Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут телекомунікаційних систем
(повна назва інституту/факультету)

Кафедра телекомунікацій
(повна назва кафедри)

«На правах рукопису»
УДК _____

До захисту допущено

В.о. завідувача кафедри

_____ Явіся В.С.___
(підпис) (ініціали, прізвище)

“___”_____2019_р.

Магістерська дисертація

на здобуття освітнього ступеня «магістр»

Спеціальність 172 Телекомунікації та радіотехніка,
(код і назва)

За освітньо-професійною програмою Інженерія та програмування інфокомунікацій.

на тему: «Розробка макету системи адаптації на основі методів машинного навчання для мобільної мережі» _____

Виконав: студент _2_ курсу, групи ТЗ - 81мп
(шифр групи)

Цмокалюк Денис Миколайович _____ _ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент, к.т.н. с. н. с. Міночкін Д.А._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем
(повна назва)

Кафедра телекомунікацій
(повна назва)

Спеціальність 172 Телекомунікації та радіотехніка
(код і назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою Інженерія та програмування інфокомунікацій.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____Явіся В.С.
(підпис) (ініціали, прізвище)
«___»_____2019 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Цмокалюку Денису Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації: «Розробка макету системи адаптації на основі методів машинного навчання для мобільної мережі»

науковий керівник дисертації: Міночкін Д.А, к.т.н., с. н. с. ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «_07_» «_11_» 2019р. № _3840-с_

2. Строк подання студентом дисертації 10 грудня 2019 р.

3. Об'єкт дослідження: система адаптацій на основі методів машинного навчання

4. Предмет дослідження методи машинного навчання у мобільній розробці

5. Перелік завдань, які потрібно розробити

- огляд технологій мобільної розробки;
- аналіз особливостей багатоплатформної розробки;
- загальний аналіз методів машинного навчання;
- розробка системи адаптації;

6. Орієнтовний перелік ілюстративного матеріалу

Презентація PowerPoint _____

_____

_____

7. Орієнтовний перелік публікацій _____

_____

_____

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 17 вересня 2018 р. _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Розробка, оформлення, узгодження та затвердження технічного завдання на магістерську дисертацію	18.09.2018 – 05.10.2018	виконано
2	Опрацювання літератури з теми досліджень	06.10.2018 – 10.11.2018	виконано
3	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	11.11.2018 – 30.11.2018	виконано
4	Огляд технологій і платформ для створення багатоплатформних додатків	01.12.2018 – 20.01.2019	виконано
5	Вибір основного шаблону проектування макету системи адаптацій	21.01.2019 – 28.02.2019	виконано
6	Огляд методів машинного навчання та їх аналіз	01.03.2019 – 30.04.2019	виконано
7	Розробка системи адаптації на основі методів машинного навчання	01.05.2019 – 31.10.2019	виконано
8	Оформлення пояснювальної записки	01.11.2019 – 30.11.2019	виконано

9	Підготовка доповіді до захисту та офомлення ілюстративного матеріалу	01.12.2019 – 09.12.2019	виконано

Студент _____ Цмокалюк Д.М.

(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Міночкін Д.А.

(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: «Розробка макету системи адаптації на основі методів машинного навчання для мобільної мережі»

Дипломна робота містить: 80 с., 12 табл., 9 рис., 17 посилань на джерела.

Метою магістерської дисертації є створення оптимальної системи навігації на підставі отриманої з найближчої базової станції інформації щодо місцеположення та розробка її реального прототипу.

Навігація в сучасних містах з використанням супутників створює великі труднощі. Високий рівень щільності міської забудови серйозно впливає на якість сигналу від супутникової системи. Перешкоди, які виникають, тягнуть за собою серйозне зменшення точності аж до помилки в кілька сотень метрів.

Актуальність диссертації полягає у тому, що незабаром проблема навігації в найбільших містах стане критичною для їх подальшого розвитку. Але з розвитком стільникового зв'язку і збільшенням кількості базових станцій зможемо дати жителям і гостям чудову можливість для навігації.

Об’єктом дослідження є система адаптації, виконана на основі методів машинного навчання.

Ключові слова: машинне навчання, мобільна розробка, багатоплаформна розробка, React Native, базова станція, нейронна мережа.

ABSTRACT

for a master's dissertation

main topic: « Development of adaptation system based on machine learning methods for mobile network»

The dissertation contains: 80 pp., 12 tables, 9 figs., 17 references to sources.

The purpose of the master's dissertation is to create an optimal navigation system based on information from the nearest base station about the location and to develop its real prototype.

Navigating in modern cities using satellites creates great difficulty. The high density of urban development seriously affects the signal quality of the satellite system. The resulting obstacles result in a serious reduction in accuracy up to a few hundred meters.

The urgency of the dissertation is that soon the problem of navigation in major cities will become critical for their further development. But with the development of cellular communication and the increase in the number of base stations, we can give residents and guests a great opportunity to navigate.

The object of the study is an adaptation system based on machine learning methods.

Keywords: machine learning, mobile development, multiplatform development, React Native, base station, neural network.

Пояснювальна записка
до магістерської дисертації

на тему: «Розробка макету системи адаптації на основі методів машинного навчання для мобільної мережі»

Київ – 2019 року

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ANN	(Artificial Neural Network) – штучна нейронна мережа
API	(Application Programming Interface) – прикладний програмний інтерфейс
CID	(Cell ID) – ідентифікатор базової станції
CSS	(Cascading Style Sheets) – каскадні таблиці стилів
GPS	(Advanced Micro Devices) – система глобального позиціонування
mcc	(Mobile Country Code) – код країни
mnc	(Mobile Network Code) – код оператора
HTML	(Hypertext Markup Language) – мова розмітки гіпертекстових документів
MVC	(Model-view-controller) – модель–представлення–контролер
MVVM	(Model-View-ViewModel) – модель–представлення–модель
ОС	операційна система
ШНМ	штучна нейронна мережа

ВСТУП

Сьогодні важко уявити життя без навігатора. Зі збільшенням числа доріг і ускладненням інфраструктури міст, стає складніше дістатися з пункту А в пункт В без відстеження своїх поточних координат.

Бажання полегшити собі життя автоматичним визначенням місцезнаходження в подорожах і не тільки з'явилося давно, як і пристрої, що дозволяють це зробити. Перші навігатори з'явилися на початку XX століття (1920 р.). Прилад нагадував звичайний наручний годинник. Він складався з невеликої, згорнутої рулоном карти і рукоятки, яку можна було обертати, в залежності від пересування по місцевості і по карті. Разом з освоєнням космосу з'явилися такі супутникові системи як GPS і GLONASS . Задля досягнення цієї ступені розвитку ми повинні бути зобов'язані військовим, які перші випробували і вдосконалили навігаційні апарати. Американський проект «TRANSIT» був запущений в 1958 р.: він став аналогом сучасної GPS-навігації. Удосконалена система визначення координат і моніторингу об'єктів, що переміщаються по поверхні нашої планети, була створена в 1960 році, завдяки виведеним на орбіту супутникам.

Також розвиток бездротових систем призвів до зростання популярності network - based navigation systems , які працюють за допомогою методів тріангуляції. Тріангуляція - один з методів створення мережі опорних геодезичних пунктів, а також сама ця мережа. Полягає в геодезичній побудові на місцевості системи пунктів, що утворюють трикутники, у яких вимірюються всі кути і довжини деяких базових (базисних) сторін.

Навігація в сучасних містах з використанням супутників створює великі труднощі. Високий рівень щільності міської забудови серйозно впливає на якість

сигналу від супутникової системи. Перешкоди, які виникають, тягнуть за собою серйозне зменшення точності аж до помилки в кілька сотень метрів.

Актуальність диссертації полягає у тому, що незабаром проблема навігації в найбільших містах стане критичною для їх подальшого розвитку. Але з розвитком стільникового зв'язку і збільшенням кількості базових станцій зможемо дати жителям і гостям чудову можливість для навігації.

Метою магістерської дисертації є створення оптимальної системи навігації на підставі отриманої з найближчої базової станції інформації щодо місцеположення та розробка її реального прототипу.

Для досягнення цієї мети необхідно вирішити кілька серйозних дослідницьких, технологічних, а також практичних завдань:

• дослідити існуючі рішення. Тема навігації далеко не нова і існує величезна кількість різних рішень. Необхідно детально вивчити вже існуючі напрацювання, виділити найбільш підходящі для практичного використання в кінцевому продукті;

• визначитися, які дані можливо отримати при підключенні до мобільного мережі. Вивчити, як отримана інформація допоможе визначити координати місцезнаходження кінцевого користувача;

• визначитися з базовими джерелами даних для нашої навігаційної системи. Будь-яка система використовує дані з різних датчиків для визначення стану, у якому знаходиться безпосередньо користувач (наприклад, стан спокою, рух і т.д.). Акселерометр, зазвичай, є базовим. Його можуть доповнювати один або кілька додаткових датчиків таких як гіроскоп, компас, магнітометр і інші. Необхідно опрацювати варіанти для кожної з комбінацій і вибрати найбільш точне і просте рішення;

• вибрати метод для обробки даних з датчиків, оптимальний по швидкості і точності;

• знайти найбільш зручний спосіб реалізації повноцінного продукту для практичного застосування користувачами. На ринку представлено кілька основних мобільних ОС, тому необхідно вивчити можливості для створення додатка під кожну з них.

1. АНАЛІЗ ТЕХНОЛОГІЙ І ПЛАТФОРМ ДЛЯ СТВОРЕННЯ КРОСПЛАТФОРМНИХ ДОДАТКІВ

1.1 Огляд методів нативної і кросплатформної розробки

Під нативною (рідної) розробкою мається на увазі використання оригінальних мов і інструментів розробки мобільної операційної системи. Для iOS додатки створюються в середовищі розробки XCode на мовах Objective -C, Swift. Для створення додатків під Android використовується середовище Android Studio і мова Java, Kotlin, Go . Кожне середовище розробки містить цілий комплекс утиліт для написання коду, проектування інтерфейсу, налагодження, профілювання (моніторингу) і збірки додатків. І середовище, і відповідний набір утиліт створені спеціально під кожну мобільну операційну систему і є максимально зручними і потужними засобами розробки мобільних додатків.

Кросплатформна (багатоплатформна) розробка має на увазі використання спеціальних утиліт (фреймворків) для створення програми на основі сімейства мов JavaScript . Вся структура і логіка додатка створюється за допомогою таких інструментів (React Native, PhoneGap, Xamarin, Cordova та ін.) на JavaScript, а потім обертається в нативний елемент, тобто інтегрується в базовий проект для XCode або Android Studio, що дозволяє створювати збірки проекту з однією і теж ж логікою під кілька операційних систем відразу.

Найближча аналогія у випадку з персональними комп'ютерами: MS Word, Skype, поштові агенти, календарі - це нативно розроблені додатки під настільну операційну систему; все, що відбувається в браузері (сайти, онлайн-редактори тексту і графіки, соціальні мережі, чати, форуми) - це кросплатформні технологіїх [1].

Плюси кросплатформної розробки

Багатоплатформовий підхід до розробки має наступні позитивні моменти:

1. Потрібно менше ресурсів для реалізації програми відразу під кілька платформ. У цьому, власне, і суть кросплатформного підходу - один і той же код працює і на iOS , і на Android. Програмістів, які займаються проектом, потрібно рівно в два рази менше. Дизайнер робить тільки один набір графіки. Все це знижує кількість робочих годин і бюджет проекту.

2. Менший час на розробку. За рахунок відсутності унікальних елементів інтерфейсу і простіших технологій час на створення простих продуктів, як правило, менше.

3. Спрощений цикл оновлення продукту. Якщо в проект потрібно щось то додати або виправити якусь помилку, це робиться відразу для всіх платформ, на які поширюється проект.

4. Можливість використання мобільної версії сайту. Більшість кросплатформних рішень використовують сімейство JavaScript мов, тому якщо у вас вже є мобільна версія сайту, значна частина коду і матеріалів може бути використана в додатку без змін.

5. Використання єдиної логіки додатка. Закладена логіка в роботу програми буде працювати гарантовано однаково для всіх платформах. Досить часто це може бути і мінусом через різну архітектуру операційних систем (яскравий приклад - кнопка Назад в навігації між екранами: в Android передбачена апаратна кнопка Back для цих цілей, в iOS - рух пальцем від лівій частині екрана або ж наявність кнопки в лівій частини навігаційної панелі; якщо кнопку не робити зовсім, користувачі iOS не зможуть повернутися назад, якщо зробити, але не на тому місці і яка буде виглядіи нестандартно, користувачам iOS буде незвично і незручно, якщо зробити як в iOS, буде незвично користувачам Android, але написана і налагоджена один раз логіка містить потенційно меншу кількість помилок і розбіжностей в своїй роботі: вам не доведеться пробляти подвійну і потрійну роботу з пошуку проблем на кожній платформі.

Плюси нативної розробки

Розробка на рідних технологіях і мовами під iOS і Android має наступні позитивні моменти:

1. Швидкість роботи програми. Так як додаток створюється з використанням оригінальних інструментів розробки (XCode, Android Studio), одержуваний в результаті компіляції проекту код є оптимальним для даної платформи. Додаток отримує повну апаратну підтримку пристрою (обробка тих же зображень здійснюється окремим процесором, спеціально для цього призначених - GPU), використовується багатопотоковість для реалізації складних завдань і завантаження контенту у фоні, в процесі розробки програмісти можуть вимірювати швидкість роботи всіх ділянок коду і при необхідності їх оптимізувати, в їх розпорядженні також є інструменти з моніторингу використання оперативної пам'яті, пошуку можливих витоків і т.д.

2. Гнучкість в реалізації. На відміну від обмежень в побудові інтерфейсу і складності візуальних ефектів, що накладаються фреймворками для кросплатформної збірки проектів, в нативній розробці реалізувати можна все, на що здатні технології тієї чи іншої мобільної операційної системи.

3. Використання останніх технологій і залежність від кросплатформних фреймворків. Новий програмний та апаратний функціонал, наданий компаніями-виробниками обладнання й операційної системи, стає доступний для реалізації відразу після випуску відповідних оновлень. Наприклад, в нових версіях iOS закладена можливість пошуку усередині додатків: в кожному з них повинен бути реалізований спеціальний метод, який повертає результати по певним пошуковим запитам. В результаті для тих додатків, в яких цей функціонал реалізований, доступна можливість пошуку контенту через системний розділ пошуку в iOS, там же, де здійснюється пошук додатків, контактів, подій та іншої інформації. У випадку з кроссплатформнною розробкою для реалізації подібного функціоналу доведеться чекати не тільки релізу iOS, але і поновлення

відповідного фреймворка, причому, коли з'явиться підтримка тих чи інших нових можливостей і чи з'явиться взагалі, передбачити неможливо.

