

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«__» _____ 20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Обробка музика за допомогою NLP»

Виконав:

студент IV курсу, групи КІ-12

Тимофєєв Валерій Олександрович

Керівник:

асистент кафедри штучного інтелекту, доктор філософії,

Гаврилович Марія Павлівна

Консультант з економічного розділу:

доцент кафедри економічної кібернетики, к.е.н., доцент,

Рощина Надія Василівна

Консультант з нормоконтролю:

фахівець першої категорії кафедри штучного інтелекту, к.т.н., доцент,

Комариста Богдана Миколаївна

Рецензент:

професор кафедри ММСА, доктор технічних наук,

Кузнєцова Наталія Володимирівна

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

Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«15» січня 2025 р.

ЗАВДАННЯ
на дипломну роботу студенту
Тимофєєву Валерію Олександровичу

1. Тема роботи «Обробка музика за допомогою NLP», керівник роботи Гаврилович Марія Павлівна, доктор філософії, затверджені наказом по університету від «26» травня 2025 р. № 1759-с
2. Термін подання студентом роботи «09» червня 2025 року.
3. Вихідні дані до роботи – набір графіків та зображень.
4. Зміст роботи: дослідження застосування методів обробки природної мови для обробки попередньо закодованих музичних даних, з метою їх класифікації та подальшого аналізу.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): схеми алгоритмів, графіки, блок-схеми, візуалізації отриманих даних.
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.		

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Вивчення літератури за темою роботи	21.04.2025	Виконано
2	Підготовка першого розділу	28.04.2025	Виконано
3	Проведення дослідження за темою роботи	05.05.2025	Виконано
4	Підготовка другого розділу	12.05.2025	Виконано
5	Розробка програмного продукту	19.05.2025	Виконано
6	Підготовка третього розділу	26.05.2025	Виконано
7	Підготовка економічної частини	02.06.2025	Виконано
8	Оформлення дипломної роботи	09.06.2025	Виконано

Студент

Валерій ТИМОФЄЄВ

Керівник

Марія ГАВРИЛОВИЧ

РЕФЕРАТ

Дипломна робота: 93 с., 14 рис., 9 табл., 31 посилання, додаток.

ОБРОБКА МУЗИКИ, ОБРОБКА ПРИРОДНОЇ МОВИ, КЛАСИФІКАЦІЯ КОМПОЗИТОРІВ, ОБРОБКА MIDI, КОДУВАННЯ МУЗИКИ.

Об'єктом дослідження є обробка музичних даних, представлених у форматі MIDI.

Предметом дослідження є застосування методів обробки природної мови для аналізу та класифікації музичних композицій.

Метою роботи є перевірка припущення про роль фундаментальних характеристик нот у формуванні унікального стилю композитора.

У роботі досліджено можливості застосування методів NLP для аналізу музичних даних з метою класифікації творів за композиторами. Розроблено комплексний метод обробки MIDI-файлів з подальшим створенням словника унікальних музичних символів. Цей метод кодує фундаментальні характеристики нот (висота, динаміка, тривалість), а також групує одночасно зіграні ноти в акорди.

Виконано порівняльний аналіз п'яти методів векторизації музичних послідовностей: усереднення, стандартного відхилення, комбінованого методу, TF-IDF та максимального об'єднання.

З використанням алгоритмів машинного навчання Random Forest, SVM, KNN та Logistic Regression було виконано класифікацію музичних творів. За її результатами встановлено, що комбінований метод усереднення та стандартного відхилення найефективніше зберігає як загальний стиль музичного твору, так і його внутрішню варіативність.

Отримані результати підтверджують гіпотезу про суттєвий вплив фундаментальних характеристик нот на формування унікального музичного стилю композитора та демонструють перспективність застосування методів NLP для аналізу музичних даних.

ABSTRACT

Bachelor's thesis: 93 p., 14 figures, 9 tables, 31 references, appendix.

MUSIC PROCESSING, NATURAL LANGUAGE PROCESSING, COMPOSERS CLASSIFICATION, MIDI PROCESSING, MUSIC ENCODING.

The object of the study is the processing of the MIDI musical data.

The subject of research is the establishment of the NLP methods for the analysis and classification of the musical compositions.

The purpose of the work is to verify the assumption about the role of the fundamental characteristics of notes in shaping the unique composers' style.

The work investigates the possibilities of applying NLP methods for musical data analysis with the purpose of classifying songs by composers. A method for processing MIDI files with subsequent creation of a dictionary of unique musical symbols has been developed. This method encodes fundamental note characteristics (pitch, dynamics, duration) as well as groups simultaneously played notes into chords.

A comprehensive analysis of five methods of vectorization of musical sequences is presented: averaging, standard deviation, combined method, TF-IDF and max pooling.

Using the Random Forest, SVM, KNN and Logistic Regression algorithms, the classification of musical works was performed. Based on the obtained results, it was determined that the combination method of averaging and standard deviation most effectively preserves both the original style of musical creativity and its internal variability.

The obtained results confirm the hypothesis about the influence of the fundamental characteristics of notes on the composers' unique musical style and demonstrates the promising future of using NLP methods for analyzing musical data.

ЗМІСТ

ВСТУП.....	9
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ГАЛУЗЕЙ NLP ТА MIR.....	11
1.1 Актуальність задачі аналізу і класифікації музичних творів.....	11
1.2 Сучасний вигляд предметної області.....	12
1.3 Аналіз існуючих підходів.....	17
1.4 Постановка задачі.....	18
Висновки до розділу 1.....	20
РОЗДІЛ 2 ОГЛЯД МЕТОДІВ РОБОТИ З МУЗИЧНИМИ ДАНИМИ.....	22
2.1 Огляд методів для комп'ютерної обробки музики.....	22
2.2 Використання технік NLP для роботи з музикою.....	28
Висновки до розділу 2.....	34
РОЗДІЛ 3 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ.....	36
3.1 Опис і передобробка датасету.....	36
3.2 Кодування музичних даних.....	36
3.3 Векторизація музичних даних.....	40
3.4 Аналіз отриманих результатів.....	43
Висновки до розділу 3.....	45
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ЗАПРОПОНОВАНОГО МЕТОДУ.....	47
4.1 Постановка задачі проектування.....	47
4.2 Обґрунтування функцій програмного продукту.....	48
4.3 Обґрунтування системи параметрів програмного продукту.....	51
4.4 Аналіз експертного оцінювання параметрів.....	52
4.5 Аналіз рівня якості варіантів реалізації функцій.....	56
4.6 Економічний аналіз варіантів розробки ПП.....	58
4.7 Вибір кращого варіанту ПП техніко-економічного рівня.....	63
Висновки до розділу 4.....	64

	7
ВИСНОВКИ.....	65
ПЕРЕЛІК ПОСИЛАНЬ.....	67
ДОДАТОК А МАТРИЦЯ НЕВІДПОВІДНОСТЕЙ.....	71
ДОДАТОК Б Т-РОЗПОДІЛЕНЕ ВКЛАДЕННЯ СТОХАСТИЧНОЇ БЛИЗКОСТІ.....	72
ДОДАТОК В ЛІСТИНГ ПРОГРАМИ.....	73

СПИСОК СКОРОЧЕНЬ

NLP – Natural language processing

MIDI – Musical Instrument Digital Interface

MIR – Music information retrieval

DAW – Digital audio workstation

ФВА – Функціонально-вартісний аналіз

ПП – Програмний продукт

ВСТУП

У сучасному світі¹ кількість доступного музичного контенту досягнула неосяжних масштабів. Щодня створюються тисячі нових пісень і композицій, до яких цифрові стримінгові платформи надають доступ мільярдам. Це створює нові виклики для систем автоматизованого аналізу, систематизації та рекомендації музики.

Паралельно з цим, методи NLP демонструють видатні результати в аналізі та класифікації текстових даних. Символьне представлення музичних MIDI-файлів надає можливості для застосування цих технологій у контексті музики. Перетин галузей MIR та NLP має значний потенціал в сфері інноваційних методів аналізу, які зможуть враховувати як структурні закономірності композицій, так і фундаментальні характеристики окремих нот.

Метою цієї роботи є дослідження та розробка методів застосування технік NLP для аналізу символьних музичних даних з метою класифікації музичних творів за композиторами. Це передбачає, насамперед, створення гнучкого і ефективного способу кодування музики для подальшого застосування на ній алгоритмів машинного навчання.

Об'єктом дослідження є обробка музичних даних, представлених у форматі MIDI, з подальшим застосуванням на ній методів та алгоритмів NLP.

Предметом дослідження є адаптація технік NLP, а саме токенізації, векторизації та класифікації, для роботи з музикою.

Наукова новизна роботи полягає в дослідженні можливості використання методів NLP для аналізу музики, з урахуванням як

¹Тут і нижче використано такий інструмент штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT, виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КІП ім. Ігоря Сікорського (протокол №11 Вченої ради КІП ім. Ігоря Сікорського від 11 грудня 2023 р.).

фундаментальних характеристик нот, так і високорівневих музичних патернів, що дозволить виявляти унікальні стилістичні особливості композиторів на всіх рівнях музичного виконання.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ГАЛУЗЕЙ NLP ТА MIR

1.1 Актуальність задачі аналізу і класифікації музичних творів

Технологія NLP пройшла значний еволюційний шлях від рудиментарних систем на основі правил, таких як ELIZA 1966 року [1], дійшовши до складних моделей глибокого навчання доступних широкій аудиторії, таких як Chat GPT. Паралельно з цими інноваціями обчислювальне дослідження музики також адаптувало використання методів обробки природної мови для обробки цифрових музичних даних як для аналітичного застосування, так і для генеративних задач. Ця ідея використання методів обробки природної мови з символічними музичними даними стає все більш поширеною всередині спільноти MIR.

З появою цифрової музики та інтернету, величезна кількість всілякого музичного контенту стала доступною для мільйонів користувачів по всьому світі. З появою мільйонів виконавців та пісень, користувачам стає дедалі важче знаходити цікавий саме для них музичний контент. Крім того, нескінченна кількість доступної музики відкрила нові можливості для дослідників, які працюють над аналізом музичної інформації. Це, в свою чергу, надає слухачам нові можливості для пошуку музики за персоналізованими рекомендаціями та музичною навігацією.

Через специфіку структури музики, її порівняння, або векторизація, є складною задачею. Ця складність пов'язана, насамперед, з неоднорідністю даних, тобто музика може зберігатись в декількох різних форматах, процес обробки яких дуже сильно відрізняється. Іншою складністю є спосіб представлення музики: під час роботи з музикою потрібно розуміти, яким чином її обробляти та які її характеристики враховувати. Ще однією проблемою є висока вартість оцінки контекстно-залежних систем — відсутність достатньо великих систематизованих наборів даних робить кожен оцінку трудомісткою.

Таким чином галузь MIR зіткнулася з численними викликами, пов'язаними з неоднорідністю даних, складністю представлення музичних структур, тощо. Використання методів обробки природної мови в контексті аналізу музики може відкрити нові можливості для подолання цих викликів. Символьне представлення музики (через MIDI-файли) дозволяє застосовувати класичні підходи NLP для вирішення задач аналізу та класифікації музичного контенту. Це особливо сьогодні, коли музичні стрімінгові сервіси потребують точних алгоритмів рекомендацій.

Застосування методів обробки природної мови до оброблених музичних даних дозволить покращити і розширити існуючі системи аналізу, пошуку та рекомендації музичного контенту.

1.2 Сучасний вигляд предметної області

На сьогоднішній день дослідженням комп'ютерного аналізу музики займається галузь MIR. MIR це новітній напрямок досліджень із комп'ютерного аналізу музики, що має спільні риси з багатьма науковими дисциплінами (рис. 1.1), наприклад, музикознавством, фізичною акустикою, психологією, комп'ютерними науками, машинним навчанням, тощо. Ця галузь охоплює теми досліджень, пов'язані з розумінням, моделюванням, обробкою та створенням музики. Дослідження MIR можуть бути умовно поділені на три основні категорії: аналіз, обробка та генерація.

Аналогічним іншим сферам, MIR поступово перейшов від парадигми орієнтованості на знання, тобто знання надаються алгоритму дослідником, до парадигми орієнтованості на дані – знання отримуються саме з даних, без обов'язкової участі дослідника. Така зміна обумовлена розвитком відповідних суміжних сфер, зокрема сфери штучного інтелекту.

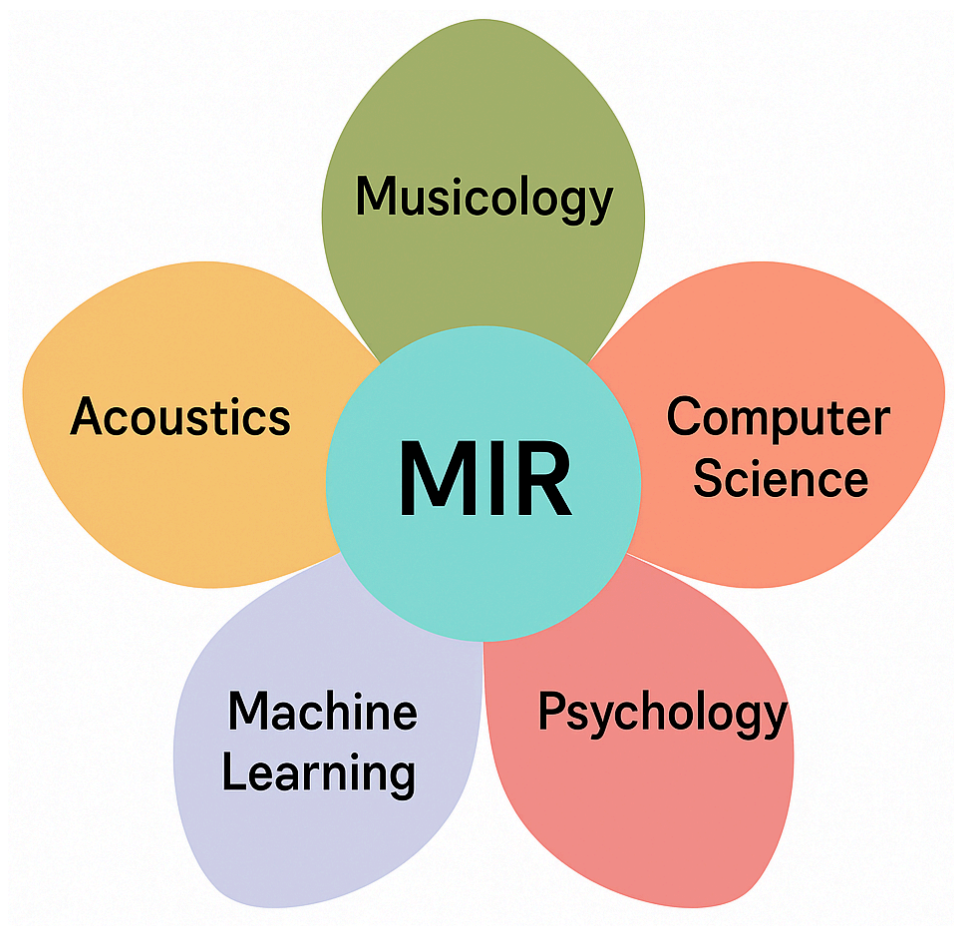


Рисунок 1.1 – Діаграма суміжних до MIR наук

Далі можна розглянути суміжну до MIR сферу – теорію музики. Фундаментальним об’єктом дослідження музичної теорії є нота. Вона описується кількома ключовими характеристиками, насамперед висотою, динамікою та довжиною. Висота ноти в музичній теорії є аналогом частоти в фізиці, динаміка та ритм в свою чергу є аналогами амплітуди та змінної тривалості нот та пауз у музичній хвилі [6].

Розглянемо, наприклад, фізичне обґрунтування гармонізації нот До та Соль. Ці дві ноти мають цілочисельні кратні їхніх основних частот, що розташовані близько одна до одної. Насамперед, нота До першої октави має частоту 261.63 Гц, а нота Соль в цій же октаві, відповідно, має частоту 392 Гц. Подвоєна частота цієї ноти Соль в свою чергу становить 784 Гц, що є

майже ідентичним потроєній частоті ноти До. Таким чином, фізичний контекст музичної теорії є необхідним для справжнього розуміння музики.

Аудіофайли фіксують та сприймають музику як набір звуків, які можна описати різними формами хвиль та спектрограмами. Водночас символні файли, наприклад музичні партитури або MIDI файли, описують музику використовуючи абстрактні нотації, які зазвичай складаються з нот, акордів, інтервалів, всіляких послідовностей, тощо. Хоча символне представлення музики і потребує складніших систем позначень, воно дозволяє досліджувати музику на високому рівні, тобто аналізувати послідовності акордів, гармонію, форму, текстуру, динамку, тощо. На практиці символне представлення музики є і залишається домінуючим в сфері цифрового продюсування музики, де зазвичай використовується формат MIDI, що є розповсюдженим стандартом серед робочих аудіо станцій (DAW).

Навіть поза межами дисципліни комп'ютерних наук, в колах дослідників музичної теорії, постійно проводяться паралелі між музикою та природною мовою людини. В такому контексті музику розглядають як окрему лінгвістичну систему, що має схожу природною мовою структурну та, що найважливіше, спільну ціль – комунікацію між людьми. Ці самі паралелі також вивчаються в чисельних роботах з галузей NLP та MIR.

Символьна музична репрезентація, як і мова людини, є семіотичною системою [4], що спирається на конкретну послідовність розташування заздалегідь визначених символів. В рамках певного музичного твору ці символи можуть бути сегментовані або згруповані на різних рівнях (рис. 1.2). Наприклад, звичайний текст може бути розбитий на пункти, підпункти, абзаци, речення, фрази, слова, літери, тощо. Так само і музичний твір може містити в собі складові одиниці, такі як секції, музичні фрази, акорди, ноти, тощо [6].

