Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ ЙМОВІРНІСНОГО ПРОГРАМУВАННЯ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-19-1
Шевченко М.С.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Кузьомін О.Я.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Шевченку Микиті Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження особливостей ймовірнісного програмування

затверджена наказом по університету від « 23 » жовтня 2020 року № 1428Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 01 грудня 2020 р.

3. Вихідні дані до роботи *Математичні моделі застосовані у мовах імовірнісного програмування*
перелік використовуваних програмних засобів: РуМС3
теоретичні відомості про мови імовірнісного програмування
реалізація на Python остаточного апостеріорного розподілу

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд імовірнісних мов програмування та їх характерних рис
2. Огляд системи імовірнісного мислення щодо прийняття рішень
3. Гаусові висновки та моделі
4. Апостеріорні прогностичні перевірки
5. Практична реалізація та оцінка з використанням *РуМС3*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Комп'ютерна презентація*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	23.10.2020	**виконано**
2	Аналіз завдання, підбір літератури	01.11.20-03.11.20	**виконано**
3	Аналіз літератури з досліджуваної проблеми	04.11.20-07.11.20	**виконано**
4	Аналіз технічних засобів	08.11.20-12.11.20	**виконано**
5	Розробка методів	13.11.20-17.11.20	**виконано**
6	Програмна реалізація	18.11.20-20.11.20	**виконано**
7	Оформлення пояснювальної записки	21.11.20-24.11.20	**виконано**
8	Перевірка на плагіат	30.11.20	**виконано**
9	Рецензування	05.12.20	**виконано**
10	Підготовка презентації та доповіді	06.12.20	**виконано**
11	Занесення роботи в електронний архів	08.12.20	**виконано**
12	Попередній захист атестаційної роботи	09.12.20	**виконано**

Дата видачі завдання 23 жовтня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Кузьомін О.Я.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до атестаційної роботи: 55 с., 37 рис., 44 джерела.

ІМОВІРНІСНІ МОВИ ПРОГРАМУВАННЯ, АПОСТЕРІОРНИЙ РОЗПОДІЛ, СТАТИСТИКА, АПОСТЕРІОРНІ ПЕРЕВІРКИ, БАЙЄСОВСЬКА ЛІНІЙНА РЕГРСІЯ.

Об’єктом дослідження є аналіз побудови моделей імовірнісного програмування.

Метою дослідження є аналіз особливостей використання імовірнісного міркування щодо імовірнісного програмування.

В роботі застосовані байєсовські методи до практичної задачі, щоб показати наскрізний байесовский аналіз, який переходить від побудови моделей до виявлення апріорних ймовірностей, а далі до реалізації в Python остаточного апостеріорного розподілу.

PROBABILISTIC PROGRAMMING LANGUAGES, POSTERIOR PROBABILITY DISTRIBUTION, STATISTICS, POSTERIOR PREDICTIVE CHECK, BAYESOV LINEAR REGRESSION.

The object of research is the analysis of construction of models of probabilistic programming.

The aim of the research is to analyze of features of the usage of probabilistic reasoning in relation to probabilistic programming.

The paper applies Bayesian methods to a practical problem to show a thorough Bayesian analysis, which goes from building models to identifying a priori probabilities, and then to the implementation in Python of the final a posteriori distribution.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

PPL – probabilistic programming language

ШІ – штучний інтелект

СІП – система імовірнісного програмування

TFP – TensorFlow Probability – бібліотека для імовірнісних міркувань

ВСТУП

Імовірнісне мислення – неймовірно цінний інструмент для прийняття рішень. Від економістів до гравців в покер, люди, які можуть мислити категоріями імовірностей, схильні приймати більш правильні рішення, коли стикаються з невизначеними ситуаціями. Області імовірностей і теорії ігор створювалися століттями і десятиліттями, але не переживають відродження зі швидкою еволюцією штучного інтелекту [1-4].

Імовірнісне програмування – це спосіб створення систем, які допомагають приймати рішення в умовах невизначеності. Багато повсякденних рішень припускають судження при визначенні відповідних факторів, які людна зазвичай не спостерігає безпосередньо. Історично склалося так, що одним із способів допомоги у прийманні рішень в умовах невизначеності було використання системи імовірнісних міркувань. Імовірнісні міркування об’єднують знання про ситуацію з законами імовірності.

До недавнього часу системи імовірнісних міркувань були обмежені за обсягом та їх важко було застосовувати до багатьох ситуацій реального світу. Імовірнісне програмування – це новий підхід, який спрощує побудову систем імовірнісних міркувань і робить їх більш широко застосовними.

Для подолання невизначеності у дослідженнях часто використовується статистика – одна з основ великого сегмента ринку машинного навчання. Імовірнісні міркування довгий час вважалися однією з основ алгоритмів виводу і представлені всіма основними фреймворками і платформами машинного навчання. Останнім часом імовірнісні міркування набули широкого поширення в технологічних гігантів, таких як Uber, Facebook або Microsoft, що допомогло просунути дослідницьку і технологічну діяльність в цій сфері. Зокрема, PPL стали однією з найбільш активних областей розвитку машинного навчання, що призвело до появи деяких нових цікавих технологій.

1 ТЕОРЕТИЧНИЙ ОГЛЯД PROBABILISIC PROGRAMMING LANGUAGES

1.1 Знайомсво з предметом дослідження

Для знайомства з імовірнісним програмуванням спочатку розглянемо модель «звичайного» програмування [5, 6]. У «звичайному» програмуванні основою є алгоритм (рис. 1.1), зазвичай детермінований, який дозволяє з вхідних даних отримати вихідні за здалегідь встановленими правилами.

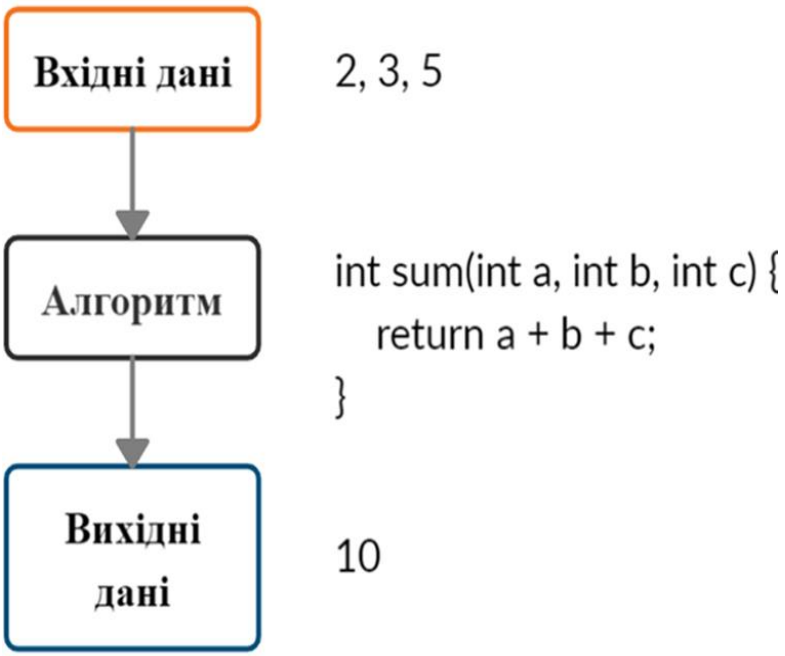


Рисунок 1.1 – Ілюстрація до «звичайного» програмування

Наприклад, у школі хлопчик Вася грає у м’яч, та відомо де він знаходиться, куди він кидає м’яч і які зовнішні умови (наприклад, сила вітру), тоді можна дізнатися, яке вікно він, на жаль, розіб’є в будівлі школи. Для цього достатньо симулювати прості закони шкільної фізики, які легко можна записати у вигляді алгоритму (рис. 1.2).

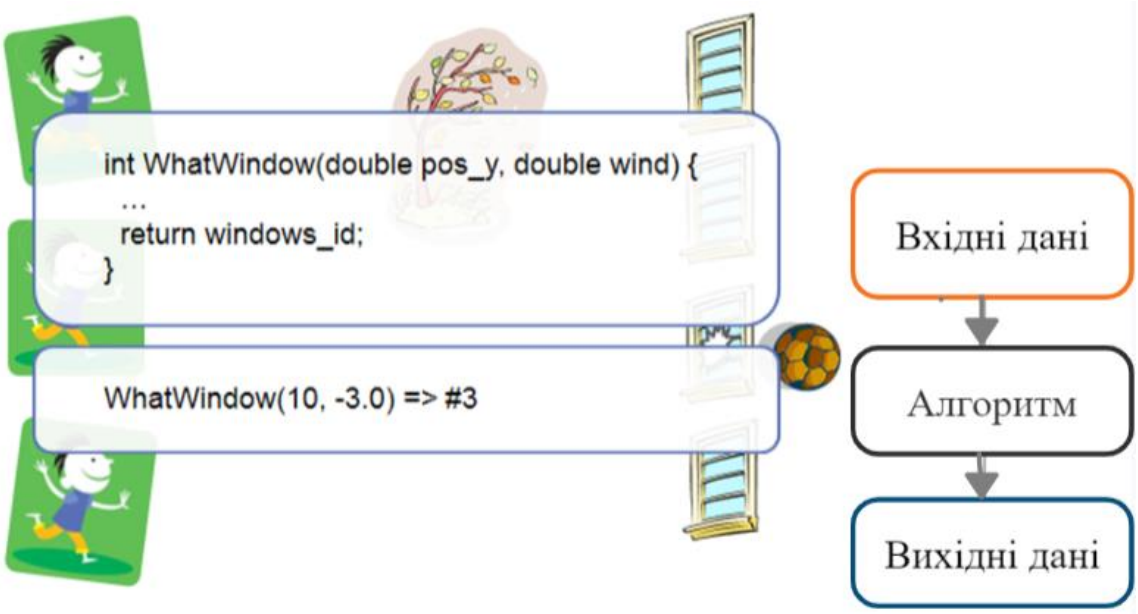


Рисунок 1.2 – Алгоритм з урахуванням зовнішніх умов

А тепер розглянемо, як ця ситуація виглядає з позиції імовірнісного програмування (рис. 1.3).

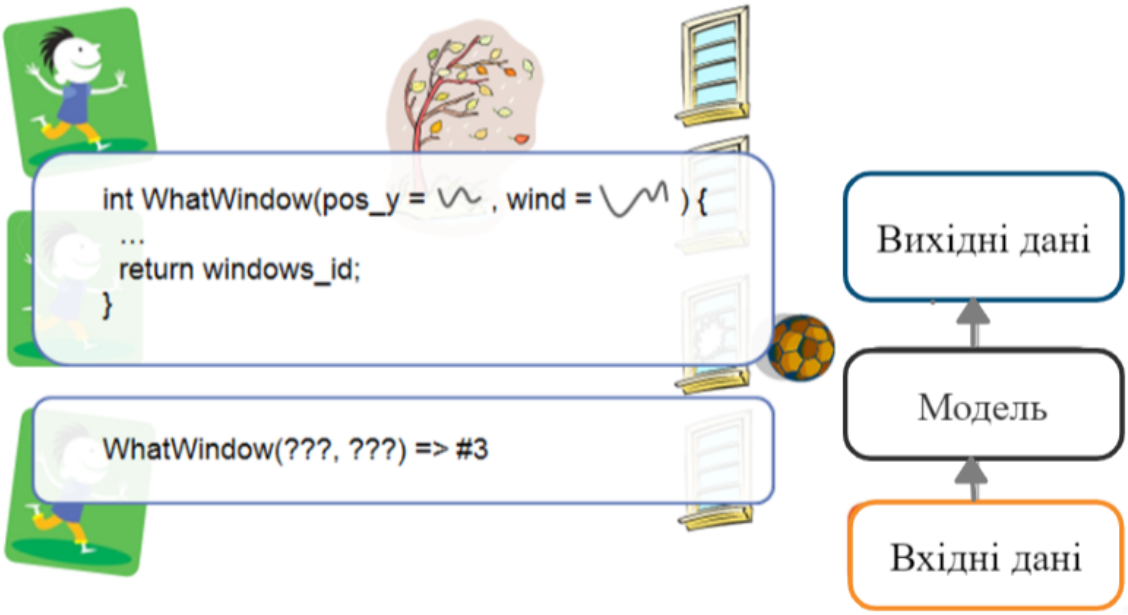


Рисунок 1.3 – Алгоритм з позиції імовірнісного програмування

Однак, частіше за все відомо тільки результат, але зацікавленість у тому, щоб дізнатися саме ті невідомі значення, які привели до цього результату [7]. Щоб відповісті на це питання, за допомогою теорії

математичного моделювання створюється імовірнісна модель, частина параметрів якої не визначені точно.

Наприклад, у випадку з хлопчиком Васею, знаючи, що він розбив вікно та маючи апріорні знання про те, що біля вікна він та його друзі зазвичай грають у футбол, і знаючи прогноз погоди на цей день, необхідно дізнатися апостеріорні розподіли розташування хлопчика Васі, тобто звідки він кидав м'яч [8, 9].