4. Легкість і якість тестування. Крім згаданого інструментарію для контролю використання додатком апаратних ресурсів пристрою в розпорядженні розробників і тестувальників є цілий комплекс технологій. По-перше, всі параметри системи в процесі роботи програми контролюються автоматично. Якщо додаток став використовувати більше пам'яті, ніж це очікується, або більше ресурсів центрального процесора, це не залишиться непоміченим. По-друге, можливості в широкому застосуванні юніт-тестів - пристрій автоматичної перевірки практично кожного методу в додатку. Якщо якась частина додатка перестала працювати коректно внаслідок будь-яких змін коду, нова версія просто не збереться, а програміст відразу побачить причину. По-третє, доступні широкі можливості в інтеграції систем віддаленого моніторингу помилок: в кожен нативний проект вбудовується відповідний функціонал, який дозволяє побачити помилку і її причину, що виникла на пристрої будь-якого користувача.

5. Повна підтримка з боку магазинів додатків App Store і Google Play . Обидві компанії, Apple і Google, зацікавлені в тому, щоб користувачі отримували максимально позитивний досвід при використанні додатків на відповідних платформах, який можливий на поточний момент. Це означає, що програма має виглядати максимально якісно (якщо у екрана висока розподільча здатність, а зображення розпливчасті, в App Store додаток просто не пропустять), має працювати настільки швидко, наскільки це можливо (якщо додаток відображає невеликий список елементів за 20-30 секунд, його так само не пропустять), і взагалі все повинно бути красиво і зручно. Якщо якісь з цих параметрів занадто низькі або взагалі не виконані, програма не пропустять в магазин. Якщо ж вони не на висоті, чого домогтися за кросплатформними технологіями вкрай складно, а часто і неможливо в принципі, то ваш додаток ніколи не буде розглянуто відповідними компаніями для розміщення в спеціальних рекламних розділах (Featured). Серед додатків, які

перебувають під Featured - Розділ і App Store, і Google Play, немає жодного, зробленого за допомогою кросплатформених технологій, за винятком ігрових проектів, в яких інтерфейс не є системним [2].

1.2 Порівняльний аналіз платформ для створення кросплатформних додатків

Як правило, вихід будь-якого бізнесу в інтернет протікає за наступним сценарієм: спочатку компанія запускає сайт, потім його адаптують під мобільні пристрої, і, якщо спостерігається приріст трафіку, з'являється сенс закріпитися серед власників мобільних ґаджетів, і компанія випускає додаток.

Порівнювати мобільний сайт і додаток немає сенсу - другий однозначно виграє за рахунок широти своїх можливостей і чуйного інтерфейсу, взаємодіяти з яким через телефон або планшет набагато комфортніше. Крім того, додаток може працювати без постійного підключення до інтернету [3].

Незалежно від того, на чому побудований ваш бізнес - на продажах, надання послуг або просвітницькій діяльності, сьогодні неможливо не враховувати час, який люди проводять перед екранами мобільних пристроїв.

Виходячи з вище сказаного проведемо порівняння популярних кросплатформних фреймворків.

Кросплатформні фреймворки React Native, PhoneGap, Xamarin, Flutter, Unity на сьогоднішній день займають 80% ринку кросплатформної розробки для мобільних пристроїв [4].

У таблиці нижче представлені основні характеристики для кожного фреймворка :

	React Native	PhoneGap	Xamarin	Unity
мови	JavaScript, HTML5, CSS3	JavaScript, HTML5, CSS3 і нативні мови (Java, Objective-C, C #)	C #, Xaml	C #, UnityScript , Boo
Підтримувані пллатформи	iOS , Android , Windows	Android , iOS , Windows Phone , Blackberry , WebOS , Symbian , Bada , Ubuntu , Firefox OS	iOS, Android, Windows Phone and Windows 8 / RT, Tizen	Android , iOS , Windows
ціни	Безкоштовна версія xCode : безкоштовно Android Studio : безкоштовно	Платна версія: від 9.99 $ Безкоштовна версія: доступна	Xamarin Studio Community : безкоштовно	Personal Edition: безкоштовно Professional Edition: від 75 $ в місяць
Open source	+	+	-	-
UI	Web	Web	Native	UI Canvas

Таблиця 1.1. Порівняння кроссплатформенних фреймворків

1.2.1 React Native

Це JS- фреймворк, заснований на JS і React - JS-бібліотеці для створення UI.

Технологія дуже перспективна, але відносно молода, тому платформа в деяких місцях ще не дороблена. Версія для Android з'явилася пізніше, тому для iOS -додатків поки є більше компонентів. Також варто враховувати, що при розгортанні додатків на пристрій користувача потрапить весь JS, тому на рівні презентації не варто тримати секретну бізнес-логіку. Можна сказати, що зараз React Native можна використовувати для швидкого прототипування мобільних версій ваших веб додатків. Причому якщо веб додаток вже написано на ReactJS , то швидкість перенесення зростає в рази.

React Native представляє можливість використовувати нативні функції мобільного пристрою по роботі з:

- акселерометром,
- камерою,
- компасом,
- контактами,
- файловим сховищем,
- геолокації,
- базою даних,
- подіями, повідомленнями, медія та ін.

Якщо програма не виходить за рамки даних пунктів, то швидкість розробки з використанням фреймворку React Native буде на порядок вище, ніж розробка нативного додатка для кожної з платформ.

Таблиця 1.2. Переваги і недоліки React Native

переваги	недоліки
• Єдиний воркфлоу і інструменти: неважливо, чи працюєте ви на Android - або iOS - версія - все одно використовуєте одні інструменти. • З цієї причини - швидкість і простота розробки. • Обв'язка успадкованого додатка в JS API і гібридні програми: припустимо, у вас вже є готовий додаток для iOS , і ви хочете перейти на React Native. Тоді можна обернути нативні компоненти так, щоб вони були доступні в React Native . Так ви можете поступово переходити на React, і виходить гібридний додаток - половина його нативна , а половина - в React , і кілька успадкованих компонентів - в JS API.	• технологія молода

1.2.2 PhoneGap

PhoneGap дозволяє створювати мобільні додатки використовуючи стандартні веб технології (HTML5, JavaScript і CSS3). В результаті це призвело до швидкого зростання популярності фреймворка, з його допомогою можна обійтися без розробки на таких мовах програмування як: Java for Android , Objective -C for iOS і C #.

PhoneGap Build дозволяє робити збірки для iOS , Android і Windows Phone одночасно, без необхідності встановлювати будь-які SDK tools . Але що більш важливо, цей сервіс дозволяє робити збірки для iOS в хмарі без наявності Mac.
У таблиці 1.3. представлені переваги та недоліки фреймворка PhoneGap.

Таблиця 1.3. Переваги і недоліки PhoneGap

переваги	недоліки
PhoneGap має просте API, що дозволить легко почати розробку, для тих хто стикався з HTML, CSS і JavaScript .	Інтерфейс візуалізується за допомогою вбудованого браузера. Це створює труднощі в отриманні зворотного зв'язку в порівнянні з нативним додатком
Можливість використання будь-яких існуючих JavaScript бібліотек (JQuery , Prototype , Sencha Touch)	Часто існуючі плагіни виявляються застарілими, тому іноді доводиться писати свої.
Підтримка всіх мобільних платформ	

1.2.3 Xamarin

Xamarin дозволяє створювати одну єдину логіку додатка із застосуванням C # і .NET.

Функціонально платформа Xamarin представляє ряд субплатформ . Ці субплатформи відіграють велику роль - через них додатки можуть направляти запити до прикладних інтерфейсів на пристроях. Визначається візуальний інтерфейс, прив'язується логіка на C #, і все це буде працювати на Android, iOS і Windows Phone .

У таблиці 1.4. представлені переваги та недоліки фреймворка Xamarin.

Таблиця 1.4. Переваги і недоліки Xamarin.

переваги	недоліки
• Велике співтовариство. • Розробники можуть використовувати TestCloud для тестування додатків автоматично. • Якщо ви вже знайомі з C # і .NET то вам не потрібно буде витрачати багато часу на вивчення кількох нових фреймворків . • Додатки під різними системами будуть виглядати дуже схоже.	• Деякі інтерфейсні патерни важко реалізувати на monodroid і дуже важко на monotouch . • Виникають проблеми з боку платформи mono , monotouch і monodroid . Ваша програма повинна задовольняти особливим вимогам стабільності. • Android сторінки неможливо розташувати як частина вже існуючого Activity/Fragment.

• Можна повторно використовувати вже написаний код. • Динамічна верстка для iOS в багато разів простіша, ніж використання constraints в ручну.	

1.2.4 Telerik AppBuilder

Однією з основних причин використовувати AppBuilder є повноцінна онлайн IDE. Вона дозволяє створювати, тестувати і навіть публікувати гібридні програми з будь-якого комп'ютера або мобільного пристрою, без необхідності в його завантаженні.

Можливість створювати iOS додатки працюючи на Windows або Linux ще одна перевага.

І наостанок, приналежність AppBuilder до Telerik Platform дає вам можливість користуватися такими функціями як: аналітика, спливаючі повідомлення, авторизація користувачів і хмарним сховищем.

Таблиця 1.5. Переваги і недоліки Telerik AppBuilder.

переваги	недоліки
• Telerik надає плагіни Visual Studio і Sublime Text для AppBuilder .	• невелике співтовариство

• AppBuilder пропонує швидкий спосіб імпорту плагінів Cordova . • Повноцінна онлайн IDE. • Легкий у використанні і вивченні	

1.2.5 Unity

Мультиплатформовий інструмент для розробки 2D і 3D додатків та ігор Unity , також один з кращих інструментів для демонстрації 3D контенту. Створені за допомогою Unity програми працюють під операційними системами Windows , OS X, Linux, Android, Apple iOS, Windows Phone, BlackBerry, а також на ігрових приставках Wii , PlayStation 3 і Xbox 360.

Таблиця 1.6. Переваги і недоліки Unity

переваги	недоліки
• Відмінний варіант для створення мобільних ігор для цілого ряду пристроїв. • 3D-движок дає високоякісні результати без будь-яких складних конфігурацій. • Є багато хороших безкоштовних плагінів.	• UI і складність у використанні для новачків. • Вихідний код недоступний. • Компілятори Unity не оптимізовані для ARM процесорів на деяких мобільних пристроях.

• Unity дозволяє розробнику зробити свої власні шейдери і змінити шлях, яким Unity візуалізує гру.	

1.2.6 Qt

Qt бібліотека для створення кросплатформних віконних додатків на C ++. Qt варто розглядати не стільки як набір класів для створення GUI, а скоріше, як повноцінний інструментарій класів на всі випадки життя. Є можливість розробляти програми не тільки на C ++, але і мовою QML, сильно схожим з JavaScript . Це особлива галузь розвитку Qt , спрямована на швидке прототипування і розробку мобільних додатків.

Таблиця 1.7. Переваги і недоліки Qt

переваги	недоліки
• Qt має безліч хороших інструментів які допоможуть в розробці, наприклад: IDE QT Creator , Qt Designer і code profiling. Він має бібліотеки, що містять інтуїтивно зрозумілі API інтерфейси для елементів, таких	• Qt складний для початківців

як мережі, анімації та багато іншого.	

1.2.7 Appcelerator Titanium

Titanium - це повністю відкрита платформа для розробки, розгортання, поширення, і, в кінцевому підсумку, для виконання веб-додатків. Appcelerator Titanium дозволяє створювати мобільні додатки на JavaScript, HTML і CSS.

Ви можете створювати сучасні, а головне - нативні додатки, використовуючи будь-яку популярну на сьогоднішній день операційну систему: Windows , GNU / Linux або MacOS X.

Додатки, створені за допомогою даного SDK будуть дійсно нативними . Контролер навігації на Андроїд буде виглядати звично і не так як на iOs . Причому не тільки вид, але і сам код програми буде теж нативний . Це до речі не заважає вам створювати і класичний WebView і наповнити його бажаним web контентом [5].

Таблиця 1.8. Переваги і недоліки Appcelerator Titanium.

переваги	недоліки
• JavaScript дозволяє легко розробляти програми без використання мов платформи. • Appcelerator дозволяє робити аналітику в режимі реального часу	• Є затримки при запуску програми через завантаження бібліотеки • Важко створювати складні додатки, так як використання JavaScript негативно позначається на продуктивності

• Використання native API дасть більш високу продуктивність для додатків, які не дуже великі.	додатків.

1.3 Технологія розробки кросплатформного додатку за допомогою React Native

Де б ви не знаходилися, куди б ви не відправилися, всюди люди використовують мобільні пристрої для зв'язку з рідними і близькими, фотографування і розміщення знімків в соціальних мережах, пошуку місця розташування ресторану або перегляду заголовків новин. Мобільні пристрої мають безліч форм і стилів. Мобільні телефони працюють на різних операційних системах, таких як Apple iOS , Google Android і Research In Motion Blackberry. Деякі мають великі екрани, фізичні клавіатури і працюють в мережах 3G, 4G або WiFi. Мобільні телефони можуть також мати датчики прискорення, розташування і навіть платежів. Деякі з цих пристроїв - навіть не телефони; це планшети з більшими екранами і підключення до мережі тільки для обміну даними [6].

Незважаючи на всі відмінності, мобільні пристрої схожі один на одного тим, що всі вони виконують мобільні додатки. Мобільні додатки можна розділити на два типи:

- Вбудовані додатки

Встановлені на пристрій вбудовані додатки є бінарні виконувані програми, створені з використанням пакетів розробки ПО (software development kit - SDK)

і поширювані через сховища додатків. SDK існують для кожної мобільної операційної системи і, на жаль, розрізняються між собою.

Наприклад, щоб створити додаток для iOS , необхідно встановити iOS SDK і засоби розробки, а написання коду виконувати на мові програмування Objective -C. Android - додаток розробляється за допомогою Android SDK і пишеться на Java . Виходить, що для створення мобільного застосування необхідно знати кожен SDK і використовувати підтримуваний мову програмування. На вивчення SDK для кожної платформи потрібно витратити чимало часу, тому розробка мобільних додатків є досить складною задачею.

