

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
МЕХАНІКО-МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА ТЕОРІЇ ЙМОВІРНОСТЕЙ,
СТАТИСТИКИ ТА АКТУАРНОЇ МАТЕМАТИКИ

КУРСОВА РОБОТА

НА ТЕМУ:

«Аналіз методів розпізнавання рукописних цифр»

Виконав:

студент 4 курсу

механіко-математичного факультету

групи «Статистика»

Гайович Богдан Ігорович

Науковий керівник:

кандидат фізико-математичних наук

асистент кафедри теорії ймовірностей,

статистики та актуарної математики

Голомозий Віталій Вікторович

КИЇВ – 2020

План

Вступ	3
Теоретичні відомості	4
К-найближчих сусідів	4
Логістична регресія.....	4
Метод опорних векторів	6
Ансамблевий класифікатор.....	8
Опис даних	9
Опис обробки даних	9
Побудова моделей	12
К-найближчих сусідів	12
Логістична регресія.....	13
Метод опорних векторів	13
Ансамблевий класифікатор.....	14
Висновок	16
Джерела	17
Додаток	18

Вступ

З настанням інформаційної ери, людство генерує з кожним роком все більше й більше інформації в цифровому вигляді. Для того, щоб її зберігати, потрібні колосальні технічні можливості, а також велика кількість інженерів, котрі цю техніку зможуть підтримувати в працездатному стані. Проте, зберігання не є основною задачею.

Більш важливою є автоматична обробка отриманих даних. З її допомогою, наприклад, можливо отримати корисну інформацію з відео та фотографій, не залучаючи для цього велику кількість людського труда. В таких прикладних задачах, як розумне сканування документів, моніторинг дотримання правил дорожнього руху, і т.д. потрібні системи автоматичного розпізнавання тексту. До того ж

Метою цієї роботи є порівняння й аналіз методів розпізнавання рукописних цифр за допомогою комп'ютера. Отримані результати далекі від впровадження в промисловість, проте описаний підхід є важливим для розуміння можливостей комп'ютерів в сфері розпізнавання тексту на зображеннях. В ході роботи було отримано, опрацьовано та розширено дані, на яких згодом було натреновано моделі на основі машинного навчання.

Теоретичні відомості

Очевидно, що перед нами стоїть задача класифікації, оскільки відгук є категорійною змінною, тобто може набувати лише 10 значень (відповідне значення для кожної цифри). Ці категорії будемо називати класами.

В ході роботи було використано три моделі, а також ансамблевий класифікатор на їх основі.

К-найближчих сусідів

Спершу був використаний алгоритм К-найближчих сусідів (*англ. KNN – K-nearest neighbors*). Це один із фундаментальних та найпростіших методів класифікації. Класифікація К-найближчого сусіда була розроблена з необхідності проведення дискримінантного аналізу у випадку, коли достовірні параметричні оцінки щільностей ймовірності невідомі або їх важко визначити.

Зазвичай цей алгоритм базується на евклідовій відстані між тестовим об'єктом та зазначеними навчальними зразками. Нехай x_i – вектор регресорів для i -го спостереження з розмірністю p , тобто $x_i = (x_{i1}, x_{i2}, \dots, x_{ip})$. Тоді відстань між двома спостереженнями рівна:

$$d(x_i, x_j) = \sqrt{\sum_{n=1}^p (x_{in} - x_{jn})^2}$$

Підрахувавши відстані від x_i до всіх x_j , $j \in \overline{1, N} \setminus \{i\}$, де N – розмір вибірки, ми обираємо лише K найближчих. Тоді класом для i -го спостереження буде той клас, що найчастіше зустрічався серед сусідніх.

Точність для цього методу підраховується як відношення суми всіх правильно класифікованих елементів до всіх елементів, що класифікувалися.

Логістична регресія

Наступним, другим за списком, алгоритмом було обрано логістичну регресію. Він також є одним із фундаментальним методів класифікації з багаторічною історією, витоки якої простежуються аж до 19 століття.

Логістична регресія – статистична модель, що використовує логістичну функцію для визначення залежної бінарної змінної. Нехай маємо випадкову величину – відгук Y , що може набувати лише двох значень (для простоти ці значення можна позначити як 0 та 1). Нехай вона залежить від множини регресорів (тобто спостереження) $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_p^{(i)})^T$. Для простоти запису вводиться нульовий елемент $x_0 = 1$.

Далі робиться припущення про те, що:

$$P\{Y = 1 \mid x\} = f(z)$$

Де $z = W^T x = w_0 + \sum_{i=1}^p w_i * x_i$, W – коефіцієнти регресії (ваги) – дійсні числа $W = (w_0, w_1, w_2, \dots, w_p)$. Логістичною функцією називається ф-ція $f: \mathbb{R} \rightarrow [0, 1]$:

$$f(z) = \frac{1}{1 + e^{-z}}$$

Її також називають сігмоїдою. По-суті, будь-яке значення x переводиться в проміжок від 0 до 1, що можна інтерпретувати як ймовірність. Оскільки Y приймає лише значення 0 або 1, то ймовірність отримати значення 0 дорівнює:

$$P\{Y = 0 \mid x\} = 1 - f(W^T x)$$

Функцію розподілу випадкової величини Y при заданому x можна записати так:

$$P\{Y \mid x\} = f(W^T x)^Y * (1 - f(W^T x))^{1-Y}, \quad Y \in \{0, 1\}$$

Для того, щоб визначити параметри W , потрібно вибрати навчальну вибірку, що складається з пар елементів спостереження – відповідний відгук y :

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)}),$$

де $x^{(i)}$ – вектор регресорів i -го спостереження, $y^{(i)}$ – відгук, m – розмір тренувальної вибірки. Потім оптимальні параметри W шукають за допомогою методу максимальної вірогідності, максимізуючи значення вибіркової функції вірогідності:

$$\hat{W} = \underset{W}{\operatorname{argmax}} L(W) = \underset{W}{\operatorname{argmax}} \prod_{i=1}^m P\{y = y^{(i)} \mid x = x^{(i)}\}$$

