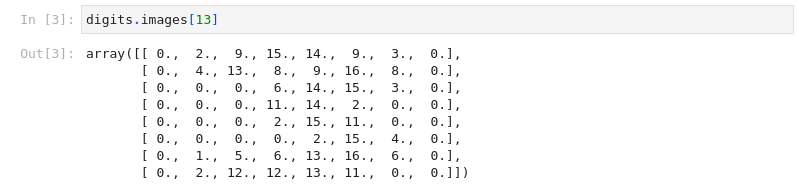
# Виконання

## Для дослідження даних, візуалізуйте їх. Виведіть зображення перших 24 і 36 цифр з набору

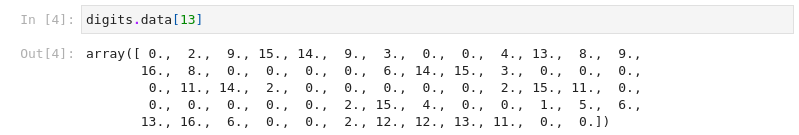
Для початку завантажимо набір даних digits за допомогою функції load\_digits з модуля sklearn.datasets, що повертає ою'єкт Bunch, що містить дані цифр.

  
  
Рисунок 3.1 - Набір даних digits

Оскільки зображення - це двовімірний об'єкт, то він володіє висотою та шириною у пікселях, то за дпомогою аотрибуту images виведемо матрицю для елементу з індексом 13.

  
  
Рисунок 3.2 - Матриця двовимірного забраження

Алгоритми машинного навчання scikit-learn потребують, щоб зразки були збережені в двовимірному масиві значень з плаваючою точкою. Проте, ми маємо масив матриць чисел з плаваючою точкою, тобто тривимірний об'єкт. Однак функція load\_digits повертає попередньо оброблені дані, які готові для машинного навчання. Набір даних Digits є числовим, тому load\_digits просто перетворює двовимірний масив на одновимірний масив.

  
  
Рисунок 3.3 - Одновимірний масив зображення

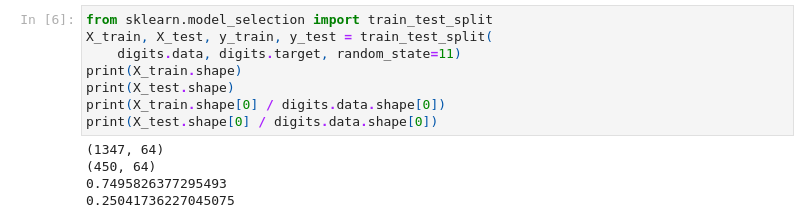
Маємо, що перші вісім елементів - це елементи рядка з індексом 0, наступні вісім - з індексом 1 і так далі.

Імпортуємо модуль matplotlib.pyplot та за допомогою методу imshow зобразимо перші 24 з 36 зображень. За допомогою методу ravel сплюснемо двовимірний масив осей до одновимірного та за допомогою zip одночасно ітеруємося як по осях, та і по картинках і їх назвах, тобто digits.target. Для зменшення padding'га між зображеннями застосуємо plt.tight\_layout.

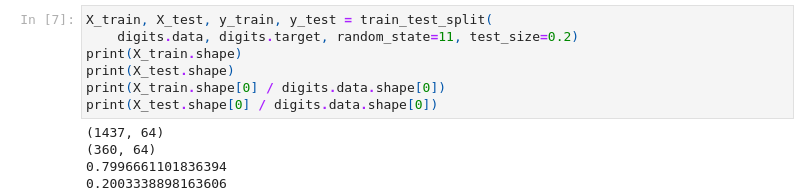
  
  
Рисунок 3.4 - Зображення перших 24 рисунків з масиву digits

## Розбийте дані на навчальні та тестові, за замовчуванням train\_test\_split резервує 75% даних для навчання і 25% для тестування, змініть це.

Розіб'ємо дані на навчальні та тестові. Імпортуємо функцію train\_test\_split з sklearn.model\_selection, що здійснює випадкову перестановку даних, а потім розбиває зразки в масиві data і цільові значення в масиві target на навчальний і тестовий набір. За замовчуванням функція розбиває тренувальні і тестові дані у співвіднощенні 75% до 25%.

  
  
Рисунок 3.5 - Розбиття даних на тренувальні та тестові у відношенні 75% до 25%

Щоб змінити це співвідношення, потрібно задати розміри навчального і тестового набору за допомогою параметрів за замовчуванням test\_size та train\_size функції train\_test\_split. Задамо розмір test\_size до 20%, а train\_size вираховується автоматично.