Отже, знаючи вихідні дані, зацікавленість у тому, щоб дізнатися найбільш імовірні значення прихованих, невідомих параметрів. В рамках машинного навчання розглядаються у тому числі породжуючі імовірнісні моделі. В рамках породжуючих імовірнісних моделей, модель описується як алгоритм, але замість точних однозначних значень прихованих параметрів і деяких вхідних параметрів використовуються імовірнісні розподіли на них .

Існує більше 15 мов імовірнісного програмування. У даній роботі наведені приклади коду на імовірнісних мовах Venture / Anglican, яка має дуже схожий синтаксис і яка бере свій початок від імовірнісного мови Church. Також наведені приклади програм на Edward, Pyro та інших мовах. Church в свою чергу заснований на мові «звичайного» програмування Lisp і Scheme.

1.2 Використання байєсівської лінійної регресії

Розглянемо задання простої імовірнісної моделі байєсівської лінійної регресії на мові імовірнісного програмування Venture / Anglican у вигляді імовірнісної програми (рис. 1.4).

```
[ASSUME t1 (normal 0 1)]
[ASSUME t2 (normal 0 1)]
[ASSUME noise 0.01]
[ASSUME noisy_x (lambda (time) (normal (+ t1 (* t2 time)) noise))]
[OBSERVE (noisy_x 1.0) 10.3]
[OBSERVE (noisy_x 2.0) 11.1]
[OBSERVE (noisy_x 3.0) 11.9]
[PREDICT t1]
[PREDICT t2]
[PREDICT (noisy_x 4.0)]
```

Рисунок 1.4 – Задання імовірнісної моделі байєсівської лінійної регресії

Приховані шукані параметри – значення коефіцієнтів *t1* і *t2* лінійної функції *x = t1 + t2 * time*. У нас є апріорні припущення про дані коефіцієнти, а саме припускаємо, що вони розподілені за законом нормального розподілу *Normal (0, 1)* із середнім *0* і стандартним відхиленням *1*. Таким чином, визначино в перших двох рядках імовірнісної програми апріорну імовірність на приховані змінні: *P (T)*. Інструкцію *[ASSUME name expression]* можна розглядати як визначення випадкової величини з ім'ям *name*, що приймає значення обчислюємого виразу (програмного коду) *expression*, який містить в собі невизначеність.

Імовірнісні мови програмування (маються на увазі конкретно Church, Venture, Anglican), як і Lisp / Scheme, є функціональними мовами програмування, і використовують польську нотацію при записі виразів для обчислення. Це означає, що у вираженні виклику функції спочатку розташовується оператор, а вже тільки потім аргументи: *(+ 1 2)*, і виклик функції обрамляється круглими дужками. На інших мовах програмування, таких як C ++ або Python, це буде еквівалентно коду *1 + 2*.

В імовірнісних мовах програмування вираз виклику функції прийнято розділяти на три різних види:

– виклик детермінованих процедур (*primitive-procedure arg1 ... argN*), які при одних і тих самих аргументах завжди повертають одне і те ж значення. До таких процедур, наприклад, відносяться арифметичні операції;

– виклик імовірнісних (стохастичних) процедур *(stochastic-procedure arg1 ... argN)*, які при кожному виклику генерують випадковим чином елемент з відповідного розподілу. Такий виклик визначає нову випадкову величину. Наприклад, виклик ймовірнісної процедури *(normal 1 10)* визначає випадкову величину, розподілену за законом нормального розподілу *Normal(1, sqrt (10))*, і результатом виконання кожен раз буде якесь дійсне число;

– виклик складових процедур *(compound-procedure arg1 ... argN)*, де *compound-procedure* – введена користувачем процедура за допомогою спеціального виразу *lambda: (lambda (arg1 ... argN) body)*, де *body* – тіло процедури, що складається з виразів. У загальному випадку складова процедура є стохастичною (недетермінованою) складовою процедурою, так як її тіло може містити виклики імовірнісних процедур.

Повернемося до вихідного коду на мові програмування Venture / Anglican. Після перших двох рядків задємо умовну ймовірність *P(X | T)*, тобто умовну ймовірність спостережуваних змінних *x1, x2, x3* при заданих значеннях прихованих змінних *t1, t2* і параметра *time*.

Перед введенням безпосередньо самих спостережень за допомогою виразу *[OBSERVE ...]* визначаємо загальний закон для спостережуваних змінних *xi* в рамках нашої моделі, а саме припускаємо, що дані випадкові величини спостерігаються при заданих *t1, t2* і заданому рівні шуму *noise* розподілені за законом нормального розподілу *Normal (t1 + t2 * time, sqrt (noise))* із середнім *t1 + t2 * time* і стандартним відхиленням *noise*. Дана умовна імовірність визначена на рядках 3 і 4 даної імовірнісної програми. *noisy_x* визначена як функція, що приймає параметр *time* і повертає випадкове значення, визначене за допомогою обчислення вираження *(normal(+ t1 (* t2 time)) noise)* і обумовлене значеннями випадкових величин *t1* і *t2* і змінної *noise*. Відзначимо, що вираз *(normal (+ t1 (* t2 time)) noise)* містить в собі невизначеність, тому кожен раз при його обчисленні будемо отримувати в загальному випадку різні значення [10-15].

У рядках 5-7 безпосередньо вводяться відомі значення *x1 = 10.3, x2 = 11.1, x3 = 11.9*. Інструкція виду *[OBSERVE expression value]* фіксує спостереження про те, що випадкова величина, що приймає значення згідно з виконанням вираження *expression*, прийняла значення *value*.

Повторимо на даному етапі все, що зроблено. На рядках 1-4 за допомогою інструкцій виду *[ASSUME ...]* задано безпосередньо саму вірогідну модель: *P (T) і P (X | T)*. На рядках 5-7 безпосередньо поставлено відомі значення спостережуваних випадкових величин *X* за допомогою інструкцій виду *[OBSERVE ...]*.

На рядках 8-9 запитуємо у системи імовірнісного програмування апостеріорне розподіл *P (T | X)* прихованих випадкових величин *t1* і *t2*. Як вже було сказано, при великому обсязі даних і досить складних моделях отримати точне аналітичне подання неможливо, тому інструкції виду *[PREDICT ...]* генерують вибірку значень випадкових величин з апостеріорного розподілу *P (T | X)* або його наближення. Інструкція виду *[PREDICT expression]* в загальному випадку генерує один елемент вибірки зі значень випадкової величини, що приймають значення згідно з виконанням вираження *expression*. Якщо перед інструкціями виду *[PREDICT ...]* розташовані інструкції виду *[OBSERVE ...]*, то вибірка буде з апостеріорного розподілу (кажучи точніше, звичайно, з наближення апостеріорного розподілу), обумовленого перерахованими раніше введеними спостереженнями.

1.3 Імовірнісні мови програмування

PPL – це предметно-орієнтовані мови, які описують імовірні моделі і механізми для виконання логічних висновків в цих моделях. PPL заснована на поєднанні можливостей логічного висновку імовірнісних методів з репрезентативною силою мов програмування.

У програмі PPL припущення кодуються за допомогою апріорних імовірностей по змінним моделі. Під час виконання програма PPL запускає процедуру виведення для автоматичного обчислення апостеріорних розподілів параметрів моделі на основі спостережуваних даних. Іншими словами, логічний висновок коригує попередня імовірність з використанням спостережуваних даних, щоб отримати більш точний результат. Результатом програми PPL є розподіл імовірностей, який дозволяє програмісту явно візуалізувати і управляти невизначеністю, пов'язаною з результатом.

Щоб проілюструвати простоту PPL, скористаємося однією з найвідоміших проблем сучасної статистики: упередженим підкиданням монети [16-18].

Ідея цього завдання – обчислити зсув монети. Припустимо, що $x_i = 0$, чи $x_i = 1$, якщо результат підкидання i-ї монети – решка, і $x_i = 0$, якщо це решка. У контексті передбачається, що окремі підкидання монети незалежні і однаково розподілені і що кожне підкидання слідує розподілу Бернуллі з параметром Θ : $p(x_i=1|\Theta)=0$ і $p(x_i=1|\Theta)=1-\Theta$. Прихована (тобто неспостережнна) змінна − це зміщення монети. Завдання полягає в тому, щоб вивести Θ за результатами раніше спостережуваних підкидань монети, тобто $p(\Theta| x_1, x_2,\ldots,x_N)$ (рис. 1.5) .

Рисунок 1.5 – Ілюстрація прикладу

Моделювання простої програми, такої як упереджене підкидання монети, на мові програмування загального призначення можно привести у

сотнях рядків коду [19-22]. Однак PPL, такі як Edward, висловлюють цю проблему в кількох простих рядах (рис. 1.6).

```
# Model
theta = Uniform(0.0, 1.0)
x = Bernoulli(probs=theta, sample_shape=10)
# Data
data = np.array([0, 1, 0, 0, 0, 0, 0, 0, 0, 1])
# Inference
qtheta = Empirical(
tf.Variable(tf.ones(1000) □ 0.5))
inference = ed.HMC({theta: qtheta},
data={x: data})
inference.run()
# Results
mean, stddev = ed.get_session().run(
[qtheta.mean(),qtheta.stddev()])
print("Posterior mean:", mean)
print("Posterior stddev:", stddev)

```

Рисунок 1.6 – Приклад програми на PPL

Протягом десятиліть простір машинного навчання було розділено на два непримиренні табори: статистика та нейронні мережі. Один табір породив імовірнісне програмування, а інший відстоював трансформаційні рухи, такі як глибоке навчання. Нещодавно дві філософські школи об’єдналися, за для того, щоб об’єднати глибоке навчання та байєсівське моделювання в єдині програми (рис. 1.7). Кінцевим виразом цих зусиль є глибокі імовірнісні мови програмування (Deep PPL).

Мова глибокого імовірнісного програмування (PPL) – це мова для визначення глибоких нейронних мереж та імовірнісних моделей. Іншими словами, глибокий PPL спирається на мови програмування, баєсівську статистику та глибоке навчання, щоб полегшити розробку потужних програм машинного навчання.

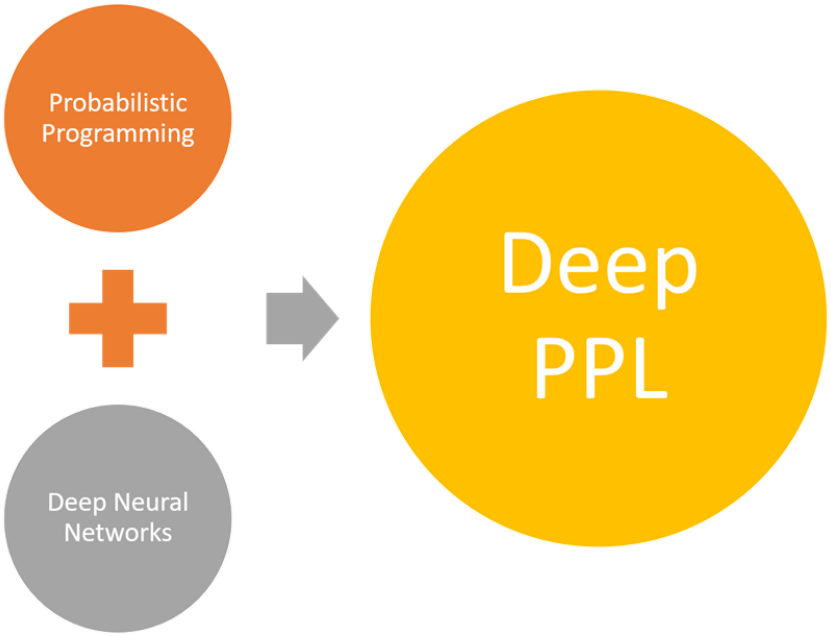


Рисунок 1.7 – Об’єднання глибокого навчання і байєсівського моделювання

Deep PPL може висловлювати байєсові нейронні мережі з імовірними вагами і зсувами. Практично кажучи, Deep PPL матеріалізувалися як нові імовірнісні мови і бібліотеки, які легко інтегруються з популярними фреймворками глибокого навчання [23-25].

В останні роки область імовірнісних мов програмування (PPL) бурхливо розвивається завдяки дослідженням та інноваціям. Велика частина цих інновацій виникла в результаті об’єднання PPL і методів глибокого навчання для створення нейронних мереж, які можуть ефективно справлятися з невизначеністю. Технологічні гіганти, такі як Google, Microsoft або Uber, несуть відповідальність за розширення меж Deep PPL в великомасштабні сценарії. Ці зусилля втілилися в абсолютно нові стеки Deep PPL, які стають все більш популярними в співтоваристві машинного навчання. Розглянемо деякі з найостанніших досягнень в області Deep PPL.

1.3.1 Імовірнісна мова програмування Edward

Edward – це повна по Тьюрингу імовірнісна мова програмування, написана на Python. Спочатку Edward був продуктом команди Google Brain, але зараз у нього великий список розробників. Оригінальна дослідницька робота над Edward була опублікована в березні 2017 року і з тих пір стек отримав велику популярність у спільноти машинного навчання. Edward об'єднує три області: байєсівську статистику і машинне навчання, глибоке навчання та імовірнісне програмування. Бібліотека легко інтегрується з фреймворками глибокого навчання, такими як Keras і TensorFlow.

