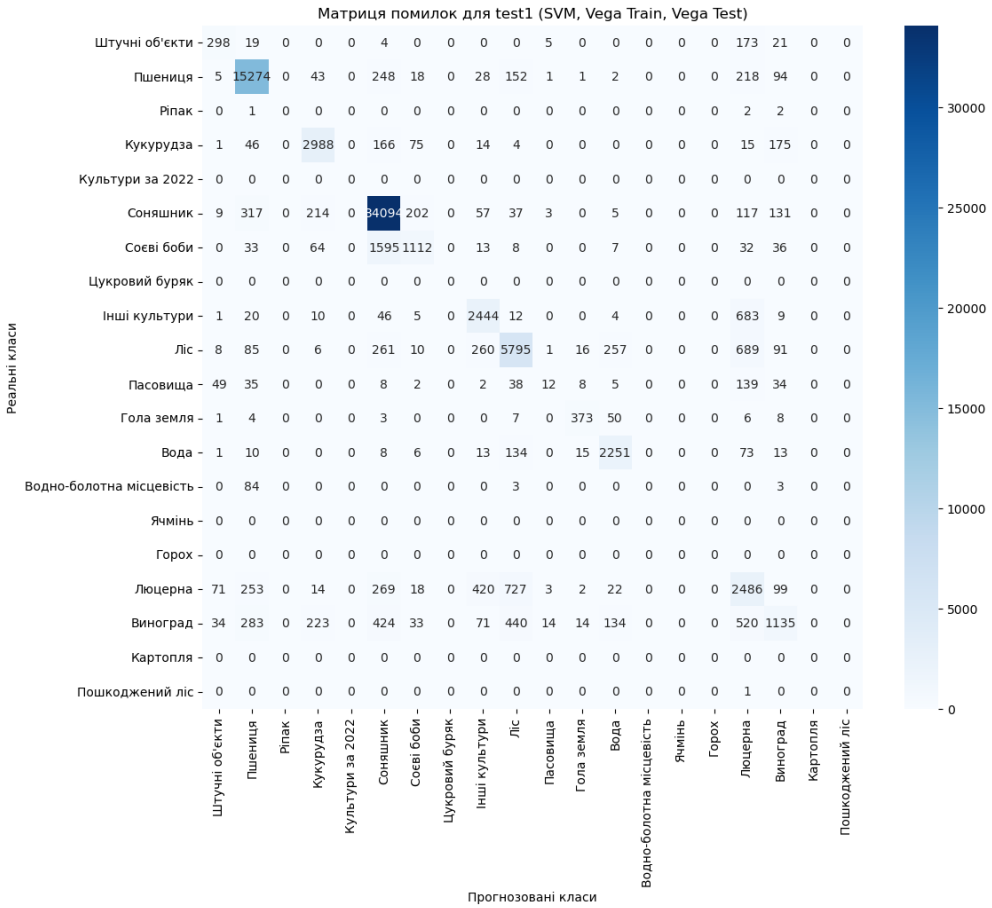


Рис 2.17 Матриця помилок test 1 для SVM з нелінійним ядром і з вегетативними індексами

2.7 Висновки до розділу 2

У цьому розділі були обрані області сільськогосподарських полів для дослідження, був зроблений опис даних, підготовлені датасети і маски Ground Truth Data для цих знімків.

Був описаний принцип додавання і обрахунків вегетаційних індексів і розділення 5 знімків на навчальні і тестові частини (NDVI, NDWI, NDMI, NDRE). Був зроблений процес стандартизації даних.

Було порівняно вплив вегетаційних індексів на точність, F1-score і швидкість моделей. Виявилося, що загалом ці індекси не мали суттєвого впливу на точність і F1-score, хоча суттєво збільшували швидкість навчання і роботи моделей.

ВИСНОВКИ

Зважаючи на сьогоднішню ситуацію, сільське господарство в нашій країні є однією з найголовніших і найстабільніших сфер. Методи аналізу сільськогосподарських полів через супутникові знімки стає ще більш важливим.

У дипломному проєкті була описана важливість моніторингу сільськогосподарських полів, описані методи їхнього аналіз від старих до більш сучасних. А також приділена увага – дистанційному зондуванню, зокрема через супутникові знімки Sentinel-2 у 12 спектральних каналах. Для цього були використанні методи машинного навчання, а саме модель: Random Forest, Логічна регресія, Support Vector Machine і XGBoost.

Також були використанні вегетаційні індекси для порівняння їхнього впливу на точність наших моделей, а саме: NDVI, NDWI, NDMI, NDRE. Для порівняння їхнього впливу були створенні 2 окремі датасети. В кожному з яких були 8 менших баз даних (у нашому аналізі вони називалися test) з ними і без них. Для цього був використаний піксельний метод, де в кожному пікселі була інформація про 12 спектральних знімків для кожного знімка місяця.

Були використані супутникові знімки протягом вегетаційного сезону (квітень - жовтень), було обрано по 1 знімку протягом цих місяців, тобто було використано по 7 знімків до кожної з областей аналізу і створений датасет на основі цих знімків. Для 2 датасету, і для кожного з цих знімків були пораховані вегетативні індекси.

Вплив вегетативних індексів на точність наших моделей виявився незначним, хоча швидкість більшості з моделе суттєво зросла (крім XGBoost). Це можна пов’язати з тим, що і без них наші моделі отримували велику кількість даних для аналізу (без вегетативних індексів по 84 значення на кожний піксель).

Отримані результати показують, що методи машинного навчання є дуже ефективними у аналізі сільськогосподарських полів, хоч для кожної з цілей потрібно брати найефективніший з методів. Для аналізу майбутнього врожаю і тенденцій в зміні посаджень фермерів це є актуальною задачею

Найкращою моделлю по відношенні час/швидкість вийшли Random Forest і XGBoost, причому для більших даних час навчання XGBoost збільшувався повільніше ніж у Random Forest. Найкраща точність була у SVM з нелінійним ядром, але через велику обчислювальну складність ця модель була найповільнішою з усіх, які ми розглядували, незважаючи на те, що навчальна і тестова вибірка цієї моделі були суттєво зменшені.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Authors (2019), 'Unmanned Aerial Vehicles in Agriculture: A Review of Perspective of Platform Control and Applications', *ResearchGate*. Available from: https://www.researchgate.net/publication/334778584_Unmanned_Aerial_Vehicles_in_Agriculture_A_Review_of_Perspective_of_Platform_Control_and_Applications [Дата звернення: 26 квітня 2025

2. groPro Drone (без дати). *Field monitoring*. [Електронний ресурс]. Доступно за адресою: https://agropro-drone.eu/?page_id=1048 [Дата звернення: 28 квітня 2025].

3. Geoawesome (2024). *The Critical Role of Ground-Based Data in Regression Model Accuracy for Remote Sensing Applications*. [Електронний ресурс]. Доступно за адресою: https://geoawesome.com/eo-hub/the-critical-role-of-ground-based-data-in-regression-model-accuracy-for-remote-sensing-applications/ [Дата звернення: 27 квітня 2025].

4. *Пропозиція* (2023), 'Супутниковий моніторинг підвищує врожайність сільськогосподарських культур', *Журнал Пропозиція*. Available from: https://propozitsiya.com/ua/suputnykovyy-monitoryng-pidvyshchuye-vrozhaynist-silskogospodarskyh-kultu [Дата звернення: 1 травня 2025].

5. Forkuor, G., Conrad, C., Thiel, M., Ullmann, T. and Zoungrana, E., (2018). Mapping crop types in heterogeneous agricultural landscapes of West Africa using Sentinel-2 time series data and machine learning algorithms. *Agriculture, Ecosystems & Environment*, 265, pp.160-169.

6. Diedrichsen, A., Musial, J. P., Fröhlich, M. & Plass, R. (2018), 'Using Sentinel-2 Data for Crop Type Classification: An Object-Based Approach', *Remote Sensing*, vol. 10, no. 11, article no. 1700. Available from: https://doi.org/10.3390/rs10111700 [Дата звернення: 6 травня 2025].

7. Stehman, S.V. and Foody, G.M., (2019). Key issues in accuracy assessment of land cover and land cover change. *Remote Sensing of Environment*, 231, p.111299.

8. Інформаційна система Державного аграрного реєстру (ІС ДАР), (2025). *Карта земельних ділянок в Інформаційній системі Державного аграрного реєстру*. [Онлайн]. Доступно за адресою: https://reg.dar.gov.ua/farmer/landparcelsonmap [Дата звернення: 15 травня 2025

9. *EAPK* (2023), 'Моніторинг сільськогосподарських угідь із застосуванням систем дистанційного зондування земель', *https://www.google.com/search?q=eapk.com.ua*. Available from: https://https://www.google.com/search?q=eapk.com.ua/en/article/read/monitoring-silskogospodarskikh-ugid-iz-zastosuvannyam-sistem-distantsiynogo-zonduvannya-zemel Дата звернення: 1 травня 2025].

10. Copernicus Sentinel-2 (processed by ESA) 2021, *MSI Level-1C TOA Reflectance Product. Collection 1*. European Space Agency. Available from: [https://sentiw iki.copernicus.eu/web/s2-mission]

11. European Space Agency. (n.d.). S2 Mission. [online] Available at: https://sentiwiki.copernicus.eu/web/s2-mission [Дата звернення: 28 квітня 2025]

12. ESA 2023, 'Introducing Sentinel-2', *European Space Agency*. Available from: https://www.esa.int/Applications/Observing_the_Earth/Copernicus/Sentinel-2/Introducing_Sentinel-2 Дата звернення: 1 травня 2025]

13. ESA 2023, *Sentinel-2 Products Specification Document - Version 14.9*, European Space Agency. Available from: https://sentinels.copernicus.eu/documents/247904/0/Sentinel-2-product-specifications-document-V14-9.pdf [Дата звернення: 28 квітня 2025].

14. Jensen, J. R. (2007). *Remote Sensing of the Environment: An Earth Resource Perspective*. 2nd ed. Pearson Prentice Hall.

15. Salii, Y., Lavreniuk, A. and Kussul, N. (2024) «Statistical methods of feature engineering for the problem of forest state classification using satellite data», 61 System research and information technologies. Igor Sikorsky Kyiv Polytechnic Institute, (1), pp. 86–98. doi: 10.20535/srit.2308-8893.2024.1.07.

16. Normalized Difference Vegetation Index (NDVI) - Landscape Toolbox, https://www.landscapetoolbox.org/remote-sensing-methods/normalized-difference-vegetation-index-ndvi/ [Дата звернення: 3 травня 2025]

17. USGS 2023, 'Normalized Difference Moisture Index (NDMI)', *U.S. Geological Survey*. Available from: https://www.usgs.gov/landsat-missions/normalized-difference-moisture-index [Дата звернення: 3 травня 2025].

18. Hub4Everybody 2023, 'Normalized Difference Red Edge Index (NDRE)', j*hub4everybody.com*. Available from: https://hub4everybody.com/normalized-difference-red-edge-index-ndre/?hs-x=16.92351&hs-y=49.17101&hs-z=11&hs-lang=en&hs-visible-layers=OpenStreetMap&map-swipe=disabled&app=PPnZvvlNqv [Дата звернення: 3 травня 2025].

19. The Classification Method Study of Crops Remote Sensing with Deep Learning, Machine Learning, and Google Earth Engine - MDPI, [Дата звернення: 28 квітня 2025], https://www.mdpi.com/2072-4292/14/12/2758

20. Shelestov, A. Yu., Yailymov, B., Yailymova, H., Bilokonska, Y., & Nivievskyi, O. V. (2020). Satellite crop monitoring for Ukraine. *ResearchGate*. [Онлайн]. Доступно за: https://www.researchgate.net/publication/349319303_Satellite_crop_monitoring_for_Ukraine [Дата звернення: 3 травня 2025].

21. Hastie, T., Tibshirani, R., & Friedman, J. (2008). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.

22. Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5-32.

scikit-learn developers. (2025). *Scikit-learn User Guide: 1.10-1.11. Decision Trees*. Available at: https://scikit-learn.org/stable/modules/tree.html (Дата звернення: 10 травня 2025).

23. Logistic Regression - ML Explained, 2025, https://ml-explained.com/blog/logistic-regression-explained [Дата звернення 10 травня 2025]

24. What Is Logistic Regression? | IBM,Доступно за https://www.ibm.com/think/topics/logistic-regression [Дата звернення 10 травня 2025]

25. Support vector machines (SVMs) - IBM, https://www.ibm.com/think/topics/support-vector-machine [Дата звернення 10 травня 2025]

26. . Scikit-learn developers, (2025) «Kernel functions. scikit-learn». Available at: https://scikit-learn.org/stable/modules/svm.html#kernel-functions [Дата звернення 10 травня 2025].

27. Chen, T. and Guestrin, C., 2016. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785-794. Available at: https://doi.org/10.1145/2939672.2939785

28. Scikit-learn developers 2024, *sklearn.preprocessing.StandardScaler*, scikit-learn. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html [Дата звернення 22 May 2025].

29. Copernicus Data Space Browser, 2025. *Інтерфейс Copernicus Data Space Browser*. Доступно за адресою: https://browser.dataspace.copernicus.eu/#error=login_required&state=97b31ebb-d6e3-4da9-8d06-5ee41749517a [Дата звернення: 25.05.2025].

30. Kussul, N., Shelestov, A., Yailymov, B., Yailymova, H., Lemoine, G. and Deininger, K. (2025) Assessment of war-induced agricultural land use changes in Ukraine using machine learning applied to Sentinel satellite data. *International Journal of Applied Earth Observation and Geoinformation*, 140, 104951. https://doi.org/10.1016/j.jag.2023.104951.