- Web -додатки

Web -додатки, що завантажуються в мобільний Web -браузер, відрізняються від вбудованих тим, що їх код пишеться з використанням Web - технологій (HTML, JavaScript і CSS), що не залежать від операційної системи пристрою. Тому зникає необхідність у вивченні різних мов програмування для кожного пристрою. HTML і JavaScript знайомі Web розробнику зі створення Web -сторінок для настільних браузерів. У більшості випадків мобільні Web -браузери можуть візуалізувати ті ж самі Web -сторінки, але Web - сайти часто надають мобільні версії з меншим обсягом інформації і більш швидким завантаженням (через меншого розміру екрану і повільнішою мережі).

Для запуску Web -додатків користувач вводить URL-адресу в мобільний Web -браузер. Після завантажується Web -сторінка, яка є точкою входу в Web -додаток. Web -додатки не поширюються через сховище додатків; вони є звичайними посиланнями, які можна включити в інші Web - сторінки, електронні повідомлення або навіть записати на папері.

Вбудовані і Web -додатків мають свої переваги і недоліки, будучи причиною багатьох дискусій про те, які програми краще. Вирішити даний спір можуть гібридні програми, які намагаються об'єднати переваги обох типів мобільних додатків.

Гібридні додатки, як і Web -додатків, програмуються з використанням Web -технологій, але пакетуються як вбудовані додатки. Гібридні додатки можна написати відразу для декількох мобільних операційних систем з використанням мови програмування, знайомого багатьом розробникам. Оскільки гібридний додаток насправді є вбудованим, ви отримуєте доступ до функцій пристрою з JavaScript , що поки недоступно для Web - додатків. Гібридні додатки можна поширювати і встановлювати через сховища додатків, подібно вбудованим [7].

React Native - це популярний набір інструментальних засобів для створення додатків. Він являє собою мобільну інфраструктуру з відкритими початковими кодами, що містить JavaScript - інтерфейси для доступу до функцій різних пристроїв, наприклад, акселерометра і камери.

Нижче описується розробка гібридного мобільного Android - додатки за допомогою наборів інструментальних засобів React Native , використовуючи емулятор Android і засоби тестування додатків.

Для того щоб приступити безпосередньо до розробки програми, необхідно переконатися чи ваше робоче місце таким вимогам:

- Операційна система Windows , OS X або Linux .
- Java Development Kit (JDK) 5 або JDK 6 (JRE недостатньо).
- Інтегроване середовище розробки Android Studio , WebStorm або інша будь-яка .
- Android SDK і платформи (r12 або більш нової версії).

Після того, як ми переконалися, що наше робоче середовище відповідає вимогам, приступимо до налаштування середовища розробки [8].

Для налаштування середовища розробки необхідно виконати наступні дії:

1. Встановити JDK.
2. Завантажити Android SDK.
3. Встановити необхідні Android -платформи.
4. Створити новий пристрій Android Virtual Device (AVD).

1.4 Вибір основного шаблону проектування

Основні шаблони проектування мобільних додатків зазвичай диктуються окремими технологічними платформами. Компанії-виробники мобільних операційних систем створюють власні API для сторонніх розробників згідно з визначеними шаблонами, чекаючи, що розробляються додатки будуть створюватися з урахуванням особливостей цільової платформи.

Одним з найпопулярніших на даний момент шаблонів є Model-View-Controller . Шаблон MVC описує простий спосіб побудови структури додатка, метою якого є відділення бізнес-логіки від призначеного для користувача інтерфейсу. В результаті, додаток легше масштабується, тестується, супроводжується і реалізується.

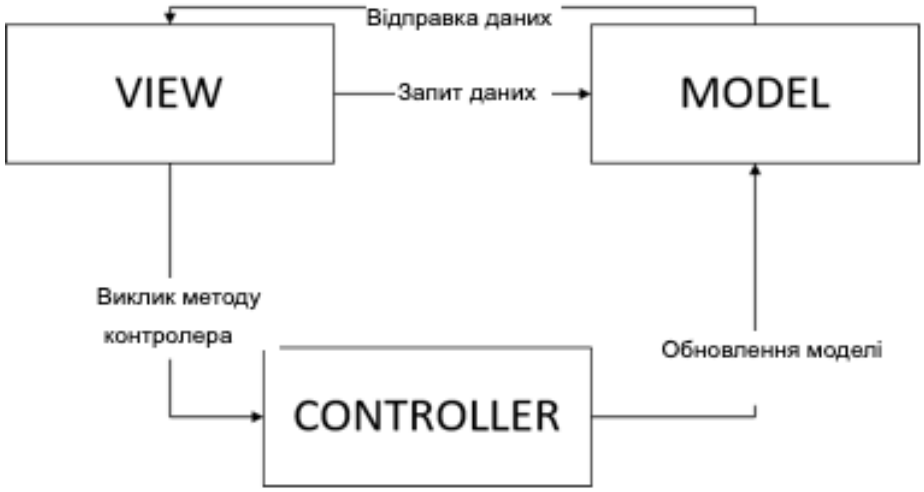

Рис.1.1. Шаблон MVC

В архітектурі MVC модель надає дані і правила бізнес-логіки, уявлення відповідає за користувальницький інтерфейс, а контролер забезпечує взаємодію між моделлю (model) і поданням (view) [9].

Типову послідовність роботи MVC-додатки можна описати таким чином:

1. При вході користувача на веб-ресурс, скрипт ініціалізації створює екземпляр додатку і запускає його на виконання. При цьому відображається вид, наприклад головної сторінки сайту.

2. Додаток отримує запит від користувача і визначає контролер і дію. У разі головної сторінки, виконується дія за замовчуванням (index).

3. Додаток створює екземпляр контролера і запускає метод дії, в якому, наприклад, містяться виклики моделі, що зчитують інформацію з бази даних.

4. Далі, дію формує уявлення з даними, отриманими з моделі і виводить результат користувачеві.

Модель - містить бізнес-логіку додатка і включає методи вибірки, обробки і надання конкретних даних, що часто робить її дуже товстою, що цілком нормально.

Модель не повинна безпосередньо взаємодіяти з користувачем. Всі змінні, що відносяться до запиту користувача повинні оброблятися в контролері.

Модель не повинна генерувати HTML або інший код відображення, який може змінюватися в залежності від потреб користувача. Такий код повинен оброблятися в видах.

Одна і та ж модель, наприклад: модель аутентифікації користувачів може використовуватися як в призначеній для користувача, так і в адміністративній частині програми. Тоді можна винести загальний код в окремий клас і успадковуватися від нього, визначаючи в спадкоємців специфічні методи [10].

Вид - використовується для задання зовнішнього відображення даних, отриманих з контролера і моделі. Види включає HTML-розмітку і невеликі вставки PHP-коду для обходу, форматування і відображення даних. Не повинні безпосередньо звертатися до бази даних. Цим повинні займатися моделі. Не повинні працювати з даними, отриманими із запиту користувача. Це завдання має виконувати контролер. Може безпосередньо звертатися до властивостей і методів контролера або моделей, для отримання готових до висновку даних. Види зазвичай поділяють на загальний шаблон, що містить розмітку,

загальну для всіх сторінок і частини шаблону, які використовують для відображення даних , що виводяться з моделі або відображення форм введення даних.

Контролер - сполучна ланка, що з'єднує моделі, види і інші компоненти в робочий додаток. Контролер відповідає за обробку запитів користувача. Контролер не повинен містити SQL-запитів. Їх краще тримати в моделях. Контролер не повинен містити HTML і іншої розмітки. Її варто виносити в види. У добре спроектованому MVC- додатку контролери зазвичай дуже тонкі і містять тільки кілька десятків рядків коду. Логіка контролера досить типова і велика її частина виноситься в базові класи. Моделі, навпаки, дуже товсті і містять велику частину коду, пов'язану з обробкою даних, так як структура даних і бізнес-логіка, що міститься в них, зазвичай досить специфічна для конкретного додатка [11].

Іншим шаблоном є Model-View-ViewModel . Основною особливістю шаблону MVVM є поєднання в собі функціональних переваг вже класичного для мобільних (і веб-) додатків підходу MVC (Model-View-Controller) з перевагами зв'язування даних (data-binding), тобто зав'язки атрибутів View на окремі властивості моделі через створення так званих Binding , які часто також передбачають внутрішні конвертери. Такий підхід дозволяє мінімізувати код, пов'язаний з частиною контролера з MVC.

Шаблон MVVM широко застосовується в програмних продуктах компанії Microsoft (WPF, Silverlight), однак, варто помітити, що сам по собі підхід незалежний від платформи. В рамках цієї роботи зроблена спроба відтворення даного шаблону проектування в поєднанні з описаним раніше шаблоном MVC з використанням власних інструментів зв'язування даних. Основні компоненти:

1. View - безпосереднє графічне представлення у вигляді UI елементів на екрані пристрою. Зі свою роль збігається з роллю з MVC.

2. Model - модель подання будь-якої предметної області, бізнес- логіка, дані. За свою роль збігається з роллю з MVC.

3. ViewModel - абстрактне уявлення View , що відкриває атрибути для зв'язки, а також описує можливі події, які будуть генеруватися при взаємодії користувача з додатком.

4. DataBinding (Binder) - покажчик-зв'язка між даними і безпосереднім їх поданням на графічному інтерфейсі. Зв'язки можливі завдяки описаним в ViewModel властивостями, до яких Model може мати доступ безпосередньо, минаючи посередників.

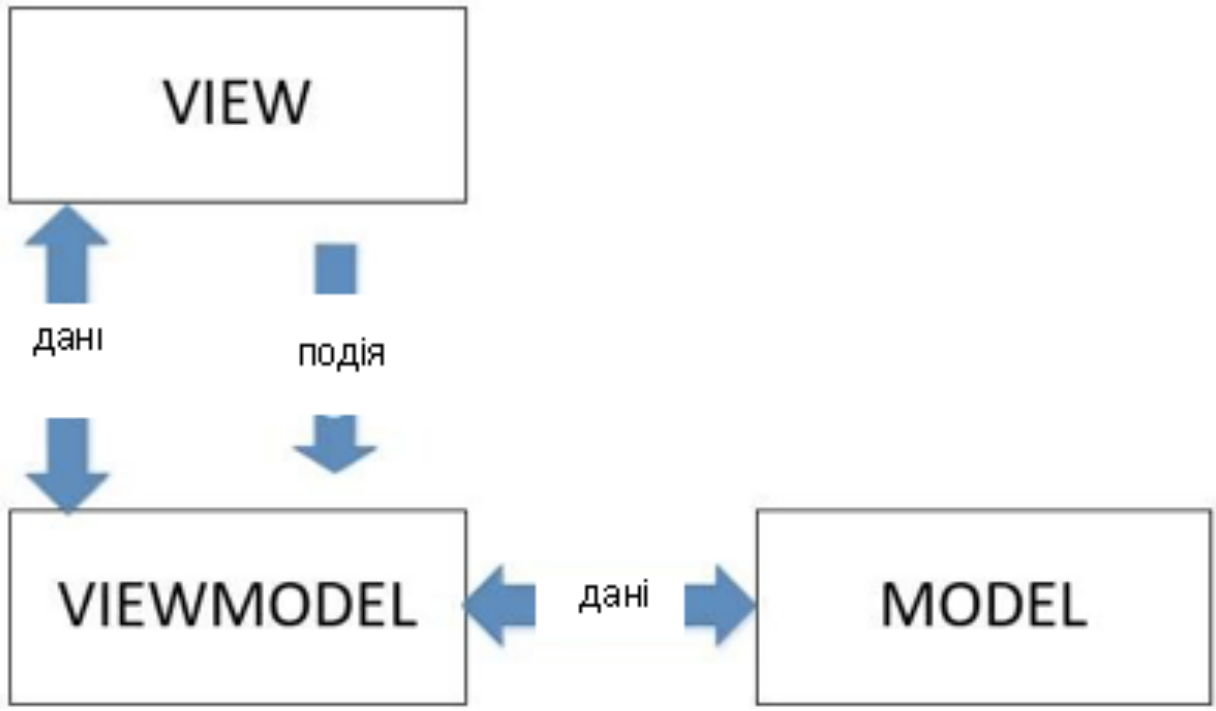

Рис.1.2. Шаблон MVVM

Виходячи з вище написаного стало ясно, що в обох шаблонах присутні свої мінуси і плюси. І було прийнято рішення комбінування двох шаблонів MVC і MVVM [12].

Основними сутностями є View, Page, Action, Procedure, EntityDescription, Binding.

View . Шаблон подання графічного інтерфейсу користувача в рамках клієнтської програми. Поточна реалізація містить наступні підкласи: TextView , ImageView , Button , WebView . Архітектурне рішення дозволяє додавати нові підкласи в модель, що в результаті автоматично

масштабується в веб-додатку - проектувальника інтерфейсу. Основні властивості класу описують розташування на екрані, розміри, колір фону. Властивості кожного з підкласів описують можливі параметри налаштування відображення, а також надають інформацію про атрибути, які доступні для встановлення зв'язків (data-binding) з моделлю.

Таблиця 1.9. Параметри відображення і атрибути, призначені для зв'язування

клас	параметри	атрибути для зв'язування
View - прямокутна область	Frame - розташування на екрані, включає в себе: origin - x, y; size - width, height Background color - колір фону	
ImageView - компонент відображення зображень	Успадковуються з View ContentMode - варіанти відображення зображення: • scale – розтягування на всю площу view • fit – розтягування зі збереженням відносини сторін	Image_url – URL картинки для відображення image - статичне зображення (placeholder)

TextView - компонент відображення текстових даних	Успадковуються з View TextAlign - розташування (вирівнювання) тексту: • center - по центру • left - по лівому краю • right - по правому краю TextStyle - стиль (накреслення) тексту: • regular - звичайний • bold - напівжирний • light - тонкий • italic - з нахилом TextSize - розмір (кегль)	text - текст для відображення
Button - компонент "кнопка", доступна для натискання користувачем	Успадковуються з TextView	title - текст для відображення на кнопці action - реакція на натискання кнопки

Page . Являє собою відображення окремого екрану (сторінки) в рамках клієнтського мобільного додатка. Є поданням частини ViewModel в рамках MVVM. Архітектура розробленої технологічної платформи передбачає

додавання нових типів екранів без значних витрат. На даний момент момент реалізовані наступні типи екранів:

• Custom - довільний екран з відображенням шаблону. Передбачає зв'язку на властивість view , де необхідно вказати створений раніше шаблон відображення, а також зв'язку на властивість item - контекст даних для відображення.