Що еквівалентно пошуку максимуму для логарифму цієї функції:

$$\begin{aligned}\ln L(W) &= \sum_{i=1}^m \ln P\{y = y^{(i)} \mid x = x^{(i)}\} \\ &= \sum_{i=1}^m y^{(i)} * \ln f(W^T x^{(i)}) + (1 - y^{(i)}) * \ln(1 - f(W^T x^{(i)}))\end{aligned}$$

Для максимізації цієї функції зазвичай застосовують метод градієнтного спуску. Суть його полягає у тому, що спочатку обираються (випадковим чином) параметри W , а потім виконуються наступні ітерації протягом певної кількості кроків:

$$W := W + \alpha * \nabla \ln L(W) = W + \alpha * \sum_{i=1}^m (y^{(i)} - \ln f(W^T x^{(i)})) * x^{(i)}, \alpha > 0$$

Нехай класів більше двох ($k > 2$). Тоді для кожної унікальної пари категорій (загальна кількість таких пар – C_k^2) навчається модель, а потім при визначенні класу обирають найбільшу вірогідність серед всіх. Точність для цього методу підраховується як відношення кількості правильно класифікованих спостережень з тестової вибірки до загального розміру тестової вибірки.

Метод опорних векторів

Метод опорних векторів (*англ. Support Vector Machines, SVM*) в сучасному вигляді був вперше представлений на конференції COLT в 1992 році вченими Босером, Гійоном та Вапником¹.

Як і у випадку логістичної регресії, маємо пари значень $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$, де $x^{(i)}$ — вектор регресорів i -го спостереження, $y^{(i)}$ — відгук для i -го об'єкту, що, на відміну від попередньої моделі, приймає значення з $\{-1, 1\}$, m — розмір вибірки. Регресори зазвичай нормалізуються, оскільки в іншому випадку точка, що матиме значне відхилення від середнього вибірки, буде занадто сильно впливати на класифікатор. Для того, щоб розділяти два класи, ми будемо гіперплощину, що має вигляд:

$$w * x + b = 0$$

¹ A training algorithm for optimal margin classifiers (Boser, Guyon and Vapnik, 1992)

Вектор w – це перпендикуляр до описаної гіперплощини, значення $\frac{b}{\|w\|}$ за модулем дорівнює відстанню від гіперплощини до початку координат.

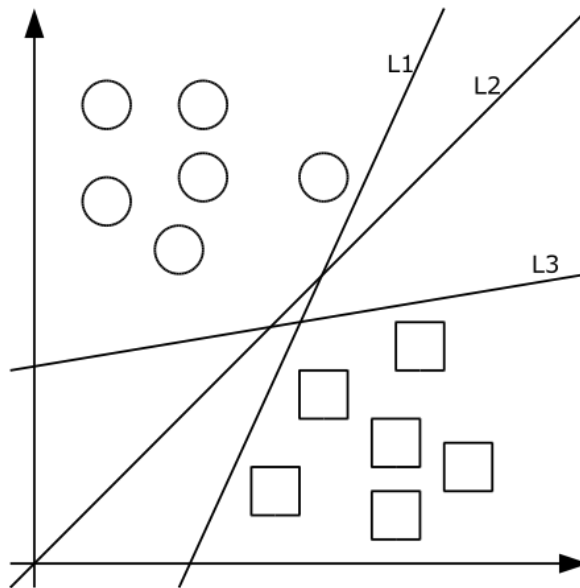


Рисунок 1

На рисунку 1 зображено декілька гіперплощин, що розділяють два класи, і лише одна з них – L2 – відповідає оптимальному розмежуванню. В основі методу опорних векторів якраз і лежить ідея знаходження цієї оптимальної гіперплощини.

Описавши оптимальну гіперплощину, для визначення опорних векторів, ми розглянемо гіперплощини, що є паралельними до неї. Їх можна описати наступними рівняннями:

$$\begin{cases} w * x - b = 1 \\ w * x - b = -1 \end{cases}$$

Якщо навчальна вибірка є лінійно сепарабельною, то можна вибрати гіперплощини таким чином, щоб жодна з точок вибірки не знаходилася між ними і після цього максимізувати відстань між ними. Ширина «полоси» при цьому рівна $\frac{2}{\|w\|}$, таким чином задача зводиться до пошуку мінімального $\|w\|$. Щоб переконатися, що всередині полоси не лежить жодна точка вибірки, для всіх i :

$$\begin{cases} w * x^{(i)} - b \geq 1, & y^{(i)} = 1 \\ w * x^{(i)} - b \leq -1, & y^{(i)} = -1 \end{cases}$$

Що еквівалентно нерівності:

$$y^{(i)} * (w * x^{(i)} - b) \geq 1, \quad 1 \leq i \leq m$$

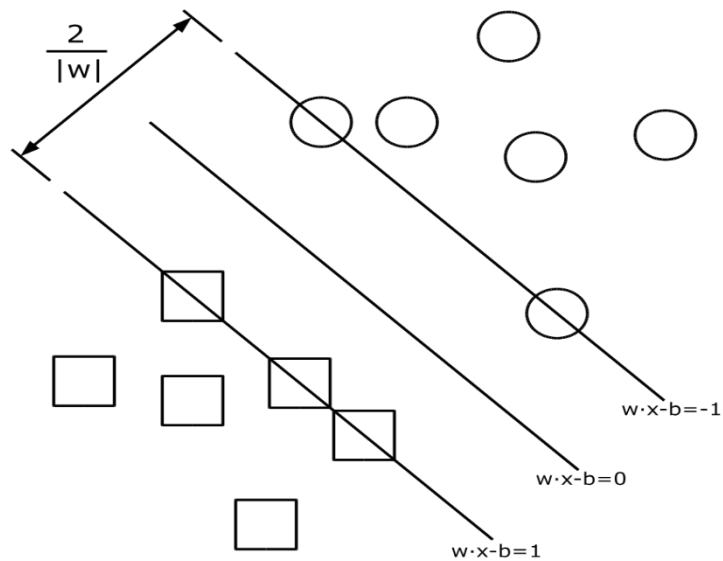


Рисунок 2

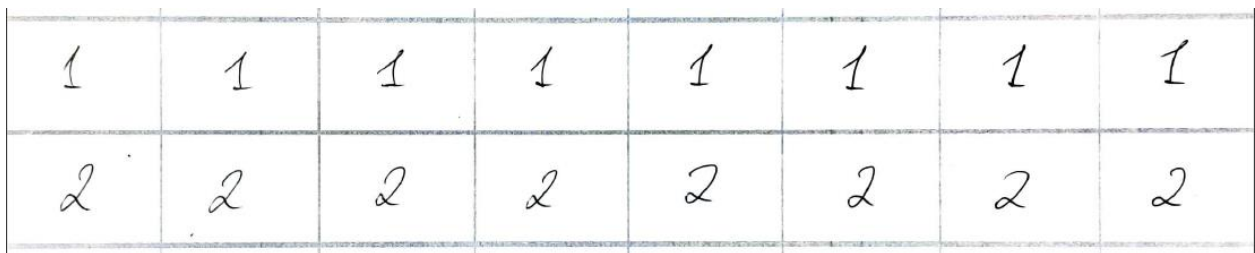
На рисунку 2 зображено оптимальну розмежувальну гіперплощину, побудовану для двох класів, а також паралельні до неї площини, що схематично утворюють «полосу». Найближчі до цих гіперплощин точки називають опорними векторами.

Ансамблевий класифікатор

Ідея ансамблевого класифікатора полягає у тому, щоб поєднати концептуально різні класифікатори машинного навчання та використовувати більшість голосів (*англ. hard voting*) або середні прогнозовані ймовірності (м'яке голосування, *англ. soft voting*) для прогнозування класів. Такий класифікатор може бути корисним для набору однаково добре працюючих моделей, щоб збалансувати їх окремі слабкі сторони.

Опис даних

Дані представляють собою набір рукописних цифр, по 72 на кожний клас. На п'яти листках А4 було створено таблиці з 8-ма стовпчиками та 18-ма рядками, в кожній клітинці відводилося місце під написання цифри. Це було зроблено для того, аби в майбутньому полегшити розбиття всього аркушу на дрібніші клітинки та для того, щоб цифри в отриманих комірках уже були вручну відцентровані з певною прийнятною похибкою (приклад зображено на рисунку 3).



1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2

Рисунок 3

Дані було відскановано та оцифровано, що дозволило їх подальшу обробку в середовищі мови програмування Python.

Опис обробки даних

Завантаживши відскановані аркуші А4 в оперативну пам'ять, з кодним із них проводилися наступні кроки задля обробки та очистки даних:

1. Кожна клітинка мала приблизно однакові висоту та ширину, що дозволяло розділити 1 фото на 144 об'єкти, в кожному з яких містилося зображення однієї цифри.
2. Початково фото мало 3 канали кольору – червоний, зелений та синій спектри (RGB). Для того, аби зображення можна було описати одним вектором, ми перетворюємо його в чорно-білу версію, а потім інвертуємо кольори, щоб пікселі, які не належать цифрі мали нульове значення (оскільки початково цифра була написана чорним на білому фоні). Оскільки під час сканування білий фон міг оцифруватися не як чисто білий, з певним «шумом», було прийнято поріг яскравості 10 і всі пікселі, яскравість яких слабша за це число, були «занулені».
3. Наступною задачею було відцентрувати всі зображення – для цього спершу знаходимо межі кожної цифри на зображенні. Це легко

зробити, знайшовши координати перших і останніх ненульових пікселів (що можливо зробити завдяки виконанню попереднього пункту). Ми отримали прямокутник з розмірами $a \times b$ пікселів (a – висота, b – ширина отриманого зображення). Потім цей прямокутник розширюється з усіх сторін на 10 пікселів, щоб створити свого роду «рамку» навколо цифри. Далі до отриманого зображення додаються рамки з нульових пікселів товщиною $\frac{|a-b|}{2}$ зверху і знизу, якщо $a < b$ і по бокам, якщо $a > b$. Таким чином отримуємо чорні квадрати, в які білими пікселями вписані цифри. Приклад можна побачити на рисунку 4



Рисунок 4

4. Далі всі зображення зводяться до єдиного розміру – 28×28 пікселів, для того, щоб встановити єдину розмірність для всіх об'єктів.

Після всі кроків вище було розбито дані на тренувальну та тестову вибірки у співвідношенні 70 до 30. Отримали 504 тренувальних та 216 тестових об'єктів.

Оскільки вручну згенерувати сотні прикладів для кожної цифри не є продуктивним, було вирішено розширити тренувальну вибірку за допомогою наступних технік:

- Випадковим чином обирається 1 зображення.
- Генерується випадкове число з рівномірного розподілу $U(-10, 10)$, що буде відповідати куту, на який слід повернути обране зображення навколо своєї осі.
- Випадковим чином обираються параметр z ($\sim U(-0.05, 0.05)$), що вказує на скільки «збільшується» зображення. Відповідно, зображення з початковим розширенням 28×28 розширюється до розмірів $(1 + z) * 28 \times (1 + z) * 28$, а потім обрізається до початкового розміру 28×28 . Для цього і створювалася рамка,

описана в пункті 3 – щоб необхідні пікселі самої цифри не були обрізані та втрачені.