На рисунку 1.8 наведено невеликий зразок программи з використанням Edward.

```
# Модель
theta = Uniform (0.0, 1.0)
x = Bernoulli (probs = theta, sample_shape = 10)
# Data
data = np.array ([0, 1, 0, 0, 0, 0, 0, 0, 0, 1])
# Висновок
qtheta = Empirical (
tf.Variable (tf.ones (1000) * 0.5))
inference = ed.HMC ({theta: qtheta},
data = {x: data})
inference.run ()
# Результати
mean, stddev = ed.get_session (). Run (
[qtheta.mean (), qtheta.stddev ()])
print ("Середнє Середнє:", Середнє)
print ("Posterior stddev:", stddev)

qalpha = tf.Variable (1.0)
qbeta = tf.Variable (1.0)
qtheta = Beta (qalpha, qbeta)
# Висновок
inference = ed.KLqp ({theta: qtheta}, {x: data})
inference.run ()

```

Рисунок 1.8 – Програма на Edward

1.3.2 Deep PPL Pyro

Pyro – це глибока імовірнісна мова програмування (Deep PPL), випущена Uber AI Labs. Pyro побудований на основі PyTorch і заснований на чотирьох фундаментальних принципах: універсальність, масштабованість, мінімальність та гнучкість [26-29].

Це універсальний PPL – він може представляти будь-який обчислюваний розподіл ймовірностей. Почавши з універсальної мови з ітерацією і рекурсією (довільний код Python), а потім додавши випадкову вибірку, спостереження і висновок.

Pyro масштабується до великих наборів даних з невеликими накладними витратами в порівнянні з написаним коду вручну, створюючи сучасні методи оптимізації чорного ящика, які використовують міні-пакети даних для приблизного виведення.

Pyro маневрений і ремонтопридатний. Pyro реалізований за допомогою невеликого ядра потужних складових абстракцій. По можливості, важка робота делегується PyTorch та іншим бібліотекам.

Pyro прагне до автоматизації, коли цього хочется, і контролю, коли це потрібно. Pyro використовує абстракції високого рівня для вираження генеративних моделей і моделей логічного висновку, дозволяючи експертам легко налаштовувати логічний висновок.

Як і інші PPL, Pyro поєднує в собі моделі глибокого навчання і статистичний висновок, використовуючи простий синтаксис, як показано на рисунку 1.9.

```
Variable (torch.Tensor ([0])),
Variable (torch.Tensor ([1]))
pyro .sample ( "x", Бернуллі (
theta * Variable (torch.ones (10)))
# Data
data = { "x": Variable (torch.Tensor (
[0, 1, 0, 0, 0, 0, 0, 0, 0, 1]))}
# Висновок
cond = pyro.condition (coin, data = data)
sampler = pyro.infer.Importance (cond,
num_samples = 1000)
post = pyro .infer.Marginal (sampler, sites = [ "theta"])
# Результат
samples = [post () [ "theta"]. data [0] for _ in range (1000)]
print ( "Середнє значення:", np.mean (зразки))
print ( "Posterior stddev:", np.std (samples)) # Керівництво по висновку
def guide ():
qalpha = pyro.param ( "qalpha", Variable (torch.Tensor ([1.0]), requires_grad = True))
qbeta = pyro.param ( "qbeta", Variable (torch.Tensor ([1.0]), requires_grad = True))
pyro.sample ( "theta", Beta (qalpha, qbeta))
# Висновок
svi = SVI (cond, guide, Adam ({}), loss = "ELBO", num_particles = 7)
for _ in range (1000):svi.step ()
```

Рисунок 1.9 – Програма на Pyro

1.3.3 Фреймворк Infer.Net

Microsoft недавно відкрила вихідний код Infer.Net – фреймворка, який спрощує імовірнісне програмування для розробників .Net. Microsoft Research працює над Infer.Net з 2004 року, але тільки недавно, з появою глибокого навчання, цей фреймворк став дійсно популярним. Infer.Net надає кілька сильних відмінностей, що робить Infer.Net відмінним вибором для розробників, які працюють на Deep PPL. Розглянемо декілька з них.

Перш за все це багата мова моделювання. Підтримка одновимірних і багатовимірних змінних, як безперервних, так і дискретних. Моделі можуть бути побудовані з широкого діапазону факторів, включаючи арифметичні операції, лінійну алгебру, обмеження діапазону і позитивності, булеві оператори, дискретні оператори Дирихле, гаусові і багато інших.

Infer.Net розроблено для великомасштабного виводу. Infer.NET компілює моделі в вихідний код, який може виконуватися незалежно без додаткових витрат. Його також можна інтегрувати безпосередньо в вашу програму [30-32].

Також Infer.Net розширюваний користувачем: розподіли ймовірностей, фактори, операції з повідомленнями та алгоритми виводу можуть бути додані користувачем. Infer.NET використовує архітектуру модулів, що робить його відкритим і адаптованим.

Приклад підкидання монети використовуючи Infer.Net зображено на рисунку 1.10.

```
1  Variable<bool> firstCoin = Variable.Bernoulli(0.5);
2  Variable<bool> secondCoin = Variable.Bernoulli(0.5);
3  Variable<bool> bothHeads = firstCoin & secondCoin;
4  InferenceEngine engine = new InferenceEngine();
5  Console.WriteLine("Probability both coins are heads: "+engine.Infer(bothHeads));
6
```

Рисунок 1.10 – Програма на Infer.Net

Сфера Deep PPL поступово стає дуже важливим та одним з основних блоків екосистеми машинного навчання. Pyro, Edward і Infer.Net – це всього лише три недавніх приклади Deep PPL, але не єдині. Перетин фреймворків глибокого навчання і PPL відкриває неймовірно великі можливості для інновацій, і нові варіанти використання, та імовірно, в найближчому майбутньому розширять кордони Deep Learning [33].

1.4 Постановка задачі дослідження

Стоїть задача розглянути особливості використання імовірнісного міркування щодо імовірнісного програмування. Також, відштовхуючись від звичайного прогрмування та беручи до уваги декілька мов імовірнісного програмування, треба розглянути їх особливості та використання.

У досліджені застосовуються байєсівські методи до практичних задач, щоб показати наскрізний байєсівський аналіз, який переходить від побудови моделей до виявлення апріорних імовірностей і реалізації на Python

остаточного апостеріорного розподілу. Закрема ці задачи дають можливість розглянути особливості машиного навчання.

Метою дослідження є аналіз методів імовірнісного програмування. Об’єктом дослідження є процес застосування спеціальніх технологій представлення даних з урахуванням статистики та байєсівської моделі. Для досягнення поставленої мети необхідно виконати наступні завдання:

– дослідити особливості найпоширеніших імовірнісних мов програмування;

– дослідити систему імовірнісного мислення щодо імовірнісного програмування;

– застосувати гаусові висновки та моделі на практиці;

– провести апостеріорні прогностичні перевірки.

2 РОЗРОБКА ІМОВІРНІСНОЇ ПРОГРМИ НА ОСНОВІ ІМОВІРНІСНИХ МІРКУВАНЬ

2.1 Підходи до програмування

Розглянемо простий приклад упередженого підкидання монети, досліджуваний усіма першокурсниками математики по всьому світу. Окремі підкидання монети описуються розподілом Бернуллі з параметром *x*, прихованої змінної або зміщенням монети [34-36]. Тому описуємо наступну функцію розподілу ймовірностей, щоб вивести приховану змінну з урахуванням результатів раніше спостережуваних підкидань монети:

$$p(x|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} .$$

Можна сказати, що мета виведення описується як *p (x | y)*. Далі можна використовувати теорему Байеса для обчислення умовної імовірності з урахуванням імовірнісного опису моделі монети як спільного розподілу наступний чином:

$$p(X|Y) = \frac{p(Y|X)p(X)}{p(Y)} = \frac{p(X,Y)}{p(Y)} = \frac{p(X,Y)}{\int p(X,Y)dX} .$$

2.1.1 Традиційний підхід до програмування

Розробка комп’ютерної програми для упередженого підкидання монети з використанням мови програмування загального призначення потребують

сотень рядків коду і являє собою складну проблему для вирішення вручну, що вимагає виконання наступних обчислень:

Крок 1. Враховуючи приховану змінну *x* і спостережуваний результат *y*, значення спільної імовірності можна обчислити таким чином:

$$p(x|y) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{y+a-1}(1-x)^{\beta-y}.$$

Крок 2. Якщо треба обчислити приховану змінну з урахуванням раніше спостережуваних змінних, а не просто спільний розподіл імовірності, можна обчислити *p (x | y)*, званий апостеріорним, наступний чином:

$$p(x|y) = Beta(\alpha+y, \beta-y+1),$$

де *Beta* відноситься до бета-біноміального розподілу, тобто безперервному розподілу на інтервалі *(0, 1)*.

Крок 3. Використовуючи вищевикладене, можна вичислити приховану змінну, тобто зміщення монети, враховуючи спостережувані «орла» або «решку» раніше. Наприклад, щоб комп'ютерній програмі вичислити імовірність того, що прихована змінна більше 0,7 при *y = 1*, програма потребує вичислення наступного:

$$p(x > 0.7|y = 1) = \int I\,(x > 0.7)p(x|y = 1)dx.$$

2.1.2 Імовірисний підход до програмування

Як вказано вище, мови та інструменти імовірного програмування дозволяють автоматизувати баєсівський вивід. Іншими словами, це дозволяє

кодувати, використовуючи імовірнісний синтаксис, специфічний для мови, для опису условних розподілених ймовірностей [37]. Наприклад, щоб закодирувати попередню програму підбрасування монети за допомогою Edward – потрібно всього декілька рядків коду (рис. 2.1)

```
1  import edward as ed
2  import numpy as np
3  import six
4  import tensorflow as tf
5  from edward.models import Bernoulli, Betaed.set_seed (42)
6  x_data = np.array ([0, 1, 0, 0, 0, 0, 0, 0, 0, 1]) # Модель
7  p = Beta (1.0, 1.0)
8  x = Bernoulli (probs = p, sample_shape = 10) # Complete Conditional
9  p_cond = ed.complete_conditional (p)
10
```

Рисунок 2.1 – Програма підбрасування монет на Edward

Імовірнісні мови програмування дозволяють інтегрувати статистику, машинне навчання і мови програмування загального призначення не тільки для опису імовірнісних моделей і виконання логічних висновків, а й для надання набору інструментів, за допомогою якого можна швидко просувати машинне навчання, автоматизуючи складні та виснажливі обчислення, пов’язані з контрольованими, неконтрольованими і напів-контрольованими висновоками, як показано вище.

Недавні досягнення в області глибокого навчання стали можливими завдяки розробці і впровадженню технологій та фреймворків, які забезпечують можливість обчислювати оптимізації на основі градієнтів швидше і ефективніше, ніж будь-коли раніше.

Перенісши складні та виснажливі обчислення в ці програмні середовища, дослідники отримали можливість досліджувати і розвивати глибоке навчання і штучний інтелект з вражаючими результатами.

2.2 Системи імовірнісного мислення щодо прийняття рішень

У реальному світі питання, які нас хвилюють, рідко мають чіткі «так» або «ні». Наприклад, якщо запускається новий продукт, хочеться знати, чи буде він добре продаватися. Можна думати, що він буде успішним, тому що, він добре розроблений і дослідження ринку вказує на необхідність цього, але ніхто у цьому не можете бути впевненим. Адже можливо конкурент випустить ще кращий продукт, або, може, у нього є фатальний недолік, який відключить ринок, або, може, економіка раптово зміниться у гіршу сторону. Якщо потрібно бути на сто відсотків впевненим, людина не зможете прийняти рішення про запуск продукту (рис. 2.2).

Рисунок 2.2 – Користувачі відносно галузі продукта

Маємо доступ до деяких знань про проблеми, які нас цікавлять. Наприклад, ви знаєте про свою продукцію і, можливо, перевірили дослідження ринку, щоб зрозуміти, хто буде його покупцем. Можливо, ви

також дізнаєтеся, що ви знаєте про конкурентів та знання з прогнозом розвитку економіки. Логіка дозволяє отримувати відповіді на питання з використанням іменованих знань.

Але необхідний спосіб опису своїх знань і логіка отримання відповідей на питання з використанням цих знань. Ймовірнісне програмування дає те й інше. Перш ніж переходити до опису системи імовірнісного програмування, опишемо більш загальну концепцію системи імовірнісних міркувань, яка надає базові засоби для специфікації знань і реалізації логіки.

Імовірнісні міркування – це підхід, в якому модель предметної області використовується для прийняття рішень в умовах невизначеності. Візьмемо приклад з області футболу. Припустимо, що за статистикою 9% кутових завершуються голом. Потрібно передбачити результат конкретного кутового удару. Зростання центрального нападника атакуючої команди дорівнює 193 сантиметри, і відомо, що він відмінно грає головою.