• List - екран для відображення списку елементів. Передбачає зв'язку на властивість item - шаблон відображення рядка елемента і зв'язку на властивість items - контекст (масив) даних для відображення в списку.

Разом, в рамках об'єкта Page передбачається два типу покажчиків зв'язок: контекст (data-context) і відображення (view). Перший тип зв'язку визначається одним з двох варіантів постачальників даних: виконання заданої процедури (об'єкта Procedure) або отримання заданих даних згідно з описом об'єкта EntityDescription . Другий тип - view - посилання на шаблон відображення, створеного за допомогою модуля опису в рамках створеного веб-програми.

Action. Опис реакції елементів інтерфейсу на взаємодію користувача. Наприклад, натискання кнопки, або успішне введення даних. Є складовою частиною зв'язки view з моделлю, спрямованої на генерацію подій і реакцію на них. У цьому проекті передбачені три типи реакцій на події:

1) відкриття внутрішньої URL - відкриває заданий URL у внутрішньому (не залишаючи додатки) веб-браузері;

2) відкриття зовнішньої URL - відкриває заданий URL в стандартному браузері мобільної операційної системи;

3) перехід на інший екран - виробляє перехід на заданий Page із зазначенням payload - прикріплених даних, які будуть розширювати контекст відкривається сторінки; з точки зору структури даних, payload - це набір пар ключ-значення, де значення може бути посиланням на будь-яке поле поточного контексту даних.

Procedure . Складова частина зв'язки ViewModel з Model . Є одним з джерел контексту даних (Data Context), використовуваного при створенні зв'язок у

об'єктів Page . Завантажений програмний код повинен мати основну функцію з ім'ям func і набором параметрів, заданих при описі процедури. Значення, що повертається повинно бути списком об'єктів (list з dict), чиї атрибути можуть бути використані в покажчиках зв'язках (data-binding) в частинах шаблону відображення. При написанні виконуваного скрипта розробники можуть користуватися розробленим в рамках цієї роботи API. Дані, що поставляються процедурами приймаються як динамічні і не можуть бути доступні без активного інтернет-з'єднання на клієнтському мобільному додатку.

EntityDescription . Аналогічна за призначенням Procedure складова моделі. Основна відмінність полягає в тому, що контент статичний і створюється модератором / адміністратором програми в рамках серверного веб-додатку, а потім синхронізується з клієнтськими додатками за допомогою локального зберігання і кешування на віддаленій стороні. Особливістю цієї моделі є runtime -генерація класів (згідно зі створеним опису), а також створення їх примірників, які зберігаються в локальному сховищі за допомогою стандартних для конкретної платформи методів зберігання об'єктів (в основному - ORM рішення).

Binding . Основна сутність-зв'язка, яка забезпечує з'єднання властивостей View і атрибутів моделі, отриманої через EntityDescription або Procedure , а також використовується для установки зв'язків між об'єктами Page і шаблонами відображення і між об'єктами Procedure і їх аргументами (при їх наявності). Поточна архітектура абстракції передбачає повне перевикористання компонентів зв'язки. У цьому проекті використовуються зв'язку наступних типів:

1. TypeValue - зв'язок між значенням атрибута елемента моделі (data context) і параметром властивості, доступним для зв'язування у елемента відображення (view). Параметрами даного типу зв'язку є:

- view - об'єкт шаблону відображення, на властивість якого буде встановлений зв'язок;

- property - властивість елемента відображення, до якого проводиться зв'язок (наприклад, text у TextView);
- value - значення, яке буде приймати вказану властивість; при встановленні значення може бути використаний будь-який елемент поточного контексту даних (payload для об'єкта Page або атрибут моделі даних).

2. TypeTemplate - зв'язок між значенням атрибута об'єкта Page , доступним для зв'язування з шаблоном, і об'єктом шаблону відображення (наприклад, item у об'єкта Page з типом List). Параметрами даного типу зв'язку є:

- property - ім'я атрибута об'єкта Page , для якого здійснюється зв'язок;
- view - об'єкт шаблону відображення, створений за допомогою конструктора опису в рамках розробленого веб додатки.

3. TypeProcedure - зв'язок між об'єктом Procedure , як елементом контексту даних, і атрибутом об'єкта Page , доступним для вказівки контексту даних (наприклад, властивість items об'єкта Page з типом List). Параметрами даного типу зв'язку є:

- procedure - процедура, яка використовується для генерації об'єктів контексту даних;
- property - ім'я атрибута об'єкта Page , для якого здійснюється зв'язок.

4. TypeArgument - використовується в комбінації з TypeProcedure типом зв'язку. Містить інформацію про значеннях аргументів процедури, що передаються при її виклику в контексті окремого об'єкта Page.

Параметрами даного типу зв'язку є:

- procedure - процедура, для якої призначений даний аргумент;
- argument - аргумент, значення якого буде вказано в зв'язку;
- value - значення аргументу, вказане довільно або з використанням елементів поточного контексту (заданого payload об'єкта Page).

5. TypeAction - зв'язок між елементом шаблону відображення, і варіантом реакції на подію, яку він генерує при призначеному для користувача взаємодії (наприклад, натискання на кнопку - елемент відображення Button). Параметрами даного типу зв'язку є:

- action - об'єкт Action , що містить в собі інформацію про варіант реакції на подію і значення, яке буде використано для цього переходу (або payload для відкриття сторінки);
- view - елемент відображення, який генерує подію в результаті користувальницького взаємодії.

6. TypeEntity - зв'язок між описом моделі даних EntityDescription і атрибутом об'єкта Page, для якого доступна установка зв'язку контексту даних (наприклад, items у об'єкта Page з типом List). При установці зв'язку надалі значенням атрибута об'єкта Page буде список об'єктів, що відповідають опису моделі EntityDescription і створених в рамках модуля розробленї програми. Параметрами даного типу зв'язку є:

- entityDescription - опис моделі даних, відповідно до якого будуть передаватися об'єкти;
- query - опціональний запит на мові SQL (фільтр - WHERE), який буде виконаний при отриманні даних з локального сховища [13].

Висновок до розділу 1

З технічної точки зору і з точки зору якості створюваного інтерфейсу нативна розробка має набагато більше плюсів. Однак є сфери, в яких кросплатформені технології є виправданими: це ігровий сектор і тестові проекти.

Сучасні ігри пишуться в переважній більшості на кросплатформних технологіях, це сильно прискорює розробку без шкоди для якості, тому що в цьому випадку використовуються спеціальні графічні фреймворки (найпопулярніший - Unity 3D). Якщо якийсь проект потрібно зробити швидко для проведення будь-яких тестів, при цьому ситуація вимагає роботи проекту саме на декількох платформах одночасно, кросплатформна реалізація може бути оптимальним рішенням.

Якщо проект не є ігровим, спрямований на довгостроковий розвиток і вимагає позитивного враження від користувачів, нативна розробка залишається більш відповідним варіантом.

Що стосується фреймворків: немає ідеального рішення, кожен фрейворк має свої переваги і недоліки. Все залежить від роду завдань, які запускає вашу програму. Для більш простих додатків, на мою думку, більше підходить React Native. А для більш серйозної розробки Xamarin , але навіть з Xamarin краще поєднувати нативную розробку для більшості елементів призначеного для користувача інтерфейсу.

Також було виявлено, що в обох шаблонах проектування мобільного додатку присутні свої мінуси і плюси. І було прийнято рішення комбінування двох шаблонів MVC і MVVM. У даній комбінації основними сутностями є View, Page, Action, Procedure, EntityDescription, Binding. Серіалізована схема буде вхідними даними для клієнтського мобільного додатку, який буде будувати свою структуру і зміст контенту відповідно до опису схеми.

В рамках розробленої технологічної платформи створення мобільних додатків основоположним об'єктом зв'язків є Binding - сутність, що відповідає за присвоєння значень між частинами контексту даних і елементами відображення з опціональною фільтрацією одержуваних об'єктів (при вказівці умов для запиту до локальної бази даних програми або при вказівці параметрів виконуваної процедури), процедурами і значеннями їх аргументів [14].

2. ОГЛЯД МЕТОДІВ МАШИННОГО НАВЧАННЯ ТА ЇХ АНАЛІЗ

У багатьох аналогах додатків для визначення місця розташування використовуються різні алгоритми машинного навчання. Звертаючи увагу на високу точність таких алгоритмів, має сенс і в додатку даної роботи використовувати машинне навчання. Для цього потрібно дати визначення машинного навчання і розглянути найбільш ефективні алгоритми, щоб зробити вибір на користь кращого.

Машинне навчання - великий підрозділ штучного інтелекту, математична дисципліна, яка використовує розділи математичної статистики, чисельних методів оптимізації, теорії ймовірностей, дискретного аналізу. Основною метою машинного навчання є повна або часткова автоматизація вирішення складних нелінійних професійних завдань в різних областях людської діяльності.

Алгоритми машинного навчання діляться на різні групи в залежності від часу навчання (попереднє навчання або навчання в процесі), а так само з методики навчання (навчання з учителем, без учителя, часткове підкріплення).

Існують різні підходи до створення алгоритмів для машинного навчання. Має сенс розглянути докладніше алгоритми, які показали високу точність і продуктивність .

2.1 Загальні поняття машинного навчання

Машинне навчання (Machine Learning) - великий підрозділ штучного інтелекту, що вивчає методи побудови алгоритмів,здатних навчатися.

Дані в математиці - деяка сукупність об'єктів . Що таке об'єкт в кожній предметної області позначається індивідуально. Наприклад, об'єктами можуть бути пацієнти в поліклініці, аудіо файли, текст, клієнт банку, код ДНК і т.д. Для математики головна вимога, яка висувається до об'єкта, це можливість описати

його якоюсь сукупністю змінних, які зможуть охарактеризувати цей об'єкт. Машинне навчання ж полягає в тому, що змінні які охарактеризує сам об'єкт ми зможемо відокремити на спостережувані і приховані. Спостережувані змінні - це змінні, яким ми явно можемо задавати значення для будь-якого об'єкта. Приховані (або латентні) - це змінні, які не можуть бути виміряні в явному вигляді, а можуть бути тільки виведені через математичні моделі з використанням спостережуваних змінних. Вважається, що між цими двома типами змінних повинна бути якась залежність один від одного. Мета машинного навчання знайти таку зв'язок.

Якщо є строгі математичні моделі, які пов'язують спостережувані і приховані складові, машинне навчання не має сенсу. Наприклад, в строгих моделях з класичної фізики. Якщо відомі спостережувані змінні маса і прискорення, то щоб визначити приховану змінну - силу, машинне навчання застосовувати не має сенсу, досить використовувати другий закон Ньютона.

Однак в задачах які зустрічаються на практиці в повсякденному житті, таких строгих математичних моделей не буває і зв'язок між змінними не зовсім очевидний. Але можливо створити досить велику колекцію даних для навчальної вибірки - набору об'єктів, у яких відомі, як спостережувані, так і приховані складові.

Класичним прикладом машинного навчання є завдання банківського кредитного скорингу - система оцінки кредитних ризиків особи, заснована на численних статистичних методах. Найчастіше використовується в споживчому кредитуванні на не дуже великі суми. Припустимо є банк в якому його клієнти хочуть отримати кредит. Напрошується питання, кому можна видавати кредит а кому ні. Спостережуваними змінними тут будуть характеристики клієнтів з заповнених ними анкетами: стать, вік, освіта, рівень доходів, склад сім'ї і т.д. Ці змінні можна дуже просто "виміряти". В якості латентної складової в найпростішому варіанті виступає бінарна величина - клієнт поверне гроші в банк або не поверне. У більш складних варіаціях передбачається, що оцінюється ризик повернення кредиту, як якась ймовірність того, що кредит не повернуть і

тоді з'являються економічні моделі прийняття рішень для видачі кредиту на порядок складніше.

Для навчаючої вибірки тут виступає сукупність клієнтів з історії. Припустимо, в банку було 400 клієнтів з яких 350 кредит повернули. Тобто це та вибірка клієнтів у яких прихована або латентна змінна приймала одне значення, а 50 клієнтів - це вибірка, у якій прихована приймала інше значення. Цього цілком достатньо, для складання завдання машинного навчання і можна пробувати математично знаходити закономірності між спостережуваними і прихованими складовими, сподіваючись, що вони допоможуть визначити для нових клієнтів варто видавати їм кредит або не варто.

У поточному розділі будуть наведені найбільш часто використовувані класичні методи машинного навчання.

Для всіх методів буде загальне те, що вони будуть перебувати в просторі ознак R^n, де n - кількість ознак, а це якраз ті самі спостережувані і латентні компоненти.

2.2 Метод к-найближчих сусідів

Суть методу найближчих сусідів полягає в тому, що об'єкт належить до класу, який був найбільш використовуваний між сусідами даного класу. Тобто буквально кажучи розташований найближче до певної сусідської групі.

Для навчання класифікатора використовуються набір об'єктів. Кожен об'єкт з вибірки зіставляється з одним або більше об'єктів з набору класів і представляється у вигляді точки в n-вимірному просторі, де – n кількість ознак, які використовуються для класифікації. Новий невідомий об'єкт можна порівняти з одним із класів наступним чином - знайти K найближчих сусідів з вибірки з навчальними даними в заданому просторі. Далі потрібно знайти до якого з класів відповідає більша частина найближчих об'єктів з навчальної

вибірки - можна зробити висновок, що до даного класу відноситься і новий невідомий елемент. Оптимальне число K в основному підбирають емпіричним шляхом. При збільшенні K, алгоритм зменшує залежність помилкових похибок в даних, але і поділ на класи стає менш грубим.

Переваги даного алгоритму в тому, що він тривіальний в реалізації і класифікацію просто інтерпретувати шляхом пред'явлення користувачеві декількох найближчих сусідів.

Професіоналам з різних областей просто доступний алгоритм і логіка функціонування, яка базується на знаходженні схожих об'єктів. Якщо ж навчальна вибірка збільшується, то з'являються складності в реалізації, так як необхідні великі обчислювальні потужності.

Алгоритм стабільний до помилкових попаданням записів в некоректні для них класи, так як ймовірність попадання такого запису в число k – найближчих сусідів не велика, а значить невелиий буде вплив на результат визначення класу для нового невідомого.

Недоліки з'являються при реалізації, не дивлячись на простоту рішення доводиться неефективно розподіляти пам'ять, що є критичним заходом при великих обсягах даних і невиправдано трудомісткі рішення, причина цього необхідність зберігання даних для навчальної вибірки.