- Аналогічно з попереднім пунктом, генеруються ще два параметри з того ж розподілу, що і z , які вказуватимуть на скільки потрібно зсунути початкове зображення по горизонталі та вертикалі відповідно.

Застосувавши вищеописані алгоритми до тренувальних зображень, ми можемо розширити навчальну вибірку в декілька разів (у нашому випадку було збільшено у два рази) і при цьому не отримати об'єктів-дублікатів. Це піде на користь генералізації нашої моделі, оскільки вона буде навчена враховувати більшу варіацію об'єктів.

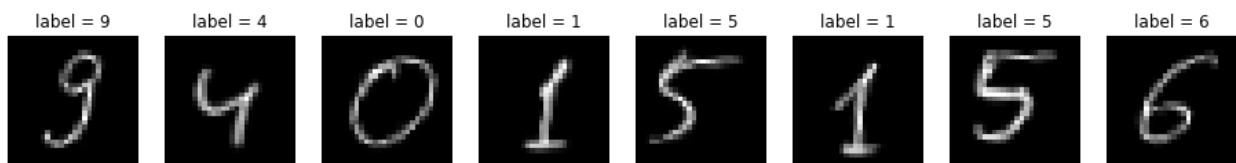


Рисунок 5

На рисунку 4 зображено приклади згенерованих вищеописаним способом об'єкти. Можна помітити, що вони менш чіткі (через повороти та збільшення) та мають іноді більші кути нахилу до вертикальної осі, але в цілому відповідають своїм категоріям.

Після пророблених вище процедур ми маємо 1512 навчальних об'єктів та 216 тестових. Кожне зображення – це матриця 28 на 28, і, для того, щоб тренувати моделі, їх необхідно перевести в одновимірний вектор – до першого рядка в кінець дописуються значення з другого, потім з третього і т.д. На виході отримуємо вектор з $28^2 = 784$ значеннями для кожного об'єкту.

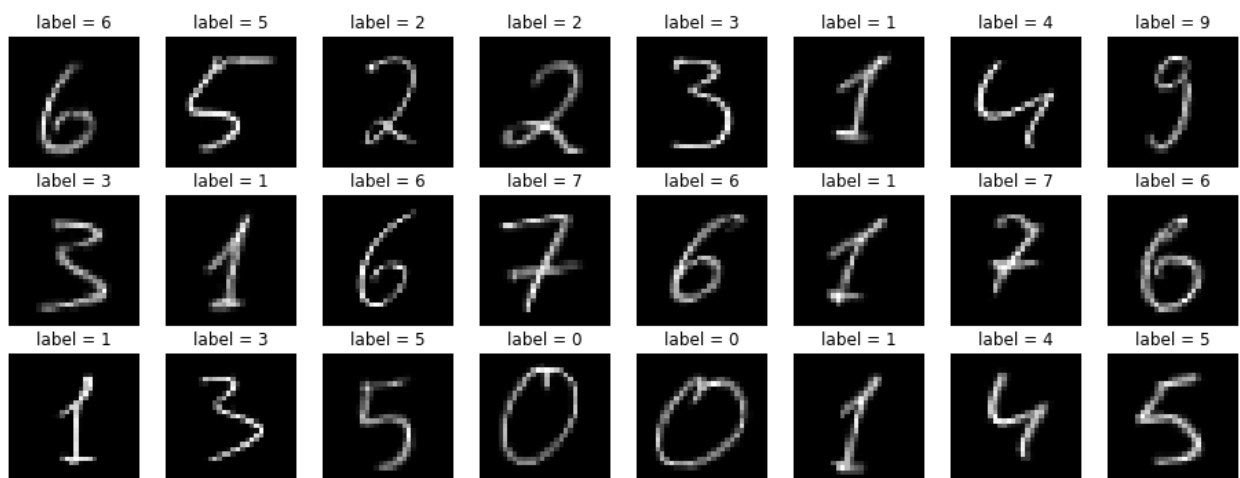
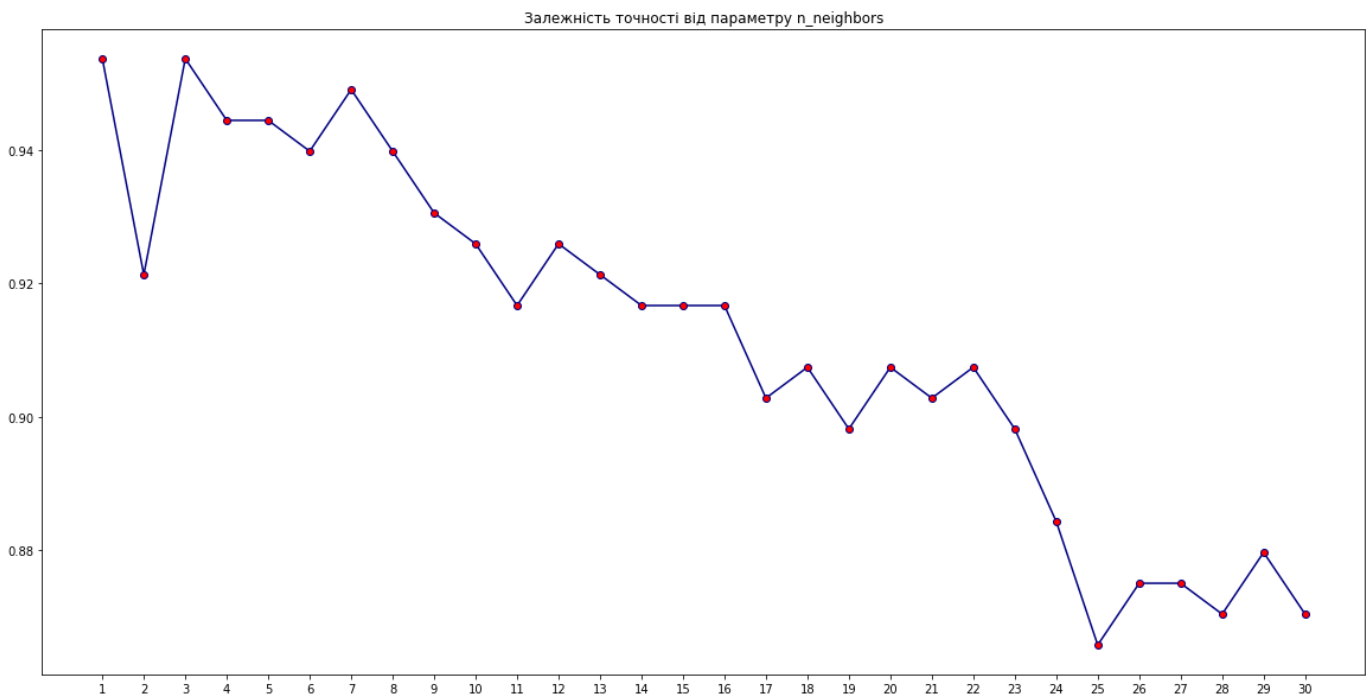