Основного воротаря команди, що захищається тільки що забрали на носилках, і на заміну йому вийшов воротар, який проводить свій перший матч. Крім того, дме сильний вітер, який ускладнює контроль над польотом м’яча.

На рисунку 2.3 показано, як отримувати відповідь за допомогою системи імовірнісних міркувань. Всі свої знання про кутові удари і релевантні чинники кодуємо у вигляді моделі кутового удару. Потім надаємо факти про конкретний кутовий удар: що центральний нападаючий високий, що воротар недосвідчений і що дме сильний вітер. Говоримо системі, що хотіли б дізнатися, чи буде забитий гол. Алгоритм виведення повертає відповідь: гол буде забитий з ймовірністю 20%.

Рисунок 2.3 – Як імовірнісна система міркувань пророкує результат кутового удару

В системі імовірнісних міркувань створюється модель, в якій відображені всі релевантні загальні знання про предметну область у вигляді ймовірностей. У даному випадку це може бути опис ситуації кутового удару і всіх релевантних характеристик гравців і навколишніх умов, які можуть вплинути на результат. Потім модель застосовується до конкретної ситуації, для якої треба отримати висновок. Для цього моделі передається наявна інформація, яка називається фактами або доказами. В даному випадку є три докази: центральний нападаючий високий, воротар недосвідчений і дме сильний вітер. Отриманий від моделі висновок допоможеу майбутньому прийнятты рішення – наприклад, що на наступну гру потрібно поставити іншого воротаря. Самі заключення постають у вигляді ймовірностей, наприклад, ймовірність інших ігрових навичок воротаря.

Співвідношення між моделлю, що надається інформацією та відповідями на запити математично строго визначено законами теорії імовірностей. Процес використання моделі для отримання відповідей на запити при відомих фактах називається імовірнісним висновком, або просто

висновком. На щастя, розроблені алгоритми, які виконують необхідні обчислення, приховуючи від вас всю математику. Вони називаються алгоритмами виведення (рис. 2.4).

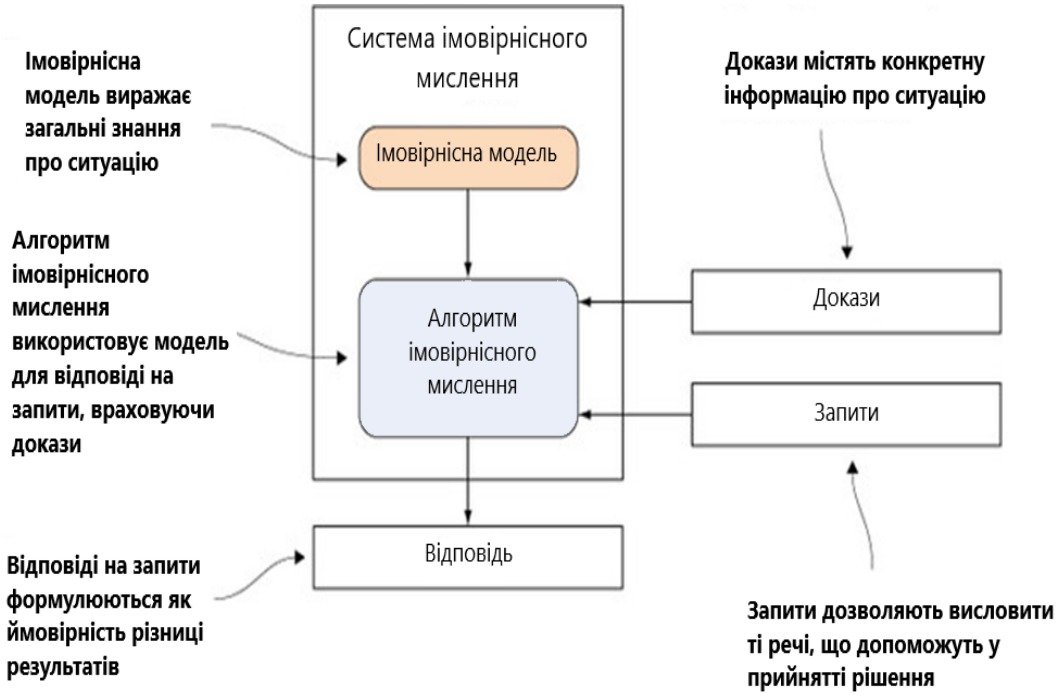


Рисунок 2.4 – Основні компоненти системи імовірнісних міркувань

Системи імовірнісних міркувань володіють гнучкістю. Вони можуть відповідати на запити у будь-якому аспекті ситуації, якщо задані факти, які мають відношення до будь-яких інших аспектів [38, 39]. На практиці система виконує міркування трьох видів:

– прогноз майбутніх подій;
– вивід причини подій;
– навчання на минулих подіях, щоб краще прогнозувати майбутні.

Необхідно зробити висновок про причини подій. Система імовірнісних міркувань має розподіл усіх міркуваннь:

– ймовірність;
– шаблон міркування;
– міркування щодо умов аргументів.

Імовірнісне міркування дозволяє використовувати інформацію про те, що трапилося під час подачі попереднього кутового, щоб краще передбачити результат наступного. На рисунку 2.5 показано, як це робиться. До складу фактів входить все те, що і минулого разу, а також нова інформація про поточну ситуацію. Відповідаючи на питання, чи буде забитий гол на цей раз, алгоритм виведення спочатку визначає властивості ситуації, які привели до голу в перший раз, наприклад, кваліфікацію нападника і воротаря. А потім ці властивості використовуються для передбачення результату в новій ситуації.

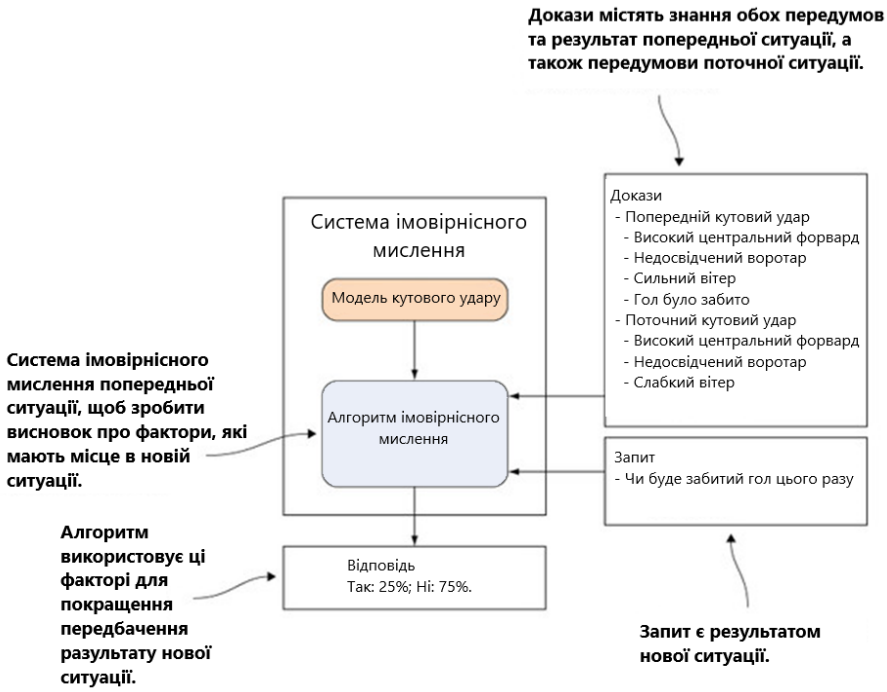


Рисунок 2.5 – Система імовірнісного міркування, у якій використовуються факти, які стосуються результату минулого кутового.

Тепер, змінівші запит и докази, система може визначити, чому був забитий гол (рис. 2.6).

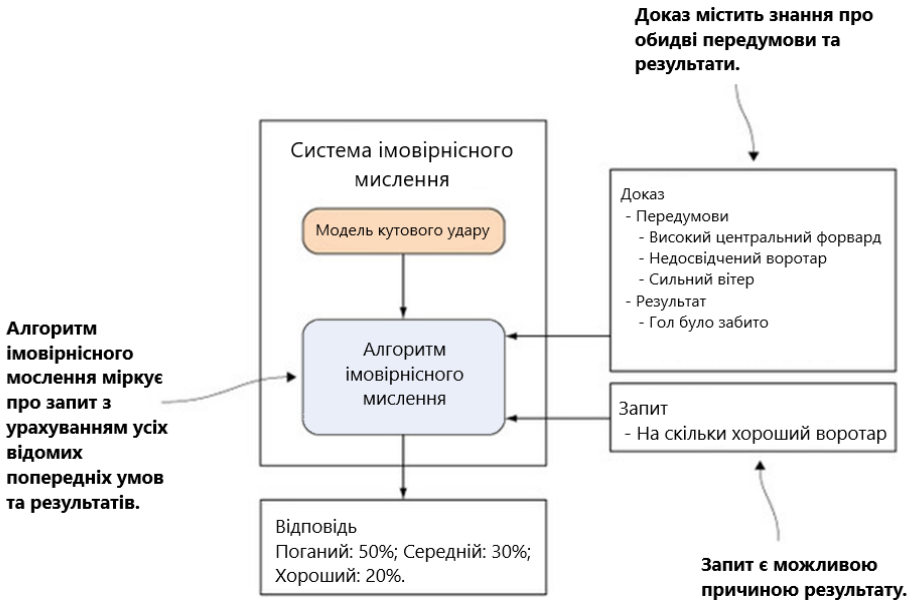


Рисунок 2.6 – Система зі зміненими фактими і запитом, яка може зробити висновок про те, чому був забитий гол

Запити перерахованих видів можуть допомогти в прийнятті різних рішень: знаючи ймовірність голу при наявності або відсутності додаткового захисника, можна вирішити, чи варто замінити атакуючого захисником; на основі оцінки кваліфікації воротаря можна вирішити, яку зарплату запропонувати йому в наступному контракті; отримавши інформацію про воротаря, можна вирішити, чи варто ставити його на наступну гру.

Вище були описані способи міркування про конкретні ситуації при відомих фактах. Але система імовірнісних міркувань дозволяє зробити ще одну річ: навчаючись на минулому досвіді, поліпшити загальні знання. Третій спосіб міркувань – це використання минулих результатів для кращого передбачення результату в конкретній нової ситуації. А зараз про поліпшення самої моделі. Якщо є великий минулий досвід, тобто інформація щодо багатьох кутових ударів, то чому б не навчити нову модель, що представляє загальні знання про те, що зазвичай відбувається при подачі кутового. На рисунку 2.7 показано, що це досягається за допомогою алгоритму навчання. На відміну від алгоритму виведення, його завдання полягає не в тому, щоб відповідати на питання, а в тому, щоб породити нову

модель. На вхід алгоритму навчання подається вихідна модель, а він оновлює її на основі досвіду. Потім нову модель можна використовувати для відповіді набудущее запити. Імовірно відповіді нової моделі будуть більш обгрунтовані, ніж відповіді вихідної.

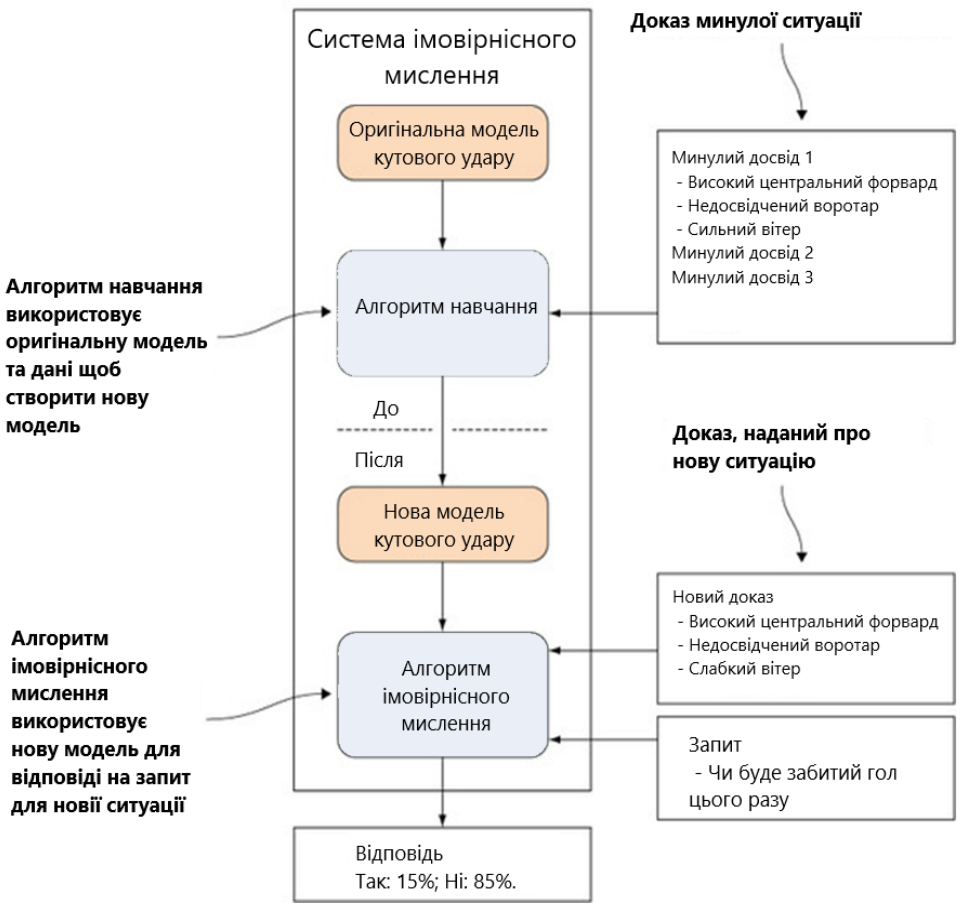


Рисунок 2.7 – Система, яка відображає, що за допомогою алгоритму навчання можна навчити нову модель на основі минулого досвіду, а потім використовувати її для майбутніх висновків

У будь-якій системі імовірнісних міркувань використовується мова представлення, на якій виражаються імовірнісні моделі. Таких мов багато. Деякі моделі доволі розповсюджені, наприклад, байесовсівські мережі (їх ще називають мережами довіри) або приховані марківські моделі. Від мови представлення залежить, які моделі зможе обробити система і як вони виглядають. Безліч моделей, які представлені мовою, називається виразною

силою мови. Для розробки додатків потрібно, щоб виразна сила була якомога більше.