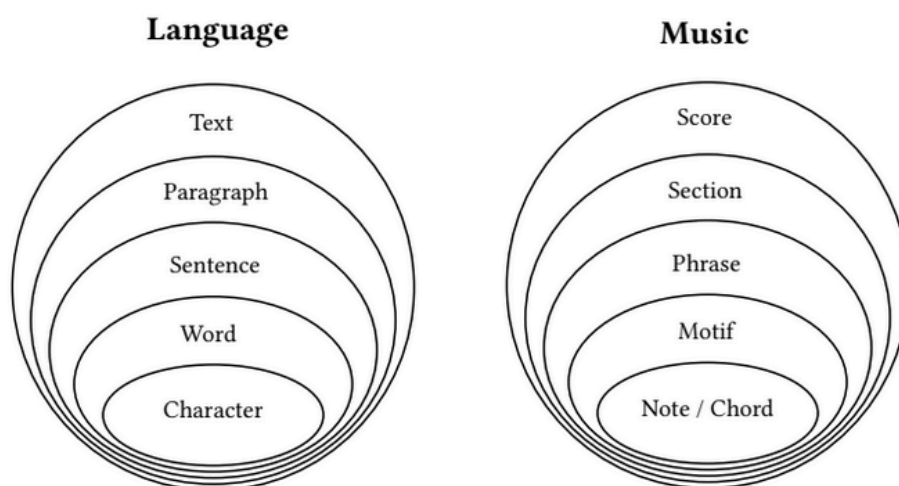


Рисунок 1.2 – Порівняння складових одиниць мови людини та музики

Аналогічно до структурування і дослідження загальних конструкцій мови людини, таких як граматики або синтаксис, було запропоновано схожі моделі і для музики [2, 3, 4]. Такі музичні граматики спираються на низькорівневі музичні поняття, такі як перехід від дисонансу до консонансу, гармонія, напруга та розв’язання, тощо. Подібні синтаксичні та граматичні правила роблять використання природної мови спільним досвідом для всіх її носіїв, тобто одні й ті самі конструкції викликать схожі або ідентичні когнітивні реакції у співрозмовника чи слухача [5]. І мова, і музика притаманні лише людям, вивчаються шляхом спостереження і повторення дій інших людей, є послідовними і протяжними у часі, спочатку з’явилися в усній формі, потім почали використовуватись і в письмовій.

Проте, разом з усіма подібностями існує і певна різниця в представленні мови і музики, насамперед музиці притаманна поліфонія (одночасне і рівноправне звучання декількох голосів), ритм, мультимодальність елементарної одиниці – ноти, яка характеризується одночасно кількома музичними особливостями (висотою, динамікою та довжиною) [6].

Окрім виявлених схожостей мови людини та музики, сфери NLP та MIR так само пов'язані між собою за природою задач які вони досліджують та розв'язують (рис. 1.3).

Одними з найбільш релевантних є задачі з використанням розмічених даних для класифікації різноманітних текстових документів або музичних творів за їхніми характеристиками [7, 8], основними з них, відповідно для MIR та NLP, є:

- класифікація музичних композиторів та визначення авторства тексту;
- класифікація походження народних пісень та виявлення мови;
- класифікація музичних жанрів та стилів тексту;
- класифікація емоцій у музиці та аналіз настрою тексту (sentiment analysis).

Аналогічно і для задач сегментації:

- відновлення музичних фраз та дискурсивний парсинг;
- сегментація музичних форм та сегментація фраз.

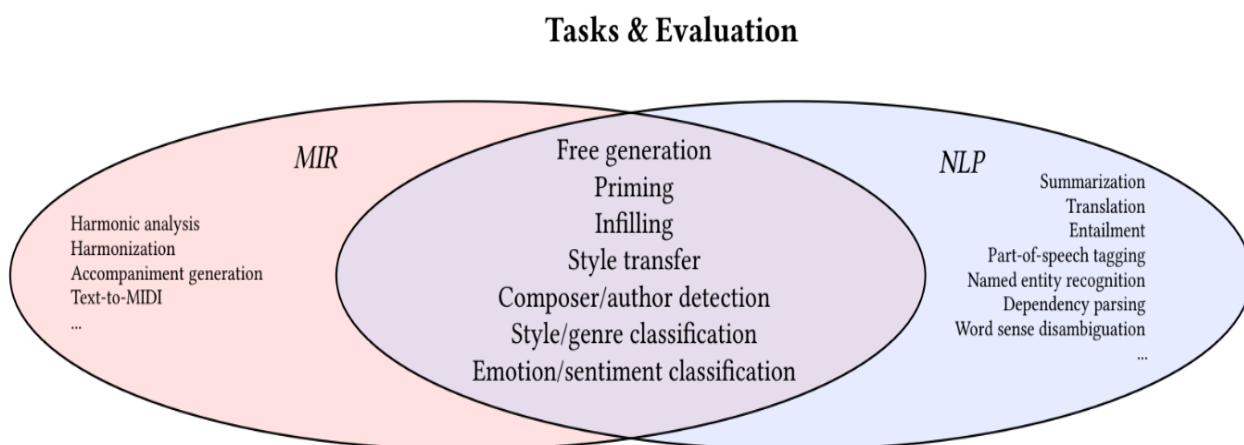


Рисунок 1.3 – Порівняння завдань MIR та NLP [3]

Також є і задачі які використовують датасети з нерозмічених даних, окрім задач кластеризації вони зазвичай використовуються для тренування генеративних моделей за методом навчання без вчителя. Ці моделі можуть бути натреновані для виконання завдань генерації відповідно музики або тексту.

Іншою групою задач є задачі перенесення стилю. Вони актуальні як в галузі MIR для зміни музичного стилю готової композиції, так і для галузі NLP для формалізації тексту [8, 10].

Ці дві сфери також містять і задачі, що відносяться тільки до однієї зі сфер. Такі задачі підкреслюють різницю між зазначеними галузями і відповідними типами комунікацій загалом. Зокрема, семантика мови є ключовою для задачі розпізнавання семантичного слідування (textual entailment); або поліфонія в музиці, що лежить в основі задачі гармонізації та генерації акомпанементу.

1.3 Аналіз існуючих підходів

Подібні роботи проводилися багатьма авторами. Деякі з досліджень так чи інакше торкалися теми класифікації композиторів [3, 4, 9], проте ніякі з них не досліджували вплив і не використовували саме низькорівневі характеристики музичної ноти для класифікації.

Насамперед робота по сентимент-аналізу музичних творів за їхніми метаданими була попередньо проведена. В її рамках було реалізовано систему рекомендації музики, що використовує аналіз настрою тексту, який є попередньо складеним з певних метатегів, для покращення точності рекомендацій на основі вподобань користувача. Система використовує Word2Vec для векторизації слів та багатошаровий перцептрон (MLP) для класифікації тексту за настроєм [9].

По класифікації музичних композиторів та жанрів також була опублікована робота, в рамках якої було використано ансамблевий мета-алгоритму машинного навчання Adaptive Boosting (AdaBoost). Метод запропонований авторами включає трьохетапний процес: вилучення акустичних характеристик з коротких фрагментів аудіо (FFTCs, MFCC, RCEPS, ZCR, тощо), агрегування характеристик у сегментні ознаки та подальшу класифікацію жанру або виконавця на основі отриманих даних [2].

Глибоке навчання також широко застосовується в дослідженнях для вирішення широкого спектру задач, в тому числі і для класифікації композиторів та музичних стилів. В одній з таких робіт було запропоновано фреймворк для класифікації композиторів за сегментами їх музичних творів використовуючи характеристики, пов'язані з нотою, а саме висоту та тривалість ноти. Дослідники використали нейронну мережу ResNet для класифікації композиторів на основі сегментів MIDI-записів з бази даних MAESTRO, фінальне рішення щодо класифікації приймається більшістю голосів. Використаний формат символного кодування дозволяє моделі зосереджуватись на особливостях нот, незалежно від тембру та акустичного середовища запису. Запропонований фреймворк дозволив ResNet вивчити специфічні закономірності, що можуть вказувати на музичні стилі композиторів [3].

1.4 Постановка задачі

Основна мета цієї роботи полягає в адаптації класичних методів обробки природної мови для їхнього подальшого застосування для вирішення вищеописаних задач пов'язаних з аналізом музики. Для цього було прийнято рішення застосовувати символне представлення елементарних одиниць музики – нот. Використання вже відомих технік обробки природної мови в

контексті опрацювання символічних музичних представлень пропонує оригінальний та перспективний погляд як на аналіз і обробку музикального контенту, так і на питання про те, наскільки аналіз музики може бути схожим на аналіз звичайної мови людини.

На відміну від текстових статей, визначення фактичного ‘сенсу’ в контексті інструментальної музики є складним і незрозумілим завданням, оскільки музика не має чіткої інтерпретованої структури. Різні види музичної граматики розроблялися на основі теорій, пов'язаних з певними музичними епохами чи жанрами. Також сама музика є суб'єктивною для різних людей в плані її форми і звучання. Тим не менш, і мови, і музичні твори є продуктами людської креативності, вони мають на меті передати знання і бути зрозумілим іншим людям. З одного боку у випадку мови, знання можуть передаватися від людини до людини напрямку, через взаємне розуміння визначень слів та структури речень. З іншого боку, не існує узгодженого стандарту для визначення ‘сенсу’ музики. Однак вона може бути передана від композитора до слухача через процес музичного виступу, під час якого відбувається передача інформації, яку можна сприймати фізично через вуха та інтерпретувати емоційно через мелодії або послідовності нот. Для передачі цієї інформації комп'ютеру можна використовувати символічне представлення.

Створення символічного музичного представлення передбачає використання спеціального алгоритму для кодування контенту (саме музичного твору) у формат, придатний для обчислювальної обробки. Адаптація моделей NLP до символічного MIR включає переважно послідовні представлення. Варто зазначити, що успіх класифікації композиторів і музичних творів напрямку залежить від ефективності запропонованого методу кодування музичних даних. Таким чином, було прийнято рішення дослідити вплив на класифікацію не тільки високорівневих конструктів музики – інтервалів, мелодій і акордів, а й основних низькорівневих параметрів нот (висоти, динаміки та довжини).

З вищесказаного можна чітко сформулювати задачу цього дослідження: перевірити правильність припущення про вплив низькорівневих характеристик та високорівневих патернів на вихідний музичний твір із застосуванням технологій обробки природної мови, і класифікувати твори композиторів у визначеній символній області. Для цього планується застосування наступних технік NLP:

- створення словника для розумного кодування та декодування одиничних музичних нот;
- окремий метод для представлення акордів, які складаються з окремих нот;
- використання токенизації;
- використання векторизації для знаходження зв'язків між музичними творами.

Для реалізації цього методу буде використано датасет MAESTRO [21], що містить понад 200 годин виконань класичних композицій. Ці виконання записані у форматі MIDI. В самому датасеті, окрім файлів з музикою, містяться і метадані, такі як ім'я композитора, назва твору та його тривалість.

Висновки до розділу 1

У цьому розділі було розглянуто актуальність і предметну область теми дослідження. Було визначено, що тема аналізу музики з використанням технологій обробки природної мови є не дуже вивченим, проте дуже перспективним напрямком досліджень, що знаходиться на межі кількох дисциплін. Такий аналіз дозволить проводити розумний аналіз величезної кількості музики, що зараз доступна кожному з нас.

Такий підхід може стати основою для, наприклад, рекомендаційних систем. Це допоможе слухачам краще розуміти закономірності в своїх

вподобаннях, знаходити нові і неочікувані музичні відкриття. Композиторам, в свою чергу, це дозволить краще розуміти взаємозв'язки між жанрами та іншими авторами, повністю розкриє їхній творчий потенціал.

Після аналізу існуючих підходів було прийнято рішення використовувати символічне представлення музики для подальшого застосування на ній класичних методів обробки природної мови. Символьне представлення допоможе зафіксувати не тільки високорівневі музичні елементи, такі як мелодії, інтервали та акорди, але й низькорівневі характеристики нот, такі як довжина і динаміка нот.

Далі було сформульовано основну задачу дослідження: обробити та проаналізувати музичний датасет, перевірити чи є обрана техніка кодування музики ефективною та дослідити вплив низькорівневих характеристик нот на результати високорівневого аналізу.

РОЗДІЛ 2 ОГЛЯД МЕТОДІВ РОБОТИ З МУЗИЧНИМИ ДАНИМИ

2.1 Огляд методів для комп'ютерної обробки музики

Музика є складним для комп'ютерної репрезентації типом даних. З одного боку музика, як вид мистецтва, вирізняється наявністю численних взаємозв'язків, що можна трактувати математично, наприклад ритм і гармонія. З іншого боку існує багато нематематичних аспектів, таких як напруга, послаблення та емоції. Музика зазвичай містить структурні зв'язки між своїми наступними складовими: висота звуку, тривалість, тембр, гармонія, темп, ритм, фразування та артикуляція. Ще однією складністю є перемінність музики, вона розвивається з кожною новою композицією. Не може існувати 'правильного' представлення, так само як не може бути і чіткого визначення музики. Всі ці елементи роблять проблему музичного представлення досить великою і неоднозначною [11].

Комп'ютери дозволяють використовувати чітко визначений формальний підхід для зберігання та обробки цифрової музики. Комп'ютерні програми вимагають, щоб кожна деталь музичного представлення була точно визначена, ця точність дозволяє тестувати та експериментувати з музичними даними. Отримані з цих експериментів знання корисні в дослідженні музики в рамках багатьох галузей, в тому числі і в теорії музики.

Музиканти мають справу з багатьма рівнями абстракції в музиці. Якщо диригент каже: «зіграйте песимістичний ритм з більшою переконаністю», то тут йдеться мова і про музичну структуру (песимічний ритм), і про емоційний зміст одночасно, в одному реченні. Можна розглядати музичні репрезентації різних рівнів, високорівневу та абстрактну, використовуючи друковані ноти, або низькорівневу і конкретну, використовуючи аудіосигнал. Виступ – це середній рівень. Ці рівні, потрібно враховувати, оскільки не існує способу унікально конвертувати музичне представлення між цими рівнями, кожен з них містить принаймні деяку інформацію, недоступну для інших.

Загалом, існує велика цінність в автоматизованих і хоча б часткових перетвореннях між цими рівнями [19].

На перший погляд проста концепція висоти ноти насправді є доволі складною. Це пов'язано з різними трактуваннями висоти, вона є одночасно акустичною властивістю, психологічна складова (сприйнятний тон) і як абстрактна сутність, що описує її відношення до інтервалів та тональностей. Психологи досліджують зв'язок між сприйняттям висоти ноти (або інтервалу) і фундаментальною частотою ноти. Цей зв'язок залежить від багатьох змінних [12]. При використанні символічного представлення висота звуку є скоріше фізичною одиницею. Щоб назвати висоту звуку, потрібно вказати тональність, октаву, та альтерації, наприклад, висота ноти $C\#$ у четвертій октаві на основі $A440$. Тут $A440$ означає конкретне налаштування музичного інструменту, де нота A є відносним орієнтиром і має висоту 440 герц. Вищеописані проблеми моделей музичної репрезентації, які є наслідками питання 'що представляти', а не 'як це представляти' [13]. Ці проблеми згадуються тут, щоб вказати на багатошаровість і невизначеність, наявну при для будь-якому загальному представленню висоти звуку чи ноти.

Взаємодія між реальним (абсолютним) часом, що вимірюється в секундах, та метричним часом, що вимірюється в тактах, часто використовується у представленні музики для того, щоб підкреслити відносність часових рамок [14]. Можна провести наступну аналогію для описання тактів в музиці: існує певна математична функція, яка співставляє такти до часі, та обернена функція, яка співставляє час до тактів. Одним із практичних прикладів такої функції залежності є відмітки часу на множині послідовності окремих тактів. В рамках цього прикладу виконавець 'відбиває' ритм в режимі реального часу, а комп'ютер фіксує час кожного відбивання, така маніпуляція створює кінцеву послідовність часових відміток. Саме таким чином, наприклад, реалізований MIDI годинник, що повертає часові відмітки з частотою 24 удари за чверть ноти. Існує декілька цікавих проблем пов'язаних з числовою точністю, що виникають через часові

обчислення темпу та ритму. Одна з них полягає в тому, що помилки округлення тривалості окремих нот не повинні спричиняти помилок у довжині їх загальної послідовності. Наприклад, припустимо, що система відображає тривалість ноти у мілісекундах. Якщо чверть ноти має тривалість 1000мс, то восьмі тріолі ($1/12$ від загальної довжини ноти) матимуть тривалість (після округлення) 333мс. Послідовність з 30 триплетів, зіграних протягом 10 чвертей, буде зміщена на 10мс.

У багатьох аспектах музики відомо, що потрібно відобразити, і питання полягає в тому, як це відобразити. Це не є релевантним, зокрема, для тембру. У звуку є дві очевидні характеристики, це висота та гучність, все інше зазвичай називають тембром [19]. Отже, тембр за визначенням – це те, що не можна пояснити, це група багатьох взаємозалежних явищ, які визначають звук [15].

Існує багато способів цифрового музичного представлення. Такі представлення обираються відповідно до поглядів продюсера на музичний контент та типу контенту. Термін ‘представлення’ стосується способу організації інформації всередині системи, тобто вигляд внутрішніх структур даних. Найбільш актуальним в рамках цього дослідження є формат MIDI.

MIDI – це протокол, який спочатку був розроблений для можливості зв’язку в реальному часі між синтезаторами. Основна ідея протоколу полягає в наступному: потрібно надсилати повідомлення щоразу, коли нота натискається (нота увімкнена) на MIDI-клавіатурі, та інше повідомлення, коли нота відпускається (нота вимкнена). Після отримання повідомлення, синтезатор-приймач знатиме, який звук потрібно зіграти [16]. Якщо зібрати всі ці повідомлення з їхніми часовими відмітками, то буде отримано MIDI файл, що містить певну послідовність нот і відтворює відповідний музичний твір (рис. 2.1).