Рисунок 6: Відібрані випадковим чином 24 тренувальних зображення

Побудова моделей

К-найближчих сусідів

Для початку в цьому методі необхідно визначити параметр К. Для цього навчалось 30 моделей зі значеннями цього параметру від 1 до 30 і оцінено їх роботу на тестовій вибірці. З *Графіку 1* нижче видно, що найкраще себе показують моделі, в яких К був рівний 3:



Графік 1

Обираємо $K=3$. Далі, щоб оцінити точність моделі, робимо перехресну перевірку (*англ. Cross-Validation*). Тобто, вся вибірка розбивається випадковим чином на тренувальну та тестову. Далі навчаємо модель на отриманих даних, її роботу оцінюємо на поточній тестовій вибірці. Потім ці ж самі кроки повторюємо 10 разів. Таким чином визначається середнє та дисперсія точності статистичної моделі.

Для методу К-найближчих сусідів, ми отримали такі результати:

- Точність моделі без розширення тренувальної вибірки:
0.9134 (+- 0.01)
- Точність моделі з розширенням тренувальної вибірки в 3 рази:
0.9537 (+- 0.01)

Для того, щоб перевірити, чи точність збільшилась значущо, проведемо t-тест Стюдента з нульовою гіпотезою про те, що точності однакові. Отримане значення $p_{value} \sim 0.00008$, що вказує на те, що точності відрізняються значущо, а отже, розширення даних допомогло покращити модель.

Логістична регресія

Аналогічним чином до попереднього методу зробимо оцінку точності логістичної регресії:

- Точність моделі без розширення тренувальної вибірки:
0.9611 (+- 0.01)
- Точність моделі з розширенням тренувальної вибірки в 3 рази:
0.9694 (+- 0.01)

Провівши t-тест Стюдента, отримали значення $p_{value} \sim 0.07$, а отже точності не відрізняються для 2-х підходів.

Метод опорних векторів

Зробивши перехресну перевірку для методу опорних векторів, ми отримали наступні результати:

- Точність моделі без розширення тренувальної вибірки:
0.9662 (+- 0.01)
- Точність моделі з розширенням тренувальної вибірки в 3 рази:
0.9852 (+- 0.01)

Провівши t-тест Стюдента, отримали значення $p_{value} \sim 0.0008$, а отже точності значущо відрізняються для 2-х підходів і штучне збільшення навчальної вибірки дало значне покращення точності.

Отже, найкращий результат показує модель методу опорних векторів, натренована на доповнених даних. Її точність складає $\sim 98.5\%$. Спробуємо покращити цей показник, застосувавши метод ансамблевого касифікатору.

Ансамблевий класифікатор

Розглянуті вище моделі ми згрупували в один класифікатор, який визначає клас об'єкту на основі передбачень всіх трьох моделей. Отримані результати:

- Ансамблевий класифікатор без розширення даних:

0.9556 (+/- 0.01) [Ensemble (Hard)]

0.9722 (+/- 0.01) [Ensemble (Soft)]

- Ансамблевий класифікатор з розширенням даних:

0.9639 (+/- 0.01) [Ensemble (Hard)]

0.9796 (+/- 0.01) [Ensemble (Soft)]

Згідно тесту Стюдента, розширення даних для типу *Hard* дає приріст точності ($p_{value} = 0.0079$), так само і для типу *Soft* точність теж значущо більша ($p_{value} = 0.0032$).