Система імовірнісного програмування – це просто система імовірнісних міркувань, для якої мовою подання є мова програмування. Говорячи «мова програмування», мається на увазі, що вона має всі можливості, які прийнято очікувати від типової мови програмування: змінними, розвиненою системою типів даних різної величини, засобами управління потоком виконання, функціями і іншими.

Як буде описано далі, мови імовірнісного програмування здатні висловити широкий спектр імовірнісних моделей, що далеко виходить за рамки більшості традиційних систем імовірнісних міркувань. Виразна сила мов імовірнісного програмування дуже висока.

На рисунку 2.8 ілюструється співвідношення між системами імовірнісного програмування і імовірнісних міркувань в загальному випадку. Основна зміна полягає в тому, що моделі виражаються у вигляді програм на деякій мові програмування, а не у вигляді математичних конструкцій типу байєсівської мережі.

Тому факти, запити та відповіді стають змінними програми. Факти можна уявити конкретними значеннями змінних, запит – це отримання значення змінної, а відповідь – ймовірності того, що змінні приймають ті чи інші значення.

Крім того, система імовірнісного програмування зазвичай включає набір алгоритмів виводу. Ці алгоритми застосовуються до моделей, написаних на мові системи.

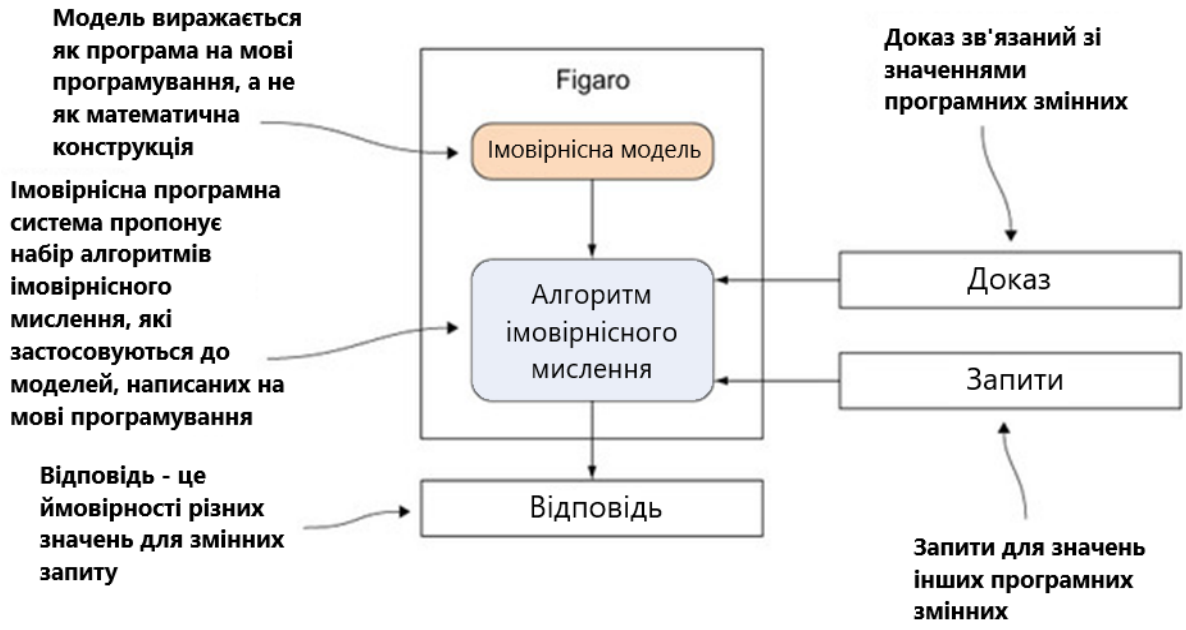


Рисунок 2.8 – СІП (система імовірнісних міркувань), в якій для подання імовірнісних моделей застосовується мова програмування

Існує багато систем імовірнісного програмування, але в роботі розглядаються тільки функціональні, повні по Тьюрингу системи. Функціональні означає, що вони засновані на функціональній мові програмування.

Тим не менш, щоб користуватися функціональною системою імовірнісного програмування, не потрібно знати, що таке лямбда-функція. Просто функціональне програмування становить теоретичну підставу, завдяки якій мова має можливість представляти імовірнісні моделі.

А повним по Тьюрингу називається мова, на якій можна закодувати будь-яке обчислення, яке здатний виконати цифровий комп’ютер. Якщо щось взагалі можна зробити за допомогою комп’ютера, то це можна зробити на будь-якій повній по Тьюрингу мові. Більшість мов програмування на слуху, наприклад C, Java і Python, є повними по Тьюрингу.

Оскільки системи імовірнісного програмування засновані на мовах програмування, повних по Тьюрингу, то вони володіють видатною гнучкістю в частині типів моделей, які можна побудувати з їх допомогою.

Головна ідея будь-якої мови програмування – виконання. Імовірнісна програма в цьому сенсі аналогічна, тільки вона може мати не один, а кілька шляхів виконання, кожен з яких породжує свій вихід.

Яким шляхом слідувати, визначається випадковим вибором, що здійснюється всередині програми. Кожен випадковий вибір має кілька можливих результатів, і в програмі закодована ймовірність кожного результату. Тому можна вважати, що імовірнісна програма – це програма, при виконанні якої випадковим чином генеруються вихідні дані.

Ця ідея ілюструється на рисунку 2.9. Тут система імовірнісного програмування містить програму кутового удару, яка описує випадковий процес генерації результату кутового удару.

Програма приймає ряд вхідних даних; в нашому прикладі це зростання центрального нападника, досвідченість воротаря і сила вітру. На основі цих даних програма випадковим чином виконується і генерує вихідні дані. Кожне випадкове виконання дає на виході конкретний результат.

Оскільки кожен випадковий вибір може мати кілька випадків, то існує багато можливих шляхів виконання, що дають різні результати. Будь-який заданий результат, наприклад взяття воріт, може бути згенероване на різних шляхах виконання.

Розглянемо, як ця програма визначає імовірнісну модель. Будь-яке конкретне виконання програми є результатом послідовності випадкових виборів. Кожен випадковий вибір відбувається з певною ймовірністю. Якщо перемножити всі ці ймовірності, то вийде ймовірність шляху виконання.

Таким чином, програма визначає ймовірність кожного шляху виконання. Якщо уявити собі, що програма проганяється багаторазово, то частка прогонів, на яких генерується заданий шлях виконання, дорівнює його ймовірності. Імовірність деякого результату дорівнює частці прогонів, на яких було отримано цей результат. На рисунку 2.9 гол виходить в 1/4 всіх прогонів, тому ймовірність голу дорівнює 1/4.

Рисунок 2.9 – Імовірнісна програма визначає процес випадкової генерації вихідних даних по відомим вхідним

На рисунку 2.9 показаний сенс ймовірнісної програми – визначення процесу випадкового виконання, а не як використовувати систему імовірнісного програмування – виконати алгоритм виведення для отримання відповіді на запит при заданих фактах. Тому хоча структурно малюнки схожі, вони ілюструють різні концепції. Насправді, випадкове виконання лежить в основі і деяких алгоритмів виводу, але є багато алгоритмів, які не засновані на простому випадковому виконанні.

Легко побачити, як можна використовувати вірогідну програму для передбачення майбутнього. Потрібно просто виконати її випадкове число раз, подаючи на вхід відомі дані про сьогодення, і подивитися, скільки разів породжується кожен результат. У прикладі з кутовим ударом на рисунок 2.9 программа виконувалась багато разів, подаючи на вхід такі дані: високий центральний нападаючий, недосвідчений воротар і сильний вітер. Оскільки в 1/4 прогонів вийшов гол, можна сказати, що при цих вхідних даних імовірність голу дорівнює 25%.

Однак принадність імовірнісного програмування полягає в тому, що його можна з тим же успіхом використовувати для будь-яких видів імовірнісних міркувань. Воно дозволяє не тільки передбачати майбутнє, але і виводити факти, що призвели до спостережуваних результатів; можна «відкрутити» програму назад і подивитися, які причини породили даний результат. Можна також застосувати програму в одній ситуації, навчитися на отриманому результаті і використовувати нову програму для отримання більш точних прогнозів в майбутньому. Імовірнісне програмування допомагає приймати будь-які рішення в рамках імовірнісних міркувань.

2.3 Використання бібліотеки TensorFlow Probability

Визначення імовірнісного програмування можна дати як статистику з використанням інструментів інформатики. На рисунку 2.10 зображено декілька підходів програмування. У лівій частині це звичайна інформатика. Написати програму, вказати значення своїх аргументів, то оцінити програму для отримання вихідного сигналу. Права частина ілюструє підхід, застосовуваний до моделювання в статистиці: почніть з виведення, спостережень або даних Y, потім вкажіть абстрактну генеративную модель $p(X, Y)$, часто позначається математично, і, нарешті, використовуйте методи алгебри і виведення для характеризують апостеріорне розподіл $p(X \mid Y)$ невідомих величин в моделі з урахуванням спостережуваних величин. У той час як в імовірнісному програмуванні: мова програмування для визначень моделей та алгоритмів статистичного висновку для обчислення умовного розподілу вхідних даних програми, які могли привести до спостережуваного виходу програми.

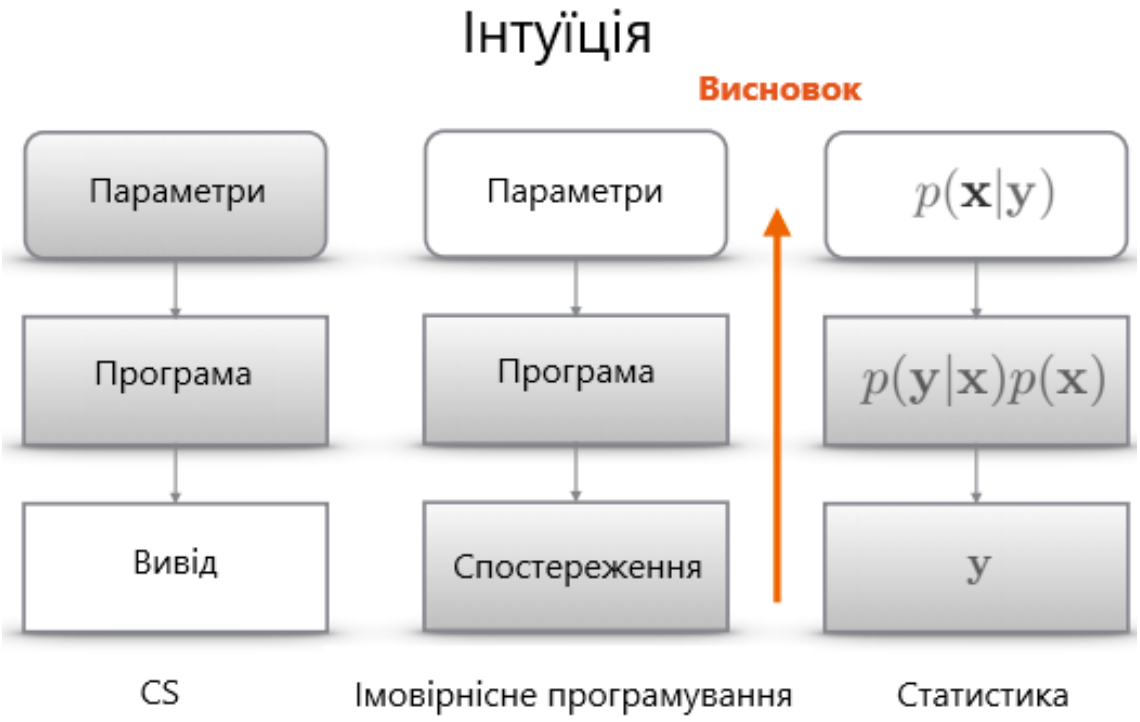


Рисунок 2.10 – Імовірнісна програма визначає процес випадкової генерації вихідних даних по відомим вхідним

Тож імовірнісне програмування – це не про написання програмного забезпечення, яке поводиться ймовірнісно. Щоб реалізувати таку архітектуру, Tensorflow вводить ймовірність Tensorflow [40].