ДОДАТОК А Лістинг коду

А.1 Обробка супутникових знімків та відповідних масок для test1

Приведений приклад для області знімку test1, ця область як і test2, test3, test4, test6 розділялися на навчальну і тестову вибірку і їхній код був майже однаковий (крім посилань на папку і дату знімків і маски). Тому приводжу приклад коду для test 1.

```
import os
import numpy as np
import rasterio
import pandas as pd
from sklearn.model_selection import train_test_split
main_folder_path = "D:/diplom/test1/"
classification_map_filename = "map24small.tif"
classification_map_path = os.path.join(main_folder_path, classification_map_filename)
output_train_bands_path = os.path.join(main_folder_path, "train_bands.csv")
output_test_bands_path = os.path.join(main_folder_path, "test_bands.csv")
output_train_vega_path = os.path.join(main_folder_path, "train_vega.csv")
output_test_vega_path = os.path.join(main_folder_path, "test_vega.csv")
output_train_labels_path = os.path.join(main_folder_path, "train_vega_labels.csv")
output_test_labels_path = os.path.join(main_folder_path, "test_vega_labels.csv")
date_folders = ["04.10", "05.05", "06.18", "07.17", "08.18", "09.17", "10.22"]
band_codes = ["01", "02", "03", "04", "05", "06", "07", "08", "8A", "09", "11", "12"]
index_bands_map = {
    "Red": "04",
    "NIR": "08",
    "Green": "03",
    "SWIR1": "11",
    "RedEdge1": "05"
}
index_names = ["NDVI", "NDWI", "NDMI", "NDRE"]
SPLIT_RANDOM_STATE = 42
TEST1_TRAIN_RATIO = 0.2 # 20% для навчання
TEST1_TEST_RATIO = 0.8 # 70% для тестування
label_mapping = {
    1: "Штучні об'єкти",
    2: "Пшениця",
    3: "Ріпак",
    4: "Гречка",
    5: "Кукурудза",
    6: "Культури за 2022",
    7: "Соняшник",
    8: "Соєві боби",
    9: "Цукровий буряк",
```

```
    10: "Інші культури",
    11: "Ліс",
    12: "Пасовища",
    13: "Гола земля",
    14: "Вода",
    15: "Водно-болотна місцевість",
    16: "Ячмінь",
    17: "Горох",
    18: "Люцерна",
    19: "Сади, парки",
    20: "Виноград",
    21: "Картопля",
    22: "Не культивується",
    23: "Пошкоджений ліс",
}
def read_band_from_date_folder(date_folder_path, band_code):
    found_path = None
    for filename in os.listdir(date_folder_path):
        if f"_B{band_code}_(Raw)" in filename and (filename.endswith(".tif") or
filename.endswith(".tiff")):
            found_path = os.path.join(date_folder_path, filename)
            break
    if found_path is None:
        raise FileNotFoundError(f"Не знайдено файл для каналу B{band_code} (Raw) у папці
{date_folder_path}")
    try:
        with rasterio.open(found_path) as src:
            data = src.read(1).astype('float32')
            meta = src.meta
            return data, meta, found_path
    except rasterio.errors.RasterioIOError as e:
        raise rasterio.errors.RasterioIOError(f"Помилка читання растра {found_path}: {e}")
    except Exception as e:
        print(f"Виникла невідома помилка при читанні {found_path}: {e}")
        raise
def calculate_indices(bands_data, index_bands_map):
    indices = {}
    epsilon = 1e-10
    # Отримуємо дані каналів за їхніми логічними назвами
    red = bands_data.get(index_bands_map.get("Red"))
    nir = bands_data.get(index_bands_map.get("NIR"))
    green = bands_data.get(index_bands_map.get("Green"))
    swir1 = bands_data.get(index_bands_map.get("SWIR1"))
    rededge1 = bands_data.get(index_bands_map.get("RedEdge1"))
    # Розрахунок NDVI
    if red is not None and nir is not None:
        indices['NDVI'] = (nir - red) / (nir + red + epsilon)
    else:
        pass
    # Розрахунок NDWI
    if green is not None and nir is not None:
        indices['NDWI'] = (green - nir) / (green + nir + epsilon)
```

```
    else:
        pass
    # Розрахунок NDMI
    if nir is not None and swir1 is not None:
        indices['NDMI'] = (nir - swir1) / (nir + swir1 + epsilon)
    else:
        pass
    # Розрахунок NDRE
    if nir is not None and rededge1 is not None:
        indices['NDRE'] = (nir - rededge1) / (nir + rededge1 + epsilon)
    else:
        pass
    return indices
print("Початок читання даних каналів та розрахунку індексів для test1...")
multitemporal_bands_data = {}
multitemporal_indices_data = {}
first_shape = None
raster_metadata = None
for date_index, date_folder_name in enumerate(date_folders):
    date_folder_path = os.path.join(main_folder_path, date_folder_name)
    print(f"\nЧитання даних для дати: {date_folder_name} (Date{date_index + 1}) з папки: {date_folder_path}")
    bands_data_for_date = {}
    for band_code in band_codes:
        try:
            data, meta, file_path = read_band_from_date_folder(date_folder_path, band_code)
            bands_data_for_date[band_code] = data # Зберігаємо дані каналу у словник
            if first_shape is None:
                raster_metadata = meta # Зберігаємо метадані першого зчитаного растра
                first_shape = data.shape # Встановлюємо очікуваний розмір

            if data.shape != first_shape:
                print(f"Критична помилка: Розмір растра для B{band_code} в папці {date_folder_name} ({data.shape}) не співпадає з очікуваним розміром ({first_shape})!")
                print("Переконайтесь, що всі ваші багаточасові знімки вирівняні між собою.")
                raise ValueError("Невідповідність розмірів растров.") # Зупиняємо виконання
        except FileNotFoundError as e:
            print(f"Помилка: {e}. Пропускаємо цей файл каналу для поточної дати.")
            continue
        except rasterio.errors.RasterioIOError as e:
            print(f"Помилка читання: {e}. Пропускаємо цей файл каналу для поточної дати.")
            continue
        except Exception as e:
            print(f"Виникла невідома помилка при читанні файлу каналу B{band_code} з папки {date_folder_name}: {e}. Пропускаємо.")
            continue
    if not bands_data_for_date:
        print(f"Попередження: Не вдалося зчитати жодного каналу для дати {date_folder_name}. Пропускаємо цю дату.")
        continue
    print(f"Розрахунок індексів для дати {date_folder_name}...")
    indices_data_for_date = calculate_indices(bands_data_for_date, index_bands_map)
```

```
    # Зберігаємо дані каналів та індексів для поточної дати в окремі словники
    multitemporal_bands_data[date_folder_name] = bands_data_for_date
    multitemporal_indices_data[date_folder_name] = indices_data_for_date
    print(f"Підготовлено дані (канали: {len(bands_data_for_date)}, індекси: {len(indices_data_for_date)}) для дати {date_folder_name}.")
print(f"\nЗчитування карти класифікації з файлу: {os.path.basename(classification_map_path)}")
try:
    with rasterio.open(classification_map_path) as src_cls:
        if first_shape is None:
            raise ValueError("Не вдалося зчитати жодного супутникового знімка для визначення очікуваного розміру.")
        if src_cls.shape != first_shape:
            raise ValueError("Невідповідність розмірів карти класифікації та супутникових знімків.")
        classification_labels_array = src_cls.read(1)
        print(f"Зчитано масив міток з розміром: {classification_labels_array.shape}")
except FileNotFoundError:
    raise FileNotFoundError(f"Файл карти класифікації не знайдено: {classification_map_path}")
except rasterio.errors.RasterioIOError as e:
    raise rasterio.errors.RasterioIOError(f"Помилка читання карти класифікації: {e}")
except Exception as e:
    raise Exception(f"Невідома помилка при читанні карти класифікації: {e}")

all_feature_types_order = band_codes + index_names
all_bands_flat = []
all_vega_flat = []
bands_column_names = []
vega_column_names = []
for date_index, date_folder_name in enumerate(date_folders):
    date_prefix = f"Date{date_index + 1}"
    # --- Збір даних каналів для поточної дати ---
    if date_folder_name in multitemporal_bands_data:
        date_bands_arrays = []
        current_date_bands_column_names = []
        for band_code in band_codes:
            if band_code in multitemporal_bands_data[date_folder_name]:
                data_array = multitemporal_bands_data[date_folder_name][band_code]
                if data_array.shape == first_shape:
                    date_bands_arrays.append(data_array.flatten())
                    current_date_bands_column_names.append(f"{date_prefix}_B{band_code}")
                else:
                    print(f"Попередження: Розмір {band_code} для дати {date_folder_name} не відповідає очікуваному.")
                    nan_array = np.full(first_shape[0] * first_shape[1], np.nan, dtype=np.float32)
                    date_bands_arrays.append(nan_array)
                    current_date_bands_column_names.append(f"{date_prefix}_B{band_code}")
            else:
                print(f"Попередження: Канал B{band_code} відсутній для дати {date_folder_name}. Заповнюємо NaN.")
                nan_array = np.full(first_shape[0] * first_shape[1], np.nan, dtype=np.float32)
```

```
            date_bands_arrays.append(nan_array)
            current_date_bands_column_names.append(f"{date_prefix}_B{band_code}")
        if date_bands_arrays and len(date_bands_arrays) == len(band_codes):
            all_bands_flat.append(np.stack(date_bands_arrays, axis=1))
            bands_column_names.extend(current_date_bands_column_names)
        else:
            print(f"Попередження: ({len(date_bands_arrays)}/{len(band_codes)}) для дати {date_folder_name}.")
            bands_column_names.extend([f"{date_prefix}_B{band_code}" for band_code in band_codes])
        if date_folder_name in multitemporal_bands_data: # Перевіряємо наявність даних каналів
            date_vega_arrays = []
            current_date_vega_column_names = []
            for feature_type in all_feature_types_order:
                data_array = None
                col_name = None #
                if feature_type in band_codes:
                    if feature_type in multitemporal_bands_data[date_folder_name]:
                        data_array = multitemporal_bands_data[date_folder_name][feature_type]
                        col_name = f"{date_prefix}_B{feature_type}"
                    else:
                        print(f"Попередження: Канал B{feature_type} відсутній для дати {date_folder_name}. Заповнюємо NaN.")
                        nan_array = np.full(first_shape[0] * first_shape[1], np.nan, dtype=np.float32)
                        date_vega_arrays.append(nan_array)
                        current_date_vega_column_names.append(f"{date_prefix}_B{feature_type}")
                        continue
                elif feature_type in index_names: # Це індекс
                    if feature_type in multitemporal_indices_data[date_folder_name]:
                        data_array = multitemporal_indices_data[date_folder_name][feature_type]
                        col_name = f"{date_prefix}_{feature_type}"
                    else:
                        print(f"Попередження: Індекс '{feature_type}' відсутній для дати {date_folder_name}. Заповнюємо NaN.")
                        nan_array = np.full(first_shape[0] * first_shape[1], np.nan, dtype=np.float32)
                        date_vega_arrays.append(nan_array)
                        current_date_vega_column_names.append(f"{date_prefix}_{feature_type}")
                        continue # Переходимо до наступної ознаки

                else:
                    # Це не очікуваний тип ознаки
                    print(f"Попередження: Невідомий тип ознаки '{feature_type}' для дати {date_folder_name}. Пропускаємо.")
                    continue # Пропускаємо цю ознаку
                # Якщо ознака присутня, перевіряємо розмір масиву
                if data_array.shape == first_shape:
                    date_vega_arrays.append(data_array.flatten())
                    current_date_vega_column_names.append(col_name)
                else:
                    print(f"Попередження: Розмір масиву для ознаки '{feature_type}' для дати {date_folder_name}.")
                    nan_array = np.full(first_shape[0] * first_shape[1], np.nan, dtype=np.float32)
```

```
                date_vega_arrays.append(nan_array)
                current_date_vega_column_names.append(col_name)
        if date_vega_arrays and len(date_vega_arrays) == len(all_feature_types_order):
            all_vega_flat.append(np.stack(date_vega_arrays, axis=1))
            vega_column_names.extend(current_date_vega_column_names)
        else:
            print(f"Попередження: Кількість зібраних масивів ознак+індексів ({len(date_vega_arrays)}/{len(all_feature_types_order)}) для дати {date_folder_name} не відповідає очікуваній. Пропускаємо цю дату для набору 'канали + індекси'.")

            temp_col_names = []
            for feature_type in all_feature_types_order:
                if feature_type in band_codes:
                    temp_col_names.append(f"{date_prefix}_B{feature_type}")
                else:
                    temp_col_names.append(f"{date_prefix}_{feature_type}")
            vega_column_names.extend(temp_col_names)
    else:
        print(f"Попередження: Дані каналів відсутні для дати {date_folder_name}.")
        # Якщо дата пропущена, додаємо порожні назви стовпців, щоб зберегти загальну кількість стовпців
        temp_col_names = []
        for feature_type in all_feature_types_order:
            if feature_type in band_codes:
                temp_col_names.append(f"{date_prefix}_B{feature_type}")
            else:
                temp_col_names.append(f"{date_prefix}_{feature_type}")
        vega_column_names.extend(temp_col_names)
# Перевіряємо, чи вдалося зібрати дані хоча б для однієї дати для кожного набору
if not all_bands_flat:
    print("Попередження: Не вдалося зібрати багаточасові ознаки)
    if first_shape is not None:
        all_bands_flat = [np.empty((first_shape[0] * first_shape[1], len(date_folders) * len(band_codes))) * np.nan]
    else:
        all_bands_flat = [np.empty((0, len(date_folders) * len(band_codes))) * np.nan] # Порожній масив
    bands_column_names = [f"Date{i+1}_B{band_code}" for i in range(len(date_folders)) for band_code in band_codes]
if not all_vega_flat:
    print("Попередження: Не вдалося зібрати багаточасові ознаки з вега")
    if first_shape is not None:
        all_vega_flat = [np.empty((first_shape[0] * first_shape[1], len(date_folders) * len(all_feature_types_order))) * np.nan]
    else:
        all_vega_flat = [np.empty((0, len(date_folders) * len(all_feature_types_order))) * np.nan] # Порожній масив
    # Генеруємо всі очікувані назви, використовуючи DateX префікс
    vega_column_names = [f"Date{i+1}_B{feat}" if feat in band_codes else f"Date{i+1}_{feat}" for i in range(len(date_folders)) for feat in all_feature_types_order]
bands_features_flat = np.concatenate(all_bands_flat, axis=1)
vega_features_flat = np.concatenate(all_vega_flat, axis=1)
```

```
print(f'Об'єднано багаточасові ознаки (тільки канали). Розмір масиву: {bands_features_flat.shape}")
print(f'Об'єднано багаточасові ознаки (канали + індекси). Розмір масиву: {vega_features_flat.shape}")
labels_flat = classification_labels_array.flatten()
# --Фільтрація пікселів ---
valid_pixels_mask = (labels_flat > 0)
nan_mask_vega = np.isnan(vega_features_flat).any(axis=1)
final_valid_mask = valid_pixels_mask & (~nan_mask_vega) # Комбінуємо маски: >0 І НЕ NaN в вега-наборі
# Застосовуємо фінальну маску до розгорнутих масивів ознак та міток
bands_features_sampled = bands_features_flat[final_valid_mask]
vega_features_sampled = vega_features_flat[final_valid_mask]
labels_sampled = labels_flat[final_valid_mask]
print(f'Витягнуто {labels_sampled.shape[0]} навчальних прикладів (пікселів) після фільтрації.")
print(f'Кількість ознак на приклад (тільки канали): {bands_features_sampled.shape[1]}")
print(f'Кількість ознак на приклад (канали + індекси): {vega_features_sampled.shape[1]}")
print(f'Кількість міток: {labels_sampled.shape[0]}")
bands_training_data_array = np.c_[bands_features_sampled, labels_sampled]
bands_column_names_final = bands_column_names + ["Label"]
training_data_df_bands = pd.DataFrame(data=bands_training_data_array, columns=bands_column_names_final)
training_data_df_bands['Label'] = training_data_df_bands['Label'].astype(int)
# --- DataFrame канали + індекси ---
vega_training_data_array = np.c_[vega_features_sampled, labels_sampled]
vega_column_names_final = vega_column_names + ["Label"]
print("Створено два DataFrame: training_data_df_bands та training_data_df_vega.")
print(f'Розмір training_data_df_bands: {training_data_df_bands.shape}")
print(f'Розмір training_data_df_vega: {training_data_df_vega.shape}")
print(f'Розділення набору 'тільки канали' на train/test")
X_train_bands, X_test_bands, y_train_bands, y_test_bands = train_test_split(
    training_data_df_bands.drop(columns=['Label']),
    training_data_df_bands['Label'],
    test_size=TEST1_TEST_RATIO,
    random_state=SPLIT_RANDOM_STATE,
    stratify=training_data_df_bands['Label']
)
print(f"  Розмір навчальної вибірки ознак (X_train_bands): {X_train_bands.shape}")
print(f"  Розмір тестової вибірки ознак (X_test_bands): {X_test_bands.shape}")
print(f"  Розмір навчальної вибірки міток (y_train_bands): {y_train_bands.shape}")
print(f"  Розмір тестової вибірки міток (y_test_bands): {y_test_bands.shape}")
print(f"\nРозділення набору 'канали + індекси' на train/test")
X_train_vega, X_test_vega, y_train_vega, y_test_vega = train_test_split(
    training_data_df_vega.drop(columns=['Label']),
    training_data_df_vega['Label'],
    test_size=TEST1_TEST_RATIO,
    random_state=SPLIT_RANDOM_STATE,
    stratify=training_data_df_vega['Label']
)
```

```
print(f"  Розмір навчальної вибірки ознак (X_train_vega): {X_train_vega.shape}")
print(f"  Розмір тестової вибірки ознак (X_test_vega): {X_test_vega.shape}")
print(f"  Розмір навчальної вибірки міток (y_train_vega): {y_train_vega.shape}")
print(f"  Розмір тестової вибірки міток (y_test_vega): {y_test_vega.shape}")
try:
    X_train_bands.to_csv(output_train_bands_path, index=False)
    y_train_bands.to_csv(output_train_labels_path, index=False, header=True)
    X_test_bands.to_csv(output_test_bands_path, index=False)
    y_test_bands.to_csv(output_test_labels_path, index=False, header=True)
    print(f'Набір 'тільки канали' збережено: {output_train_bands_path}, {output_test_bands_path} та їх мітки.")
except Exception as e:
    print(f'Помилка збереження файлів для набору 'тільки канали': {e}")
try:
    X_train_vega.to_csv(output_train_vega_path, index=False)
    X_test_vega.to_csv(output_test_vega_path, index=False)
    print(f'Набір 'канали + індекси' збережено: {output_train_vega_path}, {output_test_vega_path}.")
except Exception as e:
    print(f'Помилка збереження файлів для набору 'канали + індекси': {e}")
```

