**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ”**

**ФАКУЛЬТЕТ ІНФОРМАТИКИ**

**КАФЕДРА МАТЕМАТИКИ**

**ЗВІТ**

**з дослідницької практики на тему:**

**“Random forest”**

Студента 3-го курсу

спеціальності 113

“Прикладна математика”

Вітіска Владислава Олександровича

**Київ – 2024**

**Анотація**

Практика присвячена дослідженню методу машинного навчання Random forest.

Вперше ансамблевий алгоритм на основі дерев прийняття рішень, що носив назву «Random decision forests», був розроблений та описаний вченим IBM Тіном Кам Хо у 1995, після чого алгоритм був покращений Лео Брейманом і Аделем Катлером, що і дали йому звичну назву «Random Forest», яку запатентували. Сьогодні випадковий ліс – один з найпопулярніших методів як для задач класифікації, так і для регресії.

У даній роботі розглядається Random forest, його теоретична та практична сторона. Протягом практики було розглянуто створення моделей на основі набору даних для передбачення ядовитості грибу за його зовнішніми характеристиками. Прикладом для подібної роботи та матеріалом з більш загальним поясненням роботи методу була робота Md Atikur Rahman, Akihiko Hokugo Nobuhito Ohtsu “Household evacuation preparation time during a cyclone: Random Forest algorithm and variable degree analysis” (2021). Також оглядово було розглянуто порівняння результатів інших методів машинного навчання з алгоритмом випадкового лісу.

У першому розділі звіту розглядаються основні поняття та принципи роботи дерев прийняття рішень. У другому розділі та його підпунктах досліджуються принципи, що застосовуються для бінарних дерев для утворення методу випадкового лісу. В третьому розділі було створено різні моделі для дослідження якості передбачення Random Forest та впливу гіперпараметрів на вихідні передбачення та порівняно різні алгоритми.

**Розділ 1. Дерева прийняття рішень**

**Означення 1.1.** [1] Деревом називається граф G = (V, E), у якому довільні дві вершини (або вузли) з'єднані рівно одним шляхом.

**Означення 1.2.** [1]Деревом з коренем називається дерево, в якому одна з вершин позначена як корінь. У нашому випадку ми додатково припускаємо, що кореневе дерево є орієнтованим графом, де всі ребра спрямовані від кореня.

**Означення 1.3.** [1] Якщо існує ребро з в (тобто, якщо (, ) ∈ E), то кажуть, що вершина є батьківською для вершини , а вершина є нащадком вершини .

**Означення 1.4.** [1] У кореневому дереві вершина називається внутрішньою, якщо вона має одного або більше нащадків, і термінальною, якщо вона не має нащадків. Термінальні вершини також називають листям.

**Означення 1.5.** [1] Бінарним деревом називається кореневе дерево, у якого усі внутрішні вершини мають рівно двох нащадків.

Використовуючи вище наведені означення, ми можемо визначити дерево прийняття рішень, як модель , що представлена кореневим деревом (часто бінарним, але не обов'язково), де будь-яка вершина цього дерева t представляє підпростір  вхідного простору, з кореневою вершиною , що відповідає самому . Внутрішні вершини позначено поділом , взятим з набору запитань Q. Вона розбиває простір , який представляє вершина , на непересічні підпростори що відповідають кожному з його нащадків.

**Наприклад.**

Розглянемо задачу бінарної класифікації для точки з координатами , де точку віднесемо до класу c1 якщо вона задовольняє умову , інакше це буде точка класу c2.

Розглянемо [1] дерево прийняття рішень (рис. 1.1)

Зображення, що містить схема, ряд, коло

Автоматично згенерований опис

**рис. 1.1**

На рисунку маємо корінь , внутрішню вершину , та три термінальні вершини: , , . Не термінальні вершини та мають поділи:

* -
* -

Якщо розглядати утворені вершинами підрозділ, то наглядно це можна зрозуміти з рис. 1.2[1]

Зображення, що містить Барвистість, схема, Прямокутник, візерунок

Автоматично згенерований опис

**рис. 1.2**

У вершині нашим простором буде весь прямокутник, у вершині – підпростір обмежений червоним прямокутником, – обмежений зеленим прямокутником, після підпростір поділиться на дві частини, одна з яких обведена синім, що містить точки класу с1.