Дотримуючись алгоритму пошук найближчого сусіда передбачає порівняння нового невідомого об'єкта з об'єктами з уже категорійованої вибірки. Як відомо, такі порівняння вимагають лінійного числа операцій, що значно ускладнює завдання при великому обсязі даних.

2.3 Метод Байєса

В реальності ймовірність настання якоїсь події – є частота настання даної події, іншими словами відношення кількості настання до загальної кількості наступів при нескінченно великій загальній кількості спостережень.

Байєсівський класифікатор заснований на теоремі, в якій стверджується, що якщо щільності розподілу кожної рубрики відомі, то потрібний алгоритм ми можемо записати в аналітичному вигляді. До того ж сам алгоритм можемо вважати оптимальним, з мінімальною ймовірністю появи помилок.

Часто в реальних задачах щільності розподілу рубрик невідомі і їх потрібно оцінювати за вибіркою з навчальними даними. Тому байєсівський алгоритм може стати не оптимальним, так як, оцінювати щільність по вибірці даних можна тільки з деякою похибкою. Отже, прямо пропорційно залежить те що, чим менше навчальна вибірка, тим вищі шанси підлаштувати розподіл під конкретні документи з вибірки і наштовхнутися на те, що необхідно буде перенавчання.

Байєсова мережа (довіри) - це ациклічний орієнтований граф, в якому кожна вершина (вузол мережі) являє n- значну змінну, дуги позначають існування безпосередніх причинно-наслідкових залежностей між з'єднаними змінними, а сила цих залежностей кількісно виражається у вигляді умовних ймовірностей, зіставлених кожної з змінних.

Байєсовські мережі є чудовим інструментом для опису досить складних процесів і подій з невизначеностями. Основною ідеєю побудови мережі є розкладання складної системи на прості елементи. Для об'єднання окремих елементів в систему використовується математичний апарат теорії ймовірностей. Такий підхід забезпечує можливість будувати моделі з безліччю взаємодіючих змінних для подальшої розробки ефективних алгоритмів обробки даних і прийняття рішень. З математичної точки зору баєсова мережа - це модель для представлення імовірнісних залежностей, а також відсутність цих залежностей.

Привабливість байєсовських моделей полягає в їх високій продуктивності, а також в інтуїтивно зрозумілому поданні у вигляді графа. (Рис. 2.2.).

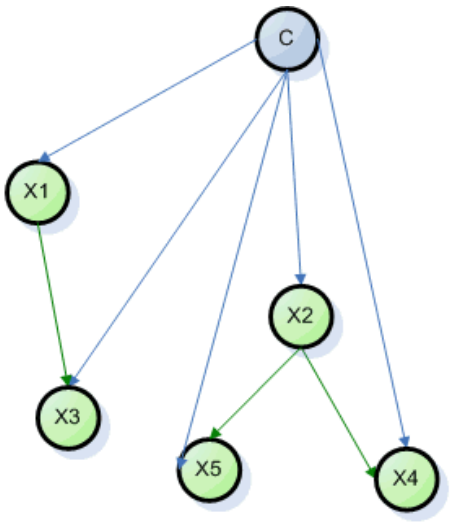

Рис. 2.2. Граф моделі Байєса

Байєсівський підхід є найбільш наочним, базовим і часто використовуваним і не дивлячись на те, що він один з найстаріших алгоритмів на ньому реалізовано досить багато вдалих алгоритмів класифікації.

Використовуючи даний метод можна визначити апостеріорну ймовірність, іншими словами отримати таку ймовірність випадкової події за умови того, що відомі дані отримані в результаті досвіду.

2.4 Дерева рішень

Дерева рішень - це метод, який дозволяє передбачати залежність деяких об'єктів від того чи іншого класу в залежності від відповідних значень однієї або декількох вже відомих змінних. Дерево рішень, подібно до його «прототипу» з живої природи, складається з «гілок» і «листя». Гілки (ребра графа) зберігають в собі значення атрибутів, від яких залежить цільова функція; на листках ж записуються значення цільової функції. Існують також і інші вузли - батьківські і нащадки - за якими відбувається розгалуження, і можна розрізнити випадки.

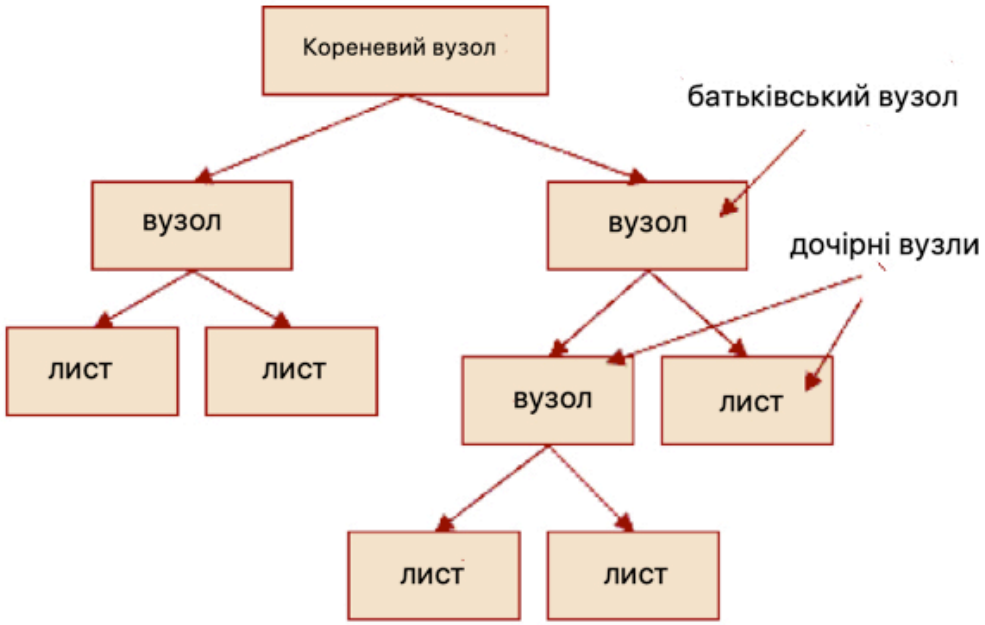

Рис. 2.1. Дерево рішень

За допомогою дерева рішень завдання класифікації виникає тоді, коли залежна змінна приймає дискретні значення.

Мета всього процесу побудови дерева прийняття рішень – створити модель, по якій можна було б класифікувати випадки і вирішувати, які значення може приймати цільова змінна, маючи на вході кілька змінних.

Дані для навчання дерева задаються зазвичай у вигляді таблиці:

Таблиця 2.1 Вхідні дані для навчання дерева

	d_1	d_2	…	d_i	y
D_1	v_{11}	v_{12}	…	v_{1i}	y_1
D_2	v_{21}	v_{22}	…	v_{2i}	y_m
…	…	…	…	….	…
D_n	v_{n1}	v_{n2}	…	v_{ni}	y_n

де $\{ D_i \} \in$ - безліч вже класифікованих данних з їх ознаками $\{ d_i \}$ для кожного, вагою $\{ v_{ij} \}$ і цільовим значенням $\{ y_n \} \in$ y.

Основний алгоритм:

- Спочатку необхідно вибрати атрибут $\{ d_i \}$ - критерій (ознака) подальшого поділу на піддерева, і помістити його в кореневий вузол.
- Потім, з наших прикладів для кожного значення атрибута і (вибираємо тільки ті, для яких $\{ d_j \}$ = і).
- Далі, рекурсивно будуємо дерево прийняття рішень.

Основна проблема, очевидно, криється в першому кроці - на якій підставі вибирається кожен наступний атрибут $\{ d_j \}$? На це питання існує кілька відповідей у вигляді приватних алгоритмів прийняття рішень - головними з яких є алгоритми *ID3* , *C4.5* і *CART* .

Дерева рішень - один з основних і найбільш популярних методів допомоги в прийнятті рішень. Дійсно, побудова дерев рішень дозволяє наочно продемонструвати іншим і розібратися самій в структурі даних, створити працюючу модель класифікації даних, якими б «великими» вони не були.

Переваги методу.

Даний метод не складний в розумінні і інтерпретації для фахівців з інших областей. Люди здатні пояснити результати роботи алгоритму дерева прийняття рішень після невеликого роз'яснення основних принципів і підходів. Метод дерева прийняття рішень по своїй суті використовує модель «білого ящика». Якщо певна ситуація спостерігається в моделі, то її можна пояснити за допомогою булевої логіки. Ця властивість дозволяє ефективно оцінювати моделі методу використовуючи статистичні тести.

Існує можливість роботи з великими даними і великими базами даних без підготовчої роботи.

Недоліки методу.

Проблема отримання оптимального дерева рішень є NP-повною.

Існує проблема «надмірної підгонки». Метод дозволяє створювати складні конструкції, які в підсумку можуть представляти не досить повні дані для прийняття рішень.

Складні концепти можуть бути важко розуміючими по опису моделі.

Для даних, які включають в себе категоріальні змінні з великим набором рівнів, більша інформаційний вага присвоюється тим атрибутам, які мають більшу кількість рівнів, що часто не є коректним рішенням.

2.5 Нейронні мережі

Штучна нейронна мережа (artificial neural network, ANN) математична модель, побудована за образом і подобою нервових клітин живого організму, а зокрема, людського мозку.

Основним прототипом штучної нейронної мережі є біологічна нейронна мережа - сукупність нейронів (структурно-функціональна одиниця нервової системи, яка обробляє і передає інформацію за допомогою хімічних і електричних сигналів.) Головного та спинного мозку центральної нервової системи (ЦНС) і ганглія периферичної нервової системи (ПНС) . Пов'язані або функціонально об'єднані в нервовій системі, вони виконують специфічні фізіологічні функції.

На відміну від біологічної нейронної мережі, яка до сих пір залишається питанням багатьох досліджень і вивчень в області біології, штучна нейронна мережа (ШНМ) являє собою спрощену математичну модель, що складається з вузлів, іменованих штучними нейронами, з'єднаних між собою зв'язками і здатними передавати та отримувати сигнали.

У біологічній нейронній мережі нейрон має відростки нервових волокон двох типів - дендрити, які беруть імпульси, і аксон, по якому нейрон може передавати імпульс. Аксон контактує з дендритами інших нейронів через спеціальні освіти - синапси, що впливають на силу імпульсу. При проходженні синапсу сила імпульсу змінюється в певну кількість разів, зване вагою синапсу. Імпульси, що надійшли до нейрона одночасно за кількома дендритам, підсумовуються. Якщо сумарний імпульс перевищує певний поріг, нейрон збуджується, формує власний імпульс і передає його далі по аксону. Варто відзначити, що ваги синапсів можуть змінюватися з часом, тобто буде змінюватися і поведінка відповідного нейрона.

Штучний нейрон, на відміну від нейрона біологічного, являє собою більш простий елемент-процесор, який отримує на вхід сигнали у вигляді будь-яких параметрів для оптимізації або вихідні сигнали інших нейронів ШНМ через кілька вхідних каналів. Кожен вхідний сигнал проходить через зв'язок, що має певну інтенсивність або вагу Кожен нейрон має якесь порогове значення. При надходженні сигналу обчислюється зважена сума входів, з неї віднімається граничне значення і в результаті виходить величина активації нейрона (постсинаптичний потенціал нейрона, PSP). Сигнал активації перетворюється за допомогою функції активації (передавальної функції) в вихідний сигнал нейрона. (Рис. 2.3.).

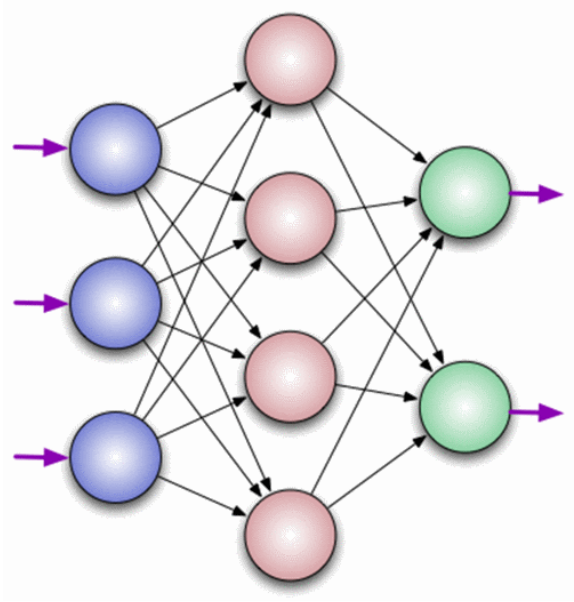

Рис. 2.3. Граф нейронної мережі

Клас задач, які можна вирішити за допомогою нейронної мережі, визначається тим, як мережа працює і тим, як вона навчається. При роботі нейронна мережа приймає значення вхідних змінних і видає значення вихідних змінних. Таким чином, мережа можна застосовувати в ситуації, коли у вас є певна відома інформація, і ви хочете з неї отримати деяку поки не відому інформацію [16].

З точки зору машинного навчання, нейронна мережа являє собою окремий випадок методів розпізнавання образів, а значить і завдання класифікації.

Для вирішення поставленого завдання нейронну мережу необхідно натренувати на підготовлених даних, для яких відомі значення як вхідні $\{ t_i \}$ так і вихідні параметри $\{ y_i \}$. Суть тренування в визначенні таких вагових коефіцієнтів міжнейронних зв'язків $\{ w \}$, які забезпечували б найімовірніший зв'язок відповідей нейронної мережі до вже правильних відповідей.

Переваги методу в тому, що у нейронних мереж дуже висока точність класифікації. А недолік - дуже низька швидкість навчання.

Висновки до розділу 2

Різні методи машинного навчання характеризуються різними властивостями. Серед основних властивостей і характеристик виділимо наступні: точність, масштабованість, інтерпретованість, трудомісткість, швидкість, широта використання, можливість перенавчання.

Під точністю будемо розуміти властивість класифікатора правильно відносити дані до однієї з категорій. Часто точність оцінюється з допомогою крос-перевірки. Крос-перевірка - це така процедура оцінки точності класифікації на даних з тестового безлічі.

Масштабованість - це властивість обчислювальної системи, яка забезпечує передбачуваний ріст системних характеристик, наприклад, швидкості реакції, загальної продуктивності та ін., при додаванні до неї обчислювальних ресурсів.