А.2 Обробка супутникових знімків та відповідних масок для test5

Приведений приклад для області знімку test5, ця область як і test7, test8, створювалися виключно як тестова вибірка і їхній код був майже однаковий (крім посилань на папку і дату знімків і маски). Тому приводжу приклад коду для test 5.

```
import os
import numpy as np
import rasterio
import pandas as pd
external_main_folder_path = "D:/diplom/test5/" # для інших тестів тут зміни
external_classification_map_filename = "Map2024_test5.tif" # для інших тестів тут зміни
external_classification_map_path = os.path.join(external_main_folder_path, external_classification_map_filename)
dates_to_process = ["04.10", "05.08", "06.22", "07.07", "08.18", "09.15", "10.02"]  # для інших тестів тут зміни
output_external_bands_features_path = os.path.join(external_main_folder_path, "test_bands.csv")
output_external_vega_features_path = os.path.join(external_main_folder_path, "test_vega.csv")
output_external_valid_labels_path = os.path.join(external_main_folder_path, "test_labels.csv")
band_codes = ["01", "02", "03", "04", "05", "06", "07", "08", "8A", "09", "11", "12"]
index_names = ["NDVI", "NDWI", "NDMI", "NDRE"]
index_bands_map = {
```

```
    "Red": "04",
    "NIR": "08",
    "Green": "03",
    "SWIR1": "11",
    "RedEdge1": "05"
}
def read_band_from_date_folder(date_folder_path, band_code):
    found_path = None
    for filename in os.listdir(date_folder_path):
        if f"_B{band_code}_(Raw)" in filename and (filename.endswith(".tif") or
filename.endswith(".tiff")):
            found_path = os.path.join(date_folder_path, filename)
            break
    if found_path is None:
        raise FileNotFoundError(f"Не знайдено файл для каналу B{band_code} (Raw) у папці
{date_folder_path}")
    try:
        with rasterio.open(found_path) as src:
            data = src.read(1).astype('float32')
            meta = src.meta
            return data, meta, found_path
    except rasterio.errors.RasterioIOError as e:
         raise rasterio.errors.RasterioIOError(f"Помилка читання растра {found_path}: {e}")
    except Exception as e:
        print(f"Виникла невідома помилка при читанні {found_path}: {e}")
        raise
def calculate_indices(bands_data, index_bands_map):
    indices = {}
    epsilon = 1e-10 # Уникнення ділення на нуль
    red = bands_data.get(index_bands_map.get("Red"))
    nir = bands_data.get(index_bands_map.get("NIR"))
    green = bands_data.get(index_bands_map.get("Green"))
    swir1 = bands_data.get(index_bands_map.get("SWIR1"))
    rededge1 = bands_data.get(index_bands_map.get("RedEdge1"))
    if red is not None and nir is not None:
        indices['NDVI'] = (nir - red) / (nir + red + epsilon)
    else:
        pass
    # Розрахунок NDWI
    if green is not None and nir is not None:
        indices['NDWI'] = (green - nir) / (green + nir + epsilon)
    else:
         pass
    # Розрахунок NDMI
    if nir is not None and swir1 is not None:
        indices['NDMI'] = (nir - swir1) / (nir + swir1 + epsilon)
    else:
         pass
    # Розрахунок NDRE
    if nir is not None and rededge1 is not None:
        indices['NDRE'] = (nir - rededge1) / (nir + rededge1 + epsilon)
    else:
```

```
        pass
    return indices
print(f"Підготовка даних для зовнішньої валідації ({os.path.basename(external_main_folder_path)}) ---")
external_multitemporal_bands_data = {}
external_multitemporal_indices_data = {}
external_first_shape = None # Для перевірки розмірів растров зовнішньої території
# Використовуємо enumerate, щоб отримати індекс дати
for date_index, date_folder_name in enumerate(dates_to_process):
    date_folder_path = os.path.join(external_main_folder_path, date_folder_name)
    print(f"\nЧитання даних для дати: {date_folder_name} (Date{date_index + 1}) з папки: {date_folder_path}")
    bands_data_for_date = {}
    for band_code in band_codes:
        try:
            data, meta, file_path = read_band_from_date_folder(date_folder_path, band_code)
            bands_data_for_date[band_code] = data # Зберігаємо дані каналу у словник
            if external_first_shape is None:
                external_first_shape = data.shape # Встановлюємо очікуваний розмір для зовнішньої території
            if data.shape != external_first_shape:
                print(f"Критична помилка: {band_code} в папці {date_folder_name} ({data.shape}) не співпадає з ({external_first_shape})")
                raise ValueError("Проблемки.") # Зупиняємо виконання
        except FileNotFoundError as e:
            continue
        except rasterio.errors.RasterioIOError as e:
            print(f"Помилка читання: {e}. Пропускаємо цей файл каналу для поточної дати.")
            continue
        except Exception as e:
            print(f"Уритична помилка{band_code} з папки {date_folder_name}: {e}. Пропускаємо.")
            continue
    if not bands_data_for_date:
        print(f"Попередження: Щось не то по коду {date_folder_name}.Нічого не завантажилось.")
        continue
    print(f"Розрахунок індексів для дати {date_folder_name}...")
    indices_data_for_date = calculate_indices(bands_data_for_date, index_bands_map)
    # Зберігаємо дані каналів та індексів для поточної дати в окремі словники
    external_multitemporal_bands_data[date_folder_name] = bands_data_for_date
    external_multitemporal_indices_data[date_folder_name] = indices_data_for_date
print(f"\nЗчитування карти класифікації з файлу: {os.path.basename(external_classification_map_path)}")
try:
    with rasterio.open(external_classification_map_path) as src_cls:
        if external_first_shape is None:
            raise ValueError("Не вдалося зчитати жодного знімка для зовнішньої території.")
        if src_cls.shape != external_first_shape:
            print(f"Критична помилка: ({src_cls.shape}) розміри не співпадають ({external_first_shape})!")
            print("Переконайтесь, що карта класифікації правильно вирівняна та обрізана.")
```

```
            raise ValueError("Невідповідність розмірів карти")
        external_classification_labels_array = src_cls.read(1)
        print(f"Зчитано масив міток з розміром: {external_classification_labels_array.shape}")
except FileNotFoundError:
    raise FileNotFoundError(f"Файл карти класифікації не знайдено: {external_classification_map_path}")
except rasterio.errors.RasterioIOError as e:
     raise rasterio.errors.RasterioIOError(f"Помилка читання карти класифікації: {e}")
except Exception as e:
    raise Exception(f"Невідома помилка при читанні карти класифікації: {e}")
all_feature_types_order = band_codes + index_names
external_bands_flat = [] # Тільки канали
external_vega_flat = [] # Канали + індекси
external_bands_column_names = []
external_vega_column_names = []
for date_index, date_folder_name in enumerate(dates_to_process):
    # Визначаємо префікс на основі індексу дати
    date_prefix = f"Date{date_index + 1}"
    # Перевіряємо, чи є дані каналів для цієї дати
    if date_folder_name in external_multitemporal_bands_data:

        date_bands_arrays = []
        current_date_bands_col_names = [] # Тимчасовий список назв для цієї дати (тільки канали)

        for band_code in band_codes:
            if band_code in external_multitemporal_bands_data[date_folder_name]:
                data_array = external_multitemporal_bands_data[date_folder_name][band_code]
                if data_array.shape == external_first_shape:
                    date_bands_arrays.append(data_array.flatten())
                    # Формуємо назву стовпця для каналів у форматі DateX_BКод
                    current_date_bands_col_names.append(f"{date_prefix}_B{band_code}")
                else:
                    print(f"Попередження: розмір {band_code} для дати {date_folder_name} не той.")
                    nan_array = np.full(external_first_shape[0] * external_first_shape[1], np.nan, dtype=np.float32)
                    date_bands_arrays.append(nan_array)
                    current_date_bands_col_names.append(f"{date_prefix}_B{band_code}")
            else:
                print(f"Попередження: Канал B{band_code} відсутній для дати {date_folder_name}.")
                nan_array = np.full(external_first_shape[0] * external_first_shape[1], np.nan, dtype=np.float32)
                date_bands_arrays.append(nan_array)
                current_date_bands_col_names.append(f"{date_prefix}_B{band_code}")
        if date_bands_arrays and len(date_bands_arrays) == len(band_codes):
            external_bands_flat.append(np.stack(date_bands_arrays, axis=1))
            external_bands_column_names.extend(current_date_bands_col_names) # Додаємо назви стовпців для цього набору
        else:
```

```
        print(f"Попередження: Не вдалося зібрати всі канали ({len(date_bands_arrays)}/{len(band_codes)}) для дати {date_folder_name}. Пропускаємо цю дату для набору 'тільки канали'.")
        external_bands_column_names.extend([f"{date_prefix}_B{band_code}" for band_code in band_codes])
    if date_folder_name in external_multitemporal_bands_data: # Перевіряємо наявність даних каналів
        date_vega_arrays = []
        current_date_vega_col_names = []
        for feature_type in all_feature_types_order:
            data_array = None # Ініціалізуємо як None
            if feature_type in external_multitemporal_bands_data[date_folder_name]:
                data_array = external_multitemporal_bands_data[date_folder_name][feature_type]
                col_name = f"{date_prefix}_B{feature_type}" if feature_type in band_codes else f"{date_prefix}_{feature_type}" # Формат DateX_BКод для каналів, DateX_Індекс для індексів
            elif feature_type in external_multitemporal_indices_data[date_folder_name]: # Має бути індекс
                data_array = external_multitemporal_indices_data[date_folder_name][feature_type]
                col_name = f"{date_prefix}_{feature_type}" # Формат DateX_Індекс для індексів
            else:
                print(f"Попередження: Ознака '{feature_type}' відсутня для дати {date_folder_name}. Заповнюємо NaN.")
                nan_array = np.full(external_first_shape[0] * external_first_shape[1], np.nan, dtype=np.float32)
                date_vega_arrays.append(nan_array)
                if feature_type in band_codes:
                    current_date_vega_col_names.append(f"{date_prefix}_B{feature_type}")
                else:
                    current_date_vega_col_names.append(f"{date_prefix}_{feature_type}")
                continue
            # Якщо ознака присутня, перевіряємо розмір масиву
            if data_array.shape == external_first_shape:
                date_vega_arrays.append(data_array.flatten())
                current_date_vega_col_names.append(col_name) # Додаємо сформовану назву
            else:
                print(f"Попередження: Розмір масиву для ознаки '{feature_type}' для дати {date_folder_name} не відповідає очікуваному. Заповнюємо NaN.")
                nan_array = np.full(external_first_shape[0] * external_first_shape[1], np.nan, dtype=np.float32)
                date_vega_arrays.append(nan_array)
                current_date_vega_col_names.append(col_name)
        if date_vega_arrays and len(date_vega_arrays) == len(all_feature_types_order):
            external_vega_flat.append(np.stack(date_vega_arrays, axis=1))
            external_vega_column_names.extend(current_date_vega_col_names) # Додаємо назви стовпців для цього набору
        else:
            print(f"Попередження: Кількість зібраних масивів ознак+індексів ({len(date_vega_arrays)}/{len(all_feature_types_order)}) для дати {date_folder_name} не відповідає очікуваній. Пропускаємо цю дату для набору 'канали + індекси'.")
            # Якщо дата пропущена, додаємо порожні назви стовпців, щоб зберегти загальну кількість стовпців
```

```
            temp_col_names = []
            for feature_type in all_feature_types_order:
                if feature_type in band_codes:
                    temp_col_names.append(f"{date_prefix}_B{feature_type}")
                else:
                    temp_col_names.append(f"{date_prefix}_{feature_type}")
            external_vega_column_names.extend(temp_col_names)
        else:
            print(f"Попередження: Дані каналів відсутні для дати {date_folder_name}. Пропускаємо цю дату для набору 'канали + індекси'.")
            # Якщо дата пропущена, додаємо порожні назви стовпців, щоб зберегти загальну кількість стовпців
            temp_col_names = []
            for feature_type in all_feature_types_order:
                if feature_type in band_codes:
                    temp_col_names.append(f"{date_prefix}_B{feature_type}")
                else:
                    temp_col_names.append(f"{date_prefix}_{feature_type}")
            external_vega_column_names.extend(temp_col_names)
if not external_bands_flat:
    print("Попередження: не вдалося зібрати масив")
    if external_first_shape is not None:
        external_bands_features_flat = np.empty((external_first_shape[0] * external_first_shape[1], len(dates_to_process) * len(band_codes))) * np.nan
    else:
        external_bands_features_flat = np.empty((0, len(dates_to_process) * len(band_codes))) * np.nan # Порожній масив
    # Генеруємо всі очікувані назви, використовуючи DateX префікс
    external_bands_column_names = [f"Date{i+1}_B{band_code}" for i in range(len(dates_to_process)) for band_code in band_codes]
if not external_vega_flat:
    # Якщо жодної дати не було успішно оброблено, створюємо порожній масив з правильною кількістю стовпців
    print("Попередження: Не вдалося зібрати масив + індекси.")
    if external_first_shape is not None:
        external_vega_features_flat = np.empty((external_first_shape[0] * external_first_shape[1], len(dates_to_process) * len(all_feature_types_order))) * np.nan
    else:
        external_vega_features_flat = np.empty((0, len(dates_to_process) * len(all_feature_types_order))) * np.nan # Порожній масив
    external_vega_column_names = [f"Date{i+1}_B{feat}" if feat in band_codes else f"Date{i+1}_{feat}" for i in range(len(dates_to_process)) for feat in all_feature_types_order]
external_bands_features_flat = np.concatenate(external_bands_flat, axis=1)
external_vega_features_flat = np.concatenate(external_vega_flat, axis=1)
external_labels_flat = external_classification_labels_array.flatten()
nan_mask_external_vega = np.isnan(external_vega_features_flat).any(axis=1)
nan_mask_external_bands = np.isnan(external_bands_features_flat).any(axis=1)
combined_nan_mask_external = nan_mask_external_vega | nan_mask_external_bands
X_external_bands_no_nan = external_bands_features_flat[~combined_nan_mask_external]
X_external_vega_no_nan = external_vega_features_flat[~combined_nan_mask_external]
y_external_raw_no_nan = external_labels_flat[~combined_nan_mask_external]
```

```
print(f"Витягнуто {X_external_bands_no_nan.shape[0]} пікселів для зовнішньої валідації після фільтрації NaN (з обох наборів).")
valid_labels_mask_external = (y_external_raw_no_nan > 0)
X_test_external_bands_valid = X_external_bands_no_nan[valid_labels_mask_external]
X_test_external_vega_valid = X_external_vega_no_nan[valid_labels_mask_external]
y_test_external_valid = y_external_raw_no_nan[valid_labels_mask_external]
print(f"Витягнуто {X_test_external_bands_valid.shape[0]} пікселів з ground truth мітками (> 0) для оцінки.")
print(f"Розмір масиву ознак для оцінки (тільки канали): {X_test_external_bands_valid.shape}")
print(f"Розмір масиву ознак для оцінки (канали + індекси): {X_test_external_vega_valid.shape}")
print(f"Розмір масиву міток для оцінки: {y_test_external_valid.shape}")
try:
    X_test_external_bands_valid_df = pd.DataFrame(X_test_external_bands_valid, columns=external_bands_column_names)
    X_test_external_bands_valid_df.to_csv(output_external_bands_features_path, index=False)
    print(f"Ознаки для зовнішньої валідації (тільки канали, ground truth > 0) збережено у {output_external_bands_features_path}")
except Exception as e:
    print(f"Помилка збереження ознак (тільки канали) для зовнішньої валідації: {e}")
try:
    X_test_external_vega_valid_df = pd.DataFrame(X_test_external_vega_valid, columns=external_vega_column_names)
    X_test_external_vega_valid_df.to_csv(output_external_vega_features_path, index=False)
    print(f"Ознаки для зовнішньої валідації (канали + індекси, ground truth > 0) збережено у {output_external_vega_features_path}")
except Exception as e:
    print(f"Помилка збереження ознак (канали + індекси) для зовнішньої валідації: {e}")
try:
    y_test_external_valid_df = pd.DataFrame(y_test_external_valid, columns=["Label"])
    y_test_external_valid_df.to_csv(output_external_valid_labels_path, index=False)
    print(f"Мітки для зовнішньої валідації (ground truth > 0) збережено у {output_external_valid_labels_path}")
except Exception as e:
    print(f"Помилка збереження міток для зовнішньої валідації: {e}")
```