Оскільки ми розглянули простий приклад задачі з класифікації, варто сказати, що для термінальних вершин в задачах регресії результатом передбачення , що прив’язаний до відповідної термінальної вершини, буде не клас, як для задач класифікації, а дійсне число, яке буде найкращим припущенням.

Формально задача передбачення за допомогою дерева прийняття рішень може бути описана наступним алгоритмом-псевдокодом.

**Алгоритм 1.1**[1] Отримання результату передбачення в дереві прийняття рішень

1: **Функція** Передбачити()

2:

3: **поки** не є термінальною вершиною виконувати

4: вершина нащадок , така що

5: **закінчення блоку**

6: **повернути**

7: **закінчення функції**

Отже, ми розібрались як за допомогою дерева рішень зробити прогноз, але очевидно, що для задач машинного навчання подібне дерево не відоме одразу і є ціллю розробки моделі.

Суть побудови дерева прийняття рішень є створення моделі, що найкраще підходитиме для відомого тренувального набору , причому серед усіх дерев прийняття рішень можуть існувати кілька найкращих. В таких випадках згідно з принципом Бритви Оккама, обирається варіант з найменшою кількістю припущень, що в нашому випадку полягає в меншому розмірі дерева, або ж це те ж саме що і менша кількість внутрішніх вершин.

Знаходження найменшого дерева є NP-повною задачею. Як наслідок, не існує не існує ефективного алгоритму для знаходження , тим самим припускаємо, що знаходження близьких до оптимальних дерев рішень є найкращим рішенням, щоб утримати вимоги до обчислень у реалістичних межах.

Введемо загальне поняття забрудненості. Визначимо міру забрудненості i(t) як функцію, яка оцінює доброякісність будь-якої вершини t. Припустимо, що чим менше i(t), тим чистіша вершина і тим краще передбачення для усіх , де – підмножина навчальних прикладів що потрапляють до t. Починаючи з однієї вершини, наближене до оптимального дерева може бути отримане жадібним ітеративним методом розподіляючи вершини на кілька чистіших, до поки існують внутрішні вершини, які можна розбити на чистіші.

Жадібне припущення про те, що отримане дерево має бути якомога меншим, тобто мати гарне узагальнення, полягає в тому, щоб розбити кожну вершину t за допомогою розбиття s, яке локально максимізує зменшення забруднення у отриманих дочірніх вершинах.

Для формалізації введемо наступне означення

**Означення** Зменшення забруднення бінарного розподілу , що ділить вершину t на праву та ліву вершини, дорівнює

Де або – відношення кількості прикладів, що потрапили до дочірньої вершини або , до кількості прикладів що були у батьківській t.

Використовуючи це означення ми можемо записати алгоритм для побудови дерева прийняття рішень з бінарними розподілами .

**Алгоритм 2.** [1] Жадібна побудова бінарного дерева рішень.

1: **Функція** ПобудуватиРішенняДерева(L)

2: **Створити** дерево рішень з кореневим вузлом

3: **Створити** порожній стек S відкритих вузлів (t,)

4: S.push((,L))

5: **поки** S не порожній робити

6: t, = S.pop()

7: **якщо** для t виконується критерій зупинки тоді

8: = деяке постійне значення

9: **інакше**

10: Знайти розділення на , яке максимізує

s\* = arg max s∈Q Δi(s, t)

11: Розділити на і відповідно до s\*

12: Створити лівий дочірній вузол для t

13: Створити правий дочірній вузол для t

14: S.push((,))

15: S.push((,))

16: **кінець якщо**

17: **кінець поки**

18: **повернути**

19: **кінець функції**

Схожим чином можна реалізувати і алгоритм побудови дерева з N-арним розподілом, втім бінарного розподілу достатньо для подальшого розуміння в контексті методу випадкового лісу.

**Розділ 2. Random Forest**

У деревах прийняття рішень є значна проблема, що понижує якість вихідного передбачення: висока дисперсія. Втім цей метод має низький зсув.