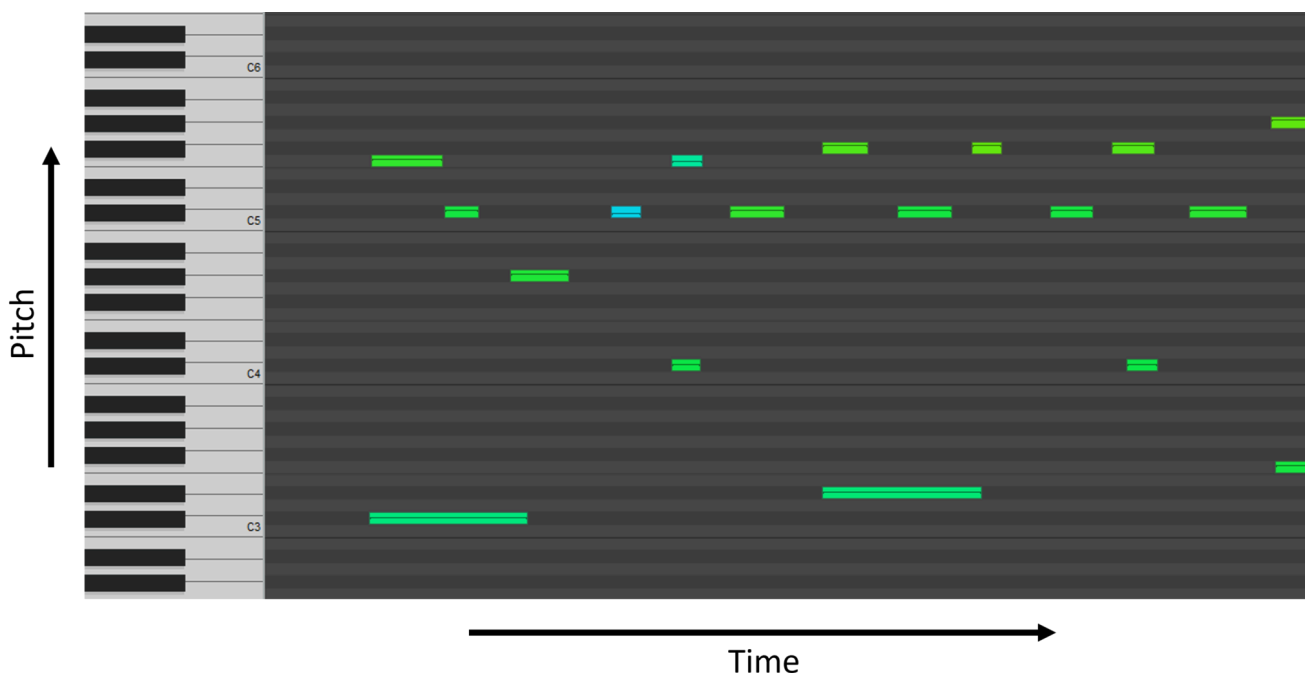


Рисунок 2.1 – Приклад послідовності MIDI нот в програмі Reaper [17, 20]

Окрім двох типів основних повідомлень існує і багато інших, в контексті піаніно це насамперед повідомлення про натискання педалі [17].

MIDI файл може бути створений декількома способами. З кожним з цих способів створюються різні типи файлів:

- граючи на MIDI-інструменті – MIDI-виступ;
- шляхом ручного редагуванні нот в спеціальній програмі – MIDI-партитура.

MIDI-виступ може надати таку інформацію про зіграні ноти:

- яка нота була зіграна (ідентифікатор ноти, або pitch);
- наскільки ‘сильно’ була зіграна нота (динаміка ноти, або velocity);
- часові відмітки початку та кінця ноти.

Висота ноти закодована цілим числом від 0 (найнижча нота) до 127 (найвища нота). Діапазон фортепіано знаходиться в інтервалі від 21 до 108, де кожне число відповідає одній з 88 клавіш. Динаміка ноти також закодована цілим числом від 0 (тиша) до 127 (найбільша гучність). Початок та зміщення ноти (і тривалість) представлені в секундах, що відповідає секундам,

протягом яких ноти були натиснуті та відпущені особою, яка грає на MIDI-інструменті. Початок та кінець ноти відображені в секундах і відображають час протягом якого нота була відповідно натиснута та відпущена особою, яка грає на MIDI-інструменті [16]. Далі буде розглянуто MIDI-виступи.

Переважає більшість MIDI-виступів виконуються на піаніно, оскільки більшість MIDI-інструментів – це MIDI-клавіатури. Інші MIDI-інструменти (наприклад, MIDI-саксофони та MIDI-гітари) існують, проте вони не є поширеними. Найбільшим датасетом MIDI-виступів є Maestro від Google Magenta. Цей датасет містить понад 200 годин аудіо та MIDI виступів за 10 років міжнародного конкурсу електронних піаніно у Міннесоті [22]. MIDI-файли містять динаміки нот та позиції педалей. Аудіо та MIDI файли синхронізовані з точністю 3 мілісекунди і поділені на окремі музичні твори, які анотовані даними про композитора, назвою, роком виконання та довжиною. Репертуар містить переважно класичні композиції XVII – початку XX століття [21].

Головною особливістю MIDI-виступів є те, що в рамках одного виступу майже ніколи не зустрічається пара нот з абсолютно однаковою часовою відміткою початку або кінця. Теоретично це можливо, але на практиці малоймовірно.

Як загально відомо, виконавець не може натиснути дві, або більше, ноти одночасно, оскільки існує певне обмеження людської точності, це саме стосується і одночасного відпускання клавіш (рис. 2.2). Більш того, зазвичай точність навіть не є пріоритетом для музиканта, оскільки невелики часові відхилення і неточності допомагають досягти ефекту більшої виразності або задорності [17]. Через це MIDI-виступи називають неквантованим MIDI (квантування це дія, перетворення якоїсь величини з неперервною шкалою значень на величину з дискретною шкалою значень [18]).

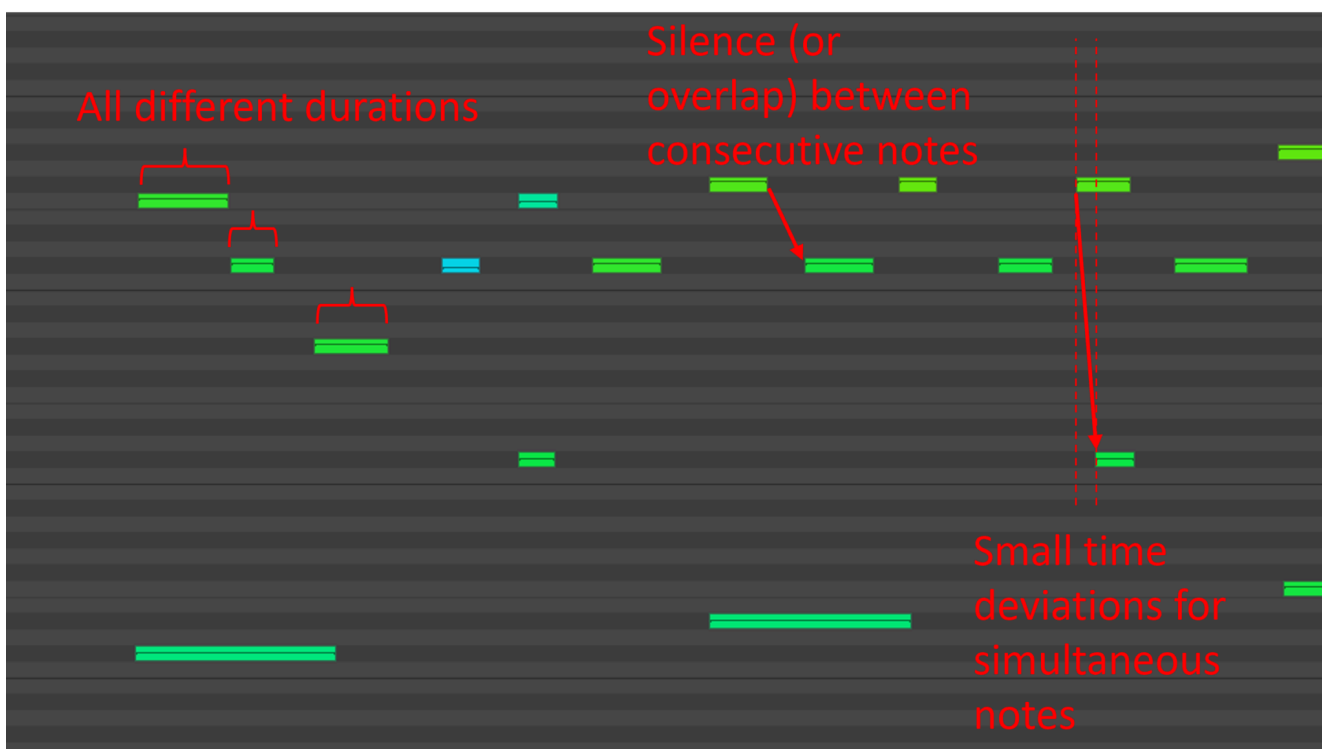


Рисунок 2.2 – Візуалізація положення нот у MIDI-виступі [17]

MIDI-партитури, в свою чергу, містять значно більше можливостей для кодування музичної інформації, зокрема, розмір такту, тональність, ключ, такт та позиції долі. MIDI-партитури є аналогами класичних музичних партитур. Інформація про довжини нот закодована без використання часових одиниць, натомість з використанням абстрактної величини – долей такту, наприклад, $\frac{1}{2}$ довжини такту, $\frac{2}{3}$ довжини такту, тощо. Головною особливістю є те, що всі початки нот вирівняні по квантованій сітці, що визначається позиціями тактів, а також її цілочисельними діленнями, як було описано раніше.

Вищеописані особливості є основними відмінностями між MIDI-виступом та MIDI-партитурою, однак існує ще кілька пов'язаних з ними нюансів. Наприклад, в деяких випадках автори MIDI-партитур вводять штучні часові неточності для кожної ноти, щоб імітувати гру реальних виконавців. Такий процес є зазвичай автоматизованим і може виконуватись

певними програмами, наприклад MuseScore [23]. В таких програмах є окрема функція експорту для MIDI-партитур, яка дозволяє імітувати реальний виступ знову ж таки, за рахунок додавання штучних затримок і дуже коротких пауз між послідовними нотами. Проте, в деяких випадках такий підхід є недосконалим, наприклад при використанні форшлагів (коротких нот вільної довжини) або трелей (швидке багаторазове чергування двох суміжних нот). У випадку форшлагів, їхня тривалість є невизначеною в музичних термінах, існує тільки загальна концепція, що вони повинні бути швидкі. Кожен виконавець буде грати їх трохи по різному, в залежності від музичної композиції та особистих переконань. Аналогічна проблема наявна і з трелями: в музичній нотації не існує способу чітко обозначити її довжину та кількість зіграних в ній нот.

MIDI файли є дуже потужний інструментом, вони надають чітку інформацію про висоту, динаміку та тривалість кожної з нот. Це означає, наприклад, що порівняно з аудіофайлами, моделі, орієнтовані на MIDI-дані потребуватимуть менших наборів даних для тренування. MIDI файли також дозволяють ігнорувати акустичні особливості приміщення, де було проведено запис. Проте MIDI-файли, все ж таки є складними форматами для використання, оскільки вони кодують багато видів інформації великою кількістю різних способів. Щоб правильно використовувати MIDI-файли для машинного навчання, потрібно як врахувати всі релевантні для конкретного завдання дані файлу, так і відкинути всю зайву інформацію.

2.2 Використання технік NLP для роботи з музикою

У цьому підрозділі окреслено схему для кодування музичних нот, а також методи обробки природної мови (NLP), які можуть бути застосовані для аналізу музичних даних.

Процес токенизації стосується процесу перетворення складного контенту в послідовність придатних для обчислювальної обробки елементів. Токенізація – це сегментація послідовності атомарних елементів (символів) шляхом їх групування в інформативні токени, такі як підслова, слова або багатослівні вирази [8]. Час відіграє особливу роль у музиці, оскільки часове відрізки початку та кінця нот суттєво впливають на структуру і звучання музичної композиції. Різні підходи до відображення часу розділяють способи токенизації музичних даних на два основні види [3]: на основі часових інтервалів та на основі подій (рис. 2.3).

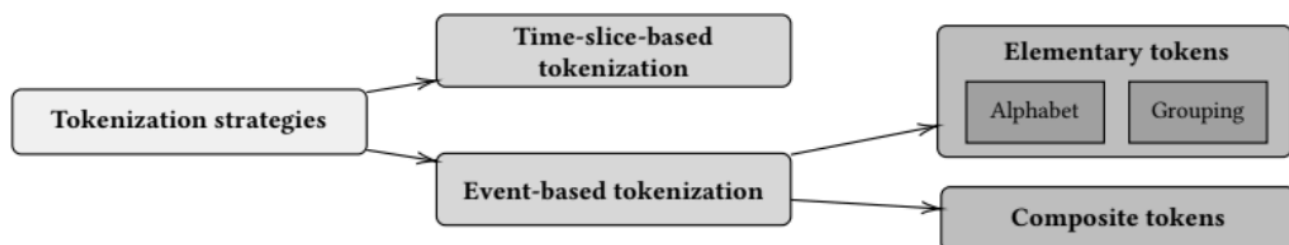


Рисунок 2.3 – Основні стратегії токенизації [3]

Спочатку розглянемо токенизацію на основі часових інтервалів. Використання поділу часу на рівні інтервали є природним підходом. Саме так композитори і почали записувати музику в партитурах, вписуючи ноти у ‘часову сітку’ відповідно до ритму, що використовується. Часовий крок можна представити як вектор, що містить певну кількість нот, наприклад, акорд буде містити як мінімум три одночасно зіграні ноти, а мелодія може містити і по одній зіграній за часовий інтервал ноті [24]. Це представлення (рис. 2.2) використовує матриці, в яких горизонтальна вісь – це час, а вертикальна вісь – висота ноти, додаткові характеристики також можуть бути додані як нові виміри.

На відміну від токенизації на основі часових інтервалів, в якій нові токени додаються кожен часовий крок, токенизація на основі подій додає токени тоді, коли стається певна нова подія, наприклад, коли була зіграна нота. Більшість стратегій токенизації використовують саме цей підхід через наявність великої кількості доступних MIDI-даних. Структура MIDI-повідомлень забезпечує природний спосіб кодування музики як послідовності подій. Однак, в цьому також є декілька нюансів:

- MIDI-повідомлення бувають різних типів;
- на відміну від символів у тексті, музичні ноти містять кілька характеристик, наприклад як висота, тривалість, динаміка, тощо.

Оскільки ці ознаки характеризують повідомлення як окрему і унікальну часову подію, то послідовне представлення MIDI-повідомлень вимагатиме введення штучних символів поверх MIDI-повідомлень (рис. 2.4). Це саме стосується і одночасного представлення декількох нот. В галузі MIR, токенизацію на основі подій зазвичай поділяють на два класи:

- токенизацію на основі подій з використанням елементарних токенів;
- токенизацію на основі подій з використанням складених токенів.

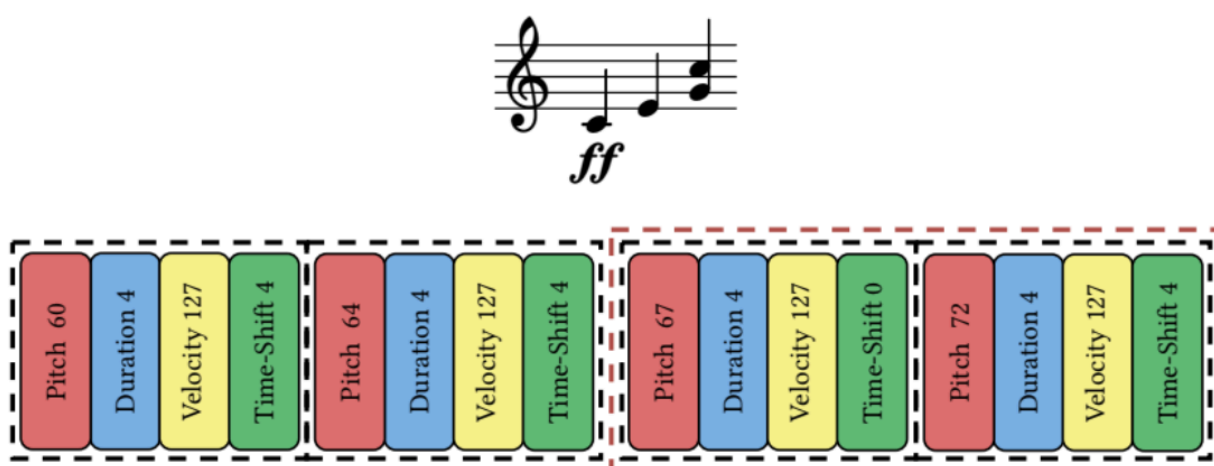


Рисунок 2.4 – Приклад використання даних MIDI-файлу [3]

Послідовності елементарних токенів вносять додаткову штучність, розділяючи ноту на окремі ознаки і кодуючи їх окремо, тобто кожен токен є окремою музичною ознакою. Наслідком цього є те, що одна й та сама подія описується декількома сусідніми токенами. Наприклад, нота описуватиметься, як мінімум двома токенами, що характеризуватимуть відповідно висоту ноти та її тривалість. З іншого боку, складені токени повністю кодують унікальну подію з усіма її характеристиками, самостійно описуючи складну подію [3, 25].

SentencePiece є одним із сучасних підходів до токенізації, який ґрунтується на техніці токенізації підслів і не вимагає попередньої обробки тексту. Цей алгоритм спочатку створює словниковий запас найчастіших підслів із вхідних даних, а потім використовує цей словник для токенізації нових вхідних даних. SentencePiece підтримує дві різні моделі сегментації: уніграмну мовну модель та кодування байтових пар (Byte-Pair Encoding, BPE) [26].