A.3 Random forest без вегетаційних індексів

Так як код RF без вегетаційних індексів і вегетаційних майже однаковий, то для скорочення приводжу тільки приклад без вегетаційних індексів.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler
import time
import os
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='sklearn')
warnings.filterwarnings('ignore', category=FutureWarning, module='sklearn')
train_areas_folders = [
    "D:/diplom/test1/",
    "D:/diplom/test2/",
    "D:/diplom/test3/",
    "D:/diplom/test4/",
    "D:/diplom/test6/",
]

test_areas_folders = [
    "D:/diplom/test5/",
    "D:/diplom/test7/",
    "D:/diplom/test9/",
]
TRAIN_BANDS_FILENAME = "train_bands.csv"
TEST_BANDS_FILENAME = "test_bands.csv"
TRAIN_VEGA_FILENAME = "train_vega.csv" # цей файл для форесту з вега, тобто зараз його не беру
TEST_VEGA_FILENAME = "test_vega.csv"   # цей файл для форесту з вега, тобто зараз його не беру
TRAIN_LABELS_FILENAME = "train_vega_labels.csv" # для вега
TRAIN_TEST_LABELS_FILENAME = "test_vega_labels.csv" # Для вега індексів
EXTERNAL_TEST_LABELS_FILENAME = "test_labels.csv" # Для зовнішніх тестових територій
label_mapping = {
    1: "Штучні об'єкти",
    2: "Пшениця",
    3: "Ріпак",
    4: "Гречка",
    5: "Кукурудза",
    6: "Культури за 2022",
    7: "Соняшник",
    8: "Соєві боби",
    9: "Цукровий буряк",
    10: "Інші культури",
    11: "Ліс",
    12: "Пасовища",
    13: "Гола земля",
    14: "Вода",
    15: "Водно-болотна місцевість",
    16: "Ячмінь",
    17: "Горох",
    18: "Люцерна",
    19: "Сади, парки",
    20: "Виноград",
    21: "Картопля",
    22: "Не культивується",
    23: "Пошкоджений ліс",
```

```
}
train_area_test_splits = {}
external_test_areas = {}
def load_data_from_folder(folder, features_filename, labels_filename):
    features_path = os.path.join(folder, features_filename)
    labels_path = os.path.join(folder, labels_filename)
    if not os.path.exists(features_path):
        print(f"    Помилка: Файл ознак не знайдено: {features_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    if not os.path.exists(labels_path):
        print(f"    Помилка: Файл міток не знайдено: {labels_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    try:
        features_df = pd.read_csv(features_path)
        labels_series = pd.read_csv(labels_path).iloc[:, 0]
        if len(features_df) != len(labels_series):
            print(f"    Попередження: Кількість рядків в ознаках ({len(features_df)}) та мітках ({len(labels_series)}) не співпадає у папці {folder}")
            return None, None, None
        print(f"    Дані з {folder} успішно завантажено. Розмір ознак: {features_df.shape}, міток: {labels_series.shape}")
        return features_df, labels_series, sorted(list(labels_series.unique()))
    except MemoryError as e:
        print(f"    Помилка MemoryError при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
    except Exception as e:
        print(f"    Помилка при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
all_train_features_dfs = []
all_train_labels_series = []
all_unique_train_labels_set = set()
for folder in train_areas_folders:
    features_path = os.path.join(folder, TRAIN_BANDS_FILENAME)
    labels_path = os.path.join(folder, TRAIN_LABELS_FILENAME)
    features_df, labels_series, unique_labels_area = load_data_from_folder(
        folder, TRAIN_BANDS_FILENAME, TRAIN_LABELS_FILENAME
    )

    if features_df is not None:
        all_train_features_dfs.append(features_df)
        all_train_labels_series.append(labels_series)
        all_unique_train_labels_set.update(unique_labels_area)
if not all_train_features_dfs:
    print("Критична помилка:")
    exit()
print("  Об'єднання завантажених навчальних даних...")
X_train_combined = pd.concat(all_train_features_dfs, ignore_index=True)
y_train_combined = pd.concat(all_train_labels_series, ignore_index=True)
```

```
unique_labels_train = sorted(list(all_unique_train_labels_set))
print(f"  Об'єднані навчальні дані завантажено. Загальний розмір ознак: {X_train_combined.shape}, міток: {y_train_combined.shape}")
target_names_train = [label_mapping.get(label, f"Код {label}") for label in unique_labels_train]
all_test_y_true = []
all_test_y_pred = []
train_area_test_y_true = []
train_area_test_y_pred = []
external_test_y_true = []
external_test_y_pred = []
# Завантажуємо тестові частини навчальних територій
for folder in train_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження тестової частини з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, TRAIN_TEST_LABELS_FILENAME
    )
    if X_test_area is not None:
        train_area_test_splits[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_train_labels_set.update(unique_labels_area)
# Завантажуємо зовнішні тестові території
for folder in test_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження даних з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, EXTERNAL_TEST_LABELS_FILENAME
    )
    if X_test_area is not None:
        external_test_areas[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_train_labels_set.update(unique_labels_area)
all_unique_labels = sorted(list(all_unique_train_labels_set))
all_target_names = [label_mapping.get(label, f"Код {label}") for label in all_unique_labels]
print(f"\nВсі унікальні числові мітки (навчання + тест): {all_unique_labels}")
print(f"Назви класів для матриць помилок та звітів: {all_target_names}")
scaler = StandardScaler()
print("  Навчання StandardScaler на навчальних даних...")
X_train_scaled = scaler.fit_transform(X_train_combined)
print("  Навчальні дані масштабовано.")
model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
print("  Навчання моделі Random Forest на масштабованих даних...")
start_time_train = time.time() # Початок вимірювання часу навчання
try:
    model.fit(X_train_scaled, y_train_combined)
    end_time_train = time.time() # Кінець вимірювання часу навчання
    training_time = end_time_train - start_time_train
    print(f"  Навчання завершено за {training_time:.2f} секунд.")

except Exception as e:
    print(f"  Помилка під час навчання моделі: {e}")
    training_time = np.nan
    print("Критична помилка: Навчання моделі не вдалося. Зупиняємо виконання.")
    exit()
```

```
print("\n--- Оцінка моделі Random Forest (Bands Only) на всіх тестових вибірках ---")
evaluation_results = [] # Для зберігання результатів оцінки
all_test_sets = {**train_area_test_splits, **external_test_areas}
if not all_test_sets:
    print("Немає тестових даних для оцінки.")
else:
    for area_name, (X_test_area, y_test_area, unique_labels_test_area) in all_test_sets.items():
        print(f"\n  Оцінка на вибірці: {area_name}")
        print("    Масштабування тестових даних...")
        try:
            X_test_area_scaled = scaler.transform(X_test_area)
            print("    Тестові дані масштабовано.")
        except Exception as e:
            print(f"    Помилка для {area_name}: {e}. Пропускаємо оцінку.")
            evaluation_results.append({
                "Model": "Random Forest",
                "Feature Set": "Bands Only",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": np.nan,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": np.nan,
            })
            continue # Переходимо до наступної вибірки
        # --- Прогнозування ---
        start_time_predict = time.time()
        try:
            y_pred_area = model.predict(X_test_area_scaled)
            end_time_predict = time.time()
            prediction_evaluation_time = end_time_predict - start_time_predict
            print(f"    Прогнозування завершено за {prediction_evaluation_time:.4f} секунд.")
            all_test_y_true.extend(y_test_area)
            all_test_y_pred.extend(y_pred_area)
            if area_name in [os.path.basename(f.rstrip('/')) for f in train_areas_folders]:
                train_area_test_y_true.extend(y_test_area)
                train_area_test_y_pred.extend(y_pred_area)
            elif area_name in [os.path.basename(f.rstrip('/')) for f in test_areas_folders]:
                external_test_y_true.extend(y_test_area)
                external_test_y_pred.extend(y_pred_area)
            accuracy_area = accuracy_score(y_test_area, y_pred_area)
            print(f"\n    Загальна точність на вибірці {area_name}: {accuracy_area:.4f}")
            # Звіт про класифікацію
            print(f"\n    Звіт про класифікацію на вибірці {area_name}:")
            try:
                report = classification_report(y_test_area, y_pred_area, labels=all_unique_labels,
target_names=all_target_names, zero_division=0)
                print(report)
            except Exception as e:
                print(f"    Помилка при генерації звіту про класифікацію для {area_name}: {e}")
            print(f"\n    Матриця помилок для вибірки {area_name}:")
            try:
                cm = confusion_matrix(y_test_area, y_pred_area, labels=all_unique_labels)
```

```
            plt.figure(figsize=(12, 10)) # Збільшено розмір для кращої читабельності
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                        xticklabels=all_target_names, yticklabels=all_target_names)
            plt.xlabel('Прогнозовані класи')
            plt.ylabel('Реальні класи')
            plt.title(f'Матриця помилок для {area_name} (Bands Only)')
            plt.show()
        except Exception as e:
            print(f"    Помилка при побудові матриці помилок для {area_name}: {e}")
        evaluation_results.append({
            "Model": "Random Forest",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": accuracy_area,
            "Training Time (s)": training_time, # Час навчання однаковий для всіх територій оцінки
            "Prediction/Evaluation Time (s)": prediction_evaluation_time,
        })
    except Exception as e:
        print(f"    Помилка під час оцінки на вибірці {area_name}: {e}")
        evaluation_results.append({
            "Model": "Random Forest",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
if evaluation_results:
    results_df = pd.DataFrame(evaluation_results)
    print("\nРезультати оцінки Random Forest на кожній тестовій вибірці:")
    print(results_df)
    # Розраховуємо середню точність по валідних тестових вибірках
    valid_results_df = results_df.dropna(subset=['Overall Accuracy'])
    if not valid_results_df.empty:
        average_results = valid_results_df.groupby("Feature Set")["Overall Accuracy"].mean().reset_index()
        average_results = average_results.rename(columns={"Overall Accuracy": "Average Overall Accuracy (All Valid Test Sets)"})
        print("\nСередня загальна точність Random Forest по всіх валідних тестових вибірках:")
        print(average_results)
    else:
        print("\nНемає валідних результатів оцінки для розрахунку середньої точності.")
    print("\n--- Агреговані метрики для груп тестових даних ---")
    all_test_y_true = np.array(all_test_y_true)
    all_test_y_pred = np.array(all_test_y_pred)
    train_area_test_y_true = np.array(train_area_test_y_true)
    train_area_test_y_pred = np.array(train_area_test_y_pred)
    external_test_y_true = np.array(external_test_y_true)
    external_test_y_pred = np.array(external_test_y_pred)
```