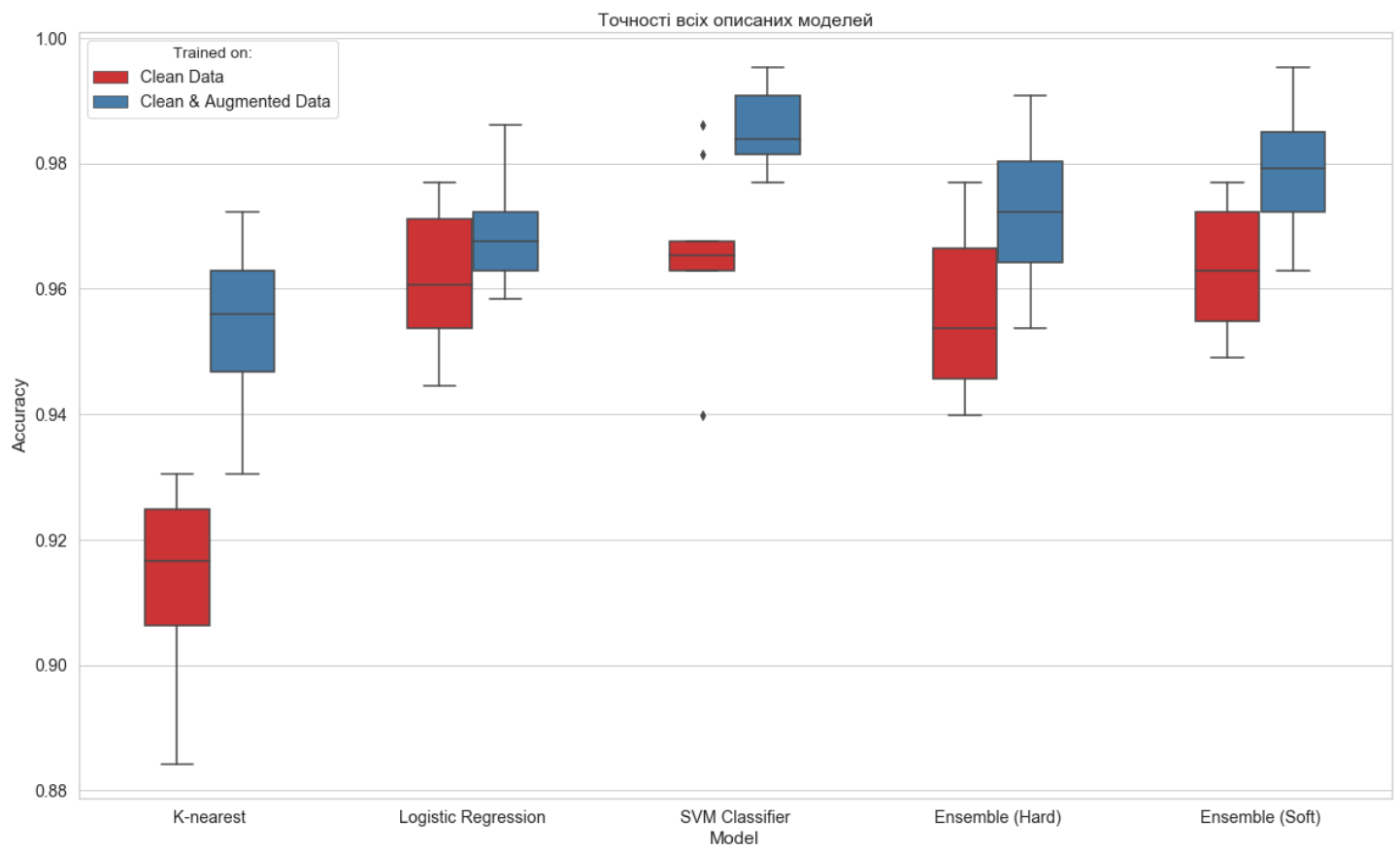
Перевіримо чи дав метод застосування ансамблевого класифікатора значущий приріст точності у порівнянні із найточнішою моделлю – методом опорних векторів, натренованого на розширеній вибірці:

P-value: 0.0094 [Ensemble (Hard)]

P-value: 0.1752 [Ensemble (Soft)]

Як видно зі значень p_{value} вище, точність разюче відрізняється в гіршу сторону при використанні методу *Hard*, і не відрізняється при використанні м'якого голосування.

В підсумку, опираючись на вищенаведені дані, можна зробити висновок, що найкращим класифікатором виявилася модель на основі методу опорних векторів, натренована на доповнених даних з точністю 98.5%. Коробки з вусами для точностей всіх розглянутих моделей зображено на Графіку 2.



Графік 2

Висновок

В ході роботи було розглянуто методи, статистичні моделі, за допомогою яких можливе розпізнавання рукописних цифр.

Для порівняння використовувалися три моделі: K-найближчих сусідів, логістична регресія та метод опорних векторів, а також їх комбінація у вигляді ансамблевого класифікатора.

В ході роботи було встановлено, що майже для всіх моделей (окрім логістичної регресії), значущий приріст у точності спостерігався, якщо тренувальну вибірку штучно розширити за допомогою незначних перетворень зображень.

Найкраща точність була отримана для класифікатора на основі методу опорних векторів, його точність склала 98.52%, що є прийнятною точністю, враховуючи той факт, що точність людей на подібній задачі складає від 97.5 до 98 відсотків². Використання ансамблевого методу з використанням підходів *hard-* та *soft-voting* не покращило точність значущо, а в деяких випадках її погіршило.

	Дані без доповнення	Дані з доповненням
К-найближчих сусідів	0.9134	0.9537
Логістична регресія	0.9611	0.9694
SVM	0.9662	0.9852
Ансамблевий класифікатор (Hard-voting)	0.9556	0.9639
Ансамблевий класифікатор (Soft-voting)	0.9722	0.9796

Таблиця 1 Таблица точностей класифікації для різних підходів.

² Efficient Pattern Recognition Using a New Transformation Distance (Patrice Simard, Yann Le Cun, John Denker, 1993)

Джерела

1. A training algorithm for optimal margin classifiers (Boser, Guyon and Vapnik, 1992)
2. Efficient Pattern Recognition Using a New Transformation Distance (Patrice Simard, Yann Le Cun, John Denker, 1993)
3. Програмний код та необхідні файли викладено на [github](#).