У контексті музики, SentencePiece можна адаптувати для створення ефективного символного представлення нотних послідовностей. Оскільки музичні твори часто містять повторювані патерни (акорди, мотиви, фрази, арпеджіо тощо), токенізація на рівні підслів є особливо ефективною. Замість токенізації окремих нот, SentencePiece дозволяє автоматично виявляти та токенізувати цілісні музичні фрагменти, що часто зустрічаються разом. Таким чином можна фіксувати високорівневі музичні структури.

Окрему увагу варто приділити ймовірнісній природі алгоритму SentencePiece, зокрема, уніграмній моделі. Вона ґрунтується на принципах баєсівської оптимізації та максимізації правдоподібності і спирається на статистичні ймовірності для формування оптимального словника токенів [6]. Кожен можливий токен у словнику має власну ймовірність. Ці ймовірності використовуються для визначення найбільш правдоподібної сегментації заданої послідовності. Математично, для послідовності символів X , алгоритм

шукає найбільш імовірний розподіл токенів і максимізує відповідну функцію, наведену у формулах (2.1) та (2.2).

$$x = (x_1, \dots, x_m), \quad (2.1)$$

$$P(x) = \prod_i P(x_i), \quad (2.2)$$

де $P(x_i)$ – ймовірність появи окремого токена x_i .

Під час тренування моделі ймовірності токенів оптимізуються за допомогою ЕМ-алгоритму (Expectation-Maximization), що ітеративно коригує ці значення для досягнення максимальної правдоподібності на тренувальному наборі даних. Цей процес забезпечує адаптивність моделі до специфічних особливостей музичного корпусу, що аналізується.

Далі, закодовані токени (символи) потрібно перетворити на вектори, щоб модель могла їх коректно обробити. Слова, речення та тексти (або ноти, акорди та пісні в контексті музики) потрібно проектувати в певний чисельний простір (рис. 2.5) для подальшої обробки.

Перші методи векторизації спиралися на базові вектор-стовпці, кожен з яких мав довжину, еквівалентну розміру словника. Документ в цьому випадку є вектором-сумою всіх цих векторів, також відомого як Bag-of-Words (BoW) [27]. Цей підхід, однак, не фіксує жодної послідовної складової наданих даних, в результаті чого, отриманий векторний простір часто є розрідженим, це, в свою чергу, не дозволяє побачити зв'язки між музичними елементами. Таким чином, для вирішення цієї проблеми у галузі NLP було розроблено чисельні методи для векторизації слів у щільному просторі.

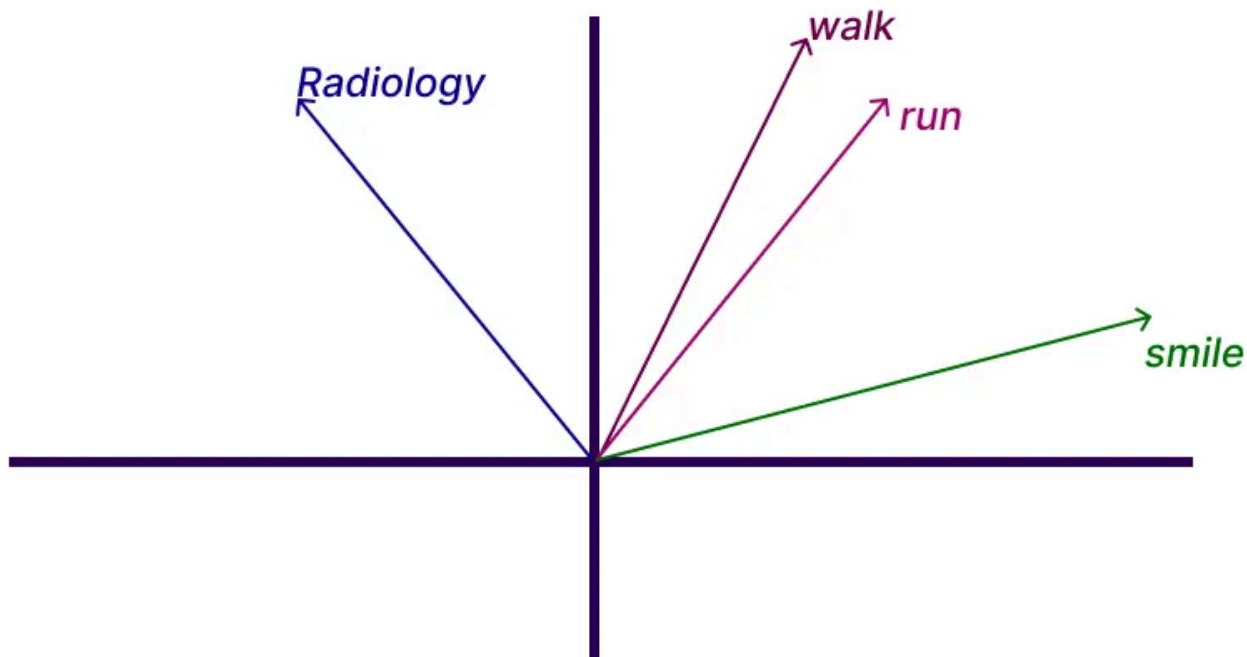


Рисунок 2.5 – Візуалізація векторизованих слів [29]

Одним з цих методів є Word2Vec. Цей метод використовує статичні вкладки, це означає, що кожне слово може бути закодовано використовуючи один і той самий вектор незалежно від контексту його використання. Word2Vec базується на нейронній мережі, яка і будує такі статичні вкладки [28]. Цей метод вже був адаптований для музики [4, 24], деякі з авторів застосовували його для неявних інтерпретацій більш складних структур, що складаються з нот, наприклад, акордів чи музичних фраз.

Word2Vec є одним із найвідоміших алгоритмів для векторизації слів. Цей метод використовує неглибокі нейронні мережі для створення векторних представлень слів таким чином, щоб слова, що з'являються в подібних контекстах, знаходилися близько одне до одного у багатовимірному просторі. Це дозволяє алгоритму фіксувати семантичні та синтаксичні зв'язки між словами [6].

Word2Vec реалізований у двох основних архітектурах: Continuous Bag of Words (CBOW) та Skip-gram. CBOW прогнозує цільове слово на основі

контексту навколишніх слів. Skip-gram, в свою чергу, працює в зворотному напрямку: прогнозує контекст на основі цільового слова. Обидві архітектури використовують метод стохастичного градієнтного спуску для оптимізації параметрів моделі [6].

При адаптації Word2Vec для аналізу музичних даних, основні елементи музичного твору (ноти, акорди, мелодії) розглядаються як "слова", а їх послідовності - як "речення". Це дозволяє приховані зв'язки між музичними елементами. Крім того, Word2Vec дозволяє виявляти більш складні музичні концепції, такі як модуляції, секвенції та інші композиційні техніки. Це особливо важливо для розуміння структурних особливостей музичних творів різних епох та стилів.

Висновки до розділу 2

У цьому розділі було розглянуто теоретичні основи обробки цифрової музики з використанням традиційних методів обробки природної мови. Було визначено, що музика, як об'єкт комп'ютерної репрезентації, є складною системою, що має багаторівневу структуру та характеризується численними взаємозв'язками між своїми елементами.

Було детально проаналізовано MIDI-формат як ефективний спосіб символного представлення музики, що дозволяє працювати з такими характеристиками нот як висота, тривалість та динаміка. Розглянуто відмінності між MIDI-виступами та MIDI-партитурами, які мають різний підхід до представлення часової складової музики.

Дослідження методів обробки природної мови для аналізу музичних даних показало, що існують різні підходи до токенизації музики: на основі часових інтервалів та на основі подій. Кожен з цих підходів має свої переваги залежно від поставленої задачі. Крім того, було розглянуто методи

векторизації музичного контенту, зокрема Word2Vec, що дозволяють створювати векторні представлення музичних елементів для подальшого аналізу.

Розглянуті теоретичні основи формують потужну методологічну базу для розробки практичних алгоритмів класифікації музичних творів та композиторів з використанням як високорівневих музичних патернів, так і низькорівневих характеристик нот.

РОЗДІЛ 3 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Опис і передобробка датасету

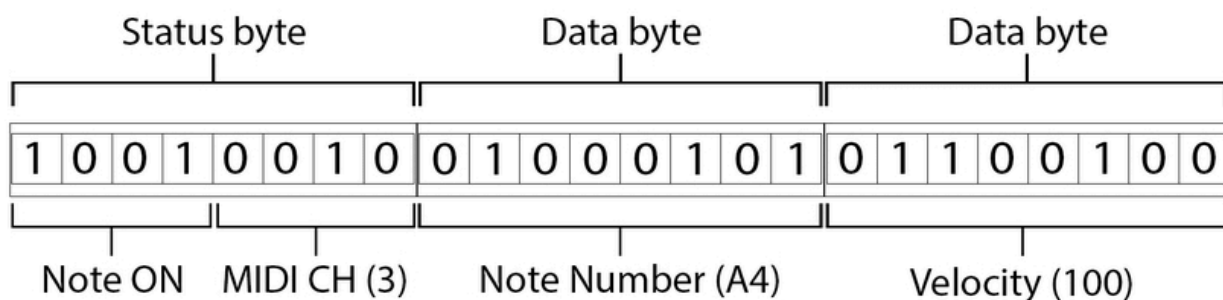
Для проведення цього дослідження було обрано датасет MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) версії 3.0.0 від Google [21]. Під час кожного з етапів конкурсу віртуозні піаністи виконують різноманітні музичні твори на акустичних роялях концертної якості Yamaha. Ці роялі також містять інтегровану високоточну систему захоплення та відтворення MIDI. Записані MIDI-дані мають високу точність, це дозволяє проводити етап прослуховування і оцінювання конкурсу дистанційно [22]. MIDI-файли містять таку інформацію про виступи: висоту нот, довжина кожної з зіграних нот, динаміки натискання на клавіші, натискання та відпускання педалей. Разом з цим датасет містить наступні метадані про виступи: назва музичного твору, повне ім'я композитора, рік виступу, загальна довжина виступу.

Було проведено попередній аналіз датасету. Датасет загалом містить 1276 записаних виступів загальною довжиною 198.7 годин. Серед авторів зіграних творів є 60 унікальних композиторів. В датасеті також наявні дуплікати деяких композицій, без їх урахування в датасеті залишається 807 унікальних композицій.

3.2 Кодування музичних даних

В цьому дослідженні використовується формат MIDI-файлів, він вже є різновидом символьного музичного представлення. Для роботи з ними було використано бібліотеку `pretty_midi`. Ця бібліотека дозволяє отримати потрібні дані нот з MIDI-файлів у вигляді структур мови python (рис. 3.1).

MIDI-message



MIDI-message is uniquely encoded into a vector

(pitch, velocity, duration)
(69, 4, 1.00)

Vector is mapped to a single symbol

怨

Рисунок 3.1 – Перетворення даних з MIDI-файлу у зручний формат

Цей процес починається з отримання фундаментальних характеристик ноти, а саме висоти, динаміки та тривалості кожної з нот музичного твору. Результатом цієї первинної обробки MIDI-файлу є масив одновимірних векторів, по одному на кожну ноту за формулою (3.1). Слід зазначити, що для зменшення кількості унікальних значень кожна зі змінних додатково оброблюється за формулами (3.2), (3.3) та (3.4):

- висота ноти лишається як є – ціле число від 21 до 108;

- динаміка нормалізується у діапазоні від 0 до 4, замість інтервалу 0-127;
- тривалість ноти округлюється з використанням спеціальної формули.

Ці перетворення застосовуються для зменшення кількості унікальних векторів – зменшення алфавіту. Надмірна кількість символів у словнику може перешкоджати здатності моделі розпізнавати патерни і залежності.

$$note = (pitch, velocity, dur), \quad (3.1)$$

$$pitch = \min(108, \max(21, pitch_{IN})), \quad (3.2)$$

$$velocity = \text{round} \left(\frac{5}{127} velocity_{IN} \right), \quad (3.3)$$

$$dur = \begin{cases} 0 & \text{if } dur_{IN} < 0 \\ \text{round} \left(\frac{20 \cdot dur_{IN}}{20} \right) & \text{if } 0 < dur_{IN} < 0.5 \\ \text{round} \left(\frac{10 \cdot dur_{IN}}{10} \right) & \text{if } 0.5 \leq dur_{IN} < 2 \\ \text{round}(dur_{IN}) & \text{if } 2 \leq dur_{IN} < 15 \\ 15 & \text{if } dur_{IN} \geq 15 \end{cases} \quad (3.4)$$

В датасеті також наявна поліфонічна музика. Це такий вид музичного виконання, коли декілька нот граються одночасно. В рамках цього дослідження було прийнято рішення групувати такі ноти в акорди. Для того щоб зрозуміти, чи потрапляє нота в акорд було додано спеціальний часовий таймаут: якщо нота була зіграна до кінця цього таймаута, то вона вважатиметься частиною акорда. Акордом в цьому випадку називатиметься будь-яка група з двох та більше нот. Для того щоб, знову ж таки, зменшити розмір словнику, ноти всередині акорда сортуються за висотою та отримують однакові значення для тривалості та динаміки (рис. 3.2). Цей підхід також дозволить забезпечити консистентність отриманих даних.

撒 抡 柳
 (78,5,4.00) (86,5,4.00) (90,5,4.00)

Notes of a chord are sorted by pitch, even if played with a small delay

怨 迺 彡 崕
 (50,4,1.00) (57,4,1.00) (62,4,1.00) (66,4,1.00)

A song is being encoded as an array of symbols: 撒抡柳怨迺彡崕

Рисунок 3.2 – Приклад кодування фрагменту музичної композиції

Після отримання всіх унікальних векторів, вони кодуються довільними символами Юнікоду, для наглядності було обрано символи китайського алфавіту (рис. 3.2). Результатом такого кодування є словник унікальних одновимірних векторів (нот), де кожному вектору відповідає певний символ.

Далі ці символи проходять обробку алгоритмом SentencePiece щоб згрупувати послідовності символів, що часто зустрічаються, у слова або підслова (рис. 3.3).

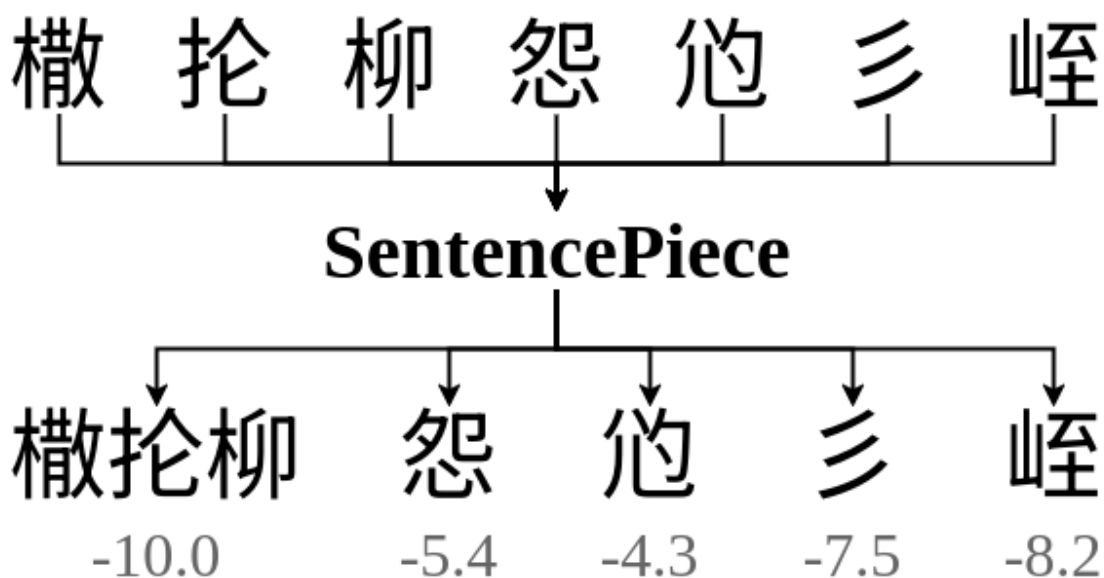


Рисунок 3.3 – Візуалізація роботи алгоритму SentencePiece

У контексті музики, це, відповідно, одиничні ноти та акорди, що граються виконавцями найчастіше.

3.3 Векторизація музичних даних

Для перетворення нот та акордів, отриманих з алгоритму SentencePiece, у вектори (рис. 3.4) було використано алгоритм Word2Vec [6].