```
    # Загальна точність та F1-score для ВСІХ тестових вибірок
    if len(all_test_y_true) > 0:
        overall_accuracy = accuracy_score(all_test_y_true, all_test_y_pred)
        overall_f1_score = f1_score(all_test_y_true, all_test_y_pred, average='weighted', labels=all_unique_labels, zero_division=0)
        print(f"Загальна точність (ВСІ тестові вибірки разом): {overall_accuracy:.4f}")
        print(f"Загальний F1-score (ВСІ тестові вибірки разом, weighted): {overall_f1_score:.4f}")
    else:
        print("Недостатньо даних для розрахунку загальних метрик для ВСІХ тестових вибірок.")
    # Точність та F1-score для тестових частин НАВЧАЛЬНИХ територій
    if len(train_area_test_y_true) > 0:
        train_area_accuracy = accuracy_score(train_area_test_y_true, train_area_test_y_pred)
        train_area_f1_score = f1_score(train_area_test_y_true, train_area_test_y_pred, average='weighted', labels=all_unique_labels, zero_division=0)
        print(f"\nТочність (тестові частини НАВЧАЛЬНИХ територій): {train_area_accuracy:.4f}")
        print(f"F1-score (тестові частини НАВЧАЛЬНИХ територій, weighted): {train_area_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для тестових частин НАВЧАЛЬНИХ територій.")
    # Точність та F1-score для ЗОВНІШНІХ тестових територій
    if len(external_test_y_true) > 0:
        external_accuracy = accuracy_score(external_test_y_true, external_test_y_pred)
        external_f1_score = f1_score(external_test_y_true, external_test_y_pred, average='weighted', labels=all_unique_labels, zero_division=0)
        print(f"\nТочність (ЗОВНІШНІ тестові території): {external_accuracy:.4f}")
        print(f"F1-score (ЗОВНІШНІ тестові території, weighted): {external_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для ЗОВНІШНІХ тестових територій.")
else:
    print("Немає зібраних результатів оцінки.")
print("\n--- Оцінка просторової переносимості Random Forest завершена ---")
```

A.4 Logistic regressiont без вегетаційних індексів

Так як код LG без вегетаційних індексів і вегетаційних майже однаковий, то для скорочення приводжу тільки приклад без вегетаційних індексів.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler
import time
import os
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='sklearn')
warnings.filterwarnings('ignore', category=FutureWarning, module='sklearn')
train_areas_folders = [
    "D:/diplom/test1/",
    "D:/diplom/test2/",
    "D:/diplom/test3/",
    "D:/diplom/test4/",
    "D:/diplom/test6/",
]
test_areas_folders = [
    "D:/diplom/test5/",
    "D:/diplom/test7/",
    "D:/diplom/test9/",
]
TRAIN_BANDS_FILENAME = "train_bands.csv"
TEST_BANDS_FILENAME = "test_bands.csv"
TRAIN_VEGA_FILENAME = "train_vega.csv" # цей файл з вега, тобто зараз його не беру
TEST_VEGA_FILENAME = "test_vega.csv"   # цей файл з вега, тобто зараз його не беру
TRAIN_LABELS_FILENAME = "train_vega_labels.csv"
TRAIN_TEST_LABELS_FILENAME = "test_vega_labels.csv" #маска вега
EXTERNAL_TEST_LABELS_FILENAME = "test_labels.csv"
label_mapping = {
    1: "Штучні об'єкти",
    2: "Пшениця",
    3: "Ріпак",
    4: "Гречка",
    5: "Кукурудза",
    6: "Культури за 2022",
    7: "Соняшник",
    8: "Соєві боби",
    9: "Цукровий буряк",
    10: "Інші культури",
    11: "Ліс",
    12: "Пасовища",
    13: "Гола земля",
    14: "Вода",
    15: "Водно-болотна місцевість",
    16: "Ячмінь",
    17: "Горох",
    18: "Люцерна",
    19: "Сади, парки",
    20: "Виноград",
    21: "Картопля",
    22: "Не культивується",
    23: "Пошкоджений ліс",
}
train_area_test_splits = {}
external_test_areas = {}
def load_data_from_folder(folder, features_filename, labels_filename):
```

```
    features_path = os.path.join(folder, features_filename)
    labels_path = os.path.join(folder, labels_filename)
    if not os.path.exists(features_path):
        print(f"    Помилка: Файл ознак не знайдено: {features_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    if not os.path.exists(labels_path):
        print(f"    Помилка: Файл міток не знайдено: {labels_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    try:
        features_df = pd.read_csv(features_path)
        labels_series = pd.read_csv(labels_path).iloc[:, 0]
        if len(features_df) != len(labels_series):
            print(f"    Попередження: Кількість рядків в ({len(features_df)}) та мітках ({len(labels_series)}) не співпадає у папці {folder}")
            return None, None, None
        print(f"    Дані з {folder} успішно завантажено. Розмір ознак: {features_df.shape}, міток: {labels_series.shape}")
        return features_df, labels_series, sorted(list(labels_series.unique()))
    except MemoryError as e:
        print(f"    Помилка MemoryError при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
    except Exception as e:
        print(f"    Помилка при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
all_train_features_dfs = []
all_train_labels_series = []
all_unique_train_labels_set = set()
for folder in train_areas_folders:
    features_path = os.path.join(folder, TRAIN_BANDS_FILENAME)
    labels_path = os.path.join(folder, TRAIN_LABELS_FILENAME)
    features_df, labels_series, unique_labels_area = load_data_from_folder(
        folder, TRAIN_BANDS_FILENAME, TRAIN_LABELS_FILENAME
    )
    if features_df is not None:
        all_train_features_dfs.append(features_df)
        all_train_labels_series.append(labels_series)
        all_unique_train_labels_set.update(unique_labels_area)
if not all_train_features_dfs:
    print("Критична помилка:")
    exit()
print("  Об'єднання завантажених навчальних даних...")
X_train_combined = pd.concat(all_train_features_dfs, ignore_index=True)
y_train_combined = pd.concat(all_train_labels_series, ignore_index=True)
unique_labels_train = sorted(list(all_unique_train_labels_set))
print(f"    Об'єднані навчальні дані завантажено. Загальний розмір ознак: {X_train_combined.shape}, міток: {y_train_combined.shape}")
target_names_train = [label_mapping.get(label, f"Код {label}") for label in unique_labels_train]
all_test_y_true = []
```

```
all_test_y_pred = []
train_area_test_y_true = []
train_area_test_y_pred = []
external_test_y_true = []
external_test_y_pred = []
for folder in train_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження тестової частини з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, TRAIN_TEST_LABELS_FILENAME
    )
    if X_test_area is not None:
        train_area_test_splits[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_train_labels_set.update(unique_labels_area)
for folder in test_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження даних з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, EXTERNAL_TEST_LABELS_FILENAME
    )
    if X_test_area is not None:
        external_test_areas[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_train_labels_set.update(unique_labels_area)
all_unique_labels = sorted(list(all_unique_train_labels_set))
all_target_names = [label_mapping.get(label, f"Код {label}") for label in all_unique_labels]
print(f"\nВсі унікальні числові мітки (навчання + тест): {all_unique_labels}")
print(f"Назви класів для матриць помилок та звітів: {all_target_names}")
scaler = StandardScaler()
print("  Навчання StandardScaler на навчальних даних...")
X_train_scaled = scaler.fit_transform(X_train_combined)
print("  Навчальні дані масштабовано.")
model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
print("  Навчання моделі Random Forest на масштабованих даних...")
start_time_train = time.time() # Початок вимірювання часу навчання
try:
    model.fit(X_train_scaled, y_train_combined)
    end_time_train = time.time() # Кінець вимірювання часу навчання
    training_time = end_time_train - start_time_train
    print(f"  Навчання завершено за {training_time:.2f} секунд.")
except Exception as e:
    print(f"  Помилка під час навчання моделі: {e}")
    training_time = np.nan
    print("Критична помилка: Навчання моделі не вдалося. Зупиняємо виконання.")
    exit()
evaluation_results = [] # Для зберігання результатів оцінки
all_test_sets = {**train_area_test_splits, **external_test_areas}
if not all_test_sets:
    print("Немає тестових даних для оцінки.")
else:
    for area_name, (X_test_area, y_test_area, unique_labels_test_area) in all_test_sets.items():
        print(f"\n  Оцінка на вибірці: {area_name}")
        print("    Масштабування тестових даних...")
```

```
        try:
            X_test_area_scaled = scaler.transform(X_test_area)
            print("    Тестові дані масштабовано.")
        except Exception as e:
            print(f"    Помилка для {area_name}: {e}. Пропускаємо оцінку.")
            evaluation_results.append({
                "Model": "Random Forest",
                "Feature Set": "Bands Only",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": np.nan,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": np.nan,
            })
            continue # Переходимо до наступної вибірки
        start_time_predict = time.time()
        try:
            y_pred_area = model.predict(X_test_area_scaled)
            end_time_predict = time.time()
            prediction_evaluation_time = end_time_predict - start_time_predict
            print(f"    Прогнозування завершено за {prediction_evaluation_time:.4f} секунд.")
            all_test_y_true.extend(y_test_area)
            all_test_y_pred.extend(y_pred_area)
            if area_name in [os.path.basename(f.rstrip('/')) for f in train_areas_folders]:
                train_area_test_y_true.extend(y_test_area)
                train_area_test_y_pred.extend(y_pred_area)
            elif area_name in [os.path.basename(f.rstrip('/')) for f in test_areas_folders]:
                external_test_y_true.extend(y_test_area)
                external_test_y_pred.extend(y_pred_area)
            accuracy_area = accuracy_score(y_test_area, y_pred_area)
            print(f"\n    Загальна точність на вибірці {area_name}: {accuracy_area:.4f}")
            print(f"\n    Звіт про класифікацію на вибірці {area_name}:")
            try:
                report = classification_report(y_test_area, y_pred_area, labels=all_unique_labels,
target_names=all_target_names, zero_division=0)
                print(report)
            except Exception as e:
                print(f"    Помилка при генерації звіту про класифікацію для {area_name}: {e}")
            print(f"\n    Матриця помилок для вибірки {area_name}:")
            try:
                cm = confusion_matrix(y_test_area, y_pred_area, labels=all_unique_labels)
                plt.figure(figsize=(12, 10)) # Збільшено розмір для кращої читабельності
                sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                            xticklabels=all_target_names, yticklabels=all_target_names)
                plt.xlabel('Прогнозовані класи')
                plt.ylabel('Реальні класи')
                plt.title(f'Матриця помилок для {area_name} (Bands Only)')
                plt.show()
            except Exception as e:
                print(f"    Помилка при побудові матриці помилок для {area_name}: {e}")
            evaluation_results.append({
                "Model": "Random Forest",
```

```
                "Feature Set": "Bands Only",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": accuracy_area,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": prediction_evaluation_time,
            })
        except Exception as e:
            print(f"    Помилка під час оцінки на вибірці {area_name}: {e}")
            evaluation_results.append({
                "Model": "Random Forest",
                "Feature Set": "Bands Only",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": np.nan,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": np.nan,
            })
if evaluation_results:
    results_df = pd.DataFrame(evaluation_results)
    print("\nРезультати оцінки Random Forest на кожній тестовій вибірці:")
    print(results_df)
    valid_results_df = results_df.dropna(subset=['Overall Accuracy'])
    if not valid_results_df.empty:
        average_results = valid_results_df.groupby("Feature Set")["Overall
Accuracy"].mean().reset_index()
        average_results = average_results.rename(columns={"Overall Accuracy": "Average Overall
Accuracy (All Valid Test Sets)"})
        print("\nСередня загальна точність Random Forest по всіх валідних тестових вибірках:")
        print(average_results)
    else:
        print("\nНемає валідних результатів оцінки для розрахунку середньої точності.")
    print("\n--- Агреговані метрики для груп тестових даних ---")
    all_test_y_true = np.array(all_test_y_true)
    all_test_y_pred = np.array(all_test_y_pred)
    train_area_test_y_true = np.array(train_area_test_y_true)
    train_area_test_y_pred = np.array(train_area_test_y_pred)
    external_test_y_true = np.array(external_test_y_true)
    external_test_y_pred = np.array(external_test_y_pred)
    if len(all_test_y_true) > 0:
        overall_accuracy = accuracy_score(all_test_y_true, all_test_y_pred)
        overall_f1_score = f1_score(all_test_y_true, all_test_y_pred, average='weighted',
labels=all_unique_labels, zero_division=0)
        print(f"Загальна точність (BCI тестові вибірки разом): {overall_accuracy:.4f}")
        print(f"Загальний F1-score (BCI тестові вибірки разом, weighted):
{overall_f1_score:.4f}")
    else:
        print("Недостатньо даних для розрахунку загальних метрик для BCIX тестових
вибірок.")
    if len(train_area_test_y_true) > 0:
        train_area_accuracy = accuracy_score(train_area_test_y_true, train_area_test_y_pred)
```

```
        train_area_f1_score = f1_score(train_area_test_y_true, train_area_test_y_pred, average='weighted', labels=all_unique_labels, zero_division=0)
        print(f"\nТочність (тестові частини НАВЧАЛЬНИХ територій): {train_area_accuracy:.4f}")
        print(f"F1-score (тестові частини НАВЧАЛЬНИХ територій, weighted): {train_area_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для тестових частин НАВЧАЛЬНИХ територій.")
    # Точність та F1-score для ЗОВНІШНІХ тестових територій
    if len(external_test_y_true) > 0:
        external_accuracy = accuracy_score(external_test_y_true, external_test_y_pred)
        external_f1_score = f1_score(external_test_y_true, external_test_y_pred, average='weighted', labels=all_unique_labels, zero_division=0)
        print(f"\nТочність (ЗОВНІШНІ тестові території): {external_accuracy:.4f}")
        print(f"F1-score (ЗОВНІШНІ тестові території, weighted): {external_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для ЗОВНІШНІХ тестових територій.")
else:
    print("Немає зібраних результатів оцінки.")
```