**Означення 2.1** [3] Зсув (англ. bias) — це похибка, викликана помилковими припущеннями в алгоритмі навчання.

Розумний підхід до зменшення помилки прогнозування полягає у зменшенні дисперсії прогнозу за умови, що відповідний зсув може залишатися незмінним або не збільшуватися надто сильно. Рішенням цієї проблеми є ансамблеві методи. Зокрема, основний принцип ансамблевих методів, заснованих на рандомізації, полягає у введенні випадкових збурень у процедуру навчання, щоб створити кілька різних моделей з однієї навчальної вибірки L, а потім об'єднати прогнози цих моделей для формування прогнозу ансамблю.

Random forest – це сімейство ансамблевих методів машинного навчання, що використовує численні дерева прийняття рішень для створення прогнозів.

Розглянемо два ключові ансамблеві мета-алгоритми, що використовуються в Random Forest – одній з найпопулярнішиї варіацій сімейства відповідних методів, розробленій Лео Брейманом і Аделем Катлером.

**2.1 Метод Bagging або ж Bootstrap aggregating**

Суть Бутстрепової агрегації полягає у спеціальній вибірці набору даних з L для тренування дерева: з відповідного початкового тренувального набору розміру береться елементів з можливими повторами. Для випадку в середньому майже 37% тренувальних прикладів не потраплять до нового набору .

**2.2 Метод випадкових підпросторів**

Дослідник Тін Кам Хо запропонував при відборі даних для тренування нових дерев звертати увагу на відбір атрибутів, оскільки важливі параметри можуть привести до кореляції значної кількості дерев, що потенційно може вплинути на якість передбачень. Для задач класифікації зазвичай використовується можливих параметрів, а для регресії .

**2.3 Прийняття рішення**

Після утворення дерев прийняття рішення для утворення єдиного передбачення для задач регресії та класифікації використовуються різні методи: у першому випадку використовується середнє значення отриманих результатів, а в другому – обирається клас з найбільшою кількістю голосів.

Узагальнюючи, метод випадкових лісів для задачі класифікації можна зобразити [2] наступним чином(рис. 2.1)

Зображення, що містить текст, схема, знімок екрана, План

Автоматично згенерований опис

**рис. 2.1**

**Розділ 3. Практичне дослідження методу випадкових лісів**

**3.1 Знайомство з набором даних**

Оскільки основною ціллю дослідження є вивчення методу випадкового лісу було обрано [7] набір даних з задачею класифікації чи ядовитий гриб через його попередню обробку автором. Цей набір – очищений від порожніх значень та оброблений, щоб перевести категоріальні буквенні змінні до числових.

Маємо 9 колонок:

* Розмір шапки гриба
* Форма шапки
* Вид кріплення пластинки гриба
* Колір пластинки
* Висота ніжки
* Товщина ніжки
* Колір ніжки
* Пора року
* Чи ядовитий гриб – цільова колонка

До набору входить 54035 записів, а кожна колонка має кількість унікальних значень зображену на рис. 3.1 з розподілом вказаним на рис 3.2

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

**рис. 3.1**

Зображення, що містить знімок екрана

Автоматично згенерований опис

**рис. 3.2**

Варто уточнити, що під час обробки інформації автор набору даних перетворив пори року на число з комою, хоча унікальних значень залишилось чотири.

**3.2 Правила тренування Моделі**

Процес тренування буде відбуватися за допомогою бібліотеки scikit-learn в середовищі Jupiter Notebook на мові програмування Python. Код програми розміщений в Додатку А.

Під час роботи над моделями було висунуте припущення, що стандартні параметри моделі задані розробниками бібліотеки є оптимальними, і порівняння різних алгоритмів буде відбуватися для базових варіантів, за виключенням одного методу, про що буде написано в частині порівняння методів.

Для відтворюваності дослідження, оскільки метод випадкового лісу, а саме алгоритми Bootstrap Aggregating та метод випадкових підпросторів, ґрунтується на випадкових процесах, було згенероване значення 215 з діапазону [0, 1000], яке буде передаватись моделі як основа для випадкової генерації.

Для тренування відбиралось 80% даних, решта 20% використовувалась для тестування моделі. Для відтворення результатів було використане значення 0 для випадкової генерації розподілу.

**3.2.1 Методи порівняння результатів передбачення**

Як критерії оцінювання якості було обрано три статистики

1. [4] Точність – ступінь наближення ряду передбачень до справжніх значень.