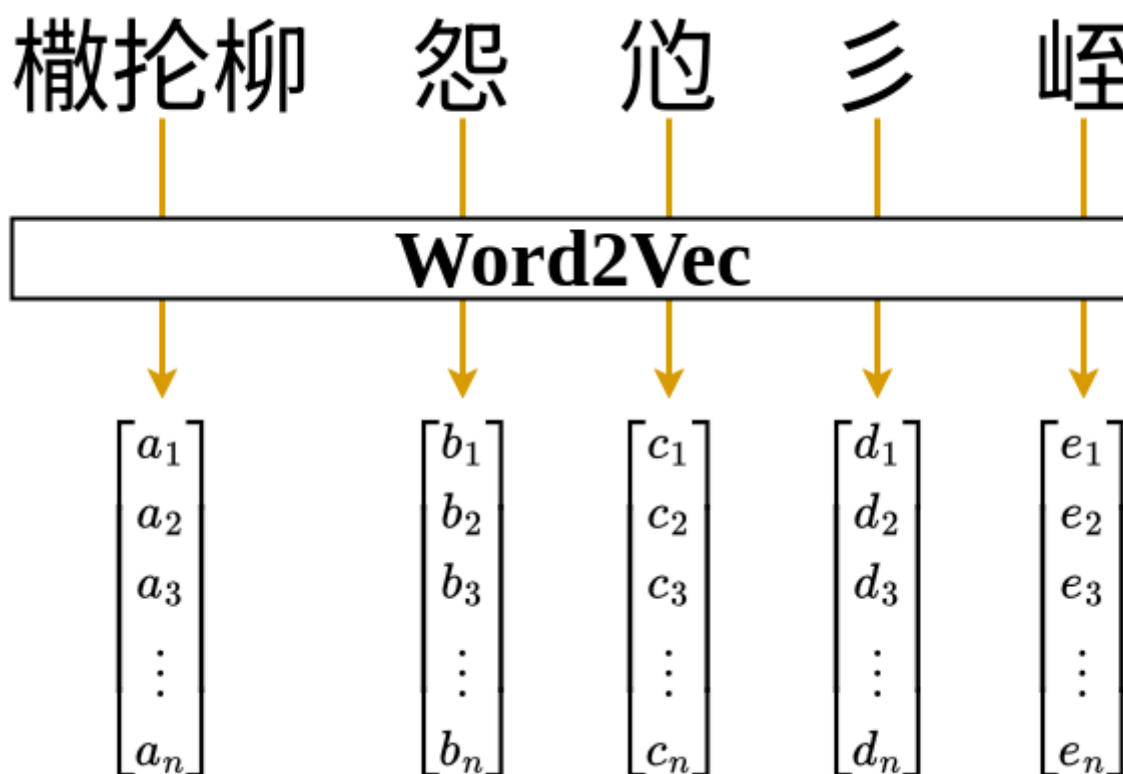


Рисунок 3.4 – Принцип роботи алгоритму Word2Vec

Головна мета векторизації полягає у створенні багатовимірних числових представлень музичних композицій, які відображають семантичні взаємозв'язки між нотами, акордами, тощо. Це дозволяє, насамперед, застосувати алгоритми машинного навчання до музичних даних.

У цьому дослідженні було реалізовано п'ять різних методів векторизації музичних послідовностей:

- векторизація методом усереднення (avg);
- векторизація методом стандартного відхилення (std);
- методом усереднення і стандартного відхилення (avg_std);
- векторизація з використанням TF-IDF (tfidf);
- векторизація за методом максимального об'єднання (max_pool).

Суть цих методів полягає у перетворення векторів окремих елементів композиції у єдиний вектор композиції (рис. 3.5). Всі з наведених методів відрізняються по характеру своєї роботи.

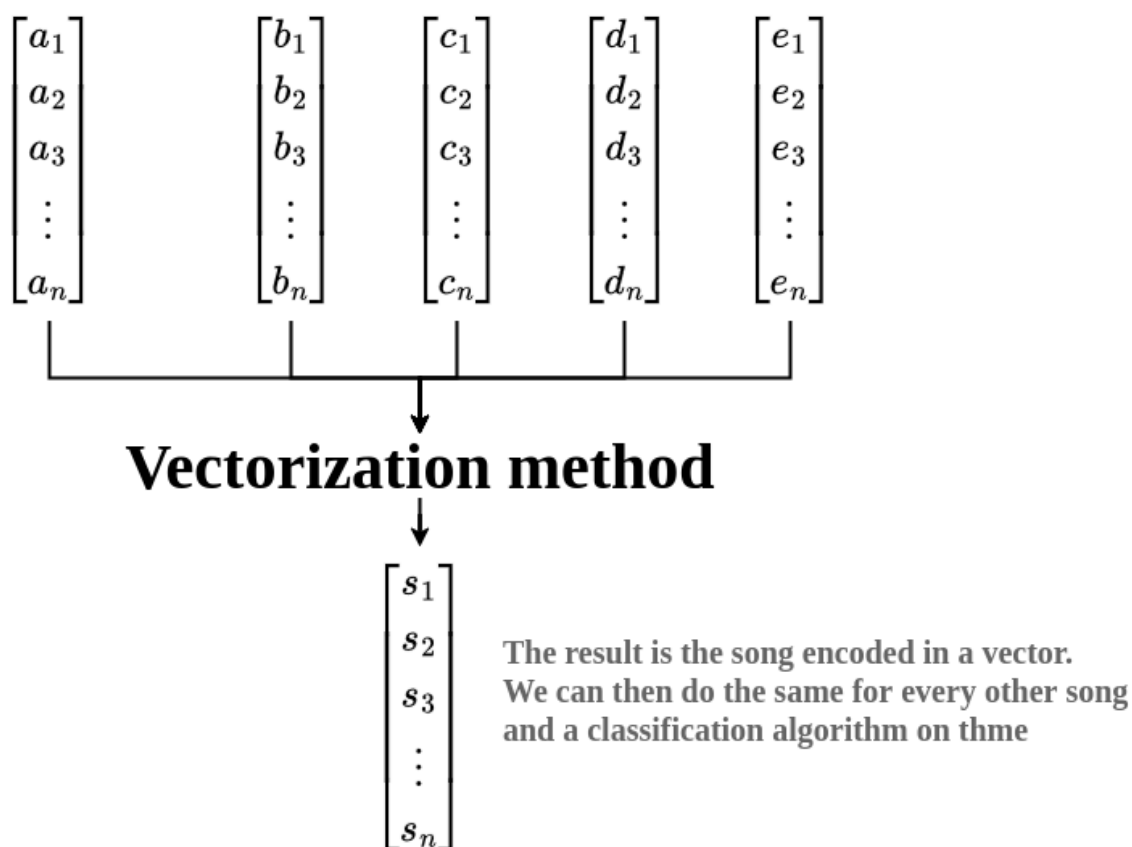


Рисунок 3.5 – Візуалізація векторизації

Метод усереднення рахує середнє значення всіх частин пісні (саме їх вектори, отримані з Word2Vec). Він зберігає основний зміст музичного твору, швидкий в обчисленні, добре передає настрій і стиль пісні.

Метод стандартного відхилення обчислює стандартне відхилення векторів усіх нот у творі. Він набагато краще фіксує мінливість і тривалість музичного твору. Це дозволяє відобразити динамічні зміни в самому творі, наприклад, зміни тональності та гармонічну складність твору.

Комбінований метод усереднення та стандартного відхилення поєднує обидва попередні підходи. Він обчислює середнє значення та стандартне відхилення нотних векторів, а потім об'єднує їх в один вектор. Це дозволяє зберегти як загальний стиль фрагмента, так і його внутрішню варіативність.

Метод з використанням TF-IDF застосовує статистичну міру TF-IDF [30] до музичних даних. Обчислюється зважене середнє векторів для всіх нот, де вагами є значення TF-IDF. Це надає більшу важливість нотам, які є характерними для конкретного твору і менш поширеними в загальному корпусі.

Метод максимального об'єднання (max pooling) створює для кожної пісні вектор, що містить максимальне значення по кожній розмірності серед усіх векторів для нот цієї композиції [31]. Цей підхід визначає найбільш виражені складові музичного твору, такі як характерні мелодії та акорди.

3.4 Аналіз отриманих результатів

Для аналізу результатів на отриманих в попередньому пункті векторах було проведено класифікацію. Для задачі класифікації було обрано 5 композиторів з найбільшою кількістю творів, зокрема, Фредерік Шопен, Франц Шуберт, Людвіг ван Бетховен, Франц Ліст та Йоганн Себастьян Бах.

Було проведено класифікацію для всіх методів векторизації. Для класифікації було використано наступні алгоритми:

- Random Forest;
- SVN;
- KNN;
- Logistic Regression.

Було обчислено результати класифікації для всіх можливих пар векторизація-класифікація (табл. 3.1).

Таблиця 3.1 – Стислі результати класифікації

Classifier	AVG	STD	AVG_STD	TFIDF	MAX_POOL
Random Forest	0.82	0.85	0.85	0.84	0.75
SVM	0.88	0.89	0.90	0.88	0.79
KNN	0.81	0.79	0.81	0.80	0.73
Logistic Regression	0.82	0.83	0.88	0.79	0.68

Найбільш якісною виявилась векторизація комбінованим методом усереднення та стандартного відхилення (avg_std), а найбільш ефективним методом класифікації – SVM (табл. 3.2).

Таблиця 3.2 – Розгорнуті результати для пари AVG_STD та SVM

Accuracy	0.8962
Precision	0.9011
Recall	0.8962
F1	0.8958

Модель SVM з використанням векторизації AVG_STD демонструє високу якість класифікації для всіх п'яти композиторів (табл. 3.3).

Найкращі результати було отримано для творів Йоганна Себастьяна Баха, де показник повноти (Recall) досяг максимального значення 1.000, що свідчить про безпомилкове розпізнавання всіх творів цього композитора. Також високі показники точності (Precision) спостерігаються для творів Фредеріка Шопена (0.968) та Франца Ліста (0.963). Водночас найгірші показники спостерігаються для Шуберта.

Таблиця 3.3 – Детальні результати класифікації AVG_STD та SVM

Composer	Precision	Recall	F1-Score	Support
Liszt	0.963	0.963	0.963	27.0
Schubert	0.857	0.828	0.842	29.0
Chopin	0.968	0.938	0.952	32.0
Bach	0.933	1.000	0.966	28.0
Beethoven	0.931	0.931	0.931	29.0

Висновки до розділу 3

У цьому розділі було проведено комплексне дослідження способів застосування технік обробки природної мови для аналізу та класифікації музичних творів. Було запропоновано, реалізовано та протестовано ряд методів для токенізації та векторизації музичних даних.

Процес обробки складався з декількох послідовних етапів.

1. Первинна обробка MIDI-файлів з датасету MAESTRO, кодування нот у китайські ієрогліфи з урахуванням їхніх низькорівневих характеристик (висота, динаміка, тривалість).
2. Групування нот в акорди.
3. Токенізація алгоритмом SentencePiece.
4. Векторизація різними методами з використанням Word2Vec.

Крім цього, було розроблено та випробувано п'ять різних методів векторизації: метод усереднення, метод стандартного відхилення,

комбінований метод усереднення та стандартного відхилення, метод з використанням TF-IDF та метод максимального об'єднання.

Результати експериментів показали, що найбільш ефективним виявився комбінований метод усереднення та стандартного відхилення (avg_std) у поєднанні з алгоритмом класифікації SVM. Ця комбінація дозволила досягти точності класифікації 93.10% на тестовій вибірці. Особливо високі показники були отримані для творів Йоганна Себастьяна Баха (recall 100%, F1-score 0.966) та Фредеріка Шопена (precision 0.968, F1-score 0.952).

Проведене дослідження підтверджує ефективність застосування методів обробки природної мови для аналізу музичних даних і демонструє, що низькорівневі характеристики нот у поєднанні з високорівневими патернами мають суттєвий вплив на результати класифікації музичних творів.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ЗАПРОПОНОВАНОГО МЕТОДУ

4.1 Постановка задачі проектування

Запропонований метод аналізу музичних композицій з використанням NLP має відповідати певним заданим базовим вимогам та стандартам якості дослідників у галузі MIR. Метод ФВА застосовується для техніко-економічного обґрунтування реалізації запропонованого підходу загалом. Оскільки кожен з етапів процесу обробки музичних даних має вплив на загальну систему, то він має відповідати заздалегідь визначеним вимогам, насамперед:

- точність класифікації – здатність правильно визначати композиторів музичних творів;
- масштабованість – робота з більшими датасетами музики без суттєвого зниження швидкості роботи застосунку;
- універсальність – можливість використання методу з різними музичними жанрами та інструментами, а також підтримка обробки декількох інструментів однієї пісні;
- відтворюваність результатів – гарантованість отримання стабільних результатів при повторних експериментах з одними й тими самими гіперпараметрами;
- простота інтеграції – гнучкість і зручність розгортання методу в існуючі системи аналізу музичного контенту;
- мінімізація витрат – забезпечення економічної ефективності розробки та експлуатації системи.

Розроблений метод може бути використаний для вирішення численних задач пов'язаних з цифровою репрезентацією музичного контенту, насамперед, для вирішення задачі класифікації музичних творів за композитором або жанром. Основна мета полягає в адаптації класичних

методів NLP для їх подальшого застосування на музичних датасетах, що, в свою чергу, відкриє новий погляд на ефективний аналіз музичних структур і фундаментальних характеристик нот.

4.2 Обґрунтування функцій програмного продукту

З вищеописаного принципу роботи розробленого застосунку, очевидно, що основною функцією є F_0 – розробка методу аналізу музичних композицій з використанням NLP для класифікації музичних композицій за їх автором.

Ця функція, в свою чергу, складається з наступних підфункцій:

- F_1 – техніка кодування фундаментальних характеристик нот;
- F_2 – метод токенизації музичних даних;
- F_3 – алгоритм векторизації музичних послідовностей;
- F_4 – алгоритм машинного навчання для класифікації.

Кожна з представлених функцій має набір варіантів своєї реалізації.

Функція F_1 :

- кодування тільки висоти нот;
- кодування висоти та тривалості;
- кодування висоти, тривалості та динаміки.

Функція F_2 :

- токенизація на основі часових інтервалів;
- токенизація на основі подій.

Функція F_3 :

- векторизація методом усереднення (AVG);
- векторизація методом стандартного відхилення (STD);
- метод усереднення і стандартного відхилення (AVG_STD);
- векторизація з використанням TF-IDF;
- векторизація методом максимального об'єднання (MAX_POOL).

Для кращого розуміння основної функції було побудовано морфологічну карту всіх варіантів її реалізації (рис. 4.1).

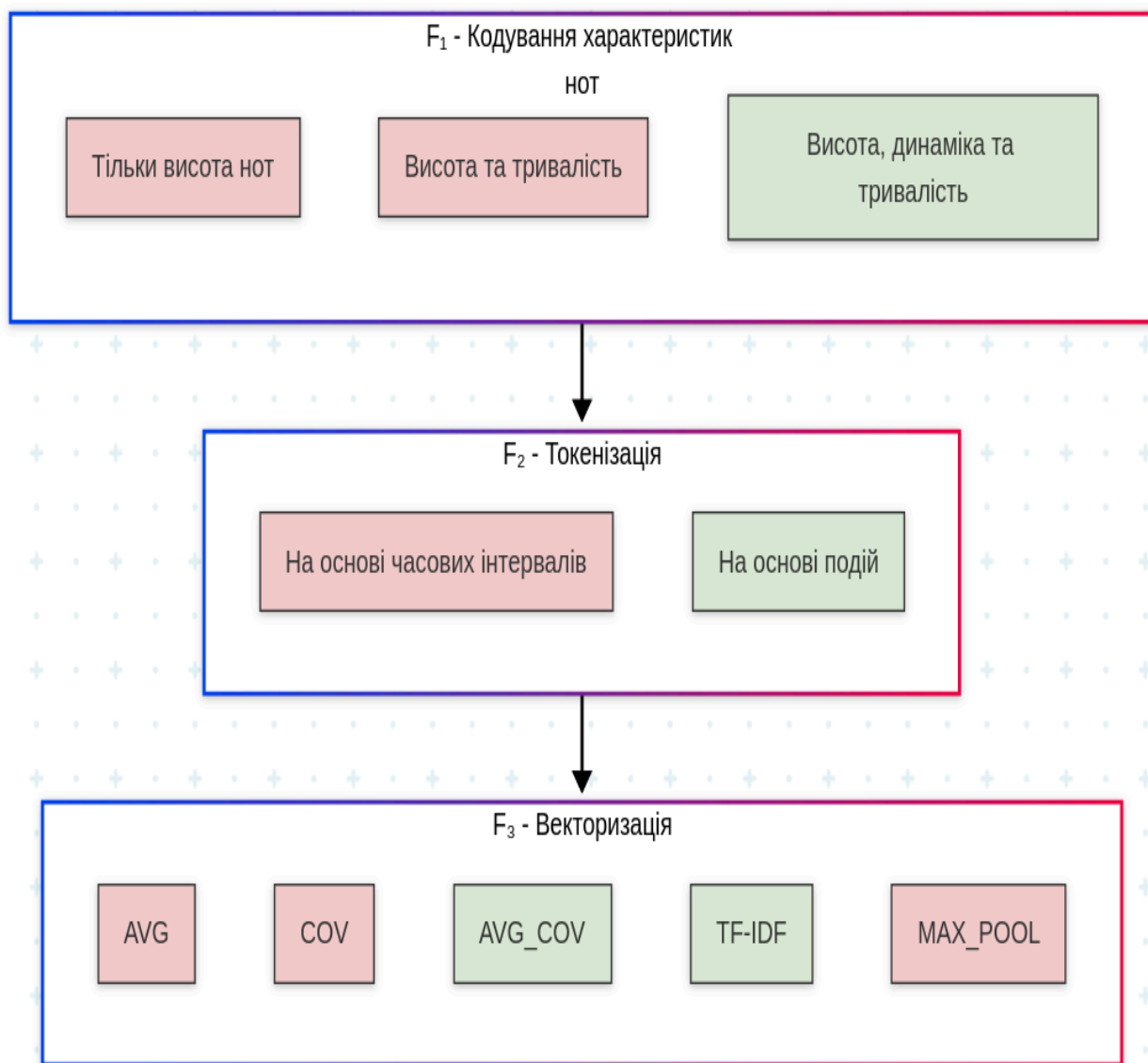


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів побудови основної функції. Спираючись на позитивно-негативну матрицю (табл. 4.1) стає зрозуміло, що використовувати деякі варіанти реалізацій не є раціональним в контексті обраної предметної області.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F ₁	A	Простота реалізації, мінімальний розмір словника.	Втрата важливої інформації про динаміку та ритм.
	B	Збереження ритмічної структури та кращий баланс між складністю та інформативністю.	Відсутність інформації про динамічні нюанси виконання, що важливо для збереження стилю пісні. Значно більший розмір словника.
	C	Максимальна інформативність, повне представлення основних характеристик нот.	Максимальний розмір словника. Можливий надлишок інформації для простих задач.
F ₂	A	Музична природність репрезентації часу. Легкість синхронізації з партитурами.	Неможливість роботи з MIDI-виступами. Складність представлення певних музичних елементів.
	B	Ефективна обробка MIDI-виступів. Краща робота з динамікою. Фіксація і представлення будь-яких музичних технік.	Складність у представлення одночасних подій. Неконстантна часова структура. Мінливість в залежності від виконавця.
F ₃	A	Простота реалізації. Швидкість обчислень. Стабільність результатів.	Втрата інформації про варіативність. Нечутливість до локальних особливостей.
	B	Фіксація варіативності та динаміки. Чутливість до змін у композиції. Відображення гармонічної складності.	Нестабільність при малих вибірках. Висока чутливість до викидів.
	C	Баланс між стабільністю та варіативністю. Комплексне представлення композиції.	Підвищена розмірність векторів. Більша обчислювальна складність.
	D	Виділення характерних музичних елементів. Зменшення впливу загальних патернів. Адаптивність до специфіки композиторів.	Складність застосування до музичних даних. Потреба в великому корпусі.
	E	Виділення найяскравіших характеристик. Стійкість до шуму. Фокус на екстремальних значеннях.	Втрата інформації про розподіл. Чутливість до викидів.