A.5 XGBoost без вегетаційних індексів

Так як код XGBoost без вегетаційних індексів і вегетаційних майже однаковий, то для скорочення приводжу тільки приклад без вегетаційних індексів.

```
import pandas as pd
import numpy as np
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
import time
import os
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='sklearn')
warnings.filterwarnings('ignore', category=FutureWarning, module='xgboost')
train_areas_folders = [
    "D:/diplom/test1/",
    "D:/diplom/test2/",
    "D:/diplom/test3/",
    "D:/diplom/test4/",
    "D:/diplom/test6/",
]
test_areas_folders = [
    "D:/diplom/test5/",
```

```
    "D:/diplom/test7/",
    "D:/diplom/test9/",
]
TRAIN_BANDS_FILENAME = "train_bands.csv"
TEST_BANDS_FILENAME = "test_bands.csv"
TRAIN_LABELS_FILENAME = "train_vega_labels.csv" # для веги, зараз не юз
TRAIN_TEST_LABELS_FILENAME = "test_vega_labels.csv" # для веги, зараз не юз
EXTERNAL_TEST_LABELS_FILENAME = "test_labels.csv" # для веги, зараз не юз

XGB_N_ESTIMATORS = 110
XGB_LEARNING_RATE = 0.07
XGB_MAX_DEPTH = 10
XGB_N_JOBS = -1
label_mapping = {
    1: "Штучні об'єкти",
    2: "Пшениця",
    3: "Ріпак",
    4: "Гречка",
    5: "Кукурудза",
    6: "Культури за 2022",
    7: "Соняшник",
    8: "Соєві боби",
    9: "Цукровий буряк",
    10: "Інші культури",
    11: "Ліс",
    12: "Пасовища",
    13: "Гола земля",
    14: "Вода",
    15: "Водно-болотна місцевість",
    16: "Ячмінь",
    17: "Горох",
    18: "Люцерна",
    19: "Сади, парки",
    20: "Виноград",
    21: "Картопля",
    22: "Не культивується",
    23: "Пошкоджений ліс",
}

train_area_test_splits = {}
external_test_areas = {}
def load_data_from_folder(folder, features_filename, labels_filename):
    features_path = os.path.join(folder, features_filename)
    labels_path = os.path.join(folder, labels_filename)
    if not os.path.exists(features_path):
        print(f"    Помилка: Файл ознак не знайдено: {features_path}. Неможливо завантажити
дані з {folder}.")
        return None, None, None
    if not os.path.exists(labels_path):
        print(f"    Помилка: Файл міток не знайдено: {labels_path}. Неможливо завантажити
дані з {folder}.")
        return None, None, None
```

```
    try:
        features_df = pd.read_csv(features_path)
        labels_series = pd.read_csv(labels_path).iloc[:, 0]
        if len(features_df) != len(labels_series):
            print(f"      Попередження: Кількість рядків в ознаках ({len(features_df)}) та мітках ({len(labels_series)}) не співпадає у папці {folder}. Неможливо завантажити дані з {folder}.")
            return None, None, None
        print(f"    Дані з {folder} успішно завантажено. Розмір ознак: {features_df.shape}, міток: {labels_series.shape}")
        return features_df, labels_series, sorted(list(labels_series.unique()))
    except MemoryError as e:
        print(f"      Помилка MemoryError при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
    except Exception as e:
        print(f"      Помилка при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
print(f"\n--- Навчання та оцінка XGBoost для просторової переносимості (Bands Only) ---")
all_train_features_dfs = []
all_train_labels_series = []
all_unique_train_labels_set_original = set()
for folder in train_areas_folders:
    features_df, labels_series, unique_labels_area = load_data_from_folder(
        folder, TRAIN_BANDS_FILENAME, TRAIN_LABELS_FILENAME
    )
    if features_df is not None:
        all_train_features_dfs.append(features_df)
        all_train_labels_series.append(labels_series)
        all_unique_train_labels_set_original.update(unique_labels_area)
if not all_train_features_dfs:
    print("Критична помилка: Не вдалося завантажити навчальні дані з жодної папки. Зупиняємо виконання.")
    exit()
X_train_combined = pd.concat(all_train_features_dfs, ignore_index=True)
y_train_combined = pd.concat(all_train_labels_series, ignore_index=True)

print(f"  Об'єднані навчальні дані (Bands Only) завантажено. Загальний розмір ознак: {X_train_combined.shape}, міток: {y_train_combined.shape}")
# Списки для збору всіх тестових даних для агрегованих метрик
all_test_y_true_original = []
all_test_y_pred_encoded = []
train_area_test_y_true_original = []
train_area_test_y_pred_encoded = []
external_test_y_true_original = []
external_test_y_pred_encoded = []
all_unique_test_labels_set_original = set()
for folder in train_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження тестової частини з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, TRAIN_TEST_LABELS_FILENAME
```

```
    )
    if X_test_area is not None:
        train_area_test_splits[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_test_labels_set_original.update(unique_labels_area)
for folder in test_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"  Завантаження даних з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, EXTERNAL_TEST_LABELS_FILENAME
    )
    if X_test_area is not None:
        external_test_areas[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_test_labels_set_original.update(unique_labels_area)
all_original_unique_labels_in_train = sorted(list(all_unique_train_labels_set_original))
all_original_unique_labels_overall = sorted(list(all_unique_train_labels_set_original.union(all_unique_test_labels_set_original)))
all_original_target_names = [label_mapping.get(label, f"Код {label}") for label in all_original_unique_labels_overall]
print(f"\nВсі унікальні ЧИСЛОВІ мітки (навчання + тест, оригінальні): {all_original_unique_labels_overall}")
print(f"Назви класів для матриць помилок та звітів (оригінальні): {all_original_target_names}")
label_encoder = LabelEncoder()
label_encoder.fit(all_original_unique_labels_overall)
y_train_encoded = label_encoder.transform(y_train_combined)
print(f"  Навчальні мітки закодовано. Оригінальні унікальні: {sorted(list(np.unique(y_train_combined)))}. Закодовані: {np.unique(y_train_encoded)}")
print(f"  Унікальні закодовані мітки у НАВЧАЛЬНИХ даних (без семплінгу, після кодування): {np.unique(y_train_encoded)}")
print(f"  Максимальна закодована мітка у НАВЧАЛЬНИХ даних: {y_train_encoded.max()}")
print(f"  Кількість унікальних закодованих міток у НАВЧАЛЬНИХ даних: {len(np.unique(y_train_encoded))}")
num_classes_for_xgb = len(label_encoder.classes_)
encoded_target_names = [label_mapping.get(original_label, f"Код {original_label}") for original_label in label_encoder.classes_]
print(f"  Назви класів для звітів (відповідно до закодованих міток): {encoded_target_names}")
scaler = StandardScaler()
print("  Навчання StandardScaler на навчальних даних (Bands Only)...")
X_train_scaled = scaler.fit_transform(X_train_combined)
print("  Навчальні дані (Bands Only) масштабовано.")
unique_train_encoded_current = np.unique(y_train_encoded)
expected_encoded_labels = set(range(num_classes_for_xgb))
missing_encoded_labels = list(expected_encoded_labels - set(unique_train_encoded_current))
if missing_encoded_labels:
    print(f"  Попередження: Навчальні дані не містять закодованих міток: {missing_encoded_labels}")
    print("  Додаємо фіктивні зразки для відсутніх класів, щоб забезпечити безперервність міток для XGBoost.")
    dummy_features = np.zeros((len(missing_encoded_labels), X_train_combined.shape[1]))
    dummy_labels_encoded = np.array(missing_encoded_labels)
    dummy_features_scaled = scaler.transform(dummy_features)
    X_train_scaled = np.vstack((X_train_scaled, dummy_features_scaled))
```

```
    y_train_encoded = np.hstack((y_train_encoded, dummy_labels_encoded))
    print(f"  Розмір навчальних даних після додавання фіктивних зразків: {X_train_scaled.shape}, міток: {y_train_encoded.shape}")
    print(f"  Унікальні закодовані мітки у НАВЧАЛЬНИХ даних після додавання фіктивних зразків: {np.unique(y_train_encoded)}")
model = XGBClassifier(
    objective='multi:softprob',
    num_class=num_classes_for_xgb,
    n_estimators=XGB_N_ESTIMATORS,
    learning_rate=XGB_LEARNING_RATE,
    max_depth=XGB_MAX_DEPTH,
    n_jobs=XGB_N_JOBS,
    random_state=42,
    use_label_encoder=False,
    eval_metric='mlogloss'
)
print(f"  Навчання моделі XGBoost (n_estimators={XGB_N_ESTIMATORS}, learning_rate={XGB_LEARNING_RATE}, max_depth={XGB_MAX_DEPTH}) на масштабованих даних (Bands Only)...")
start_time_train = time.time()
try:
    model.fit(X_train_scaled, y_train_encoded)
    end_time_train = time.time()
    training_time = end_time_train - start_time_train
    print(f"  Навчання завершено за {training_time:.2f} секунд.")
except Exception as e:
    print(f"  Помилка під час навчання моделі: {e}")
    training_time = np.nan
    print("Критична помилка: Навчання моделі не вдалося. Зупиняємо виконання.")
    exit()
evaluation_results = []
all_test_sets = {**train_area_test_splits, **external_test_areas}
if not all_test_sets:
    print("Немає тестових даних для оцінки.")
else:
    for area_name, (X_test_area, y_test_area_original, unique_labels_test_area) in all_test_sets.items():
        print(f"\n  Оцінка на вибірці: {area_name} (Bands Only)")

        # --- Кодування тестових міток ---
        print("    Кодування тестових міток...")
        try:
            # Перетворюємо тестові мітки за допомогою того ж LabelEncoder, що був навчений на всіх можливих мітках
            y_test_area_encoded = label_encoder.transform(y_test_area_original)
            print("    Тестові мітки закодовано.")
        except ValueError as e:
            print(f"    Помилка під час кодування тестових міток для {area_name}: {e}. Можливо, в тестових даних є мітки, яких немає в навчальних. Пропускаємо оцінку.")
            evaluation_results.append({
                "Model": "XGBoost",
                "Feature Set": "Bands Only",
```

```
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
        continue
    except Exception as e:
        print(f"    Невідома помилка під час кодування тестових міток для {area_name}: {e}. Пропускаємо оцінку.")
        evaluation_results.append({
            "Model": "XGBoost",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
        continue
    try:
        X_test_area_scaled = scaler.transform(X_test_area)
        print("    Тестові дані (Bands Only) масштабовано.")
    except Exception as e:
        print(f"    Помилка під час масштабування тестових даних для {area_name}: {e}. Пропускаємо оцінку.")
        evaluation_results.append({
            "Model": "XGBoost",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
        continue
    # --- Прогнозування ---
    start_time_predict = time.time()
    try:
        y_pred_proba_area_encoded = model.predict_proba(X_test_area_scaled)
        y_pred_area_encoded = np.argmax(y_pred_proba_area_encoded, axis=1) # Отримуємо клас з найвищою ймовірністю
        end_time_predict = time.time()
        prediction_evaluation_time = end_time_predict - start_time_predict
        print(f"    Прогнозування завершено за {prediction_evaluation_time:.4f} секунд.")
        all_test_y_true_original.extend(y_test_area_original) # Зберігаємо оригінальні мітки
        all_test_y_pred_encoded.extend(y_pred_area_encoded) # Зберігаємо закодовані прогнози
        if area_name in [os.path.basename(f.rstrip('/')) for f in train_areas_folders]:
            train_area_test_y_true_original.extend(y_test_area_original)
            train_area_test_y_pred_encoded.extend(y_pred_area_encoded)
        elif area_name in [os.path.basename(f.rstrip('/')) for f in test_areas_folders]:
```

```
            external_test_y_true_original.extend(y_test_area_original)
            external_test_y_pred_encoded.extend(y_pred_area_encoded)
        accuracy_area = accuracy_score(y_test_area_encoded, y_pred_area_encoded)
        print(f"\n    Загальна точність на вибірці {area_name}: {accuracy_area:.4f}")
        print(f"\n    Звіт про класифікацію на вибірці {area_name}:")
        try:
            unique_test_labels_encoded = np.unique(y_test_area_encoded)
            filtered_target_names = [encoded_target_names[i] for i in unique_test_labels_encoded
if i < len(encoded_target_names)]

            report = classification_report(y_test_area_encoded, y_pred_area_encoded,
                                           labels=unique_test_labels_encoded,
                                           target_names=filtered_target_names,
                                           zero_division=0)
            print(report)
        except Exception as e:
            print(f"    Помилка при генерації звіту про класифікацію для {area_name}: {e}")
        print(f"\n    Матриця помилок для вибірки {area_name}:")
        try:
            cm = confusion_matrix(y_test_area_encoded, y_pred_area_encoded,
                                  labels=unique_test_labels_encoded)
            plt.figure(figsize=(12, 10))
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                        xticklabels=filtered_target_names, yticklabels=filtered_target_names)
            plt.xlabel('Прогнозовані класи')
            plt.ylabel('Реальні класи')
            plt.title(f'Матриця помилок для {area_name} (XGBoost, Bands Only)')
            plt.show()
        except Exception as e:
            print(f"    Помилка при побудові матриці помилок для {area_name}: {e}")
        evaluation_results.append({
            "Model": "XGBoost",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": accuracy_area,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": prediction_evaluation_time,
        })
    except Exception as e:
        print(f"    Помилка під час оцінки на вибірці {area_name}: {e}")
        evaluation_results.append({
            "Model": "XGBoost",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
if evaluation_results:
    results_df = pd.DataFrame(evaluation_results)
```

```
    print("\nРезультати оцінки XGBoost на кожній тестовій вибірці:")
    print(results_df)
    valid_results_df = results_df.dropna(subset=['Overall Accuracy'])
    if not valid_results_df.empty:
        average_results = valid_results_df.groupby("Feature Set")["Overall Accuracy"].mean().reset_index()
        average_results = average_results.rename(columns={"Overall Accuracy": "Average Overall Accuracy (All Valid Test Sets)"})
        print("\nСередня загальна точність XGBoost по всіх валідних тестових вибірках:")
        print(average_results)
    else:
        print("\nНемає валідних результатів оцінки для розрахунку середньої точності.")
    all_test_y_true_encoded = label_encoder.transform(all_test_y_true_original)
    train_area_test_y_true_encoded = label_encoder.transform(train_area_test_y_true_original)
    external_test_y_true_encoded = label_encoder.transform(external_test_y_true_original)
    if len(all_test_y_true_encoded) > 0:
        overall_accuracy = accuracy_score(all_test_y_true_encoded, all_test_y_pred_encoded)
        overall_f1_score = f1_score(all_test_y_true_encoded, all_test_y_pred_encoded, average='weighted', labels=label_encoder.classes_, zero_division=0)
        print(f"Загальна точність (ВСІ тестові вибірки разом): {overall_accuracy:.4f}")
        print(f"Загальний F1-score (ВСІ тестові вибірки разом, weighted): {overall_f1_score:.4f}")
    else:
        print("Недостатньо даних для розрахунку загальних метрик для ВСІХ тестових вибірок.")
    if len(train_area_test_y_true_encoded) > 0:
        train_area_accuracy = accuracy_score(train_area_test_y_true_encoded, train_area_test_y_pred_encoded)
        train_area_f1_score = f1_score(train_area_test_y_true_encoded, train_area_test_y_pred_encoded, average='weighted', labels=label_encoder.classes_, zero_division=0)
        print(f"\nТочність (тестові частини НАВЧАЛЬНИХ територій): {train_area_accuracy:.4f}")
        print(f"F1-score (тестові частини НАВЧАЛЬНИХ територій, weighted): {train_area_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для тестових частин НАВЧАЛЬНИХ територій.")
    if len(external_test_y_true_encoded) > 0:
        external_accuracy = accuracy_score(external_test_y_true_encoded, external_test_y_pred_encoded)
        external_f1_score = f1_score(external_test_y_true_encoded, external_test_y_pred_encoded, average='weighted', labels=label_encoder.classes_, zero_division=0)
        print(f"\nТочність (ЗОВНІШНІ тестові території): {external_accuracy:.4f}")
        print(f"F1-score (ЗОВНІШНІ тестові території, weighted): {external_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для ЗОВНІШНІХ тестових територій.")
else:
    print("Немає зібраних результатів оцінки.")
```