Для задач класифікації в загальному вигляді можна записати наступну формулу:

1. [4] Влучність – ступінь наближення вимірювань один до одного.

Для задач бінарної класифікації описується наступним чином:

1. [5] F1-міра – середнє гармонійне влучності та повноти.

Формула:

**3.3 Дослідження випадкового лісу**

**3.3.1 Тренування базової моделі**

Після тренування моделі були отримані наступні результати:

* Точність - 0.9907467382252244;
* Влучність - 0.9907376220949815;
* F1-міра - 0.9915725602561941.

Ці результати будуть використані для подальших порівняннь.

**3.3.2 Дослідження впливу гіперпараметрів**

**Означення 3.1** [6] Гіперпараметр — це параметр, значення якого використовується для керування процесом навчання.

В наступних пунктах буде досліджено вплив кількості дерев прийняття рішень, використання Bootstrap Aggregating, кількості параметрів при виборі випадкового простору та вплив максимальної глибини дерева.

**3.3.3 Вплив кількості дерев**

Базовим значенням для реалізації класифікатора, що використовує випадковий ліс, з бібліотеки scikit-learn є 100 дерев. Розглянемо, як вплине зміна кількості дерев на точність передбачення. Було обрано перевірити випадки з 1, 2, 5, 10, 25, 50, 100, 200, 500 деревами. Результати цього(рис 3.3) та подальших порівнянь будуть у вигляді HeatMap створених за допомогою бібліотеки Seaborn.

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

**рис. 3.3**

Як бачимо, значна просадка якості передбачень є тільки для 1 та 2 дерев.

**3.3.4 Вплив Bootstrap Aggregating**

Як було описано в теоретичній частині, цей алгоритм допомагає підвищити точність передбачень. Незначне зниження точності можна побачити графічно на рис 3.4, втім різниця вимірюється в тисячних, що не сильно впиває на якість отриманих результатів. Протягом усіх подібних досліджень цього методу на цьому наборі даних різниця буде мала, причини чого ми розглянемо пізніше.

Зображення, що містить текст, знімок екрана, Шрифт, дизайн

Автоматично згенерований опис

**рис. 3.3**

**3.3.5 Вплив числа параметрів**

Другим важливим алгоритмом, що є складовою випадкових лісів, є метод випадкових просторів. Теорія каже нам для задач класифікації використати корінь з числа параметрів значень, що є значенням за замовчуванням в цій бібліотеці. На рис 3.4 можна побачити результати для 1, 6, 8 та фактично 2, оскільки кількість параметрів – 8, а ціла частина від кореня буде 2. Корінь від числа параметрів показав найкращий результат, але знову різниця в тисячних.

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

**рис. 3.4**

**3.3.5 Влив максимальної глибини дерева**

Одним з гіперпараметрів доступних для модифікації є максимальна глибина дерева – найбільша можлива відстань від кореня дерева до найвіддаленішого нащадка. На рис 3.5 можна побачити вплив максимальної глибини заданої значеннями 1, 2, 5, 10, 25, 50, 100, 200, 500. Варто сказати, що за замовчуванням обмежень на цей параметр немає. Як бачимо після 25 точність незмінна.

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

**рис. 3.5**

**3.4 Порівняння методу випадкового лісу з іншими алгоритмами**

Для порівняння було відібрано 5 інших методів класифікації, серед яких:

* Дерево прийняття рішень;
* Логістична регресія;
* Метод опорних векторів;
* Наївний баєсів класифікатор;
* Метод k-найближчих сусідів.

Відбір алгоритмів був здійснений на основі різних статтей, щодо популярних методів класифікації, а дерево прийняття рішень буде корисним джерелом інформації про вплив покращень застосованих в «випадкових лісах». Всі методи були натреновані зі стандартними параметрами, окрім логістичної регресії, для якої було збільшено кількість можливих ітерацій до 10000 через недостатню кількість в стандартній реалізації, про що повідомляла програма.

Дивлячись на результати(рис 3.6), можна помітити, що зі значним відривом найкращими моделями є дерево прийняття рішень та Random forest, що пояснює низький вплив зміни гіперпараметрів, оскільки алгоритм на якому ґрунтується випадковий ліс дає гарні передбачення, а покращений метод хоч і має кращі передбачення, втім простір для покращень досить обмежений. Решта методів дали схожі результати, що значно відстають від перших.