Додаток

```
#!/usr/bin/env python
# coding: utf-8

# # Аналіз методів розпізнавання рукописних цифр

# In[1]:

import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from copy import deepcopy

# In[2]:

class config:
    IM_RES = (28, 28)
    RAW_IM_PATH = "Handwritten_numbers"
    NFOLDS = 10
    SCORES = []

# In[3]:

class DataLoader():
    def __init__(self, path):
        self.folder = path
        self.raw_images = []
        self.samples = None
        self.labels = None

        self.Xtrain = None
        self.Xtest = None
        self.ytrain = None
        self.ytest = None

    def load_raw_ims(self):
        self.raw_images = []
        for f in os.listdir(self.folder):
            self.raw_images.append(cv2.imread(os.path.join(self.folder, f)))

        print("Loaded {} raw images from disk".format(len(self.raw_images)))

    def process_raw_ims(self):
        width = self.raw_images[0].shape[1] // 8
        height = self.raw_images[0].shape[0] // 18
        self.samples = []
        self.labels = []
        for i, im in enumerate(self.raw_images):
```

```

        for vertical_step in range(18):
            for horizontal_step in range(8):
                # cut raw image so we get the image of a number
                im1 = im[vertical_step*height:(vertical_step+1) * height,
                        horizontal_step*width : (horizontal_step +
1)*width].copy()
                im1_neg = 255 - im1.mean(axis=2)
                im1_neg[im1_neg <= 10] = 0
                x = np.argmax(im1_neg, axis=0)
                y = np.argmax(im1_neg, axis=1)

                # find coordinates of the smallest square that contains the
number
                # add small frame around the number
                top = np.nonzero(y)[0][0] - 10
                bot = np.nonzero(y)[0][-1] + 10
                h = bot - top
                left = np.nonzero(x)[0][0] - 10
                right = np.nonzero(x)[0][-1] + 10
                w = right - left

                # make image square
                if h > w:
                    side = (h - w) // 2
                    left -= side
                    right += side
                else:
                    side = (h - w) // 2
                    top -= side
                    bot += side

                # cut image by the coordinates and resize it ti standart size
                im_cutted = im1_neg[top:bot, left:right].copy()
                im_cutted = cv2.resize(im_cutted, config.IM_RES)

                label = (i * 2 + 1 + vertical_step // 9) % 10

                self.samples.append(im_cutted)
                self.labels.append(label)

            self.samples = np.array(self.samples)
            self.samples = self.samples.reshape(self.samples.shape[0],
                                                config.IM_RES[0],
                                                config.IM_RES[1], 1)
            self.labels = np.array(self.labels)

        print("Successfully cut raw images into {}
samples".format(self.samples.shape[0]))

    def plot_numbers(self, num=8, images=None, labels=None):
        num_rows = max(1, num // 8)
        if num % 8 > 0:
            num_rows += 1

        fig, axs = plt.subplots(num_rows, 8, figsize=(16, 2 * num_rows),
sharey=True, squeeze=False)

        if images is not None and labels is not None:

```

```

        print("Plotting random {} images from the given set".format(num))
        rand_ind = np.random.randint(images.shape[0], size=num)
        images = images[rand_ind]
        labels = labels[rand_ind]
    else:
        rand_ind = np.random.randint(self.samples.shape[0], size=num)
        images = self.samples[rand_ind]
        labels = self.labels[rand_ind]

    for i in range(num):
        row = i // 8
        col = i % 8
        axs[row][col].imshow(images[i].reshape(config.IM_RES[0],
                                                config.IM_RES[1]), cmap='gray')

        axs[row][col].axis("off")
        axs[row][col].set_title("label = {}".format(labels[i]))

plt.show()

def data_augmentation(self, times=0.3, show=True):
    if show:
        print("Augmenting data by {} %".format(times * 100))
    aug_size = round(self.Xtrain.shape[0] * times)

    generator = ImageDataGenerator(
        rotation_range=10,
        zoom_range = 0.05,
        width_shift_range = 0.05,
        height_shift_range = 0.05,
        horizontal_flip=False,
        vertical_flip=False,
        data_format="channels_last")

    # get transformed images
    rand_ind = np.random.randint(self.train_size, size=aug_size)
    samples_aug = self.Xtrain[rand_ind].copy()
    labels_aug = self.ytrain[rand_ind].copy()

    samples_aug = generator.flow(samples_aug, np.zeros(aug_size),
                                batch_size=aug_size, shuffle=False).next()[0]
    # append augmented data to trainset
    self.Xtrain = np.concatenate((self.Xtrain, samples_aug))
    self.ytrain = np.concatenate((self.ytrain, labels_aug))
    self.train_size = self.Xtrain.shape[0]

    if show:
        print("Successfully augmented data... Created {} new train
samples".format(samples_aug.shape[0]))
        self.plot_numbers(images=samples_aug, labels=labels_aug)

    def split_data(self, test_size, show=True):
        self.Xtrain, self.Xtest, self.ytrain, self.ytest =
train_test_split(self.samples,

self.labels,

test_size=test_size)
        self.train_size = self.Xtrain.shape[0]
        self.test_size = self.Xtest.shape[0]

```

```

        if show:
            print("Created {} train samples\n\t{} test
samples".format(self.train_size, self.test_size))

# # Завантажуємо та обробляємо дані

# In[4]:

data = DataLoader(config.RAW_IM_PATH)

# In[5]:

data.load_raw_imgs()
data.process_raw_imgs()
data.plot_numbers()

# In[14]:

data.split_data(test_size=0.3)

# In[7]:

data.data_augmentation(times=2)

# In[8]:

data.plot_numbers(num=24, images=data.Xtrain, labels=data.ytrain)

# # ПОБУДОВА МОДЕЛЕЙ

# # KNN (K-найближчих сусідів)

# Спершу визначимо параметр K

# In[24]:

from sklearn.neighbors import KNeighborsClassifier as KNN
from scipy.stats import ttest_ind

# In[21]:

scores = []
data.split_data(test_size=0.3)
for n in range(1, 31):
    model_knn = KNN(n_neighbors=n)