А.6 SVM модель з лінійним ядром і без вегетаційних індексів

Так як код SVM без вегетаційних індексів і вегетаційних майже однаковий, то для скорочення приводжу тільки приклад без вегетаційних індексів.

```
import pandas as pd
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
import time
import os
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='sklearn')
warnings.filterwarnings('ignore', category=FutureWarning, module='sklearn')
train_areas_folders = [
    "D:/diplom/test1/",
    "D:/diplom/test2/",
    "D:/diplom/test3/",
    "D:/diplom/test4/",
    "D:/diplom/test6/",
]
test_areas_folders = [
    "D:/diplom/test5/",
    "D:/diplom/test7/",
    "D:/diplom/test9/",
]
TRAIN_BANDS_FILENAME = "train_bands.csv"
TEST_BANDS_FILENAME = "test_bands.csv"
TRAIN_LABELS_FILENAME = "train_vega_labels.csv" #для вега, зараз не користуюся
TRAIN_TEST_LABELS_FILENAME = "test_vega_labels.csv" #для вега, зараз не користуюся
EXTERNAL_TEST_LABELS_FILENAME = "test_labels.csv" #для вега, зараз не користуюся
LINEAR_SVM_C = 1.0
LINEAR_SVM_LOSS = 'squared_hinge'
LINEAR_SVM_DUAL = False
LINEAR_SVM_RANDOM_STATE = 42
LINEAR_SVM_MAX_ITER = 10000
label_mapping = {
    1: "Штучні об'єкти",
    2: "Пшениця",
    3: "Ріпак",
    4: "Гречка",
    5: "Кукурудза",
    6: "Культури за 2022",
    7: "Соняшник",
    8: "Соєві боби",
    9: "Цукровий буряк",
```

```
    10: "Інші культури",
    11: "Ліс",
    12: "Пасовища",
    13: "Гола земля",
    14: "Вода",
    15: "Водно-болотна місцевість",
    16: "Ячмінь",
    17: "Горох",
    18: "Люцерна",
    19: "Сади, парки",
    20: "Виноград",
    21: "Картопля",
    22: "Не культивується",
    23: "Пошкоджений ліс",
}
train_area_test_splits = {}
external_test_areas = {}
def load_data_from_folder(folder, features_filename, labels_filename):
    features_path = os.path.join(folder, features_filename)
    labels_path = os.path.join(folder, labels_filename)
    if not os.path.exists(features_path):
        print(f"    Помилка: Файл ознак не знайдено: {features_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    if not os.path.exists(labels_path):
        print(f"    Помилка: Файл міток не знайдено: {labels_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    try:
        features_df = pd.read_csv(features_path)
        labels_series = pd.read_csv(labels_path).iloc[:, 0]
        if len(features_df) != len(labels_series):
            print(f"    Попередження: Кількість рядків в ознаках ({len(features_df)}) та мітках ({len(labels_series)}) не співпадає у папці {folder}. Неможливо завантажити дані з {folder}.")
            return None, None, None
        print(f"    Дані з {folder} успішно завантажено. Розмір ознак: {features_df.shape}, міток: {labels_series.shape}")
        return features_df, labels_series, sorted(list(labels_series.unique()))
    except MemoryError as e:
        print(f"    Помилка MemoryError при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
    except Exception as e:
        print(f"    Помилка при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
all_train_features_dfs = []
all_train_labels_series = []
all_unique_train_labels_set_original = set()
for folder in train_areas_folders:
    features_df, labels_series, unique_labels_area = load_data_from_folder(
        folder, TRAIN_BANDS_FILENAME, TRAIN_LABELS_FILENAME
```

```
    )
    if features_df is not None:
        all_train_features_dfs.append(features_df)
        all_train_labels_series.append(labels_series)
        all_unique_train_labels_set_original.update(unique_labels_area)
if not all_train_features_dfs:
    print("Критична помилка: Не вдалося завантажити навчальні дані з жодної папки. Зупиняємо виконання.")
    exit()
X_train_combined = pd.concat(all_train_features_dfs, ignore_index=True)
y_train_combined = pd.concat(all_train_labels_series, ignore_index=True)
print(f"  Об'єднані навчальні дані (Bands Only) завантажено. Загальний розмір ознак: {X_train_combined.shape}, міток: {y_train_combined.shape}")
all_test_y_true_original = []
all_test_y_pred_encoded = []
train_area_test_y_true_original = []
train_area_test_y_pred_encoded = []
external_test_y_true_original = []
external_test_y_pred_encoded = []
all_unique_test_labels_set_original = set()
for folder in train_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження тестової частини з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, TRAIN_TEST_LABELS_FILENAME
    )
    if X_test_area is not None:
        train_area_test_splits[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_test_labels_set_original.update(unique_labels_area)
for folder in test_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження даних з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_BANDS_FILENAME, EXTERNAL_TEST_LABELS_FILENAME
    )
    if X_test_area is not None:
        external_test_areas[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_test_labels_set_original.update(unique_labels_area)
all_original_unique_labels_in_train = sorted(list(all_unique_train_labels_set_original))
all_original_unique_labels_overall = sorted(list(all_unique_train_labels_set_original.union(all_unique_test_labels_set_original)))
all_original_target_names = [label_mapping.get(label, f"Код {label}") for label in all_original_unique_labels_overall]
print(f"\nВсі унікальні ЧИСЛОВІ мітки (навчання + тест, оригінальні): {all_original_unique_labels_overall}")
print(f"Назви класів для матриць помилок та звітів (оригінальні): {all_original_target_names}")
label_encoder = LabelEncoder()
label_encoder.fit(all_original_unique_labels_overall)
y_train_encoded = label_encoder.transform(y_train_combined)
print(f"  Навчальні мітки закодовано. Оригінальні унікальні: {sorted(list(np.unique(y_train_combined)))}. Закодовані: {np.unique(y_train_encoded)}")
```

```
print(f"  Унікальні закодовані мітки у НАВЧАЛЬНИХ даних (без семплінгу, після кодування): {np.unique(y_train_encoded)}")
print(f"  Максимальна закодована мітка у НАВЧАЛЬНИХ даних: {y_train_encoded.max()}")
print(f"  Кількість унікальних закодованих міток у НАВЧАЛЬНИХ даних: {len(np.unique(y_train_encoded))}")
encoded_target_names = [label_mapping.get(original_label, f"Код {original_label}") for original_label in label_encoder.classes_]
print(f"  Назви класів для звітів (відповідно до закодованих міток): {encoded_target_names}")
scaler = StandardScaler()
print("  Навчання StandardScaler на навчальних даних (Bands Only)...")
X_train_scaled = scaler.fit_transform(X_train_combined)
print("  Навчальні дані (Bands Only) масштабовано.")
model = LinearSVC(
    C=LINEAR_SVM_C,
    loss=LINEAR_SVM_LOSS,
    dual=LINEAR_SVM_DUAL,
    random_state=LINEAR_SVM_RANDOM_STATE,
    max_iter=LINEAR_SVM_MAX_ITER
)
print(f"  Навчання моделі Linear SVM (C={LINEAR_SVM_C}, loss='{LINEAR_SVM_LOSS}', dual={LINEAR_SVM_DUAL}, max_iter={LINEAR_SVM_MAX_ITER}) на масштабованих даних (Bands Only)...")
start_time_train = time.time()
try:
    model.fit(X_train_scaled, y_train_encoded)
    end_time_train = time.time()
    training_time = end_time_train - start_time_train
    print(f"  Навчання завершено за {training_time:.2f} секунд.")
except Exception as e:
    print(f"  Помилка під час навчання моделі: {e}")
    training_time = np.nan
    print("Критична помилка: Навчання моделі не вдалося. Зупиняємо виконання.")
    exit()
evaluation_results = []
all_test_sets = {**train_area_test_splits, **external_test_areas}
if not all_test_sets:
    print("Немає тестових даних для оцінки.")
else:
    for area_name, (X_test_area, y_test_area_original, unique_labels_test_area) in all_test_sets.items():
        print(f"\n  Оцінка на вибірці: {area_name} (Bands Only)")
        print("    Кодування тестових міток...")
        try:
            y_test_area_encoded = label_encoder.transform(y_test_area_original)
            print("    Тестові мітки закодовано.")
        except ValueError as e:
            print(f"    Помилка під час кодування тестових міток для {area_name}: {e}. Можливо, в тестових даних є мітки, яких немає в навчальних. Пропускаємо оцінку.")
            evaluation_results.append({
                "Model": "Linear SVM",
                "Feature Set": "Bands Only",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
```

```
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
        continue
    except Exception as e:
        print(f"    Невідома помилка під час кодування тестових міток для {area_name}: {e}. Пропускаємо оцінку.")
        evaluation_results.append({
            "Model": "Linear SVM",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
        continue
    print("    Масштабування тестових даних (Bands Only)...")
    try:
        X_test_area_scaled = scaler.transform(X_test_area)
        print("    Тестові дані (Bands Only) масштабовано.")
    except Exception as e:
        print(f"    Помилка під час масштабування тестових даних для {area_name}: {e}. Пропускаємо оцінку.")
        evaluation_results.append({
            "Model": "Linear SVM",
            "Feature Set": "Bands Only",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
        continue
    start_time_predict = time.time()
    try:
        y_pred_area_encoded = model.predict(X_test_area_scaled)
        end_time_predict = time.time()
        prediction_evaluation_time = end_time_predict - start_time_predict
        print(f"    Прогнозування завершено за {prediction_evaluation_time:.4f} секунд.")
        all_test_y_true_original.extend(y_test_area_original) # Зберігаємо оригінальні мітки
        all_test_y_pred_encoded.extend(y_pred_area_encoded) # Зберігаємо закодовані прогнози
        if area_name in [os.path.basename(f.rstrip('/')) for f in train_areas_folders]:
            train_area_test_y_true_original.extend(y_test_area_original)
            train_area_test_y_pred_encoded.extend(y_pred_area_encoded)
        elif area_name in [os.path.basename(f.rstrip('/')) for f in test_areas_folders]:
            external_test_y_true_original.extend(y_test_area_original)
            external_test_y_pred_encoded.extend(y_pred_area_encoded)
        accuracy_area = accuracy_score(y_test_area_encoded, y_pred_area_encoded)
```

```
            print(f"\n    Загальна точність на вибірці {area_name}: {accuracy_area:.4f}")
            print(f"\n    Звіт про класифікацію на вибірці {area_name}:")
            try:
                unique_test_labels_encoded = np.unique(y_test_area_encoded)
                filtered_target_names = [encoded_target_names[i] for i in unique_test_labels_encoded
if i < len(encoded_target_names)]
                report = classification_report(y_test_area_encoded, y_pred_area_encoded,
                                               labels=unique_test_labels_encoded,
                                               target_names=filtered_target_names,
                                               zero_division=0)
                print(report)
            except Exception as e:
                print(f"    Помилка при генерації звіту про класифікацію для {area_name}: {e}")
            print(f"\n    Матриця помилок для вибірки {area_name}:")
            try:
                cm = confusion_matrix(y_test_area_encoded, y_pred_area_encoded,
                                      labels=unique_test_labels_encoded)
                plt.figure(figsize=(12, 10))
                sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                            xticklabels=filtered_target_names, yticklabels=filtered_target_names)
                plt.xlabel('Прогнозовані класи')
                plt.ylabel('Реальні класи')
                plt.title(f'Матриця помилок для {area_name} (Linear SVM, Bands Only)')
                plt.show()
            except Exception as e:
                print(f"    Помилка при побудові матриці помилок для {area_name}: {e}")
            # Зберігаємо результат
            evaluation_results.append({
                "Model": "Linear SVM",
                "Feature Set": "Bands Only",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": accuracy_area,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": prediction_evaluation_time,
            })
        except Exception as e:
            print(f"    Помилка під час оцінки на вибірці {area_name}: {e}")
            evaluation_results.append({
                "Model": "Linear SVM",
                "Feature Set": "Bands Only",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": np.nan,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": np.nan,
            })
if evaluation_results:
    results_df = pd.DataFrame(evaluation_results)
    print("\nРезультати оцінки Linear SVM на кожній тестовій вибірці:")
    print(results_df)
    valid_results_df = results_df.dropna(subset=['Overall Accuracy'])
```

```
    if not valid_results_df.empty:
        average_results = valid_results_df.groupby("Feature Set")["Overall Accuracy"].mean().reset_index()
        average_results = average_results.rename(columns={"Overall Accuracy": "Average Overall Accuracy (All Valid Test Sets)"})
        print("\nСередня загальна точність Linear SVM по всіх валідних тестових вибірках:")
        print(average_results)
    else:
        print("\nНемає валідних результатів оцінки для розрахунку середньої точності.")
    all_test_y_true_encoded = label_encoder.transform(all_test_y_true_original)
    train_area_test_y_true_encoded = label_encoder.transform(train_area_test_y_true_original)
    external_test_y_true_encoded = label_encoder.transform(external_test_y_true_original)
    if len(all_test_y_true_encoded) > 0:
        overall_accuracy = accuracy_score(all_test_y_true_encoded, all_test_y_pred_encoded)
        overall_f1_score = f1_score(all_test_y_true_encoded, all_test_y_pred_encoded, average='weighted', labels=label_encoder.classes_, zero_division=0)
        print(f"Загальна точність (ВСІ тестові вибірки разом): {overall_accuracy:.4f}")
        print(f"Загальний F1-score (ВСІ тестові вибірки разом, weighted): {overall_f1_score:.4f}")
    else:
        print("Недостатньо даних для розрахунку загальних метрик для ВСІХ тестових вибірок.")
    if len(train_area_test_y_true_encoded) > 0:
        train_area_accuracy = accuracy_score(train_area_test_y_true_encoded, train_area_test_y_pred_encoded)
        train_area_f1_score = f1_score(train_area_test_y_true_encoded, train_area_test_y_pred_encoded, average='weighted', labels=label_encoder.classes_, zero_division=0)
        print(f"\nТочність (тестові частини НАВЧАЛЬНИХ територій): {train_area_accuracy:.4f}")
        print(f"F1-score (тестові частини НАВЧАЛЬНИХ територій, weighted): {train_area_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для тестових частин НАВЧАЛЬНИХ територій.")
    if len(external_test_y_true_encoded) > 0:
        external_accuracy = accuracy_score(external_test_y_true_encoded, external_test_y_pred_encoded)
        external_f1_score = f1_score(external_test_y_true_encoded, external_test_y_pred_encoded, average='weighted', labels=label_encoder.classes_, zero_division=0)
        print(f"\nТочність (ЗОВНІШНІ тестові території): {external_accuracy:.4f}")
        print(f"F1-score (ЗОВНІШНІ тестові території, weighted): {external_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для ЗОВНІШНІХ тестових територій.")
else:
    print("Немає зібраних результатів оцінки.")
```

А.7 SVM з не лінійним ядром з вегетаційними індексами

Так як код SVM без вегетаційних індексів і вегетаційних майже однаковий, то для скорочення приводжу тільки приклад з вегетаційними індексами.