Зображення, що містить текст, знімок екрана

Автоматично згенерований опис

**рис. 3.6**

**Висновки**

Під час практики було досліджено метод Random Forest.

В першому розділі роботи були розглянуті основні поняття пов’язані з деревами прийняття рішень, необхідними для розуміння досліджуваного методу.

В другому розділі були розглянуті основні алгоритми застосовані в методі Random Forest та принцип його роботи.

В третьому розділі було розглянуто набір даних для передбачення ядовитості грибів за їх зовнішніми характеристиками. Було розглянуто принципи, що використовувались при розробці моделей, було досліджено вплив гіперпараметрів та порівняно результати якості передбачень з іншими методами.

Джерела

[1] Louppe G. Understanding Random Forests: From Theory to Practice: PhD dissertation. Liège, 2014.

[2] Rahman M. A., Hokugo A., Ohtsu N. Household evacuation preparation time during a cyclone: Random Forest algorithm and variable degree analysis. Progress in disaster science. 2021. Vol. 12. P. 100209. URL: <https://doi.org/10.1016/j.pdisas.2021.100209>

[3] Учасники проектів Вікімедіа. Компроміс зсуву та дисперсії – Вікіпедія. Вікіпедія. URL: [https://uk.wikipedia.org/wiki/Компроміс\_зсуву\_та\_дисперсії](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%80%D0%BE%D0%BC%D1%96%D1%81_%D0%B7%D1%81%D1%83%D0%B2%D1%83_%D1%82%D0%B0_%D0%B4%D0%B8%D1%81%D0%BF%D0%B5%D1%80%D1%81%D1%96%D1%97)

[4] Учасники проектів Вікімедіа. Точність – Вікіпедія. Вікіпедія. URL: [https://uk.wikipedia.org/wiki/Точність](https://uk.wikipedia.org/wiki/%D0%A2%D0%BE%D1%87%D0%BD%D1%96%D1%81%D1%82%D1%8C) (дата звернення: 13.06.2024).

[5] Учасники проектів Вікімедіа. F-міра – Вікіпедія. Вікіпедія. URL: [https://uk.wikipedia.org/wiki/F-міра](https://uk.wikipedia.org/wiki/F-%D0%BC%D1%96%D1%80%D0%B0)

[6] Учасники проектів Вікімедіа. Гіперпараметр (машинне навчання) – Вікіпедія. Вікіпедія. URL: [https://uk.wikipedia.org/wiki/Гіперпараметр\_(машинне\_навчання)](https://uk.wikipedia.org/wiki/%D0%93%D1%96%D0%BF%D0%B5%D1%80%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80_(%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B5_%D0%BD%D0%B0%D0%B2%D1%87%D0%B0%D0%BD%D0%BD%D1%8F))

[7] Mushroom Dataset (Binary Classification). Kaggle: Your Machine Learning and Data Science Community. URL: <https://www.kaggle.com/datasets/prishasawhney/mushroom-dataset>

Додаток A (обов’язковий)

Текст програми “Research.ipynb”

**In [1]**

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

**In [2]**  
DATA\_FOLDER = "DATA/"  
FILE\_NAME = "mushroom\_cleaned.csv"  
  
file\_path = DATA\_FOLDER + FILE\_NAME

**In [3]**  
mushroom\_data = pd.read\_csv(file\_path)  
mushroom\_data.head(5)

**In [4]**  
mushroom\_data.info()

**In [5]**  
mushroom\_data.nunique()

**In [6]**  
mushroom\_data.describe()

**In [7]**  
mushroom\_data["season"].unique()

**#%% md  
## Basic training**

**In [8]**  
X = mushroom\_data.iloc[:, :-1].values  
y = mushroom\_data.iloc[:, -1].values

**In [9]**  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

**In [10]**

#import random  
  
#random.randint(0, 1000)  
## 215

**In [11]**  
from sklearn.metrics import accuracy\_score, precision\_score, f1\_score  
  
def generate\_result(model\_name, model):  
 model.fit(X\_train, y\_train)  
 prediction = model.predict(X\_test)  
 accuracy = accuracy\_score(y\_test, y\_pred=prediction)  
 precision = precision\_score(y\_test, y\_pred=prediction)  
 f1 = f1\_score(y\_test, y\_pred=prediction)  
 return [model\_name, accuracy, precision, f1]