Для алгоритмів класифікації так само важливо на скільки складно класифікатор реалізувати, тобто загальна трудомісткість, адже на рівні з цим необхідні і інструменти для роботи з об'єктами.

Швидкість навчання класифікатора так само є важливим і необхідною властивістю при реалізації алгоритму, вона так само визначає на якому обсязі даних класифікатор починає коректно працювати.

Під широтою використання будемо мати на увазі можливість методу бути застосованим до найбільш широкого спектру об'єктів.

Можливість коригування теж є важливою особливістю класифікатора, адже даних багато і неможливо відразу навчити правильно, тому під цим будемо розуміти перенавчання.

Так як найчастіше даними методами користуються люди, які не є фахівцями в галузі аналізу даних, то інтерпретованість алгоритму також є важливим критерієм.

Як видно з таблиці 2.2 оцінка кожної з категорій була проведена наступними характеристиками: низька, нейтральна, висока. У кожного методу позначилися свої переваги і недоліки.

Таблиця 2.2 - Порівняльна характеристика популярних методів.

Алгоритм	**Точність**	**Масштабованість**	**Інтепретованість**	**Трудомісткість в реалізації**	**Швидкість**	**Широта використання**	**Можливість перенавчання**
Нейронна мережа	висока	низька	низька	висока	низька	висока	низька
Дерево рішень	низька	висока	висока	висока	висока	висока	висока
Метод Байєса	нейтральна	висока	нейтральна	нейтральна	висока	нейтральна	нейтральна
Метод к-найближчих сусідів	нейтральна	низька	висока	нейтральна	низька	нейтральна	висока

3. РОЗРОБКА СИСТЕМИ АДАПТАЦІЇ НА ОСНОВІ МЕТОДІВ МАШИННОГО НАВЧАННЯ

3.1 Проблеми проектування

Незважаючи на те що, проблема визначення місця розташування існує давно і аналоги систем визначення вже використовуються, вони до сих пір не мають сто процентної точності.

Існує ряд складнощів, з якими стикаються і які повинні вирішити якомога ефективнішим способом, щоб збільшити надійність результуючої програми:

1) різні моделі датчиків дають різні картинки того, що відбувається, через це одна і теж дія на різних пристроях може виглядати по-різному. Щоб в результаті додаток добре функціонував, необхідно розглядати всі можливі ситуації;

2) потрібно враховувати велику похибку даних з датчиків, тому що це може сильно вплинути на результат. Для цього можна усереднювати значення;

3) ще одна проблема - це конфіденційність. Так як додаток відправляє дані про місце знаходження користувача, це може привести до порушення конфіденційності. Тому програма має зашифровувати і приховувати ідентифікацію користувача, щоб підтримати його конфіденційність [17].

Аналізуючи теорію, описану вище, можна зробити висновки про архітектуру майбутнього програми і сформулювати більш конкретні завдання для його написання.

Архітектура системи для відстежування місцеположення повинна включати в себе наступні компоненти: збір даних з датчиків, збір даних з найближчих базових станцій, обробка отриманих даних, відображення поточного положення користувача .

Кращим варіантом пристрою для збору даних і відображення положення є мобільний телефон. Сьогодні такий ґаджет є практично у будь-якої людини. Основною перевагою цього варіанту є наявність вбудованих датчиків, достатніх для визначення руху. Має сенс використовувати такі датчики, як акселерометр, гіроскоп, компас.

Для обробки даних найлогічніше буде використовувати машинне навчання. Самим простим і ефективним алгоритмом для машинного навчання будуть дерева рішень.

Так як додаток планується реалізувати з мінімум підключення до мережі Інтернет, варто відзначити, будуть задіяні обчислювальні ресурси телефону і заряд його батареї. Клієнтський додаток буде збирати дані рухів і за допомогою машинного навчання будуть класифікувати і обчислюватися нове місце розташування.

Більш конкретні завдання можна сформулювати так:

1) написати додаток для мобільного пристрою, який буде збирати дані стільникового зв'язку, акселерометр, компаса і гіроскопа;

2) додаток, за допомогою якого можна буде розмічати дані для навчання, тобто записувати показники датчиків і явно вказувати, яка дію зараз відбувається;

3) додаток, який буде аналізувати дані за допомогою дерева рішень.

3.2 Архітектура додатка

Принцип дії визначення місцеположення по точкам мобільної мережі полягає в тому, що стільниковий телефон (або модуль стільникового зв'язку) знає, яким приймачем базової станції він обслуговується і маючи базу даних координат передавачів базової станції можна приблизно визначити своє місце розташування.

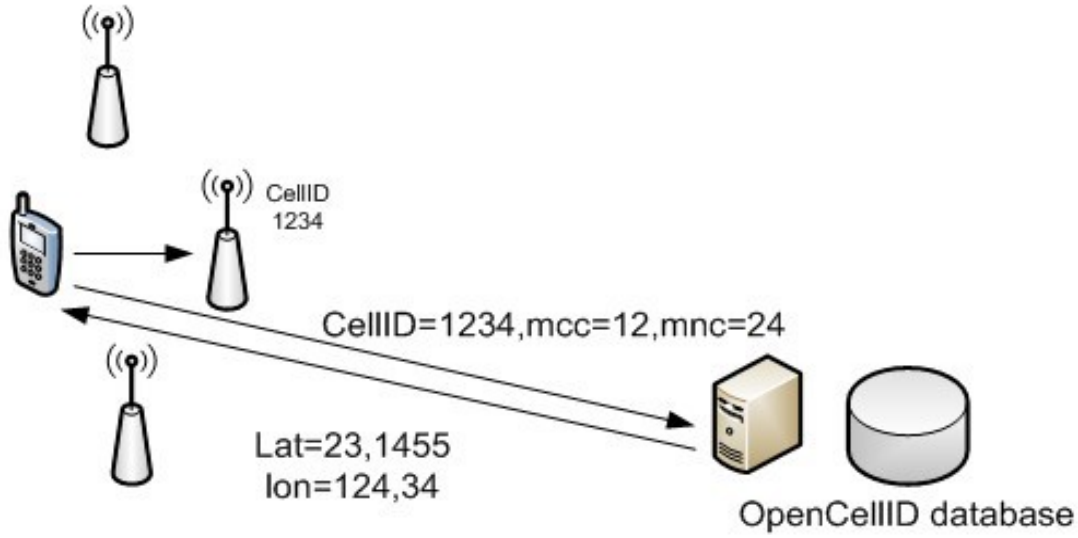

Рис. 3.1. Принцип визначення місцеположення за допомогою базової станції

Спершу необхідно підібрати відкриту базу з даними базових станції. Проаналізувавши інформацію, можна навести найбільш повні бази з необхідними даними у таблицю 3.1.

Таблиця 3.1. Порівняння відкритих баз даних

Найменування	**К-сть точок**	**К-сть точок в Україні**	**К-сть операторів**	**Безкоштовно**
Combain Mobile	100 млн.	Немає даних	1700	Ні
LocationAPI.org	157 млн.	Немає даних	1712	Ні
Mozilla Location Service	28 млн.	122 тис.	Немає даних	Так
OpenCellID	39	245	753	Так

	млн.	тис.		

Виходячи з наведених результатів, можемо виділити дві бази - Mozilla Location Service та OpenCellID, так як вони є безкоштовними та з достатньою кількістю точок в Україні. Для прикладу наведемо фрагмент карти Києва з сайту opencellid.org їх розташування.

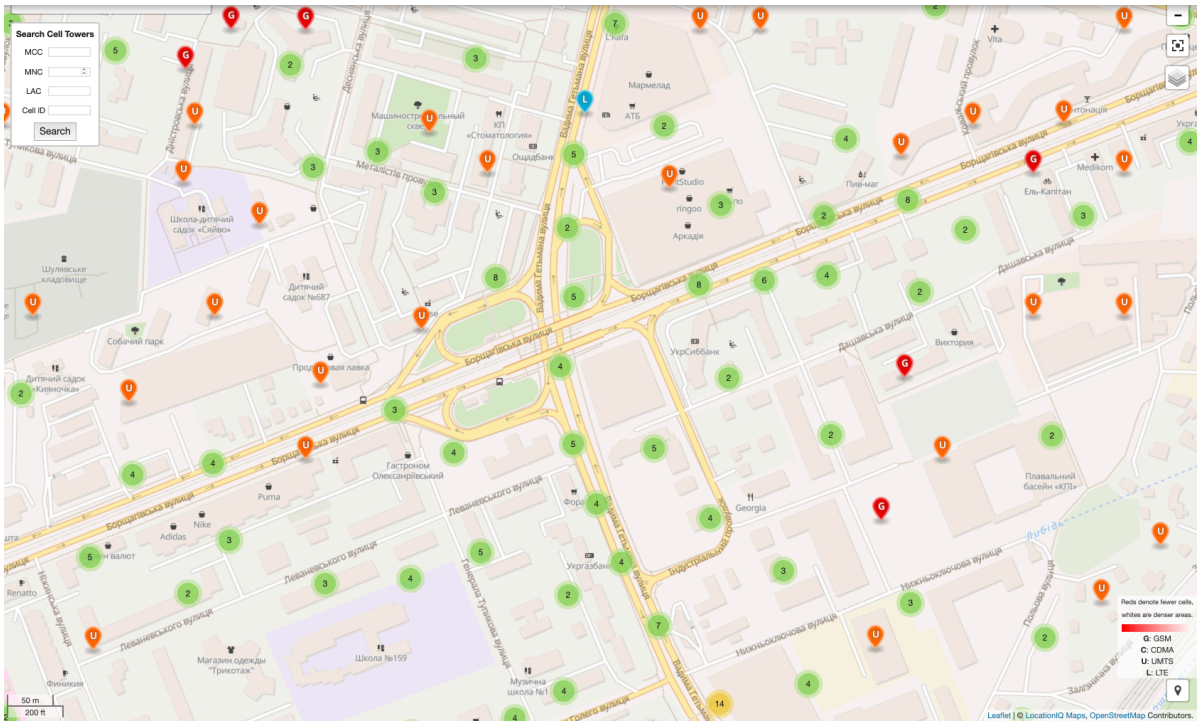

Рис. 3.2. Загальний вигляд частини міста з нанесеними базовими станціями

Тепер трохи про те, що таке передатчик в розумінні OpenCellID і яким чином наповнюється база даних OpenCellID. Ця БД наповнюється різними способами, найбільш простий - це установка на смартфон програми, яка записує координати телефону та обслуговуючу базову станцію, а потім відсилає на сервер всі вимірювання. На сервері OpenCellID відбувається обчислення приблизного місця розташування базової станції на підставі великого числа вимірювань. Таким чином, координати бездротової мережі обчислюються автоматично і є дуже приблизними.

На Рис. 3.3. наведемо детальну схему роботи модуля з визначення приблизних даних координат базової станції, до якої підключений мобільний пристрій.

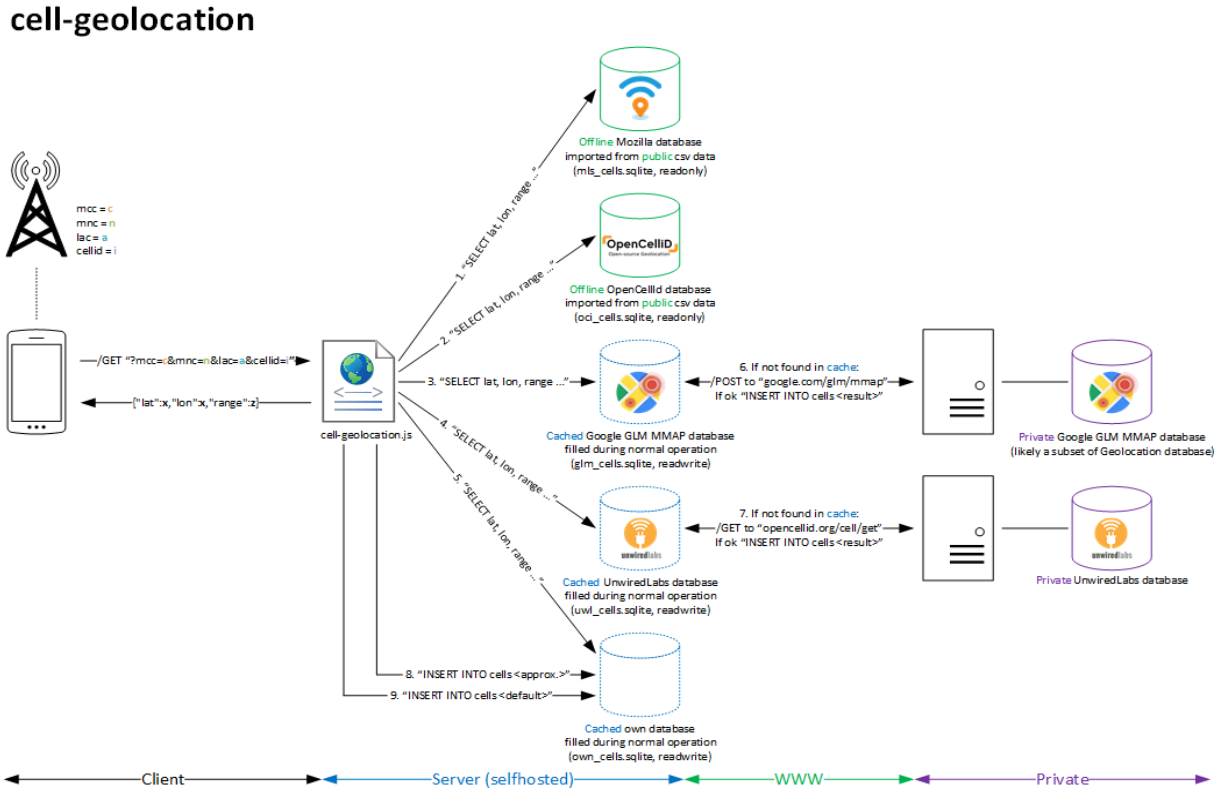

Рис. 3.3. Схема роботи модуля з визначення приблизних даних координат базової станції

Центральним елементом макету є сам мобільний пристрій. Спершу, він отримує дані з базової станції, а саме mcc (Mobile Country Code - код країни), mnc (Mobile Network Code – код оператора), lac (Location Area Cod – код локальної зони), cellid(CID – ідентифікатор базової станції). Отримавши необхідну інформацію, здійснюється пошук запису у локально розміщених базах даних для отриманння координат цієї базової станції. Пошук відбувається у такому порядку:

- Mozilla Location Service: локальна база;
- OpenCellId: локальна база;
- Google GLM MMAP та OpenCellIdCach: локальні бази з інформацією, яка буде знайдена на відповідних ресурсах;

- Google GLM MMAP: онлайн сервіс;
- OpenCellId: онлайн сервіс;
- Примерное местоположение в соответствии со средней точкой башен из автономной базы данных OpenCellId с теми же MCC, MNC и LAC (диапазон: 2147483648) ;
- Дефолтне місцерозташування (lat: 50.442843, lon: 30.443213, range: 4294967295).