TensorFlow Probability – це бібліотека для імовірнісних міркувань та статистичного аналізу в TensorFlow. Як частина екосистеми TensorFlow, TensorFlow Probability забезпечує інтеграцію ймовірнісних методів з глибокими мережами, висновок на основі градієнта за допомогою автоматичної диференціації та масштабованість до великих наборів даних та моделей за допомогою апаратного прискорення (наприклад, графічних процесорів) та розподілених обчислень. Почнемо з імпорту необхідних модулів.

Tfp.distributions.Distribution – це клас із двома основними методами: *sample* та *log_prob*. Цей клас містить багато розподілів, які можна побачити, написавши:

print_subclasses_from_module(tfp.distributions, tfp.distributions.Distribution)

Проаналізуємо, як відібрати нормальний розподіл, який є хорошим початком у *stat101*, використовуючи tf-ймовірність (рис. 2.11).

```
# A standard normal
# mean=0, std=3samples = normal.sample(1000)
normal = tfd.Normal(loc=0., scale=1.)
sns.distplot(samples)
plt.title("Samples from a standard Normal")
plt.show()'''
log of the probability density/mass function
evaluated at the given sample value.
'''
print("log(PDF):",normal.log_prob(0.))
```

Рисунок 2.11 – Підбір нормального розподілу використовуючи tf-ймовірність

Далі для обчислення інших статистичних параметрів, таких як кумулятивна функція розподілу та множинні розподіли, все ще можна використовувати власні класи tf-ймовірностей (рис. 2.12).

```
# Define a single scalar Normal distribution.
dist = tfd.Normal(loc=0., scale=3.)
# mean=0, std=3# Evaluate the cdf at 1, returning a scalar.
dist.cdf(1.)# Define a batch of two scalar valued Normals.
# The first has mean 1 and standard deviation 11, the second 2 and 22.
dist = tfd.Normal(loc=[1, 2.], scale=[11, 22.])
# Evaluate the pdf of the first distribution on 0, and the second on 1.5,
# returning a length two tensor.
dist.prob([0, 1.5])# Get 3 samples, returning a 3 x 2 tensor.
dist.sample([3])
```

Рисунок 2.12 – Обчислення CDF та множинних звичайних розподілів

Використовуючи наведений вище код, можно обчислювати CDF та множинні звичайні розподіли на льоту. Використовуючи статистичні

інструменти у своєму проекті, також може знадобитися оголошення багатовимірних розподілів, що tf-ймовірність охоплює (рис. 2.13).

```
mvn = tfd.MultivariateNormalDiag(loc=[0., 0.], scale_diag = [1., 1.])
print("Batch shape:", mvn.batch_shape)
print("Event shape:", mvn.event_shape)samples = mvn.sample(1000)
print("Samples shape:", samples.shape)g =
    sns.jointplot(samples[:, 0], samples[:, 1], kind='scatter')
plt.show()
```

Рисунок 2.13 – Вивід багатовимірних розподілів

У модулі *tfp* є безліч розподілів: *LogNormal*, *Logistic*, *LogitNormal Mixture*, *Multinomial*, *MultivariateNormalDiag*, щоб назвати декілька. Кожен розподіл має безліч статистичних висновків та функцій.

Бієктори представляють собою зворотні, плавні функції [41]. Вони можуть бути використані для перетворення розподілів, зберігаючи можливість брати вибірки та обчислювати *log_probs*. До них можна отримати доступ з модуля *tfp.bijectors*. Кожен бієктор реалізує принаймні 3 методи: *forward*, *inverse* та (принаймні) один із *forward_log_det_jacobian* та *inverse_log_det_jacobian*.

Використовуючи код: *print_subclasses_from_module(tfp.bijectors, tfp.bijectors.Bijector)*, можна перерахувати всі функції бієктора, доступні з tf-ймовірністю. А тепер подивимося, як оголосити *NormalCDF* за допомогою бієкторів та обчислити журнал детермінанта Jacobian в прямому розповсюдженні (рис. 2.14).

```
normal_cdf = tfp.bijectors.NormalCDF()
xs = np.linspace(-4., 4., 200)
plt.plot(xs, normal_cdf.forward(xs))
plt.show()plt.plot(xs,
    normal_cdf.forward_log_det_jacobian(xs, event_ndims=0))
plt.show()
```

Рисунок 2.14 – Вивід *NormalCDF*

Бієктори в основному використовуються для перетворення розподілів, як на рисунку 2.15.

```
# creates a Y=g(X)=exp(X) transform
exp_bijector = tfp.bijectors.Exp()
# declare a Normal Distribution and Transform it
log_normal = exp_bijector(tfd.Normal(0., .5))
samples = log_normal.sample(1000)
xs = np.linspace(1e-10, np.max(samples), 200)
sns.distplot(samples, norm_hist=True, kde=False)
plt.plot(xs, log_normal.prob(xs), c='k', alpha=.75)
plt.show()
```

Рисунок 2.15 – Застосування трансформації

Після запуску вищезазначеного фрагмента побачимо трансформований розподіл та графік оцінки щільності ймовірності. Отже, на даному етапі успішно впроваджено будівельні блоки статистичного висновку в Tensorflow Probability [42-44].

TensorFlow-Probability (TFP) має вбудовану підтримку для встановлення та прогнозування за допомогою структурних моделей часових рядів. Ця підтримка включає баєсівський висновок моделі. Оскільки вони вбудовані в TensorFlow, ці методи, природно, використовують переваги векторизованого обладнання (GPU і TPU), можуть ефективно обробляти багато часових рядів паралельно і можуть бути інтегровані в глибокі нейронні мережі.

3 РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

3.1 Опис даних

В роботі застосовані байєсовські методи до практичної задачі, щоб показати наскрізний байесовский аналіз, який переходить від побудови моделей до виявлення апріорних ймовірностей, а далі до реалізації в Python остаточного апостеріорного розподілу.

Байєсовські моделі також відомі як імовірнісні моделі бо вони побудовані з використанням імовірностей. І байєсовські імовірності використовуються в якості інструменту для кількісної оцінки невизначеності. Таким чином, отримані відповіді розподілів не точкові оцінки.

У роботі застосовуються байєсовські методи до набору даних про ціни на квитки на іспанські високошвидкісні залізниці (рис. 3.1).

	insert_date	origin	destination	start_date	end_date	train_type	price	train_class	fare
420104	2019-04-22 08:00:25	MADRID	SEVILLA	2019-04-28 08:30:00	2019-04-28 11:14:00	ALVIA	NaN	Turista	Flexible
431888	2019-04-22 10:03:24	MADRID	VALENCIA	2019-05-20 06:45:00	2019-05-20 08:38:00	AVE	21.95	Turista	Promo
791293	2019-04-25 19:19:46	MADRID	SEVILLA	2019-05-29 06:20:00	2019-05-29 09:16:00	AV City	38.55	Turista	Promo

Рисунок 3.1 – Приклад набору данних

Такий набір даних можна імпортувати завдяки коду, показаному на рисунку 3.2.

```
from scipy import stats
import arviz as az
import numpy as np
import matplotlib.pyplot as plt
import pymc3 as pm
import seaborn as sns
import pandas as pd
from theano import shared
from sklearn import preprocessingprint(
    'Running on PyMC3 v{}'.format(pm.__version__))
data = pd.read_csv('renfe.csv')
data.drop('Unnamed: 0', axis = 1, inplace=True)
data = data.sample(frac=0.01, random_state=99)
data.head(3)
```

Рисунок 3.2 – Імпорт даних

3.2 Гаусові висновки та моделі

KDE має ділянку рейкової ціни квиток і показує гаусовий розподіл (рис. 3.3), як, за винятком порядку кілька десятків точок даних, які знаходяться далеко від середнього.

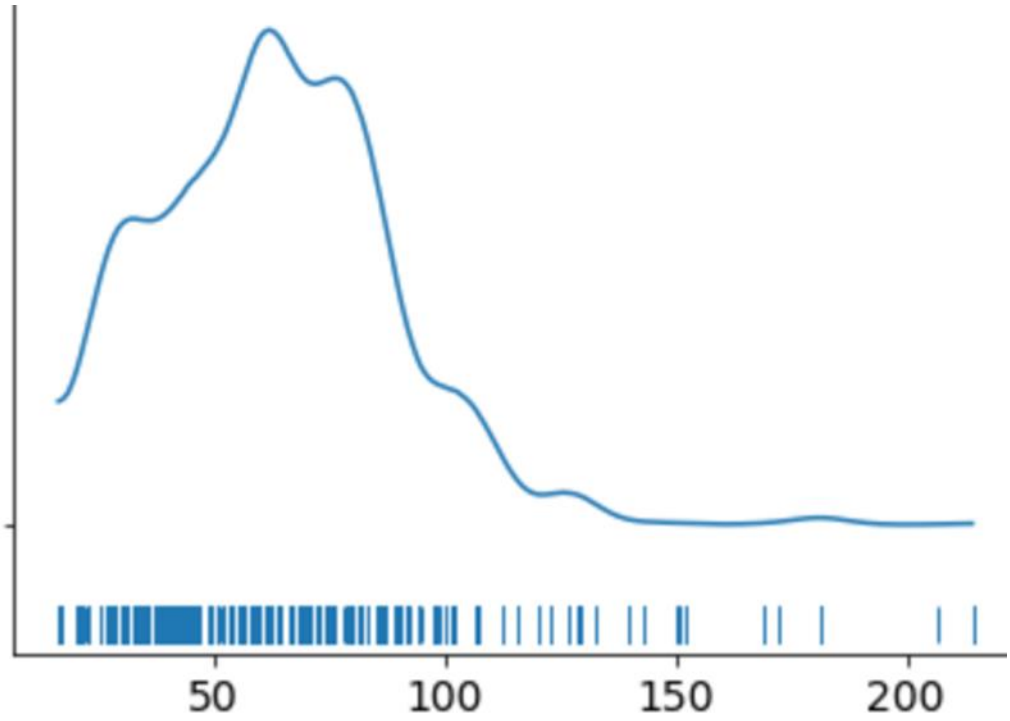


Рисунок 3.3 – Ілюстрація до гаусового висновку

Припустимо, що розподіл Гауса є правильним описом ціни залізничного квитка. Оскільки не відомо середнього або стандартного відхилення, треба встановити апріорні значення для них обох. Отже, розумною моделлю може бути наступне.

Виконаємо гаусів висновок даних про ціни на квитки. Ось деякі з можливих варіантів моделювання.

Створимо екземпляри моделей в PyMC3 наступним чином:

Вибір пріорі:

– μ, середнє значення за сукупністю. Поширення нормальне, дуже широке. Можливих значення μ неідомі, можна встановити апріорні значення,

що відображають незнання. З досвіду зрозуміло, що ціна квитка на потяг не може бути нижче 0 або вище 300, тому встановлюються кордони рівномірного розподілу рівними 0 і 300. Може бути різний досвід і різні кордони. Це абсолютно нормально;

– σ, стандартне відхилення генеральної сукупності. Може бути тільки позитивним, тому використовується HalfNormal distribution. Знову ж таки, дуже широкий.

При виборі функції ймовірності ціни квитка спостерігається змінна, що представляє дані, отримані з нормального розподілу з параметрами μ і σ; Далі малюється 1000 задніх зразків, використовуючи вибірку NUTS.

Використовуючи PyMC3, можна написати модель наступним чином: *model_g.py*.

Це спосіб, яким повідомляється PyMC3, що треба встановити умову для невідомого в відомих (даних).

Будуємо графік гаусівської моделі (рис. 3.4). Це працює на графіку Theano. Для цього виконуємо наступний код: *az.plot_trace (trace_g).*

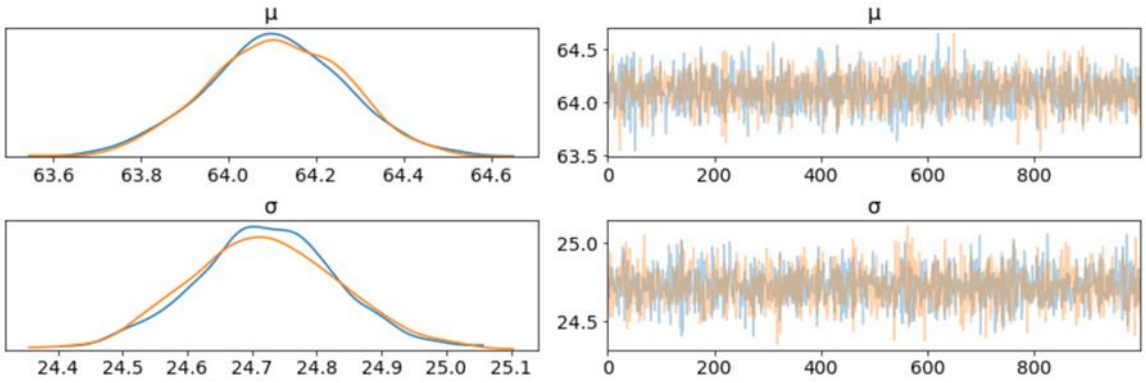


Рисунок 3.4 – Графік гаусівської моделі

Зліва маємо графік KDE – для кожного значення параметра на осі *x* отримуємо ймовірність на осі *y*, яка повідомляє, наскільки ймовірно це значення параметра.