Функція F_1 .

Враховуючи, що використаний датасет є досить великим, то перевагу слід віддати максимальній інформативності та повноті представлення фундаментальних характеристик нот. Хоча інші варіанти і є простішими у реалізації, проте вони призводять до втрати ключової інформації про музичний твір. Зокрема, динаміка є ключовою характеристикою, що відрізняє як і стилі окремих композиторів, так і звучання різних епох. Варіанти А та В відкидаються через неповноту музичного представлення.

Функція F_2 .

При виборі підходу токенизації музичних даних важливою є природа походження досліджуваних даних, а саме чи є вони MIDI-партитурами або MIDI-виступами, різниця між якими була описана вище. Використаний датасет MAESTRO містить саме MIDI-виступи, таким чином, варіант А розглядатися не буде.

Функція F_3 .

Всі запропоновані методи векторизації мають свої унікальні переваги. Було відкинуто варіанти А, В, Е через їхню концептуальну примітивність для поставленої задачі. Через це розглянемо їх далі.

Таким чином, будемо розглядати такий варіанти реалізації програмного продукту:

$$F_1C - F_2B - F_3C, \quad (4.1)$$

$$F_1C - F_2B - F_3D. \quad (4.2)$$

4.3 Обґрунтування системи параметрів програмного продукту

Після проведення аналізу вимог та умов експлуатації розробленого методу аналізу музичних даних, необхідно визначити параметри, за якими

буде розраховано коефіцієнт технічного рівня різних варіантів реалізації. При виборі параметрів було враховано специфіку поставленої задачі.

Параметрами було обрано:

- X_1 – точність (accuracy), загальна частка правильно класифікованих прикладів серед усіх прикладів у наборі даних;
- X_2 – влучність (precision), частка позитивних прикладів серед всіх, що були класифіковані як позитивні;
- X_3 – повнота (recall), частка знайдених позитивних прикладів серед всіх, що насправді є позитивними;
- X_4 – міра-F1 (F1) – гармонічне середнє між точністю та повнотою, що забезпечує збалансовану оцінку якості класифікації.

Гірші, середні і кращі значення параметрів вибираються на основі поставлених вимог та наданого датасету (табл. 4.2).

Таблиця 4.2 – Основні параметри програмного продукту

Назва	Позначення	Значення параметра		
		гірші	середні	кращі
Точність	X_1	0.7	0.8	0.9
Влучність	X_2	0.7	0.8	0.9
Повнота	X_3	0.7	0.8	0.9
F1	X_4	0.7	0.8	0.9

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – аналіз і

класифікація музичних композицій, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей (табл. 4.3). Визначення коефіцієнтів значущості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Таблиця 4.3 – Результати ранжування параметрів

Параметр	Назва параметра	Ранг параметра за оцінкою експерта							Сума рангів R _i	Відхилення - Δ _i	Δ _i ²
		1	2	3	4	5	6	7			
X1	Точність	1	2	1	1	2	1	1	9	-8.5	72.25
X2	Влучність	3	3	3	3	3	4	2	21	3.5	12.25
X3	Повнота	1	2	1	3	2	3	2	14	-3.5	12.25
X4	F1	4	4	4	3	4	3	4	26	8.5	72.25
	Разом	10	10	10	10	10	10	10	70	0	169

Для перевірки ступеня достовірності експертних оцінок, визнаються такі параметри.

Сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.3)$$

де N – число експертів;

n – кількість параметрів.

Середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5 . \quad (4.4)$$

Відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T . \quad (4.5)$$

Сума відхилень по всіх параметрах повинна дорівнювати 0.

Загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 169 . \quad (4.6)$$

Далі порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 169}{7^2(4^3 - 4)} = 0.69 > W_k = 0.67 . \quad (4.7)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67. Числові значення (табл. 4.4), що визначають ступінь переваги i -го параметра над j -тим, a_{ij} визначаються по (4.8).

$$a_{ij} = \begin{cases} 1.5 & X_i > X_j \\ 1.0 & X_i = X_j \\ 0.5 & X_i < X_j \end{cases} . \quad (4.8)$$

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
$X1 \text{ і } X2$	<	<	<	<	<	<	<	<	0.5
$X1 \text{ і } X3$	=	=	=	<	=	<	<	<	0.5
$X1 \text{ і } X4$	<	<	<	<	<	<	<	<	0.5
$X2 \text{ і } X3$	>	>	>	=	=	>	=	>	1.5
$X2 \text{ і } X4$	<	<	<	=	<	>	<	<	0.5
$X3 \text{ і } X4$	<	<	<	=	<	=	<	<	0.5

З отриманих числових оцінок переваги складемо матрицю A . Для кожного параметра зробимо розрахунок вагомості за наступними формулами:

$$K_{bi} = b_i / \sum_{i=1}^n b_i, \quad (4.9)$$

$$b_i = \sum_{i=1}^N a_{ij}. \quad (4.10)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K'_{bi} = b'_i / \sum_{i=1}^n b'_i, \quad (4.11)$$

$$b'_i = \sum_{i=1}^N a'_{ij} b_i. \quad (4.12)$$

Різниця значень коефіцієнтів вагомості не перевищує 2% (табл. 4.5), тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.	
	$X1$	$X2$	$X3$	$X4$	b_i	K_{Bi}	b_i^1	K_{Bi}^1
$X1$	1	0.5	0.5	0.5	2.5	0.16	9.25	0.16
$X2$	1.5	1	1.5	0.5	4.5	0.28	16.25	0.28
$X3$	1.5	0.5	1	0.5	3.5	0.22	12.25	0.20
$X4$	1.5	1.5	1.5	1	5.5	0.34	21.25	0.36
Всього:					16	1	59	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Абсолютні значення параметрів $X1$ (точність), $X2$ (влучність), $X3$ (повнота) та $X4$ (F1) відповідають технічним вимогам умов функціонування даного ПП.

Рахуємо коефіцієнт технічного рівня для кожного варіанта реалізації ПП вказаний (табл. 4.6). Він розраховується за формулою:

$$K_K(j) = \sum_{i=1}^n K_{Bi,j} B_{i,j}, \quad (4.13)$$

де n – кількість параметрів;

K_{Bi} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Функція	Варіант реалізації	Параметр	Абсолютне значення	Бальна оцінка	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	C	X_1	0.83	9.00	0.16	1.44
F2	B	X_4	0.86	26.00	0.36	9.36
F3	C	X_2	0.85	21.00	0.28	5.88
	D	X_3	0.74	14.00	0.20	2.80

За отриманими даними за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}] , \quad (4.14)$$

визначається рівень якості кожного з варіантів.

$$K_{K1} = 1.44 + 9.36 + 5.88 = 16.68 , \quad (4.15)$$

$$K_{K2} = 1.44 + 9.36 + 2.80 = 13.6 . \quad (4.16)$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

- розробка проекту програмного продукту;
- розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи Б, завдання 2 до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_{\Pi} \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.16)$$

де T_P – трудомісткість розробки ПП;

K_{Π} – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P = 48$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання:

$K_{\Pi} = 1.4$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.8$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 48 \cdot 1.4 \cdot 1 \cdot 0.8 = 53.76 \text{ людино-днів.} \quad (4.16)$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_p = 38$ людино-днів, $K_{\Pi} = 0.9$, $K_{СК} = 1$, $K_{СТ} = 0.6$:

$$T_2 = 38 \cdot 0.9 \cdot 1 \cdot 0.6 = 20.52 \text{ людино-днів.} \quad (4.17)$$

Отже, найбільш високу трудомісткість має варіант 1.

В розробці беруть участь один програміст з окладом 40000 грн, один аналітик з окладом 32000 грн. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{M}{T_m \cdot t} \text{ грн,} \quad (4.18)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$СЧ = \frac{40000 + 32000}{2 \cdot 18 \cdot 8} = 250 \text{ грн.} \quad (4.18)$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{3П} = C_{\text{ч}} \cdot T_i \cdot КД, \quad (4.18)$$

де $C_{\text{ч}}$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$КД$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$C_{3П1} = 250 \cdot 53.76 \cdot 1.2 = 16128, \quad (4.18)$$

$$C_{3П2} = 250 \cdot 20.52 \cdot 1.2 = 6156. \quad (4.18)$$

Відрахування на єдиний соціальний внесок становить 22%:

$$C_{\text{ВІД1}} = C_{3П1} \cdot 0.22 = 16128 \cdot 0.22 = 3548.16, \quad (4.19)$$

$$C_{\text{ВІД2}} = C_{3П2} \cdot 0.22 = 6156 \cdot 0.22 = 1354.32. \quad (4.20)$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного програміста з окладом 40000 грн, з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_{\Gamma} = 12 \cdot M \cdot K_3 = 12 \cdot 40000 \cdot 0.2 = 96000 \text{ грн.} \quad (4.21)$$

З урахуванням додаткової заробітної плати:

$$C_{3П} = C_{\Gamma} \cdot (1 + K_3) = 96000 \cdot (1 + 0.2) = 115200 \text{ грн.} \quad (4.22)$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 115200 \cdot 0.22 = 25344 \text{ грн.} \quad (4.23)$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 50000 грн.

$$C_A = K_{\text{ТМ}} \cdot K_A \cdot C_{\text{ПР}} = 1.1 \cdot 0.25 \cdot 50000 = 13750 \text{ грн,} \quad (4.24)$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$C_{\text{ПР}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{\text{ТМ}} \cdot C_{\text{ПР}} \cdot K_P = 1.1 \cdot 50000 \cdot 0.05 = 2750 \text{ грн,} \quad (4.25)$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{\text{ЕФ}} = (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B, \quad (4.26)$$

$$T_{\text{ЕФ}} = (365 - 105 - 21 - 16) \cdot 8 \cdot 0.87 = 1552,08 \text{ години,} \quad (4.27)$$

де D_K – календарна кількість днів у році;

D_B, D_C – відповідно кількість вихідних та святкових днів;

D_P – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_{\text{С}} \cdot K_{\text{З}} \cdot \Pi_{\text{ЕН}}, \quad (4.28)$$

$$C_{\text{ЕЛ}} = 1552,08 \cdot 0,3 \cdot 0,5 \cdot 9,43 = 2195,42 \text{ грн}, \quad (4.29)$$

де $N_{\text{С}}$ – середньо-споживча потужність приладу;

$K_{\text{З}}$ – коефіцієнтом зайнятості приладу;

$\Pi_{\text{ЕН}}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_{\text{Н}} = \Pi_{\text{ПР}} \cdot 0,67 = 50000 \cdot 0,67 = 33500 \text{ грн}. \quad (4.30)$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}, \quad (4.31)$$

$$C_{\text{ЕКС}} = 115200 + 25344 + 13750 + 2750 + 2195 + 33500 = 192739 \text{ грн}. \quad (4.32)$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 192739 / 1552,08 = 124,2 \text{ грн/год}. \quad (4.33)$$

Оскільки в цьому випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_{\text{М}} = C_{\text{М-Г}} \cdot T, \quad (4.33)$$

$$C_{M1} = 124.2 \cdot 53.76 = 6676.992 \text{ грн}, \quad (4.34)$$

$$C_{M2} = 124.2 \cdot 20.52 = 2548.584 \text{ грн}. \quad (4.35)$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{3П} \cdot 0.67, \quad (4.36)$$

$$C_{H1} = 16128 \cdot 0.67 = 10805.76 \text{ грн}, \quad (4.37)$$

$$C_{H2} = 6156 \cdot 0.67 = 4124.52 \text{ грн}. \quad (4.38)$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{3П} + C_{ВІД} + C_M + C_H, \quad (4.39)$$

$$C_{ПП1} = 6156 + 3548 + 6677 + 10806 = 27187 \text{ грн}, \quad (4.40)$$

$$C_{ПП2} = 6156 + 1354 + 2549 + 4125 = 14184 \text{ грн}. \quad (4.41)$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{Kj} / C_{\Phi j}, \quad (4.42)$$

$$K_{\text{ТЕР}1} = 16.68 / 27187 = 6 \cdot 10^{-4}, \quad (4.43)$$

$$K_{\text{ТЕР}2} = 13.6 / 14184 = 10 \cdot 10^{-4}. \quad (4.44)$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що

залишилися після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості.

Цей варіант реалізації програмного продукту має такі параметри:

- кодування нот з урахуванням висоти, динаміки та тривалості;
- використання токенизації на основі подій;
- вибір методу векторизації TF-IDF.

Цей варіант виконання програмного комплексу є компромісом між складністю обчислень і розробки, а також гнучкістю і ефективності виконання поставлених задач.

Висновки до розділу 4

В цій частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту. В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано другий варіант реалізації програмного продукту.

ВИСНОВКИ

У цій роботі було проведено дослідження можливості застосування методів обробки природної мови для аналізу та класифікації музичних творів. Основна ідея полягала в тому, щоб розглянути музику як спосіб комунікації, подібний до мови людини, і застосувати до неї відповідні алгоритми аналізу та обробки.

Ключові результати дослідження:

- було встановлено, що символічне представлення музики у форматі MIDI дозволяє ефективно враховувати низькорівневі характеристики нот (висоту, динаміку, тривалість) та високорівневі музичні патерни, що зробило MIDI файли придатними для подальшої обробки методами NLP;
- запропонований метод кодування музичних даних включає в себе групування нот в акорди, нормалізацію динаміки нот та округлення їх тривалості, що дозволило зменшити розмір словника унікальних символів без суттєвої втрати важливої інформації про музичні твори;
- алгоритм SentencePiece, що було використано для токенізації, виявив часто повторювані музичні фрагменти, що відображають характерні для різних композиторів патерни;
- порівняльний аналіз п'яти різних методів векторизації музичних даних (усереднення, стандартного відхилення, комбінований метод, TF-IDF та максимальне об'єднання) показав, що комбінований метод усереднення та стандартного відхилення найкраще зберігає як загальний стиль музичного твору, так і його внутрішню варіативність;

- алгоритм SVM, що було застосовано до отриманих векторизованих даних, дозволило досягти високої точності класифікації для творів п'яти класичних композиторів (93.10%).

Ці результати підтверджують як наявність унікальних стилістичних особливостей у творчості кожного з них, так і можливість їх виявлення з використанням методів обробки природної мови.

Практична значимість дослідження полягає у можливості застосування досліджуваного підходу для вирішення різноманітних задач аналізу музичного контенту, насамперед, для автоматичної каталогізації музичних творів, розробки рекомендаційних систем, аналізу музичних стилів та жанрів, а також для створення здатних імітувати стиль конкретних композиторів генеративних моделей.

Перспективними напрямками подальших досліджень можуть бути:

- проведення аналогічного дослідження з більшою кількістю композиторів та музичних жанрів;
- дослідження можливостей покращення кодування музичних даних з урахуванням додаткових характеристик, наприклад, використання педалі;
- застосування більш складних архітектур нейронних мереж (зокрема, трансформерів) для моделювання музичного контексту;
- дослідження можливостей генерації нових музичних творів у стилі певних композиторів на основі отриманих моделей.

Отримані результати підтверджують гіпотезу про те, що низькорівневі характеристики нот мають суттєвий вплив на формування унікального музичного стилю композитора і можуть бути ефективно використані для класифікації музичних творів. Дослідження демонструє перспективність застосування методів обробки природної мови в галузі музичного аналізу.