```

```

    model_knn.fit(data.Xtrain.reshape(data.train_size, config.IM_RES[0] ** 2),
data.ytrain)
    scores.append([n, model_knn.score(data.Xtest.reshape(data.test_size,
config.IM_RES[0] ** 2),
                                data.ytest)])

scores = np.array(scores)

# In[22]:

plt.figure(figsize=(20, 10))
plt.plot(scores[:, 0], scores[:, 1], color='navy', marker='o', markerfacecolor='red')
plt.xticks(ticks=scores[:, 0])
plt.title("Залежність точності від параметру n_neighbors")
plt.show()

# Обираємо K = 3
#
# Тепер робимо крос-валідацію з 10 фолдами для 2-х вибірок: зі збільшенням та без
нього, щоб зрозуміти наскільки добре модель класифікує та чи впливає збільшення
вибірки на результат

# # KNN vs Logistic vs SVM vs Soft-voting vs Hard-voting

# In[27]:

from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression as LR
from sklearn.svm import SVC

# In[28]:

def build_voting_classifier(type_voting = 'hard'):
    assert type_voting in ['hard', 'soft']

    VC = VotingClassifier(estimators=[('knn', KNN(n_neighbors=3)),
                                     ('lr', LR(penalty='l2', tol=0.00001,
                                              solver='newton-cg', max_iter=1000,
                                              multi_class='auto')),
                                     ('svm', SVC(gamma='scale', probability=True))],
                           voting=type_voting)

    return VC

# Із чистою вибіркою

# In[31]:

model_knn = KNN(n_neighbors=3)
model_lr = LR(penalty='l2', tol=0.00001, solver='newton-cg', max_iter=1000,
multi_class='auto')
model_svm = SVC(gamma='scale')
ensemble_h = build_voting_classifier(type_voting='hard')

```

```

ensemble_s = build_voting_classifier(type_voting='soft')

all_scores_clean = []
for model, label in zip([model_knn, model_lr, model_svm, ensemble_h, ensemble_s],
['K-nearest', 'Logistic Regression', 'SVM Classifier', 'Ensemble (Hard)', 'Ensemble (Soft)']):
    scores = []
    for f in range(config.NFOLDS):
#         split the data
        data.split_data(test_size=0.3, show=False)

        #         train a model
        clf = deepcopy(model)
        clf.fit(data.Xtrain.reshape(data.train_size, config.IM_RES[0] ** 2),
data.ytrain)
        scores.append(clf.score(data.Xtest.reshape(data.test_size, config.IM_RES[0]
** 2), data.ytest))
    scores = np.array(scores)
    print("Accuracy: %0.4f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
    all_scores_clean.append([label, scores])

# In[32]:

ensemble_h = build_voting_classifier(type_voting='hard')
ensemble_s = build_voting_classifier(type_voting='soft')

all_scores_aug = []
for model, label in zip([model_knn, model_lr, model_svm, ensemble_h, ensemble_s],
['K-nearest', 'Logistic Regression', 'SVM Classifier',
'Ensemble (Hard)', 'Ensemble (Soft)']):
    scores = []
    for f in range(config.NFOLDS):

#         split the data
        data.split_data(test_size=0.3, show=False)

        #         enlarge train set by 200%
        data.data_augmentation(2, False)

        #         train a model
        clf = deepcopy(model)
        clf.fit(data.Xtrain.reshape(data.train_size, config.IM_RES[0] ** 2),
data.ytrain)
        scores.append(clf.score(data.Xtest.reshape(data.test_size, config.IM_RES[0]
** 2), data.ytest))

    scores = np.array(scores)
    print("Accuracy: %0.4f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
    all_scores_aug.append([label, scores])

# # Порівняємо результати з та без доповнення даних
#
# За допомогою тесту Ст'юдента

# In[33]:

for i in range(len(all_scores_aug)):

```

```

l = all_scores_aug[i][0]
s_aug = all_scores_aug[i][1]
s_clean = all_scores_clean[i][1]

p_value = ttest_ind(a=s_clean, b=s_aug, equal_var=False).pvalue

print("P-value: %.4f [%s]" % (p_value, l), end='\t')
if p_value >= 0.05:
    print("Mean accuracies are equal")
else:
    print("Mean accuracies are not equal")

# Подивимося чи змінилася точність з та без використання ансамблю порівняно до SVM
(який давав найбільші показники)

# Для чистих даних

# In[34]:

max_row = all_scores_clean[2]
for i in range(2, 0, -1):
    l = all_scores_clean[-i][0]
    s = all_scores_clean[-i][1]

    p_value = ttest_ind(a=s, b=max_row[1], equal_var=False).pvalue

    print("P-value: %.4f [%s]" % (p_value, l), end='\t')
    if p_value >= 0.05:
        print("Mean accuracies are equal")
    else:
        print("Mean accuracies are not equal")

# Для розширених даних

# In[35]:

max_row = all_scores_aug[2]
for i in range(2, 0, -1):
    l = all_scores_aug[-i][0]
    s = all_scores_aug[-i][1]

    p_value = ttest_ind(a=s, b=max_row[1], equal_var=False).pvalue

    print("P-value: %.4f [%s]" % (p_value, l), end='\t')
    if p_value >= 0.05:
        print("Mean accuracies are equal")
    else:
        print("Mean accuracies are not equal")

# Зобразимо всі отримані точності

# In[36]:

import seaborn as sns
import pandas as pd

```



```

# In[50]:

data_to_plot = []
for row in range(5):
    data_temp = all_scores_clean[row][1]
    l = all_scores_clean[row][0]
    for d in data_temp:
        data_to_plot.append([l, d, 'Clean Data'])

    data_temp = all_scores_aug[row][1]
    l = all_scores_aug[row][0]
    for d in data_temp:
        data_to_plot.append([l, d, 'Clean & Augmented Data'])

data_to_plot = pd.DataFrame(data_to_plot, columns=['Model', 'Accuracy', 'Trained
on:'])

plt.figure(figsize=(20, 12))
sns.set(style="whitegrid", font_scale=1.3)
plt.title("Точності всіх описаних моделей")

ax = sns.boxplot(x="Model", y="Accuracy", hue="Trained on:", data=data_to_plot,
palette="Set1", width=0.5)

```