Праворуч отримуємо окремі значення вибірки на кожному кроці під час вибірки. З графіка сліду можно візуально отримати правдоподібні значення від апостеріорного.

На наведеному вище графіку є один рядок для кожного параметра. Для цієї моделі апостеріорі є двовимірним, тому на наведеному вище рисунку показані граничні розподіли кожного параметра.

Тут слід зазначити кілька моментів:

1. Ланцюжки вибірки для окремих параметрів (зліва) здаються добре сходяться і стаціонарними (немає великих відхилень або інших дивних закономірностей).

2. Максимальна апостериорная оцінка кожної змінної (пік в лівому розподілі) дуже близька до істинним параметрам.

Тоді можна побудувати спільний розподіл параметрів (рис. 3.5), виконави наступний код: *az.plot_joint (trace_g, kind = 'kde', fill_last = False).*

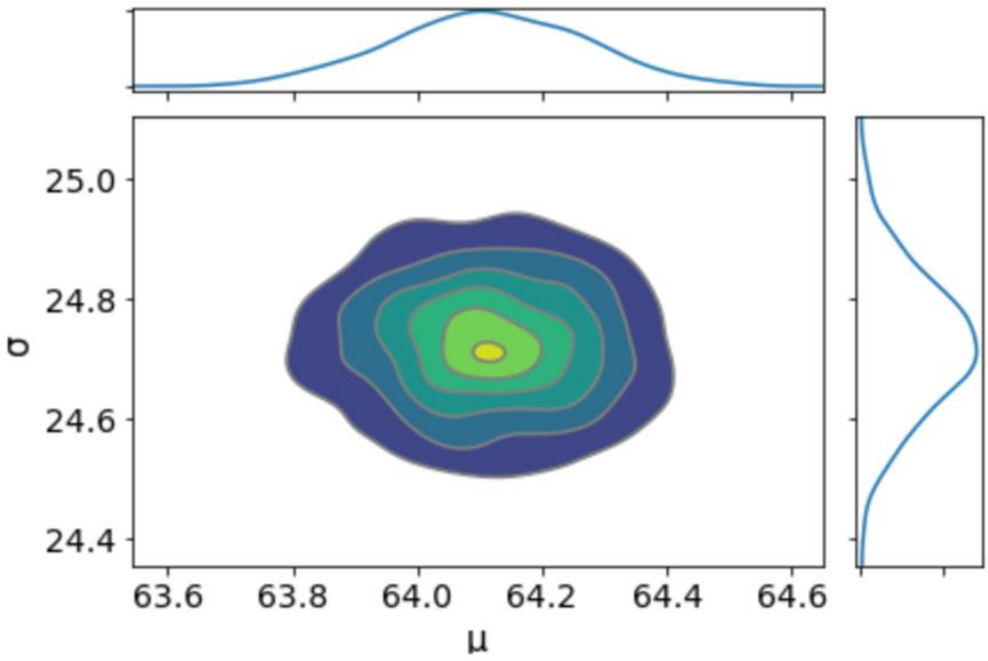


Рисунок 3.5 – Спільний розподіл параметрів

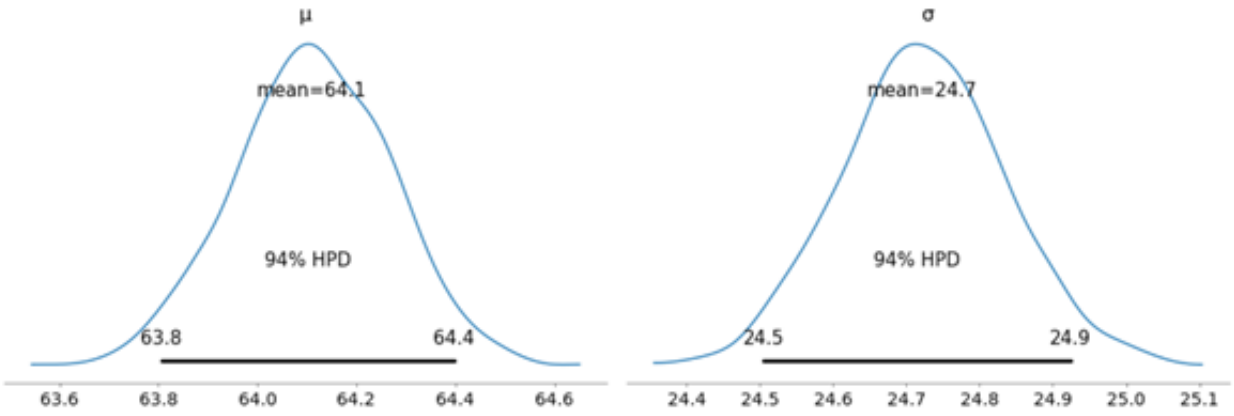
Так, на рисунку 3.5 не видно кореляції між цими двома параметрами. Це означає, що у нас, ймовірно, немає колінеарності у моделі. Це добре.

Також можна отримати детальні зведення апостеріорного розподілу для кожного параметра (рис. 3.6).

	mean	sd	hpd_3%	hpd_97%	mcse_mean	mcse_sd	ess_mean	ess_sd	ess_bulk	ess_tail	r_hat
μ	64.109	0.158	63.807	64.398	0.003	0.002	2058.0	2058.0	2063.0	1576.0	1.0
σ	24.721	0.113	24.505	24.924	0.002	0.002	2058.0	2056.0	2054.0	1367.0	1.0

Рисунок 3.6 – Детальні зведення

На рисунку 3.7 можна побачити наведену вище зведення візуально, побудувавши графік із середньою і максимальною апостеріорної щільністю (HPD) розподілу, а також інтерпретувати і повідомити результати байєсівського виведення: *az.plot_posterior (трасировка_g).*



а) б)

Рисунок 3.7 – Розподіли: а) розподіл з mean=64,1; б) розподіл з mean=24,7

На відміну від виведення Frequentist, у висновку Bayesian отримуємо повний розподіл значень. Кожного разу, коли ArviZ обчислює і повідомляє HPD, за замовчуванням він буде використовувати значення 94%. Зверніть увагу, що інтервали HPD – це не те ж саме, що довірчі інтервали. Тут можно інтерпретувати як такі, що існує 94% імовірності, що віра знаходиться між 63,8 євро і 64,4 євро для середньої ціни квитка.

Формально збіжність ланцюгів можна перевірити за допомогою тесту Гельмана-Рубіна (рис. 3.8). Значення, близькі до 1,0 означають збіжність.

```
pm.gelman_rubin (trace_g)
bfmi = pm.bfmi (trace_g)
max_gr = max (np.max (gr_stats) for
gr_stats в pm.gelman_rubin (trace_g) .values ())
(pm.energyplot (trace_g, legend = False, figsize = (6, 4))
.set_title ("BFMI = {} \ nGelman-Rubin = {}"
.format (bfmi, max_gr)));

```

Рисунок 3.8 – Тест Гельмана-Рубіна

Тоді отримаємо графік приведений на рисунку 3.9.

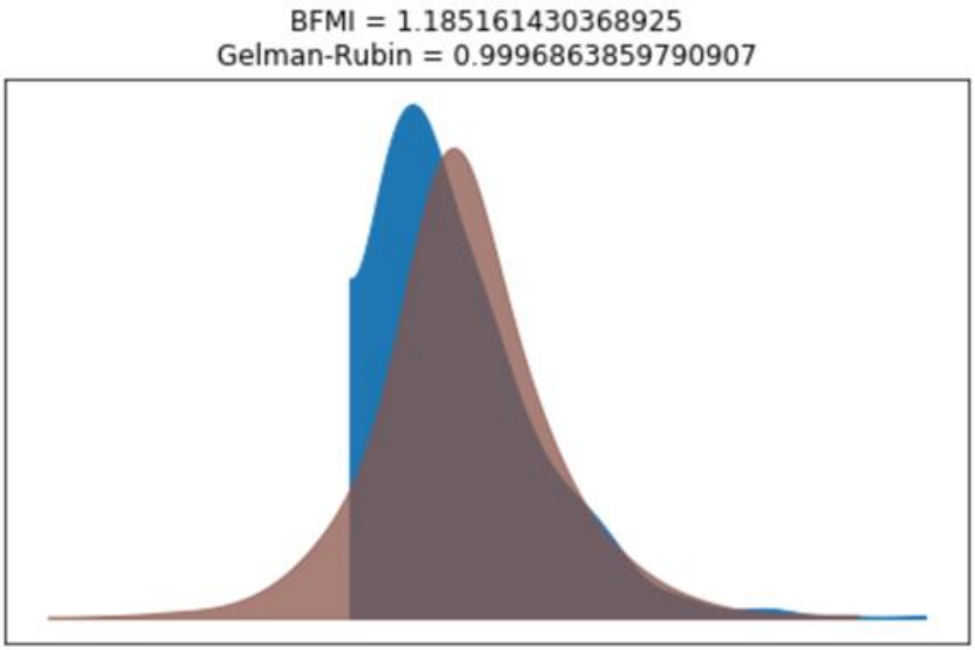


Рисунок 3.9 – Статистика Гельмана-Рубіна

Наша модель добре сходиться, і статистика Гельмана-Рубіна виглядає нормально (рис. 3.9).

3.3 Апостеріорні прогностичні перевірки

Апостеріорні прогностичні перевірки – відмінний спосіб перевірити модель. Ідея полягає в тому, щоб генерувати дані з моделі з використанням параметрів з апостеріорного креслень.

Тепер, коли апостеріорна оцінка вирахувана, проілюструємо, як використовувати результати моделювання для отримання прогнозів.

Наступна функція буде випадковим чином витягувати 1000 вибірок параметрів з траси. Потім для кожної вибірки буде відібрано 25798 випадкових чисел з нормального розподілу, визначеного значеннями μ і σ в цій вибірці (рис. 3.10).

```
1  ppc = pm.sample_posterior_predictive(
2      trace_g, samples=1000, model=model_g)
3  np.asarray(ppc['y']).shape
```

Рисунок 3.10 – Функція, яка випадковим чином витягує 1000 вибірок

Отримуємо (1000, 25798). Тепер в ppc міститься 1000 згенерованих наборів даних (що містять по 25798 вибірок кожна), кожна з яких використовує значення параметра, відмінне від апостеріорного. Отримаємо апостеріорне прогнозування середнього значення (рис. 3.11).

```
1  _, ax = plt.subplots (figsize = (10, 5))
2  ax.hist ([y.mean () for
3      y in ppc ['y']], bins = 19, alpha = 0.5)
4  ax.axvline (data .price.mean ())
5  ax.set (title = 'Апостеропное прогнозування среднього',
6  xlabel = 'mean (x)', ylabel = 'Частота');
7
```

Рисунок 3.11 – Апостеріорне прогнозування

Передбачуване середнє значення дуже близько до фактичного середнього значення ціни на залізничний квиток (рис. 3.12).

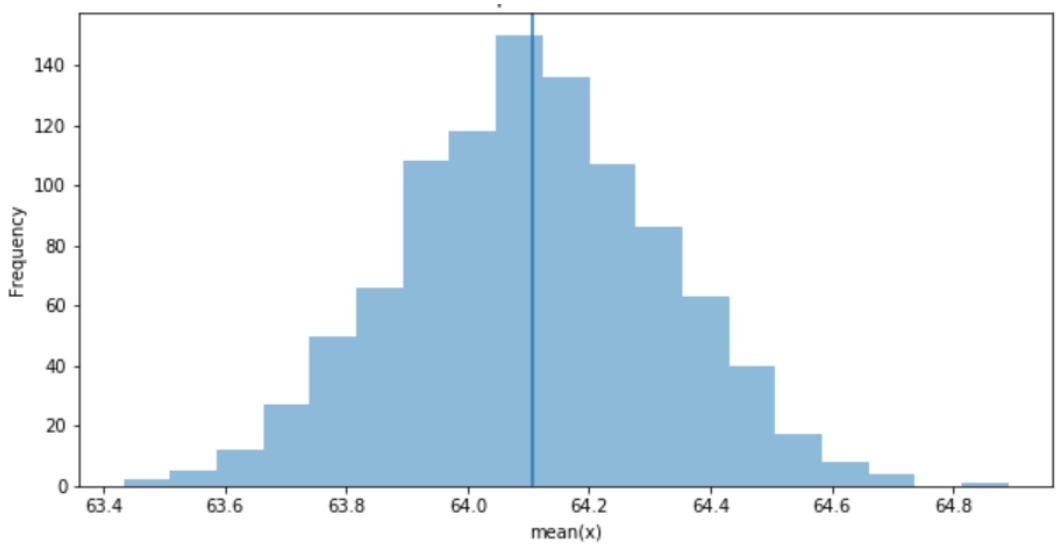


Рисунок 3.12 − Гістограма передбачуваного середнього значення

ВИСНОВКИ

У ході роботи були досліджені деякі фреймфорки та мови імовірнісного програмування, порівняння імовірнісного програмування з классичним програмуванням. Також була досліджена математична залежність імовірнісного програмування, а саме використання байєсівської лінійної регресії.