**In [12]**

def generate\_df(results):  
 df = pd.DataFrame(results, columns=["name","accuracy", "precision", "f1"])  
 df.set\_index("name", inplace = True)  
 return df

**In [13]**  
from sklearn.ensemble import RandomForestClassifier  
  
RF\_base = RandomForestClassifier(random\_state = 215)  
RF\_result = generate\_result("Simple Random Forest",RF\_base)  
RF\_result

**#%% md  
## Testing impact of number of trees**

**In [14]**  
number\_of\_trees = [1, 2, 5, 10, 25, 50, 100, 200, 500]  
results\_num\_of\_trees = []  
  
def test\_RF(RF\_classifier, result\_array, name, param):  
 name = name + str(param)  
 RF\_results = generate\_result(name,RF\_classifier)  
 result\_array.append(RF\_results)  
  
for number in number\_of\_trees:  
 RF = RandomForestClassifier(n\_estimators = number, random\_state = 215)  
 test\_RF(RF, results\_num\_of\_trees, "Number of trees: ", number)  
  
sns.heatmap(generate\_df(results\_num\_of\_trees), annot = True)  
plt.show()

**#%% md  
## Testing impact of bootstrap**

**In [15]**  
RF\_no\_bootstrap = RandomForestClassifier(bootstrap = False, random\_state = 215)  
RF\_no\_bs\_result = generate\_result("Random Forest No BS",RF\_no\_bootstrap)  
sns.heatmap(generate\_df([RF\_no\_bs\_result, RF\_result]), annot = True)  
plt.show()

**#%% md  
## Testing impact of number of features**

**In [16]**  
number\_of\_features = [1, "sqrt", 6, 8]  
results\_features = []  
  
for number in number\_of\_features:  
 RF = RandomForestClassifier(max\_features = number, random\_state = 215)  
 test\_RF(RF, results\_features, "Number of features: ", number)  
sns.heatmap(generate\_df(results\_features), annot = True)  
plt.show()

**#%% md  
## Testing impact of max depth**

**In [17]**  
max\_depth = [1, 2, 5, 10, 25, 50, 100, 200, 500]  
results\_depth = []  
  
for number in number\_of\_trees:  
 RF = RandomForestClassifier(max\_depth = number, random\_state = 215)  
 test\_RF(RF, results\_depth, "Max depth: ", number)  
sns.heatmap(generate\_df(results\_depth), annot = True)  
plt.show()

**#%% md  
# Comparing with another models**

**#%% md  
## Decision Tree Classifier**

**In [18]**  
from sklearn.tree import DecisionTreeClassifier  
DT\_base = DecisionTreeClassifier()  
Dt\_result = generate\_result("Decision Tree",DT\_base)  
Dt\_result

**#%% md  
## Logistic Regression**

**In [19]**  
from sklearn.linear\_model import LogisticRegression  
LR\_base = LogisticRegression(max\_iter = 10000)  
LR\_result = generate\_result("Logistic Regression",LR\_base)  
LR\_result

**#%% md  
## Support Vector Machines(SVC)**

**In [20]**  
from sklearn.svm import SVC  
SVC\_base = SVC()  
SVC\_result = generate\_result("SVC",SVC\_base)  
SVC\_result

**#%% md  
## Naive Bayes**

**In [21]**  
from sklearn.naive\_bayes import GaussianNB  
NB\_base = GaussianNB()  
NB\_result = generate\_result("Naive Bayes",NB\_base)  
NB\_result

**#%% md  
## K-nearest neighbors**

**In [22]**  
from sklearn.neighbors import KNeighborsClassifier  
KNN\_base = KNeighborsClassifier()  
KNN\_result = generate\_result("K-nearest neighbors",KNN\_base)  
KNN\_result

**In [23]**  
merged\_models = [RF\_result, Dt\_result, LR\_result, SVC\_result, NB\_result, KNN\_result]  
  
sns.heatmap(generate\_df(merged\_models), annot = True)  
plt.show()