Створення локальних баз данних

Спершу, для роботи з базами необхідно інсталювати програмне забезпечення SQLite.

```
    SQLite extension-functions.c
    sudo apt-get install -y libsqlite3-dev
    wget -O extension-functions.c https://www.sqlite.org/contrib/download/extension-functions.c?get=25
    gcc -fPIC -lm -shared extension-functions.c -o libsqlitefunctions.so
```

Створимо базу на основі отриманих даних з сервісу Mozilla Location Service

```
    wget -O mls_cells.csv.gz "https://d17pt8qph6ncyq.cloudfront.net/export/MLS-full-cell-export-$(date -u "+%Y-%m-%d")T000000.csv.gz"
    cat mls_cells.csv.gz | gunzip - > mls_cells.csv
    cat schema.sql | sqlite3 mls_cells.sqlite
    cat mls_import.sql | sqlite3 mls_cells.sqlite
```

Створимо базу на основі отриманих даних з сервісу OpenCellId

```
    wget                -O                oci_cells.csv.gz
"https://download.unwiredlabs.com/ocid/downloads?token=<Y
OUR_OPENCELLID_API_KEY>&file=cell_towers.csv.gz"
    cat oci_cells.csv.gz | gunzip - > oci_cells.csv
    cat schema.sql | sqlite3 oci_cells.sqlite
    cat oci_import.sql | sqlite3 oci_cells.sqlite
    cat oci_cells-cleanup.sql | sqlite3 oci_cells.sqlite
```

Створимо базу на основі отриманих даних з сервісу Google GLM MMAP

```
    cat cache_schema.sql | sqlite3 glm_cells.sqlite
```

Створимо базу на основі отриманих даних з сервісу OpenCellIdCache

```
    cat cache_schema.sql | sqlite3 uwl_cells.sqlite
```

Створимо базу на основі наближених результатів

```
    cat cache_schema.sql | sqlite3 own_cells.sqlite
```

Створені бази даних помістимо у проект додатку у директорію ~/www. Модуль, у якому будуть здійснюватися запити у уже створені бази даних матиме наступну конфігурацію:

```
    // cell-geolocation.js

    import SQLite from 'react-native-sqlite-storage';
    const request = require('./request.js');

    let mlsDb = SQLite.openDatabase({
      name: 'mls_cells.sqlite',
      location: 'default',
      createFromLocation: '~www/mls_cells.sqlite',
    });
    let ociDb = SQLite.openDatabase({
      name: 'oci_cells.sqlite',
```

```
      location: 'default',
      createFromLocation: '~www/oci_cells.sqlite',
    });
    let glmDb = SQLite.openDatabase({
      name: 'glm_cells.sqlite',
      location: 'default',
      createFromLocation: '~www/glm_cells.sqlite',
    });
    let uwlDb = SQLite.openDatabase({
      name: 'uwl_cells.sqlite',
      location: 'default',
      createFromLocation: '~www/uwl_cells.sqlite',
    });
    let ownDb = SQLite.openDatabase({
      name: 'own_cells.sqlite',
      location: 'default',
      createFromLocation: '~www/own_cells.sqlite',
    });

    function getInOciDb(mcc, mnc, lac, cellid) {
      return new Promise((resolve, reject) => {
        ociDb.transaction(
          (tx) => {
            tx.executeSql(`SELECT lat, lon, range FROM
cells WHERE mcc = ${mcc} AND mnc = ${mnc } AND lac = ${lac}
AND cellid = ${cellid}`, [], (tx, results) => {
              const rows = results.rows;
              let data = [];

              if (rows.length > 0) {
                for (let i = 0; i < rows.length; i++) {
                  data.push({
                    ...rows.item(i),
                  });
                }
                resolve(data);
              } else {
                resolve(getInGlmDb(mcc,     mnc,     lac,
cellid));
              }
```

```
            });
          },
          (error) => {
            resolve(getInGlmDb(mcc, mnc, lac, cellid));
          },
        );
      })
    }

    function getInGlmDb(mcc, mnc, lac, cellid) {
      return new Promise((resolve, reject) => {
        glmDb.transaction(
          (tx) => {
            tx.executeSql(`SELECT lat, lon, range FROM
cells WHERE mcc = ${mcc} AND mnc = ${mnc } AND lac = ${lac}
AND cellid = ${cellid}`, [], (tx, results) => {
              const rows = results.rows;
              let data = [];

              if (rows.length > 0) {
                for (let i = 0; i < rows.length; i++) {
                  data.push({
                    ...rows.item(i),
                  });
                }
                resolve(data);
              } else {
                resolve(getInUwlDb(mcc,     mnc,     lac,
cellid));
              }
            });
          },
          (error) => {
            resolve(getInUwlDb(mcc, mnc, lac, cellid));
          },
        );
      })
    }

    function getInUwlDb(mcc, mnc, lac, cellid) {
```

```
      return new Promise((resolve, reject) => {
        uwlDb.transaction(
          (tx) => {
            tx.executeSql(`SELECT lat, lon, range FROM
cells WHERE mcc = ${mcc} AND mnc = ${mnc } AND lac = ${lac}
AND cellid = ${cellid}`, [], (tx, results) => {
              const rows = results.rows;
              let data = [];

              if (rows.length > 0) {
                for (let i = 0; i < rows.length; i++) {
                  data.push({
                    ...rows.item(i),
                  });
                }
                resolve(data);
              } else {
                resolve(getInOwnDb(mcc,     mnc,     lac,
cellid));
              }
            });
          },
          (error) => {
            resolve(getInOwnDb(mcc, mnc, lac, cellid));
          },
        );
      })
    }

    function getInOwnDb(mcc, mnc, lac, cellid) {
      return new Promise((resolve, reject) => {
        ownDb.transaction(
          (tx) => {
            tx.executeSql(`SELECT lat, lon, range FROM
cells WHERE mcc = ${mcc} AND mnc = ${mnc } AND lac = ${lac}
AND cellid = ${cellid}`, [], (tx, results) => {
              const rows = results.rows;
              let data = [];

              if (rows.length > 0) {
```

```
                for (let i = 0; i < rows.length; i++) {
                  data.push({
                    ...rows.item(i),
                  });
                }
                resolve(data);
              } else {
                resolve(requestOSI(mcc,     mnc,     lac,
cellid));
              }
            });
          },
          (error) => {
            resolve(requestOSI(mcc, mnc, lac, cellid));
          },
        );
      })
    }

    function requestOSI(mcc, mnc, lac, cellid) {
      return new Promise((resolve, reject) => {
        request.osi(mcc, mnc, lac, cellid)
          .then((res) => {
            uwlDb.transaction(
              (tx) => {
                const   currentTime   =   Math.floor(new
Date().getTime() / 1000 | 0);
                tx.executeSql(`INSERT  INTO  cells  (mcc,
mnc, lac, cellid, lat, lon, range, created_at, updated_at)
VALUES(${mcc},${mnc},${lac},${cellid},${res.lat},${res.lo
ng},${res.range},${currentTime},${currentTime})`,     [],
(tx, results) => {

                });
              },
              (error) => {
                getInOwnDb(mcc, mnc, lac, cellid);
              },
            );
            resolve(res);
```

```
            })
            .catch((err) => {})
        })
    }

    export default function getCell(mcc, mnc, lac, cellid)
{
      return new Promise((resolve, reject) => {
        mlsDb.transaction(
          (tx) => {
            tx.executeSql(`SELECT lat, lon, range FROM
cells WHERE mcc = ${mcc} AND mnc = ${mnc } AND lac = ${lac}
AND cellid = ${cellid}`, [], (tx, results) => {
              const rows = results.rows;
              let data = [];

              if (rows.length > 0) {
                for (let i = 0; i < rows.length; i++) {
                  data.push({
                    ...rows.item(i),
                  });
                }
                resolve(data);
              } else {
                resolve(getInOciDb(mcc,     mnc,     lac,
cellid));
              }
            }, (err) => {});
          },
          (error) => {
            resolve(getInOciDb(mcc, mnc, lac, cellid));
          },
        );
      })
    }
```

Нижче наведено конфігурацію модуля, який відповідає за запити до зовнішніх сервісів у разі відсутності збігів у локальних базах даних.

```
    //request.js

    import axios from 'axios';
    import { Buffer } from 'buffer'

    let CONNECTION_TIMEOUT = 3000;

    const E_NOTFOUND = 'BTS not found';
    const E_REQERROR = 'Request error';

    const   RE_FETCH_OPENCELLID_LAT      =   /\slat="([+\-
\d\.]+)"/i;
    const   RE_FETCH_OPENCELLID_LON      =   /\slon="([+\-
\d\.]+)"/i;
    const  RE_FETCH_OPENCELLID_RANGE    =  /\srange="([+\-
\d\.]+)"/i;
    const   RE_FETCH_OPENCELLID_CODE    =   /\scode="([+\-
\d\.]+)"/i;

    const               RE_OPENCELLID_ERROR              =
/err\s+info="[^"]+"\s+code="/i;

    const RE_OPENCELLID_QUOTA = /exceeded/i

    const RE_UNWIREDLABS_QUOTA = /free/i

    /**
     * Perform request to Location Service.
     * Taken from https://github.com/kolonist/bscoords and
kept unchaged.
     *
     * @param {object} options      Node.js HTTPS request
options.
     * @param {*} request_body      Request body for POST
requests. Can be String or
     *                              Buffer. If you do not
need it you can pass null or
     *                              empty string ''.
```

```
     * @param {*} response_encoding Can be 'utf8' or 'hex'
(for Google response).
     *   @param   {*}   response_parser          Callback
function(response) where `response` is
     *                                String with data from
Location server. Callback
     *                               function should return
object like
     *                                `{lat: 23.12345, lon:
50.12345, range: 1000}` or
     *                                 null if there are no
coordinates in the answer.
     */
    const request = (options, response_parser) => {
      return new Promise((resolve, reject) => {
        axios(options)
          .then(function (response) {
            if (response && response.data) {
              const                coords                =
response_parser(response.data);
              if (coords !== null) {
                return resolve(coords);
              } else {
                return reject(new Error(E_NOTFOUND));
              }
            }
          })
          .catch((err) => {
            reject(new Error(E_REQERROR));
          })
      })
    };

    module.exports = {
      /**
       * Get geographical coordinates from Google GLM MMAP
(unofficial API).
       * Taken from https://github.com/kolonist/bscoords
and modified to parse range.
       *
```

```
        * @param  {Number}  mcc Mobile Country Code
        * @param  {Number}  mnc Mobile Network Code
        * @param  {Number}  lac Location area code
        * @param  {Number}  cid Cell Identity
        * @return {Promise} Object containing lat, lon and
range. If cell can not be resolved null.
        */
       glm: function (mcc, mnc, lac, cid) {
         const options = {
           url: 'http://www.google.com/glm/mmap',
           method  : 'POST',
         };

         const request_body = Buffer.from(

'000e000000000000000000000000001b000000000000000000000000
030000' +
           ('00000000'                                    +
Number(cid).toString(16)).substr(-8) +
           ('00000000'                                    +
Number(lac).toString(16)).substr(-8) +
           ('00000000'                                    +
Number(mnc).toString(16)).substr(-8) +
           ('00000000'                                    +
Number(mcc).toString(16)).substr(-8) +
           'ffffffff00000000',
           'hex'
         );

         const response_encoding = 'hex';


         /**
          * Convert 32-bit hex string into signed integer.
          * @param {String} hex Hex string like 'fab1c2d3'.
          */
         const hex2int = hex => {
           let int = parseInt(hex, 16);

           // negative number
```

```
          if ((int & 0x80000000) !== 0) {
            int = int - 0x100000000;
          }

          return int;
        };


        const response_parser = buf => {
          try {
            if (buf.length < 38) {
              return null;
            }

            const coords = {
              lat: hex2int(buf.slice(14, 22)) / 1000000,
              lon: hex2int(buf.slice(22, 30)) / 1000000,
              range: hex2int(buf.slice(30, 38))
            };

            if (coords.lat === 0 && coords.lon === 0) {
              return null;
            }

            return coords;
          } catch(err) {
            return null;
          }
        };

        return request({...options, data: request_body},
response_encoding, response_parser);
      },

      /**
       * Get geographical coordinates from OpenCellId.
       * Taken from https://github.com/kolonist/bscoords
and modified to parse range and code.
       *
       * @param  {Number}  mcc Mobile Country Code
```

```
        * @param  {Number}  mnc Mobile Network Code
        * @param  {Number}  lac Location area code
        * @param  {Number}  cid Cell Identity
        * @param  {String}  key OpenCellId API key
        *  @return {Promise} Object containing lat, lon,
range and status code. If there is a severe error null.
        */
       oci: function (mcc, mnc, lac, cid, key) {
         const options = {
           url:
`http://opencellid.org/cell/get?key=d5fc4198beb996&mnc=${
mnc}&mcc=${mcc}&lac=${lac}&cellid=${cid}`,
           method  : 'GET',
         };

         const request_body = null;
         const response_encoding = 'utf8';

         const response_parser = buf => {
           try {
             if      (RE_OPENCELLID_QUOTA.test(buf)      ||
RE_UNWIREDLABS_QUOTA.test(buf)) {
               const coords = {
                 lat: 0,
                 lon: 0,
                 range: 0,
                 statusCode: 429
               };

               return coords;
             } else if (RE_OPENCELLID_ERROR.test(buf)) {
               const coords = {
                 lat: 0,
                 lon: 0,
                 range: 0,
                 statusCode: 404
               };

               return coords;
             } else {
```

```
              const coords = {
                lat:
Number(RE_FETCH_OPENCELLID_LAT.exec(buf)[1]),
                lon:
Number(RE_FETCH_OPENCELLID_LON.exec(buf)[1]),
                range:
Number(RE_FETCH_OPENCELLID_RANGE.exec(buf)[1]),
                statusCode: 200
              };

              return coords;
            }
          } catch(err) {
            return null;
          }
        };

        return       request(options,       request_body,
response_encoding, response_parser);
      }
    };
```

```
    // index.js

    import getCell from './cell-geolocation';
    import Telephony from 'react-native-telephony'

     Telephony.getCellInfo((cellInfos) => {
          const    {    mcc,    mnc,    lac,    cid    }    =
cellInfos[0].cellIdentity;
          getCell(mcc, mnc, lac, cid)
            .then(res => {
              //get results
            })
            .catch(() => {
            });
        });
```

В результаті наведених вище операцій планується отримувати приблизні координати місцезнаходження користувача мобільного пристрою у наступному форматі:

```
{"lat":46.9226916,"lon":7.4132636,"range":1225}
```

3.3 Збір даних для визначення руху мобільного пристрою

Мета збору даних: записувати показники акселерометра, гіроскопа, компаса в відповідні файли для подальшої обробки.