ПЕРЕЛІК ПОСИЛАНЬ

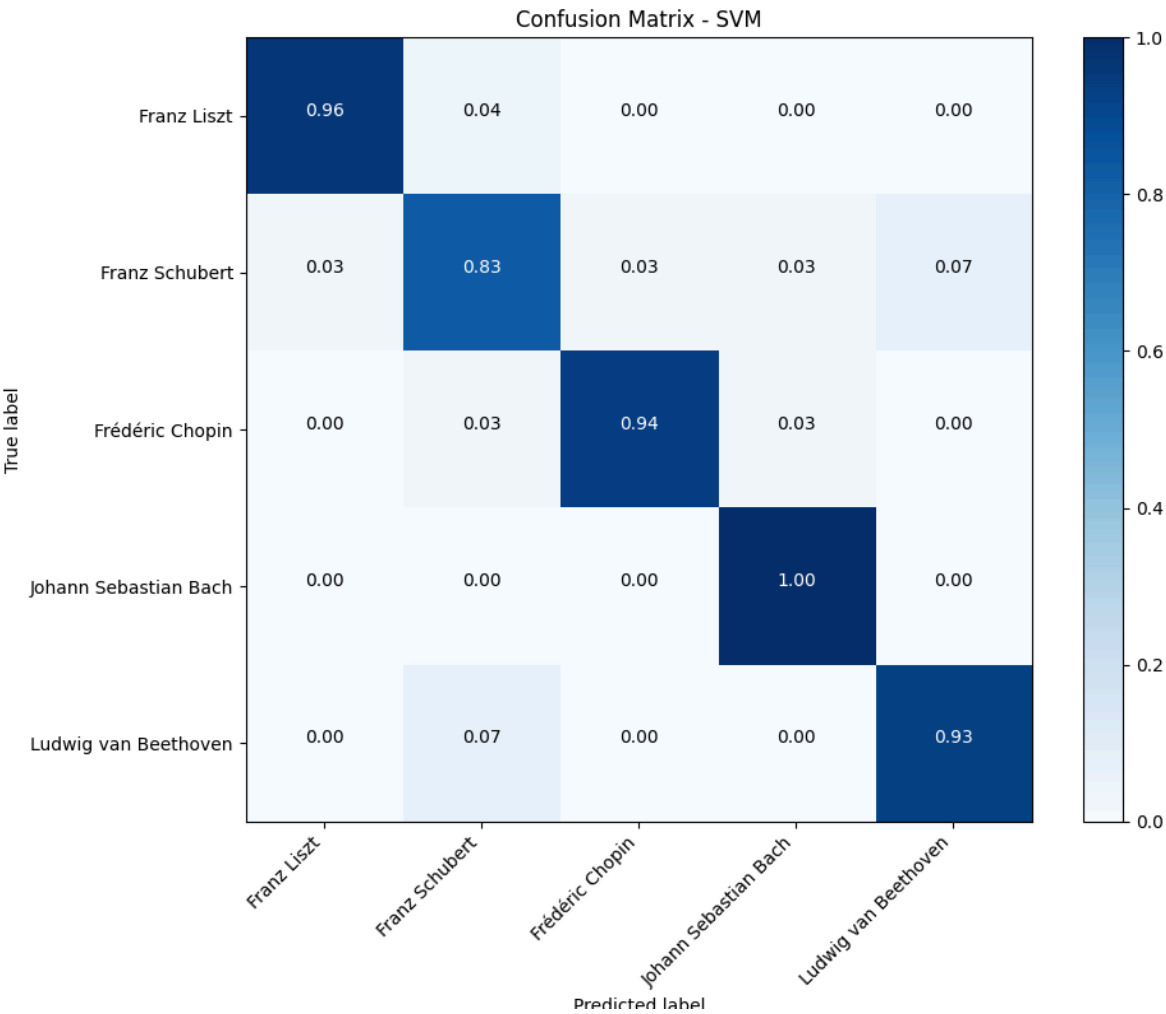
1. Weizenbaum J. ELIZA—a Computer Program for the Study of Natural Language Communication Between Man and Machine. *Communications of the ACM*. 1966. Vol. 9, No. 1. P. 36-45.
2. Bod R. A Unified Model of Structural Organization in Language and Music. *Journal of Artificial Intelligence Research*. 2002. Vol. 17. P. 289–308.
3. Le D.-V.-T., Bigo L., Keller M., Herremans D. Natural Language Processing Methods for Symbolic Music Generation and Information Retrieval: a Survey. *Journal of Artificial Intelligence Research*. 2024. Vol. 61. P. 937-968.
4. Deepaisarn S., Chokphantavee S., Chokphantavee S., Prathipasen P., Buaruk S., Sornlertlamvanich V. NLP-based music processing for composer classification. *Computer Music Journal*. 2023. Vol. 47. P. 58-75.
5. Besson M., Schön D. Comparison Between Language and Music. *Annals of the New York Academy of Sciences*. 2006. Vol. 930, No. 1. P. 232–258.
6. Le D.-V.-T., Bigo L., Keller M., Herremans D. Natural Language Processing Methods for Symbolic Music Generation and Information Retrieval: a Survey. *Journal of Artificial Intelligence Research*. 2024. Vol. 61. P. 937-968.
7. Peeters G., Rafii Z., Fuentes M., Duan Z., Benetos E. et al. Twenty-Five Years of MIR Research: Achievements, Practices, Evaluations, and Future Challenges. *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Hyderabad, 2025. P. 1-5.
8. Jurafsky D., Martin J. H. *Speech and Language Processing*. 3rd ed. New York : Pearson, 2025. 680 p.
9. Medina A. L. S., Leon R. S., Calderón-Vilca H. D. Music Recommender System based on Sentiment Analysis Enhanced with Natural Language Processing Technics. *International Journal of Advanced Computer Science and Applications*. 2023. Vol. 14, No. 3. P. 142-151.

10. Wu Y., Yang Y. MuseMorphose: Full-Song and Fine-Grained Piano Music Style Transfer With One Transformer VAE. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. 2023. Vol. 31. P. 1953–1967.
11. Dannenberg R. B. A Brief Survey of Music Representation Issues, Techniques, and Systems. *Computer Music Journal*. 1993. Vol. 17, No. 3. P. 20-30.
12. Krumhansl C. L. Music Psychology: Tonal Structures in Perception and Memory. *Annual Review of Psychology*. 1991. Vol. 42. P. 277-303.
13. Shepard R. N. Geometrical approximations to the structure of musical pitch. *Psychological Review*. 1982. Vol. 89, No. 4. P. 305-333.
14. Desain P., Honing H. Towards a calculus for expressive timing in music : технічний звіт. Утрехт : Center for Knowledge Technology, 1991. 23 p.
15. Grey J. M. An Exploration of Musical Timbre : дис. ... д-ра філософії / Stanford University. Стенфорд, 1975. 155 p. Department of Music Report STAN-M-2.
16. MIDI Association : офіційний сайт. URL: <https://midi.org/> (дата звернення: 18.05.2025).
17. Barren D. MIDI Files as Training Data. Towards Data Science. 2022. URL: <https://towardsdatascience.com/midi-files-as-training-data-b67852c8b291/> (дата звернення: 18.05.2025).
18. MIDI Explained : навчальний ресурс. PirateMIDI. URL: <https://learn.pirateMIDI.com/learn/lesson/1-midi-explained> (дата звернення: 18.05.2025).
19. Benfield A., Kinsella N. The Sounds of Development: Musical Representation as A(nother) Source of Development Knowledge. *Journal of Development Studies*. 2021. Vol. 57, No. 4. P. 582-599. URL: <https://www.tandfonline.com/doi/full/10.1080/00220388.2020.1772975> (дата звернення: 18.05.2025).
20. REAPER : офіційний сайт аудіоредактора. URL: <https://www.reaper.fm/> (дата звернення: 18.05.2025).

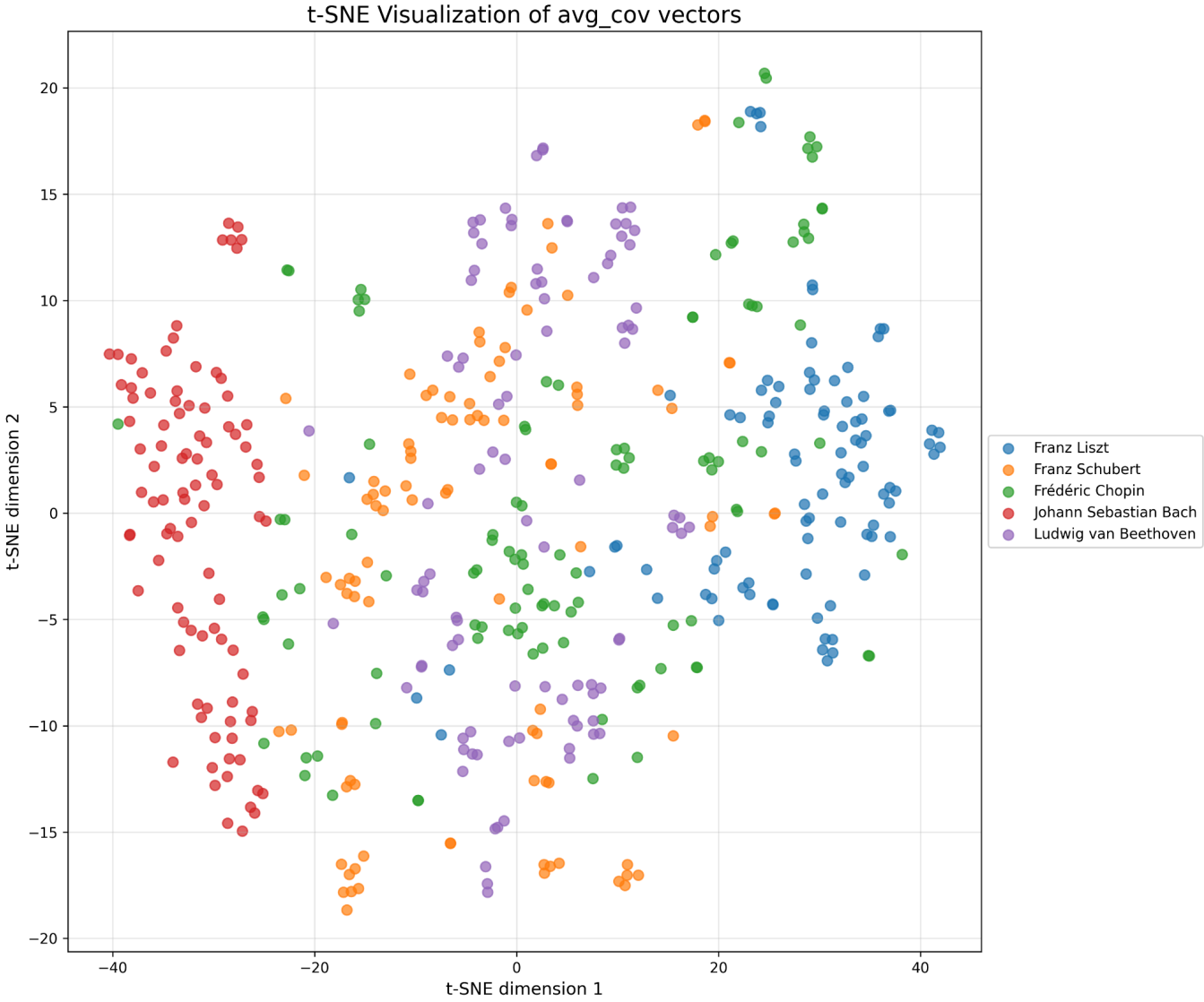
21. MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) : репозиторій даних. Google Magenta. URL:
<https://magenta.tensorflow.org/datasets/maestro> (дата звернення: 18.05.2025).
22. The International Piano-e-Competition : офіційний сайт. URL:
<http://www.piano-e-competition.com> (дата звернення: 18.05.2025).
23. MuseScore : офіційний сайт програми нотного запису. URL:
<https://musescore.com> (дата звернення: 18.05.2025).
24. Herremans D., Chuan C.-H. Modeling Musical Context Using Word2vec. arXiv preprint. 2017. URL: <https://arxiv.org/abs/1706.09088v2> (дата звернення: 18.05.2025).
25. Le D.-V.-T., Bigo L., Keller M. Evaluating Interval-based Tokenization for Pitch Representation in Symbolic Music Analysis. arXiv preprint. 2025. URL: <https://arxiv.org/abs/2501.04630v1> (дата звернення: 18.05.2025).
26. Kudo T., Richardson J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2018. P. 66-71. URL: <https://arxiv.org/abs/1808.06226> (дата звернення: 18.05.2025).
27. Qader W. A., Ameen M. M., Ahmed B. I. An Overview of Bag of Words: Importance, Implementation, Applications, and Challenges. Fifth International Engineering Conference on Developments in Civil & Computer Engineering Applications (IEC2019). Erbil, 2019. P. 215-227.
28. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. Proceedings of the International Conference on Learning Representations (ICLR 2013). 2013. URL: <https://arxiv.org/abs/1301.3781> (дата звернення: 18.05.2025).
29. Sanchez A. Cracking the Code of Context: Word Vectorization Techniques in NLP. Towards AI. 2023. URL: <https://pub.towardsai.net/cracking-the-code-of-context-word-vectorization-techniques-in-nlp-9fa3a90a3368> (дата звернення: 18.05.2025).

30. Kumar D., Sharma A. A Comparative Study on TF-IDF feature Weighting Method and its Analysis using Unstructured Dataset. International Journal of Computer Applications. 2021. Vol. 173, No. 1. P. 21-25.
31. Joulin A., Grave E., Bojanowski P., Mikolov T. Bag of Tricks for Efficient Text Classification. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics. 2017. Vol. 2. P. 427-431.

ДОДАТОК А МАТРИЦЯ НЕВІДПОВІДНОСТЕЙ



ДОДАТОК Б Т-РОЗПОДІЛЕНЕ ВКЛАДЕННЯ СТОХАСТИЧНОЇ
БЛИЗКОСТІ



ДОДАТОК В ЛІСТИНГ ПРОГРАМИ

song_encoding/__init__.py

```
from .corpus import create_corpus

__all__ = [
    "create_corpus",
]
```

song_encoding/corpus.py

```
import os
import pandas as pd
from tqdm import tqdm
import pretty_midi
import data

from .note import Note
from .song import Song

def create_corpus(dataset: pd.DataFrame) -> tuple[str, dict[Note,
str], dict[str, Note]]:
    songs = list[Song]()
    all_notes = list[Note]()
    for i in tqdm(range(dataset.shape[0]), desc="Processing MIDI
files", unit="file"):
        suffix_path = dataset.iloc[i]["midi_filename"]
        path = os.path.join(data.DATA_FOLDER_PATH, suffix_path)
        song = Song(pretty_midi.PrettyMIDI(path))

        songs.append(song)
        all_notes.extend(song)

    unique_sorted_notes = sorted(set(all_notes))

    note2char = {note: encodeIntAsChinese(i) for i, note in
```

```

enumerate(unique_sorted_notes)}
    char2note = {encodeIntAsChinese(i): note for i, note in
enumerate(unique_sorted_notes)}

    corpus = list[str]()
    for song in tqdm(songs, desc="Processing Songs"):
        encoded_song = list[str]()
        for note in song:
            encoded_song.append(note2char[note])
        corpus.append("".join(encoded_song))

    return "\n".join(corpus), note2char, char2note

# encodeIntAsChar encodes an integer to a chinese character.
# Since all chinese characters are located in the range of 0x4E00 to
0x9FFF,
# the max value of the index number is 0x9FFF - 0x4E00 = 20,991.
def encodeIntAsChinese(index_number):
    val = index_number + 0x4E00
    return chr(val)

```

song_encoding/note.py

```

from functools import total_ordering

@total_ordering
class Note:
    pitch: int
    velocity: int
    duration: float

    def __init__(self, pitch: int, velocity: int, duration: float):
        pitch = max(21, pitch)
        pitch = min(108, pitch)
        self.pitch = pitch

    # By default, MIDI velocity is in the range of 0-127.
    # Here it is being converted to a range of 0-4 for the internal
representation.

```

```

self.velocity = round(velocity * 5 / 127)

duration = round(duration, 2)
duration = max(0.0, duration)
duration = min(15.0, duration)
if duration < 0.5:
    # Round to 2 decimal places, then to nearest 0.05
    duration = round(duration * 20) / 20
elif 0.5 <= duration < 2:
    # Round to 1 decimal place
    duration = round(duration, 1)
elif 2 <= duration < 15:
    # Round to the nearest integer
    duration = round(duration)

self.duration = duration

def __eq__(self, other):
    return (
        self.pitch == other.pitch
        and self.velocity == other.velocity
        and self.duration == other.duration
    )

def __lt__(self, other):
    if self.pitch == other.pitch:
        if self.velocity == other.velocity:
            return self.duration < other.duration
        return self.velocity < other.velocity
    return self.pitch < other.pitch

def __hash__(self):
    return hash((self.pitch, self.velocity, self.duration))

```

song_encoding/song.py

```

import pretty_midi
from .note import Note

CHORD_TIMEOUT = 0.005

```

```

class Song(list[Note]):
    def __init__(self, midi: pretty_midi.PrettyMIDI):
        super().__init__()
        if not midi.instruments or not midi.instruments[0].notes:
            return

        sorted_midi_notes = sorted(
            midi.instruments[0].notes,
            key=lambda note: note.start,
        )

        chord = list[Note]()
        chord_start_time = sorted_midi_notes[0].start

        for i, midi_note in enumerate(sorted_midi_notes):
            note = Note(
                pitch=midi_note.pitch,
                velocity=midi_note.velocity,
                duration=midi_note.end - midi_note.start,
            )

            is_note_part_of_chord = midi_note.start - chord_start_time
            <= CHORD_TIMEOUT

            if not is_note_part_of_chord:
                self._standardize_and_add_chord(chord)
                chord.clear()
                chord_start_time = midi_note.start

            chord.append(note)

            is_note_last_in_song = i == len(sorted_midi_notes) - 1
            if is_note_last_in_song:
                self._standardize_and_add_chord(chord)

        def _standardize_and_add_chord(self, chord: list[Note]):
            if not chord:
                return

            # Calculate average velocity and max duration for the chord
            avg_velocity = sum(note.velocity for note in chord) //
            len(chord)
            max_duration = max(note.duration for note in chord)

```

```

        # Standardize velocity and duration for all notes in the chord
        for note in chord:
            note.velocity = avg_velocity
            note.duration = max_duration

        self.extend(sorted(chord, key=lambda note: note.pitch,
reverse=True))

```

processing/tokenize.py

```

import sentencepiece as spm
import os
import cache

MODEL_NAME = os.path.join(cache.FOLDER_PATH, "2_spm")

def tokenize(
    corpus_filepath,
    vocabSize,
    maxSenLength,
) -> list:
    if not os.path.exists(MODEL_NAME + ".model") or not
os.path.exists(MODEL_NAME + ".vocab"):
        spm.SentencePieceTrainer.Train(
            input=corpus_filepath,
            model_prefix=MODEL_NAME,
            vocab_size=vocabSize,
            max_sentence_length=maxSenLength,
        )

    sp = spm.SentencePieceProcessor()
    sp.Load(MODEL_NAME + ".model")

    with open(corpus_filepath, "r") as f:
        corpus = f.read()

    return [sp.EncodeAsPieces(song) for song in corpus.split("\n")]