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
import time
import os
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore', category=UserWarning, module='sklearn')
warnings.filterwarnings('ignore', category=FutureWarning, module='sklearn')
train_areas_folders = [
    "D:/diplom/test1/",
    "D:/diplom/test2/",
    "D:/diplom/test3/",
    "D:/diplom/test4/",
    "D:/diplom/test6/",
]
test_areas_folders = [
    "D:/diplom/test5/",
    "D:/diplom/test7/",
    "D:/diplom/test9/",
]
TRAIN_VEGA_FILENAME = "train_vega.csv"
TEST_VEGA_FILENAME = "test_vega.csv"
TRAIN_LABELS_FILENAME = "train_vega_labels.csv" # тут  вега взяв
TRAIN_TEST_LABELS_FILENAME = "test_vega_labels.csv" # тут вега взяв
EXTERNAL_TEST_LABELS_FILENAME = "test_labels.csv" # тут вега взяв
# --- Налаштування: Параметри Семплінгу ---
TRAINING_SAMPLE_SIZE_PER_AREA = 40000 # Кількість зразків з кожного навчального датасету
TESTING_SAMPLE_SIZE_PER_AREA = 80000 # Кількість зразків з кожного тестового датасету
SVM_KERNEL = 'rbf'
SVM_GAMMA = 'scale'
SVM_C = 1.0
SVM_RANDOM_STATE = 42
label_mapping = {
    1: "Штучні об'єкти",
    2: "Пшениця",
    3: "Ріпак",
    4: "Гречка",
    5: "Кукурудза",
    6: "Культури за 2022",
    7: "Соняшник",
    8: "Соєві боби",
    9: "Цукровий буряк",
```

```
    10: "Інші культури",
    11: "Ліс",
    12: "Пасовища",
    13: "Гола земля",
    14: "Вода",
    15: "Водно-болотна місцевість",
    16: "Ячмінь",
    17: "Горох",
    18: "Люцерна",
    19: "Сади, парки",
    20: "Виноград",
    21: "Картопля",
    22: "Не культивується", # Додано
    23: "Пошкоджений ліс",
}
train_area_test_splits = {}
external_test_areas = {}
def load_data_from_folder(folder, features_filename, labels_filename, sample_size=None):
     features_path = os.path.join(folder, features_filename)
    labels_path = os.path.join(folder, labels_filename)

    if not os.path.exists(features_path):
        print(f"    Помилка: Файл ознак не знайдено: {features_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    if not os.path.exists(labels_path):
        print(f"    Помилка: Файл міток не знайдено: {labels_path}. Неможливо завантажити дані з {folder}.")
        return None, None, None
    try:
        features_df = pd.read_csv(features_path)
        labels_series = pd.read_csv(labels_path).iloc[:, 0]
        if len(features_df) != len(labels_series):
            print(f"    Попередження: Кількість рядків в ознаках ({len(features_df)}) та мітках ({len(labels_series)}) не співпадає у папці {folder}. Неможливо завантажити дані з {folder}.")
            return None, None, None
        print(f"  Дані з {folder} успішно завантажено. Розмір ознак: {features_df.shape}, міток: {labels_series.shape}")
        if sample_size is not None and len(features_df) > sample_size:
            sampled_indices = features_df.sample(n=sample_size, random_state=SVM_RANDOM_STATE).index
            features_df = features_df.loc[sampled_indices].reset_index(drop=True)
            labels_series = labels_series.loc[sampled_indices].reset_index(drop=True)
            print(f"    Застосування семплінгу до даних з {folder}: зменшення до {sample_size} зразків...")
            print(f"    Дані з {folder} після семплінгу. Розмір ознак: {features_df.shape}, міток: {labels_series.shape}")
        return features_df, labels_series, sorted(list(labels_series.unique()))
    except MemoryError as e:
        print(f"      Помилка MemoryError при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
```

```
    except Exception as e:
        print(f"  Помилка при завантаженні даних з папки {folder}: {e}. Неможливо завантажити.")
        return None, None, None
all_train_features_dfs = []
all_train_labels_series = []
all_unique_train_labels_set_original = set()
for folder in train_areas_folders:
    features_df, labels_series, unique_labels_area = load_data_from_folder(
        folder, TRAIN_VEGA_FILENAME, TRAIN_LABELS_FILENAME, TRAINING_SAMPLE_SIZE_PER_AREA
    )
    if features_df is not None:
        all_train_features_dfs.append(features_df)
        all_train_labels_series.append(labels_series)
        all_unique_train_labels_set_original.update(unique_labels_area)
if not all_train_features_dfs:
    print("Критична помилка: Не вдалося завантажити навчальні дані з жодної папки. Зупиняємо виконання.")
    exit()
print("  Об'єднання завантажених навчальних даних...")
X_train_combined = pd.concat(all_train_features_dfs, ignore_index=True)
y_train_combined = pd.concat(all_train_labels_series, ignore_index=True)
print(f"  Об'єднані навчальні дані (Vega Only) завантажено. Загальний розмір ознак: {X_train_combined.shape}, міток: {y_train_combined.shape}")
all_test_y_true = []
all_test_y_pred = [] #
train_area_test_y_true = []
train_area_test_y_pred = []
external_test_y_true = []
external_test_y_pred = []
all_unique_test_labels_set_original = set()
for folder in train_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження тестової частини з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_VEGA_FILENAME, TRAIN_TEST_LABELS_FILENAME, TESTING_SAMPLE_SIZE_PER_AREA
    )
    if X_test_area is not None:
        train_area_test_splits[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_test_labels_set_original.update(unique_labels_area)
for folder in test_areas_folders:
    area_name = os.path.basename(folder.rstrip('/'))
    print(f"    Завантаження даних з {area_name}...")
    X_test_area, y_test_area, unique_labels_area = load_data_from_folder(
        folder, TEST_VEGA_FILENAME, EXTERNAL_TEST_LABELS_FILENAME, TESTING_SAMPLE_SIZE_PER_AREA
    )
    if X_test_area is not None:
        external_test_areas[area_name] = (X_test_area, y_test_area, unique_labels_area)
        all_unique_test_labels_set_original.update(unique_labels_area)
```

```
all_original_unique_labels_in_train = sorted(list(all_unique_train_labels_set_original))
all_original_unique_labels_overall = sorted(list(all_unique_train_labels_set_original.union(all_unique_test_labels_set_original)))
all_original_target_names = [label_mapping.get(label, f"Код {label}") for label in all_original_unique_labels_overall]
print(f"\nВсі унікальні ЧИСЛОВІ мітки (навчання + тест, оригінальні): {all_original_unique_labels_overall}")
print(f"Назви класів для матриць помилок та звітів (оригінальні): {all_original_target_names}")
label_encoder = LabelEncoder()
label_encoder.fit(all_original_unique_labels_overall)
y_train_encoded = label_encoder.transform(y_train_combined)
print(f" Навчальні мітки закодовано. Оригінальні унікальні: {sorted(list(np.unique(y_train_combined)))}. Закодовані: {np.unique(y_train_encoded)}")
print(f" Унікальні закодовані мітки у НАВЧАЛЬНИХ даних (після семплінгу та кодування): {np.unique(y_train_encoded)}")
print(f" Максимальна закодована мітка у НАВЧАЛЬНИХ даних: {y_train_encoded.max()}")
print(f" Кількість унікальних закодованих міток у НАВЧАЛЬНИХ даних: {len(np.unique(y_train_encoded))}")
encoded_target_names = [label_mapping.get(original_label, f"Код {original_label}") for original_label in label_encoder.classes_]
print(f" Назви класів для звітів (відповідно до закодованих міток): {encoded_target_names}")
scaler = StandardScaler()
print(" Навчання StandardScaler на навчальних даних (Vega Only)...")
X_train_scaled = scaler.fit_transform(X_train_combined)
model = SVC(
    kernel=SVM_KERNEL,
    gamma=SVM_GAMMA,
    C=SVM_C,
    random_state=SVM_RANDOM_STATE
)
print(f" Навчання моделі SVM (kernel='{SVM_KERNEL}', gamma='{SVM_GAMMA}', C={SVM_C}) на масштабованих даних (Vega Only)...")
start_time_train = time.time()
try:
    model.fit(X_train_scaled, y_train_encoded)
    end_time_train = time.time()
    training_time = end_time_train - start_time_train
    print(f" Навчання завершено за {training_time:.2f} секунд.")
except Exception as e:
    print(f" Помилка під час навчання моделі: {e}")
    training_time = np.nan
    print("Критична помилка: Навчання моделі не вдалося. Зупиняємо виконання.")
    exit()
evaluation_results = []
all_test_sets = {**train_area_test_splits, **external_test_areas}
if not all_test_sets:
    print("Немає тестових даних для оцінки.")
else:
    for area_name, (X_test_area, y_test_area_original, unique_labels_test_area) in all_test_sets.items():
        print(f"\n Оцінка на вибірці: {area_name} (Vega Only)")
```

```
        print("    Кодування тестових міток...")
        try:
            y_test_area_encoded = label_encoder.transform(y_test_area_original)
            print("    Тестові мітки закодовано.")
        except ValueError as e:
            print(f"    Помилка під час кодування тестових міток для {area_name}: {e}. Можливо,
в тестових даних є мітки, яких немає в навчальних. Пропускаємо оцінку.")
            evaluation_results.append({
                "Model": "SVM",
                "Feature Set": "Vega Train, Vega Test",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": np.nan,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": np.nan,
            })
            continue
        except Exception as e:
            print(f"    Невідома помилка під час кодування тестових міток для {area_name}: {e}.
Пропускаємо оцінку.")
            evaluation_results.append({
                "Model": "SVM",
                "Feature Set": "Vega Train, Vega Test",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": np.nan,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": np.nan,
            })
            continue
        print("    Масштабування тестових даних (Vega Only)...")
        try:
            X_test_area_scaled = scaler.transform(X_test_area)
            print("    Тестові дані (Vega Only) масштабовано.")
        except Exception as e:
             print(f"     Помилка під час масштабування тестових даних для {area_name}: {e}.
Пропускаємо оцінку.")
             evaluation_results.append({
                 "Model": "SVM",
                 "Feature Set": "Vega Train, Vega Test", # Оновлено на Vega Train, Vega Test
                 "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                 "Test Set": area_name,
                 "Overall Accuracy": np.nan,
                 "Training Time (s)": training_time,
                 "Prediction/Evaluation Time (s)": np.nan,
             })
             continue
        start_time_predict = time.time()
        try:
            y_pred_area_encoded = model.predict(X_test_area_scaled) # SVM predict повертає
безпосередньо клас
            end_time_predict = time.time()
```

```
            prediction_evaluation_time = end_time_predict - start_time_predict
            print(f"    Прогнозування завершено за {prediction_evaluation_time:.4f} секунд.")
            all_test_y_true.extend(y_test_area_encoded)
            all_test_y_pred.extend(y_pred_area_encoded)

            if area_name in [os.path.basename(f.rstrip('/')) for f in train_areas_folders]:
                train_area_test_y_true.extend(y_test_area_encoded)
                train_area_test_y_pred.extend(y_pred_area_encoded)
            elif area_name in [os.path.basename(f.rstrip('/')) for f in test_areas_folders]:
                external_test_y_true.extend(y_test_area_encoded)
                external_test_y_pred.extend(y_pred_area_encoded)
            accuracy_area = accuracy_score(y_test_area_encoded, y_pred_area_encoded)
            print(f"\n    Загальна точність на вибірці {area_name}: {accuracy_area:.4f}")
            print(f"\n    Звіт про класифікацію на вибірці {area_name}:")
            try:
                unique_test_labels_encoded = np.unique(y_test_area_encoded)
                report = classification_report(y_test_area_encoded, y_pred_area_encoded,

labels=label_encoder.transform(all_original_unique_labels_overall),
                                               target_names=all_original_target_names,
                                               zero_division=0)
                print(report)
            except Exception as e:
                print(f"    Помилка при генерації звіту про класифікацію для {area_name}: {e}")
            print(f"\n    Матриця помилок для вибірки {area_name}:")
            try:
                cm = confusion_matrix(y_test_area_encoded, y_pred_area_encoded,
                                      labels=label_encoder.transform(all_original_unique_labels_overall))
                plt.figure(figsize=(12, 10))
                sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                            xticklabels=all_original_target_names, yticklabels=all_original_target_names)
                plt.xlabel('Прогнозовані класи')
                plt.ylabel('Реальні класи')
                plt.title(f'Матриця помилок для {area_name} (SVM, Vega Train, Vega Test)')
                plt.show()
            except Exception as e:
                print(f"    Помилка при побудові матриці помилок для {area_name}: {e}")

            evaluation_results.append({
                "Model": "SVM",
                "Feature Set": "Vega Train, Vega Test",
                "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
                "Test Set": area_name,
                "Overall Accuracy": accuracy_area,
                "Training Time (s)": training_time,
                "Prediction/Evaluation Time (s)": prediction_evaluation_time,
            })

        except Exception as e:
            print(f"    Помилка під час оцінки на вибірці {area_name}: {e}")
            evaluation_results.append({
                "Model": "SVM",
```

```
            "Feature Set": "Vega Train, Vega Test",
            "Training Areas": "+".join([os.path.basename(f.rstrip('/')) for f in train_areas_folders]),
            "Test Set": area_name,
            "Overall Accuracy": np.nan,
            "Training Time (s)": training_time,
            "Prediction/Evaluation Time (s)": np.nan,
        })
print("\n--- Зведені результати оцінки просторової переносимості SVM (Vega Train, Vega Test) ---")
if evaluation_results:
    results_df = pd.DataFrame(evaluation_results)
    print("\nРезультати оцінки SVM на кожній тестовій вибірці:")
    print(results_df)
    valid_results_df = results_df.dropna(subset=['Overall Accuracy'])
    if not valid_results_df.empty:
        average_results = valid_results_df.groupby("Feature Set")["Overall Accuracy"].mean().reset_index()
        average_results = average_results.rename(columns={"Overall Accuracy": "Average Overall Accuracy (All Valid Test Sets)"})
        print("\nСередня загальна точність SVM по всіх валідних тестових вибірках:")
        print(average_results)
    else:
        print("\nНемає валідних результатів оцінки для розрахунку середньої точності.")
    if all_test_y_true:
        all_test_y_true_np = np.array(all_test_y_true)
        all_test_y_pred_np = np.array(all_test_y_pred)
    else:
        all_test_y_true_np = np.array([])
        all_test_y_pred_np = np.array([])
    if train_area_test_y_true:
        train_area_test_y_true_np = np.array(train_area_test_y_true)
        train_area_test_y_pred_np = np.array(train_area_test_y_pred)
    else:
        train_area_test_y_true_np = np.array([])
        train_area_test_y_pred_np = np.array([])
    if external_test_y_true:
        external_test_y_true_np = np.array(external_test_y_true)
        external_test_y_pred_np = np.array(external_test_y_pred)
    else:
        external_test_y_true_np = np.array([])
        external_test_y_pred_np = np.array([])
    if len(all_test_y_true_np) > 0:
        overall_accuracy = accuracy_score(all_test_y_true_np, all_test_y_pred_np)
        overall_f1_score = f1_score(all_test_y_true_np, all_test_y_pred_np, average='weighted', labels=label_encoder.transform(all_original_unique_labels_overall), zero_division=0)
        print(f"Загальна точність (BCI тестові вибірки разом): {overall_accuracy:.4f}")
        print(f"Загальний F1-score (BCI тестові вибірки разом, weighted): {overall_f1_score:.4f}")
    else:
        print("Недостатньо даних для розрахунку загальних метрик для BCIX тестових вибірок.")
    if len(train_area_test_y_true_np) > 0:
```

```
        train_area_accuracy = accuracy_score(train_area_test_y_true_np, train_area_test_y_pred_np)
        train_area_f1_score = f1_score(train_area_test_y_true_np, train_area_test_y_pred_np, average='weighted', labels=label_encoder.transform(all_original_unique_labels_overall), zero_division=0)
        print(f"\nТочність (тестові частини НАВЧАЛЬНИХ територій): {train_area_accuracy:.4f}")
        print(f"F1-score (тестові частини НАВЧАЛЬНИХ територій, weighted): {train_area_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для тестових частин НАВЧАЛЬНИХ територій.")
    if len(external_test_y_true_np) > 0:
        external_accuracy = accuracy_score(external_test_y_true_np, external_test_y_pred_np)
        external_f1_score = f1_score(external_test_y_true_np, external_test_y_pred_np, average='weighted', labels=label_encoder.transform(all_original_unique_labels_overall), zero_division=0)
        print(f"\nТочність (ЗОВНІШНІ тестові території): {external_accuracy:.4f}")
        print(f"F1-score (ЗОВНІШНІ тестові території, weighted): {external_f1_score:.4f}")
    else:
        print("\nНедостатньо даних для розрахунку метрик для ЗОВНІШНІХ тестових територій.")
else:
    print("Немає зібраних результатів оцінки.")
```