При запуску програми починається збір даних і запис в файл. У файл записуються рядки, що містять час, що минув з моменту запуску програми і дані з датчиків.

Додаток отримує дані з акселерометра, гіроскопа і компаса кожні 25 мс .

Запис в файл відбувається через буфер. Спочатку дані записуються у відповідні буфери, при досягнення максимальної довжини буфера, він записується в файл і обнуляється. Такий метод запису в файли економить час: звернення до файлу на запис довга за часом виконання операція, записуючи буфер в файл, а не окремі дані, число звернень до файлу скорочується, а отже ефективність поліпшується.

Також в додатку для збору даних реалізований перерахунок показників акселерометра.

Осі акселерометра повинні збігатися з осями телефону: вісь X вказувати в напрямку руху, у вправо, а z наверх. Так як телефон не завжди знаходиться в потрібному положенні, напрямки осей телефону і руху можуть розходитися. Це може призводити до серйозних розбіжностей реальних даних і отриманих. Щоб не зіткнутися з такою проблемою був реалізований перерахунок даних акселерометра по осях. Для перерахунку береться середнє значення гіроскопа по осях за 1 секунду, обчислюються матриці повороту для розрахунку нових показників акселерометра.

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \quad M_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix} \quad M_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Потім ці матриці перемножуються: $M = M_z * (M_x * M_y)$ і показники акселерометра записані в матрицю множать на матрицю M .

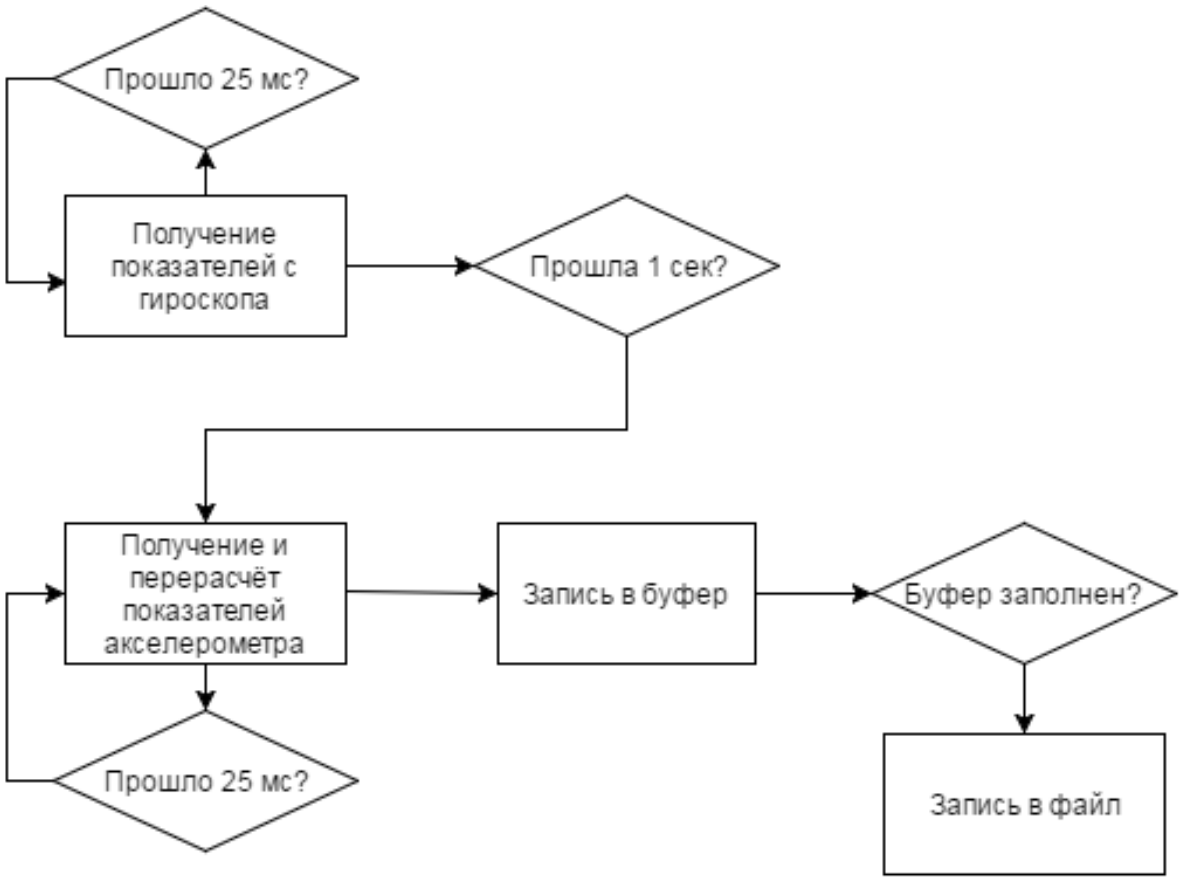

Рис. 3.4. Алгоритм запису показників акселерометра в файл

3.4 Аналізатор даних

Для навчання було вирішено використовувати Python з бібліотекою SciKit - Learn . Ця бібліотека включає в себе велику кількість алгоритмів машинного навчання, а також має відмінну документацію, що робить її більш привабливою для тих, хто тільки починає працювати в області аналізу даних. В ході попереднього дослідження було вирішено сконцентруватися на аналізі алгоритмів Decision Tree (дерева рішень), так як вони показували

найкращі результати. Ці алгоритми часто використовуються в задачах, в яких об'єкти мають категоріальні ознаки і використовується для задач регресії і класифікації.

Машинне навчання буде використовуватися тільки для швидкості і аналізувати дані акселерометра по осі X. розпізнавати будуть 4 стану: користувач стоїть, користувач йде/їде з рівномірною швидкістю, користувач прискорюється і користувач гальмує.

Насамперед потрібно підготувати дані для навчання. Для цього:

1) Округлюються значення;

2) Створюється новий масив, в якому міститься середнє значення і середньоквадратичне відхилення для вибірки з 10.

Коли користувач стоїть середнє значення показників акселерометра і середньоквадратичне відхилення знаходяться в районі нуля.

Коли користувач йде / їде з рівномірною швидкістю, середнє значення перебуває в районі нуля, а середньоквадратичне відхилення вище нуля.

Коли користувач прискорюється, середнє значення значно вище нуля, коли сповільнюється, значно нижче. У цих двох випадках середньоквадратичне значення не так важливо.

Потім для кожного нового значення обчислюється його лейбл . Лейбл 1 означає, що користувач стоїть, лейбл 2, що користувач рівномірно їде, лейбл 3, що користувач прискорюється і лейбл 4, що користувач сповільнюється. Зберігається значення лейблу , який найчастіше зустрічався в вибірці з 10 елементів.

Після будується регресія RandomForestClassifier за даними отриманими вище. Алгоритм Random Forest називається «Випадковий Ліс», тому що для отриманих даних він створює безліч дерев прийняття рішень і потім усереднює результат їх прогнозів. Важливим моментом тут є елемент випадковості у створенні кожного дерева. Адже зрозуміло, що якщо ми створимо багато однакових дерев, то результат їх усереднення буде володіти точністю одного дерева.

Показники акселерометра для розрахунку і час, відповідне їм, завантажуються в масиви, які повинні бути підготовлені для подальшого аналізу. Для кожного значення викликається predict, який повертає лейбл, що вказує дію, яка відбувалась в момент зняття даних. Для обчислення поточної швидкості використовується середнє значення акселерометра, час і лейбл .

Якщо лейбл дорівнює 1, то і швидкість дорівнює нулю, тому що користувач знаходиться в стані спокою.

Якщо лейбл дорівнює 2, то користувач їде рівномірно, і швидкість дорівнює попередньої швидкості.

Якщо лейбл дорівнює 3, то користувач прискорюється, і формула для обчислення швидкості $v_{new} = v_0 + acc * time$, де - це нове значення швидкості, - попереднє значення швидкості, - середнє значення акселерометра за час time.

Якщо лейбл дорівнює 4, то користувач гальмує і формула для обчислення виглядає як при лейблі 3, тільки значення негативно.

Похибка такого алгоритму вийшла відносно невеликою. Звичайно у даного алгоритму є мінуси, він погано працює на кругових розв'язках і різких поворотах. Похибку і помилки можна зменшити, збільшивши кількість даних для навчання.

Висновки до розділу 3

Архітектура системи для відстежування місцеположення включає в себе наступні компоненти: збір даних з датчиків, збір даних з найближчих базових станцій, обробка отриманих даних, відображення поточного положення користувача .

Кращим варіантом пристрою для збору даних і відображення положення є мобільний телефон. Сьогодні такий ґаджет є практично у будь-якої людини. Основною перевагою цього варіанту є наявність вбудованих датчиків, достатніх для визначення руху.

Для обробки даних було використано машинне навчання. Самим простим і ефективним алгоритмом для машинного навчання є дерева рішень.

ВИСНОВКИ

Результати даної роботи показують, що написання визначення місцеположення без доступу до GPS є складною і комплексним завданням. Використання алгоритмів машинного навчання здатне в рази збільшити точність визначення нової координати.

В даній роботі було на створено додаток для мобільного пристрою, який збирає дані стільникового зв'язку, акселерометр, компаса і гіроскопа. За допомогою машинного навчання було класифіковано дію, яка зараз відбувається.

Принцип дії визначення місцеположення по базовим станціям полягає в тому, що стільниковий телефон (або модуль стільникового зв'язку) знає, яким приймачем базової станції він обслуговується і маючи базу даних координат передавачів базової станції можна приблизно визначити своє місце розташування.

Написану в ході цієї роботи програму, неможна використовувати масово, так як у нього є ряд недоліків:

1) невисока точність. Не дивлячись на те, що додаток має хорошу точність, погрішність в обчисленнях все ще велика. Поліпшити точність визначення місця розташування можна, наприклад, розширивши базу для машинного навчання.

2) дані для навчання збиралися з використанням декількох пристроїв. Для масового використання це число повинне доходити до декількох десятків, тому що різні пристрої матимуть різні показання акселерометрів на одному і тому ж місцезнаходженні, що може призводити до неправильної класифікації дій. Ця проблема теж вирішувана розширенням бази машинного навчання.

Незважаючи на свої недоліки, додаток все-таки показує непогані результати, щодо аналогів і може бути використано в якості основи для подальших досліджень.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Гріффітс Д., Гріффітс Д. Head First. Програмування для Android [Текст]. - СПб .: Пітер, 2016. - 704 с .: іл. - (Серія «Head First O'Reilly»).

2. Комаровський Ю.А. Погіршення точності GPS-приймача поблизу високих об'єктів / Вчені записки Комсомольського-на-Амурі Державного технічного університету. - 2012. - No1. - с. 28-35.

3. Лебедєв О.А. Картографічні проекції [Текст]: методичний посібник. - Новосибірськ: Новосибірський навчально-методичний центр по ГІС і ДЗ, 2000. - 37 с.

4. Макконнел С. Досконалий код. Майстер клас. [Текст] / Пер. з англ. - М .: Видавничо-торговий дім «Російська Редакція»; СПб .: Пітер, 2005. - 896 с .: іл.

5. Паттерни проектування. [Текст] / Е. Фрімен, Е. Фрімен, К. Сьєрра, Б. Бейтс. - СПб .: Пітер, 2016. - 656 с .: іл. - (Серія «Head First O'Reilly»).

6. Програмування під Android. 2-е изд. [Текст] / З. Меднікс, Л. Дорнін, Б. Мік, М. Накамура. - СПб .: Пітер, 2013. - 560 с .: іл. - (Серія «Бестселери O'Reilly»).

7. Сієрра К., Бейтс Б. Вивчаємо Java. [Текст] - М .: Ексмо, 2016. - 720 с .: іл. - (Серія «Head First O'Reilly»).

8. Furht B. Handbook of Augmented Reality. [Текст] / Florida Atlantic University (USA): Springer, 2011. - 769 с.

9. OpenStreetMap in GIScience. [Текст] / J.J. Arsanjani, A. Zipf, P. Mooney, M. Helbich. - New York: Springer, 2015. - 324 с.

10. Salychev. O.S.Applied Inertial Navigation: problems and solutions. [Текст] - М .: BMSTU Press, 2004. - 304 с. (Геогр коорд)

11. Градусна мережа і її елементи [Електронний документ]. - (http://www.geo-site.ru/index.php/2011-01-21-10-59-05/106-2011-01-21-10-54-52/775-gradys-set. html). Перевірено 25.11.2019.

12. Іванов О.Є., Лясін Д.Н. Дослідження методів геоприв'язки даних для сервісів доповненої реальності. [Електронний документ] // Novainfo. Технічні науки. - 2016. - No42-3. - (http://novainfo.ru/article/4895). Перевірено 25.09.2019.

13. Правила і стандарти оформлення коду програм. Iguania [Електронний документ]. - (http://iguania.ru/article/programming-style). Перевірено 19.10.2019.

14. Сенсори. Прискорення, орієнтація. Startandroid. Урок 137 [Електронний документ]. - (http://startandroid.ru/ru/uroki/vse-uroki-spiskom/287- urok-137-sensory-uskorenie-orientatsija.html). Перевірено 27.11.2019.

15. Android. Graphics architecture [Електронний документ]. - (http://source.android.com/devices/graphics/architecture.html). Перевірено 12.11.2016.

16. Android Developers. CameraDevice [Електронний документ]. - (https://developer.android.com/reference/android/hardware/camera2/ CameraDevice.html). Перевірено 27.09.2019.

17. Android Developers. Connecting to the Network [Електронний документ]. - (https://developer.android.com/training/basics/network- ops / connecting.html).