У досліджені показано, як багаторівнева ієрархічна байєсівська модель відпрацьовує, коли у існує кілька наборів вимірювань, які, як очікується, будуть мати схожість.

Наївний підхід або об’єднує всі дані разом і ігнорує індивідуальні відмінності, або розглядає кожен набір як повністю окремий, що призводить до зашумлених оцінок.

Припускаючи, що кожен окремий набір даних розподілений згідно групового розподілу, який одночасно оцінюється, отримуємо вигоду від збільшення статистичної потужності та інтелектуальної регуляризації за рахунок ефекту стиснення, тоді імовірнісне програмування в PyMC3 робить байєсівську оцінку моделі тривіальною.

В результаті практичного дослідження завдяки апостеріорній прогностичній перевірці було виявлено, що передбачуване середнє значення дуже близько до фактичного середнього значення щодо аналізу ціни на залізничний квиток.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The patient organism modeling for diagnosis with the usage of a multi agent representation /Kuzomin, O., Dudka, O., Vasylenko, O., Lyashenko, V. // International Journal of Emerging Trends in Engineering Research, 2020, 8(9), pp. 5733-5739

2. Mobile expert system for diagnostic human state in emergency situations./ Kuzomin, O., Dudka, O., Vasylenko, O., Shylo, R., Lyashenko, V.// International Journal of Advanced Trends in Computer Science and Engineering, 2020, 9(4), pp. 6484-6489, 334

3. Using of ontologies for building databases and knowledge bases for consequences management of emergency/Kuzomin, O., Dudka, O., Vasylenko, O., Radchenko, V., Lyashenko, V.//International Journal of Advanced Trends in Computer Science and Engineering, 2020, 9(4), pp. 5040-5045

4. Intelligent geoinformatic expert system for providing emergency help during extreme situations./Kuzomin, O., Stukin, M., Bozhkov, D.//International Multidisciplinary Scientific GeoConference Surveying Geology and Mining Ecology Management, GEM, 2019, 18(2.2), pp. 269-276

5. Kuzomin, Bohdan Maliar //International Journal "Information Models and Analyses" Volume 7, Number 4. PP.2019. 327 – 338.

6. Research of medical diagnostic data search methods / Oleksii Vasilenko, Oleksandr Kuzomin, Oleksandr Shapoval // International Journal "Information Models and Analyses" Volume 7, Number 4. PP.2019. 339 – 349.

7. Intellectual Models and Means of the Biometric System Dynamics of Rinosinusite / Oleksii Vasilenko, Oleksandr Kuzomin, Tatyana Khripushina // International Journal "Information Models and Analyses" Volume 7, Number 4. PP.2019. 350 – 361.

8. Forming Medical Database and Knowledge for Diagnostic Disease / Oleksii Vasilenko, Oleksandr Kuzomin Vladislav Shvets. //International Journal

"Information Models and Analyses" Volume 7, Number 4. PP.2019. 362 – 372.

9. Automated Tests for Errors in Computer System ‘Environment’/ Oleksii Vasilenko, Oleksandr Kuzomin. // International Journal"Information Models and Analyses" Volume 7, Number 4. PP.2019. 373 – 384.

10. Developing methods based on text mining technology to Improve the quality and speed of automatic clustering of Documents / Oleksii Vasilenko, Oleksandr Kuzomin, Artem Mertsalov // International Journal "Information Models and Analyses" Volume 7, Number 4.PP.2019. 385 – 397.

11. Jaynes, Edwin T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. ISBN 0-521-59271-2. (англ.)

12. Bessière, P.; Mazer, E.; Ahuactzin, J-M.; Mekhnacha, K. (2013). *Bayesian Programming*. Chapman & Hall/CRC. ISBN 9781439880326. (англ.)

13. Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME--Journal of Basic Engineering* **82**: 33—45. doi:10.1115/1.3662552. (англ.)

14. Bessière, P.; Laugier, C. & Siegwart, R. (2008). *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*. Springer. ISBN 978-3-540-79007-5. (англ.)

15. Lebeltel, O.; Bessière, P.; Diard, J.; Mazer, E. (2004). Bayesian Robot Programming. *Advanced Robotics* **16** (1): 49—79. doi:10.1023/b:auro.0000008671.38949.43. (англ.)

16. Diard, J.; Gilet, E.; Simonin, E.; Bessière, P. (2010). Incremental learning of Bayesian sensorimotor models: from low-level behaviours to large-scale structure of the environment. *Connection Science* **22** (4): 291—312. doi:10.1080/09540091003682561.(англ.)

17. Pradalier, C.; Hermosillo, J.; Koike, C.; Braillon, C.; Bessière, P.; Laugier, C. (2005). The CyCab: a car-like robot navigating autonomously and

safely among pedestrians. *Robotics and Autonomous Systems* **50** (1): 51—68. doi:10.1016/j.robot.2004.10.002. (англ.)

18. Ferreira, J.; Lobo, J.; Bessière, P.; Castelo-Branco, M.; Dias, J. (2012). A Bayesian Framework for Active Artificial Perception. *IEEE Transactions on Systems, IEEE Transactions on Systems, Man, and Cybernetics, Part B* **99**: 1—13. (англ.)

19. Ferreira, J. F.; Dias, J. M. (2014). *Probabilistic Approaches to Robotic Perception*. Springer. (англ.)

20. Mekhnacha, K.; Mazer, E.; Bessière, P. (2001). The design and implementation of a Bayesian CAD modeler for robotic applications. *Advanced Robotics* **15** (1): 45—69. doi:10.1163/156855301750095578. (англ.)

21. Coué, C.; Pradalier, C.; Laugier, C.; Fraichard, T.; Bessière, P. (2006). Bayesian Occupancy Filtering for Multitarget Tracking: an Automotive Application. *International Journal of Robotics Research* **25** (1): 19—30. doi:10.1177/0278364906061158. (англ.)

22. Vasudevan, S.; Siegwart, R. (2008). Bayesian space conceptualization and place classification for semantic maps in mobile robotics. *Robotics and Autonomous Systems***56** (6): 522—537. doi:10.1016/j.robot.2008.03.005. (англ.)

23. Perrin, X.; Chavarriaga, R.; Colas, F.; Seigwart, R.; Millan, J. (2010). Brain-coupled interaction for semi-autonomous navigation of an assistive robot. *Robotics and Autonomous Systems* **58** (12): 1246—1255. doi:10.1016/j.robot.2010.05.010. (англ.)

24. Rett, J.; Dias, J.; Ahuactzin, J-M. (2010). Bayesian reasoning for Laban Movement Analysis used in human-machine interaction. *Int. J. of Reasoning-based Intelligent Systems* **2** (1): 13—35. doi:10.1504/IJRIS.2010.029812. (англ.)

25. Möbus, C.; Eilers, M.; Garbe, H.; Zilinski, M. (2009). Probabilistic and Empirical Grounded Modeling of Agents in (Partial) Cooperative Traffic Scenarios. У Duffy, Vincent G. *Digital Human Modeling*. Lecture Notes in Computer Science, Volume 5620. Second International Conference, ICDHM 2009,

San Diego, CA, USA: Springer. с. 423–432. ISBN 978-3-642-02808-3. doi:10.1007/978-3-642-02809-0_45. (англ.)

26. Möbus, C.; Eilers, M. (2009). Further Steps Towards Driver Modeling according to the Bayesian Programming Approach. У Duffy, Vincent G. *Digital Human Modeling*. Lecture Notes in Computer Science, Volume 5620. Second International Conference, ICDHM 2009, San Diego, CA, USA: Springer. с. 413–422. ISBN 978-3-642-02808-3. doi:10.1007/978-3-642-02809-0_44. (англ.)

27. Eilers, M.; Möbus, C. (2010). Lernen eines modularen Bayesian Autonomous Driver Mixture-of-Behaviors (BAD MoB) Modells. У Kolrep, H.; Jürgensohn, Th. *Fahrermodellierung – Zwischen kinematischen Menschmodellen und dynamisch-kognitiven Verhaltensmodellen*. Fortschrittsbericht des VDI in der Reihe 22 (Mensch-Maschine-Systeme). Düsseldorf, Germany: VDI-Verlag. с. 61 – 74. ISBN 978-3-18-303222-8. (нім.)

28. Möbus, C.; Eilers, M. (2011). Prototyping Smart Assistance with Bayesian Autonomous Driver Models. У Mastrogiovanni, F.; Chong, N.-Y. *Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives*. Hershey, Pennsylvania (USA): IGI Global publications. с. 460–512. ISBN 9781616928575. doi:10.4018/978-1-61692-857-5.ch023. (англ.)

29. Eilers, M.; Möbus, C. (2011). Learning the Relevant Percepts of Modular Hierarchical Bayesian Driver Models Using a Bayesian Information Criterion. У Duffy, V.G. *Digital Human Modeling*. LNCS 6777. Heidelberg, Germany: Springer. с. 463–472. ISBN 978-3-642-21798-2. doi:10.1007/978-3-642-21799-9_52. (англ.)

30. Eilers, M.; Möbus, C. (2011). Learning of a Bayesian Autonomous Driver Mixture-of-Behaviors (BAD-MoB) Model. У Duffy, V.G. *Advances in Applied Digital Human Modeling*. LNCS 6777. Boca Raton, USA: CRC Press, Taylor & Francis Group. с. 436–445. ISBN 978-1-4398-3511-1. (англ.)

31. Le Hy, R.; Arrigoni, A.; Bessière, P.; Lebetel, O. (2004). Teaching Bayesian Behaviours to Video Game Characters. *Robotics and Autonomous Systems* **47** (2–3): 177—185. doi:10.1016/j.robot.2004.03.012. (англ.)

32. Synnaeve, G. (2012). *Bayesian Programming and Learning for Multiplayer Video Games*. (англ.)

33. Colas, F.; Droulez, J.; Wexler, M.; Bessière, P. (2008). A unified probabilistic model of the perception of three-dimensional structure from optic flow. *Biological Cybernetics*: 132—154. (англ.)

34. Laurens, J.; Droulez, J. (2007). Bayesian processing of vestibular information. *Biological Cybernetics* **96** (4): 389—404. doi:10.1007/s00422-006-0133-1. (англ.)

35. Colas, F.; Flacher, F.; Tanner, T.; Bessière, P.; Girard, B. (2009). Bayesian models of eye movement selection with retinotopic maps. *Biological Cybernetics* **100** (3): 203—214. doi:10.1007/s00422-009-0292-y. (англ.)

36. Serkhane, J.; Schwartz, J-L.; Bessière, P. (2005). Building a talking baby robot A contribution to the study of speech acquisition and evolution. *Interaction Studies* **6** (2): 253—286. doi:10.1075/is.6.2.06ser. (англ.)

37. Moulin-Frier, C.; Laurent, R.; Bessière, P.; Schwartz, J-L.; Diard, J. (2012). Adverse conditions improve distinguishability of auditory, motor and percep-tuo-motor theories of speech perception: an exploratory Bayesian modeling study. *Language and Cognitive Processes* **27** (7–8): 1240—1263. doi:10.1080/01690965.2011.645313. (англ.)

38. Gilet, E.; Diard, J.; Bessière, P. (2011). Bayesian Action–Perception Computational Model: Interaction of Production and Recognition of Cursive Letters. У Sporns, Olaf. *Plos ONE* **6** (6): e20387. Bibcode:2011PLoSO...620387G. doi:10.1371/journal.pone.0020387. (англ.)

39. Zadeh, Lofti, A. (1965). Fuzzy sets. *Information and Control* **8** (3): 338—353. doi:10.1016/S0019-9958(65)90241-X. (англ.)

40. Zadeh, Lofti, A. (1975). Fuzzy logic and approximate reasoning. *Synthese* **30** (3—4): 407—428. doi:10.1007/BF00485052. (англ.)

41. Dubois, D.; Prade, H. (2001). Possibility Theory, Probability Theory and Multiple-Valued Logics: A Clarification. *Ann. Math. Artif. Intell.* **32** (1—4): 35—66. doi:10.1023/A:1016740830286. (англ.)

42. Poole, D. (1993). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence* **64**: 81–129. doi:10.1016/0004-3702(93)90061-F. (англ.)

43. Poole, D. (1997). The Independent Choice Logic for modelling multiple agents under uncertainty. *Artficial Intelligence* **94**: 7–56. doi:10.1016/S0004-3702(97)00027-1. (англ.)

44. Sato, T.; Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* **15**: 391––454. Архів оригіналу за 12 липня 2014. Процитовано 18 жовтня 2015. (англ.)