```

processing/__init__.py

```

from .tokenize import tokenize
from .vectorize import (
    vectorize_avg,
    vectorize_std,
    vectorize_avg_std,
    vectorize_tfidf,
    vectorize_max_pool,
)
from .scale import scale_vectors
from .visualize import visualize_vectors

__all__ = [
    "tokenize",
    "vectorize_avg",
    "vectorize_std",
    "vectorize_avg_std",
    "vectorize_tfidf",
    "vectorize_max_pool",
    "scale_vectors",
    "visualize_vectors",
]

```

processing/vectorize.py

```

import numpy as np
from tqdm import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec

def vectorize_avg(tokenized_sentences: list, model: Word2Vec) -> list:
    vector_size = model.wv.vector_size
    vectorized_sentences = []

    for sentence in tqdm(
        tokenized_sentences,
        desc="Vectorizing sentences with average",
    ):

```

```

    unit="sentence",
):
    temp = np.zeros(vector_size)
    for word in sentence:
        temp += model.wv[word]
    vectorized_sentences.append(temp / len(sentence))

    return vectorized_sentences

```

```

def vectorize_std(tokenized_sentences: list, model: Word2Vec) ->
list:
    vectorized_sentences = []

    for sentence in tqdm(
        tokenized_sentences,
        desc="Vectorizing sentences with standard deviation",
        unit="sentence",
    ):
        word_vectors = []
        for word in sentence:
            word_vectors.append(model.wv[word])
        data = np.array(word_vectors)
        sd = np.std(data, axis=0)
        z = sd.tolist()
        z = np.array(z)
        vectorized_sentences.append(z)

    return vectorized_sentences

```

```

def vectorize_avg_std(tokenized_sentences: list, model: Word2Vec) ->
list:
    vector_size = model.wv.vector_size
    vectorized_sentences = []

    for sentence in tqdm(
        tokenized_sentences,
        desc="Vectorizing sentences with average and standard
deviation",
        unit="sentence",
    ):
        temp = np.zeros(vector_size)
        word_vectors = []

```

```

    for word in sentence:
        temp += model.wv[word]
        word_vectors.append(model.wv[word])
    data = np.array(word_vectors)
    sd = np.std(data, axis=0)
    z = temp / len(sentence)
    z = z.tolist()
    z += sd.tolist()
    z = np.array(z)
    vectorized_sentences.append(z)

    return vectorized_sentences

def vectorize_tfidf(tokenized_sentences: list, model: Word2Vec) ->
list:
    # First create a corpus of detokenized sentences for TF-IDF
    detokenized = [" ".join(sentence) for sentence in
tokenized_sentences]

    # Fit TF-IDF
    tfidf = TfidfVectorizer().fit(detokenized)
    word_to_tfidf = dict(zip(tfidf.get_feature_names_out(),
tfidf.idf_))

    vector_size = model.wv.vector_size
    vectorized_sentences = []

    for sentence in tqdm(
tokenized_sentences, desc="Vectorizing with TF-IDF weights",
unit="sentence"
    ):
        weights = np.zeros(len(sentence))
        vectors = np.zeros((len(sentence), vector_size))

        for i, word in enumerate(sentence):
            if word in word_to_tfidf and word in model.wv:
                weights[i] = word_to_tfidf[word]
                vectors[i] = model.wv[word]

        if weights.sum() > 0:
            weighted_avg = np.average(vectors, weights=weights,
axis=0)
        else:

```

```

        weighted_avg = np.zeros(vector_size)

    vectorized_sentences.append(weighted_avg)

    return vectorized_sentences

def vectorize_max_pool(tokenized_sentences: list, model: Word2Vec) ->
list:
    vector_size = model.wv.vector_size
    vectorized_sentences = []

    for sentence in tqdm(tokenized_sentences, desc="Vectorizing
with max pooling", unit="sentence"):
        if not sentence:
            vectorized_sentences.append(np.zeros(vector_size))
            continue

        # Initialize with the first word's vector or zeros
        if sentence[0] in model.wv:
            max_vector = model.wv[sentence[0]].copy()
        else:
            max_vector = np.zeros(vector_size)

        # Element-wise maximum of all word vectors
        for word in sentence[1:]:
            if word in model.wv:
                max_vector = np.maximum(max_vector, model.wv[word])

    vectorized_sentences.append(max_vector)

    return vectorized_sentences

```

processing/scale.py

```

from sklearn.preprocessing import StandardScaler

def scale_vectors(vectors: list) -> list:
    PredictorScaler = StandardScaler()
    PredictorScalerFit = PredictorScaler.fit(vectors)

```

```
return PredictorScalerFit.transform(vectors)
```

processing/visualize.py

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter
import matplotlib.colors as mcolors
import seaborn as sns

def visualize_vectors(
    data_vectors: np.ndarray,
    labels: list,
    title: str,
    save_path: str,
    perplexity: int = 30,
    random_state: int = 42,
    figsize: tuple = (12, 10),
):
    unique_composers = sorted(list(set(labels)))
    label_to_index = {composer: i for i, composer in
enumerate(unique_composers)}
    numeric_labels = np.array([label_to_index[composer] for
composer in labels])

    print(f"Performing t-SNE with perplexity={perplexity}...")
    tsne = TSNE(n_components=2, perplexity=perplexity,
random_state=random_state)
    embedded_vectors = tsne.fit_transform(data_vectors)

    plt.figure(figsize=figsize)

    # use default seaborn color palette
    palette = sns.color_palette("tab10")

    plt.figure(figsize=figsize)
    for i, composer in enumerate(unique_composers):
        mask = numeric_labels == i
        plt.scatter(
```

```

        embedded_vectors[mask, 0],
        embedded_vectors[mask, 1],
        label=composer,
        color=palette[i % len(palette)],
        alpha=0.7,
        s=50, # Point size
    )

    plt.title(title, fontsize=16)
    plt.xlabel("t-SNE dimension 1", fontsize=12)
    plt.ylabel("t-SNE dimension 2", fontsize=12)
    plt.grid(alpha=0.3)

    plt.legend(loc="center left", bbox_to_anchor=(1, 0.5),
              fontsize=10)

    plt.tight_layout()

    plt.savefig(save_path, dpi=300, bbox_inches="tight")

    plt.close()

```

cache/__init__.py

```

import os

FOLDER_PATH = "local/cache"

os.makedirs(FOLDER_PATH, exist_ok=True)

```

data/visualize.py

```

import matplotlib.pyplot as plt
import os
import pandas as pd
import seaborn as sns
import cache

```

```

def composer_song_counts(dataset: pd.DataFrame) -> None:
    composer_counts =
dataset["canonical_composer"].value_counts().head(10).iloc[::-1]
    plt.figure(figsize=(10, max(6, len(composer_counts) * 0.4)))
    bars = plt.barh(composer_counts.index,
composer_counts.values.astype(float), height=0.7)
    for bar in bars:
        plt.text(
            bar.get_width() + 0.5,
            bar.get_y() + bar.get_height() / 2,
            str(int(bar.get_width())),
            va="center",
        )
    plt.title("Number of Pieces per Composer (Top 10)",
fontsize=14)
    plt.xlabel("Number of Pieces")
    plt.tight_layout()
    plt.savefig(
os.path.join(cache.FOLDER_PATH, "composer_song_counts.png"),
dpi=300,
bbox_inches="tight",
)
    plt.close()

def song_durations(dataset: pd.DataFrame) -> None:
    duration_minutes = dataset["duration"] / 60
    mean_duration = dataset["duration"].mean()
    median_duration = dataset["duration"].median()

    plt.figure(figsize=(10, 6))
    sns.histplot(duration_minutes.to_numpy(), bins=30, kde=True)
    plt.axvline(
mean_duration / 60,
color="r",
linestyle="--",
label=f"Mean: {mean_duration / 60:.1f} min",
)
    plt.axvline(
median_duration / 60,
color="g",
linestyle=":",
label=f"Median: {median_duration / 60:.1f} min",
)

```

```
plt.title("Distribution of Piece Durations", fontsize=14)
plt.xlabel("Duration (minutes)")
plt.ylabel("Count")
plt.legend()
plt.tight_layout()
plt.savefig(
    os.path.join(cache.FOLDER_PATH, "song_durations.png"),
    dpi=300,
    bbox_inches="tight",
)
plt.close()
```

data/__init__.py

```
from .dataset import DATA_FOLDER_PATH, new_dataset, filter_dataset

__all__ = [
    "DATA_FOLDER_PATH",
    "new_dataset",
    "filter_dataset",
]
```

data/dataset.py

```
import pandas as pd
import os
from . import visualize

DATA_FOLDER_PATH = "local/maestro-v3.0.0"

def new_dataset() -> pd.DataFrame:
    df = pd.read_csv(os.path.join(DATA_FOLDER_PATH,
    "maestro-v3.0.0.csv"))

    # Add an ID column to track the original index
    df["id"] = df.index

    # Remove the extra fields
```

```

df = df.astype(
{
    "id": pd.Int64Dtype(),
    "canonical_composer": pd.StringDtype(),
    "canonical_title": pd.StringDtype(),
    "duration": pd.Float64Dtype(),
    "midi_filename": pd.StringDtype(),
}
)

visualize.composer_song_counts(df)
visualize.song_durations(df)

return df

def filter_dataset(df: pd.DataFrame) -> pd.DataFrame:
    # Remove songs with multiple composers
    df = df[~df["canonical_composer"].str.contains("/")]

    # Remove duplicate recordings
    df = (
        df.sort_values("duration")
        .groupby(["canonical_composer", "canonical_title"])
        .first()
        .reset_index()
    )["id", "canonical_composer", "canonical_title", "duration",
    "midi_filename"]

    composer_names = [
        "Frédéric Chopin",
        "Franz Schubert",
        "Ludwig van Beethoven",
        "Franz Liszt",
        "Johann Sebastian Bach",
    ]
    df = df[df["canonical_composer"].isin(composer_names)]

    return df

```

src/classify.py

```

import os
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,
cross_val_score, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

import data
import cache
import processing

dataset = data.filter_dataset(data.new_dataset())

vectors_types = ["avg", "std", "avg_std", "tfidf", "max_pool"]
vectors = {}
for vector_type in vectors_types:
    filepath = os.path.join(cache.FOLDER_PATH,
f"4_songs_vectors_{vector_type}.pkl")
    if os.path.exists(filepath):
        with open(filepath, "rb") as f:
            vectors[vector_type] = pickle.load(f)

def evaluate_classifier(X, y, classifier_name, classifier):
    X = processing.scale_vectors(X)

    skf = StratifiedKFold(n_splits=5, shuffle=True,
random_state=42)
    cv_scores = cross_val_score(classifier, X, y, cv=skf,
scoring="accuracy")

    X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.3, random_state=42, stratify=y
)

```

```

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

detail_results = {
    "cv_accuracy_mean": cv_scores.mean(),
    "cv_accuracy_std": cv_scores.std(),
    "test_accuracy": accuracy_score(y_test, y_pred),
    "classification_report": classification_report(y_test, y_pred,
output_dict=True),
}

return classifier, cv_scores.mean(), detail_results

results = {}
detailed_results = {}

composers = dataset["canonical_composer"].to_list()
unique_composers = sorted(set(composers))
print(f"Number of unique composers: {len(unique_composers)}")
print(f"Classes: {unique_composers}")

classifiers = {
    "Random Forest": RandomForestClassifier(n_estimators=100,
random_state=42),
    "SVM": SVC(kernel="rbf", C=10, gamma="scale", random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Logistic Regression": LogisticRegression(max_iter=1000, C=1.0,
random_state=42),
}

trained_classifiers = {}

for vector_type, X in vectors.items():
    results[vector_type] = {}
    detailed_results[vector_type] = {}

    for cls_name, classifier in classifiers.items():
        trained_classifier, accuracy, details = evaluate_classifier(
            X, composers, cls_name, classifier
        )

```

```

        results[vector_type][cls_name] = accuracy
        detailed_results[vector_type][cls_name] = details
        trained_classifiers[cls_name] = trained_classifier

summary = pd.DataFrame(results)
print("\nSummary of classification accuracies:")
print(summary)

summary_csv_path = os.path.join(cache.FOLDER_PATH,
"classification_summary.csv")
summary.to_csv(summary_csv_path)
print(f"Summary table saved to: {summary_csv_path}")

best_accuracy = 0
best_vector = None
best_classifier = None

for vector_type, classifiers in results.items():
    for cls_name, accuracy in classifiers.items():
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_vector = vector_type
            best_classifier = cls_name

print(f"\nBest accuracy: {best_accuracy:.4f}")
print(f"Best vector type: {best_vector}")
print(f"Best classifier: {best_classifier}")

best_model = trained_classifiers[best_classifier]
best_X = vectors[best_vector]
best_X_scaled = processing.scale_vectors(best_X)

X_train, X_test, y_train, y_test = train_test_split(
    best_X_scaled, composers, test_size=0.3, random_state=42,
    stratify=composers
)

best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

```

```

cm = confusion_matrix(y_test, y_pred)
cm_normalized = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
plt.figure(figsize=(12, 10))
plt.imshow(cm_normalized, interpolation="nearest", cmap=plt.cm.Blues)
plt.title(f"Normalized Confusion Matrix: {best_vector} +
{best_classifier}")
plt.colorbar()
tick_marks = np.arange(len(unique_composers))
plt.xticks(tick_marks, unique_composers, rotation=90)
plt.yticks(tick_marks, unique_composers)
thresh = cm_normalized.max() / 2.0
for i, j in np.ndindex(cm_normalized.shape):
    plt.text(
        j,
        i,
        f"{cm_normalized[i, j]:.2f}",
        horizontalalignment="center",
        color="white" if cm_normalized[i, j] > thresh else "black",
    )
plt.tight_layout()
plt.ylabel("True Composer")
plt.xlabel("Predicted Composer")
cm_plot_path = os.path.join(
    cache.FOLDER_PATH,
    f"confusion_matrix_{best_vector}_{best_classifier.replace(' ',
    '_')}.png"
)
plt.savefig(cm_plot_path)
plt.close()
print(f"Confusion matrix saved to: {cm_plot_path}")

```

src/vectorize.py

```

import os
import gensim
import pickle
import logging
from typing import Callable
import numpy as np
from gensim.models import Word2Vec

```

```

import data
import song_encoding
import processing
import cache

logging.basicConfig(
    format="%(asctime)s - %(levelname)s - %(message)s",
    level=logging.INFO,
    datefmt="%H:%M:%S",
)

logging.info("The script has started...")

dataset = data.new_dataset()
logging.info("Dataset loaded")

CORPUS_FILEPATH = os.path.join(cache.FOLDER_PATH, "1_corpus.txt")
NOTE2CHAR_FILEPATH = os.path.join(cache.FOLDER_PATH,
    "1_note2char.pkl")
CHAR2NOTE_FILEPATH = os.path.join(cache.FOLDER_PATH,
    "1_char2note.pkl")
if (
    os.path.exists(CORPUS_FILEPATH)
    and os.path.exists(NOTE2CHAR_FILEPATH)
    and os.path.exists(CHAR2NOTE_FILEPATH)
):
    logging.info("Corpus already exists, skipping creation.")
else:
    corpus, note2char, char2note =
song_encoding.create_corpus(dataset)

    with open(CORPUS_FILEPATH, "w", encoding="utf-8") as f:
        f.write(corpus)
    with open(NOTE2CHAR_FILEPATH, "wb") as f:
        pickle.dump(note2char, f)
    with open(CHAR2NOTE_FILEPATH, "wb") as f:
        pickle.dump(char2note, f)

    logging.info("Corpus created")

TOKENIZED_SONGS_FILEPATH = os.path.join(cache.FOLDER_PATH,

```

```

"2_tokenized_songs.pkl")
if os.path.exists(TOKENIZED_SONGS_FILEPATH):
    with open(TOKENIZED_SONGS_FILEPATH, "rb") as f:
        tokenized_songs = pickle.load(f)

    logging.info("Tokenized songs already exist, skipping
tokenization.")
else:
    tokenized_songs = processing.tokenize(
        corpus_filepath=CORPUS_FILEPATH,
        vocabSize=25_000,
        maxSenLength=100_000,
    )

    with open(TOKENIZED_SONGS_FILEPATH, "wb") as f:
        pickle.dump(tokenized_songs, f)

    logging.info("Tokenization complete")

WORD2VEC_FILEPATH = os.path.join(cache.FOLDER_PATH,
"3_word2vec.model")
if os.path.exists(WORD2VEC_FILEPATH):
    word2vec = gensim.models.Word2Vec.load(WORD2VEC_FILEPATH)
    logging.info("Word2Vec model loaded.")
else:
    word2vec = gensim.models.Word2Vec(
        sentences=tokenized_songs,
        window=10,
        min_count=1,
        workers=4,
        sg=1,
    )
    word2vec.save(WORD2VEC_FILEPATH)

    logging.info("Word2Vec model trained and saved.")

dataset = data.filter_dataset(dataset)
relevant_ids = dataset["id"].to_list()
tokenized_songs = [tokenized_songs[i] for i in relevant_ids]

def save_vectors(vectorisation_method, vectorize_func:

```

```

Callable[[list, Word2Vec], list]):
    filepath = os.path.join(cache.FOLDER_PATH,
f"4_songs_vectors_{vectorisation_method}.pkl")
    if not os.path.exists(filepath):
        vectors = vectorize_func(tokenized_songs, word2vec)

    with open(filepath, "wb") as f:
        pickle.dump(vectors, f)

    processing.visualize_vectors(
        data_vectors=np.array(processing.scale_vectors(vectors)),
        labels=dataset["canonical_composer"].to_list(),
        title=f"t-SNE Visualization of {vectorisation_method}
vectors",
        save_path=os.path.join(cache.FOLDER_PATH,
f"tsne_{vectorisation_method}.png"),
    )

vectors_avg = save_vectors("avg", processing.vectorize_avg)
vectors_std = save_vectors("std", processing.vectorize_std)
vectors_avg_std = save_vectors("avg_std",
processing.vectorize_avg_std)
vectors_tfidf = save_vectors("tfidf", processing.vectorize_tfidf)
vectors_max_pool = save_vectors("max_pool",
processing.vectorize_max_pool)

```