  
  
Рисунок 3.6 - Розбиття даних на тренувальні та тестові у відношенні 80% до 20%

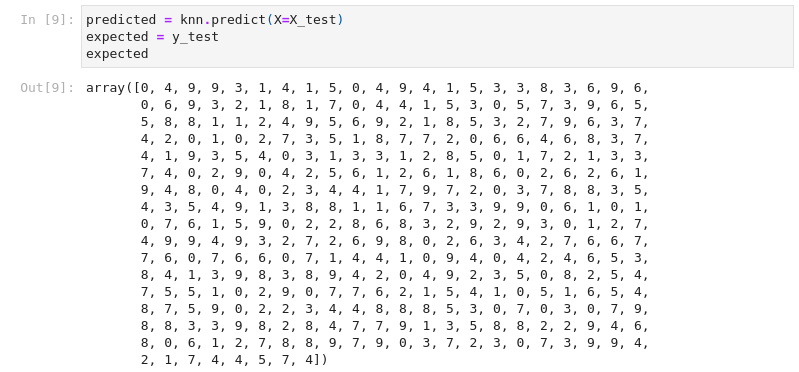
## Створити та навчити модель

Імпортуємо оцінювач KNeighborsClassifier з модуля sklearn.neighbors, який реалізує алгоритм K-Nearest Neighbors. Створимо екземпляр та викличемо метод fit, передавши в нього X\_train та y\_train. k - відповідає за кількість сусідів, яких буде враховано під час класифікації. За замовчуванням їх п'ять.

  
  
Рисунок 3.7 - Тренування моделі K-Nearest Neighbors

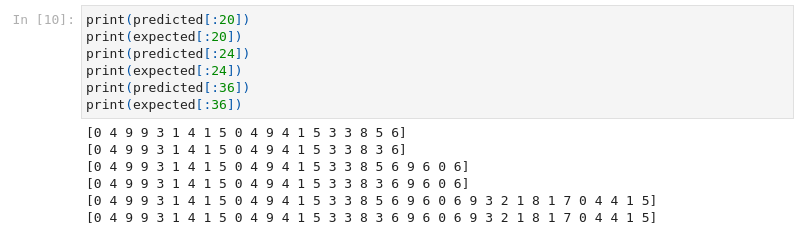
## Виконайте прогнозування класів

Викличемо метод fit в об'єкта knn, передавши в нього значення для пронозування, виведемо їх.

  
  
Рисунок 3.8 - Прогнозовані значення

## Порівняйте прогнозовані цифри з очікуваними для перших 20, 24, 36 тестових зразків

Виведемо перші 20 елементів масивів predicted та expected. Бачимо, що тільки передостанній елемент неправильно пронозований: замість п'ятірки маємо трійку.

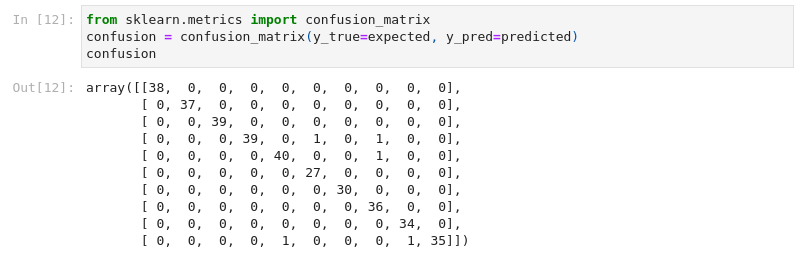
  
  
Рисунок 3.9 - Порівняння прогнозованих даних з очікуваними

## Поясніть результат, застосуйте метрики точності моделі

Оскільки кожен оцінювач має метод score, то виведемо оцінку результатів.

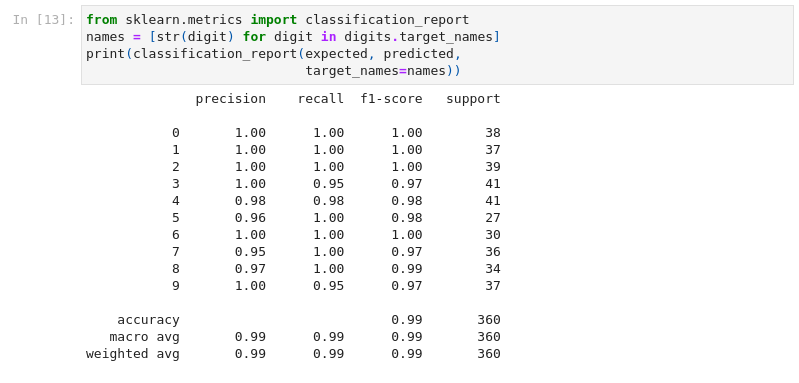
  
  
Рисунок 3.10 - Оцінка точності моделі

Виведемо інформацію про правильно і неправильно прогнозованих значеннях за допомогою матриці невідповідностей для заданого класу.

  
  
Рисунок 3.11 - Матриця невідповідностей

## Виведіть звіт класифікації

За допомогою функції classification\_report з модуля sklearn.metrics виведемо таблицю метрик класифікації, заснованих на очікуваних і пронозованих значеннях.

  
  
Рисунок 3.12 - Звіт класифікації

## Використайте декілька моделей KNeighborsClassifier, SVC і GaussianNB для пошуку найкращої

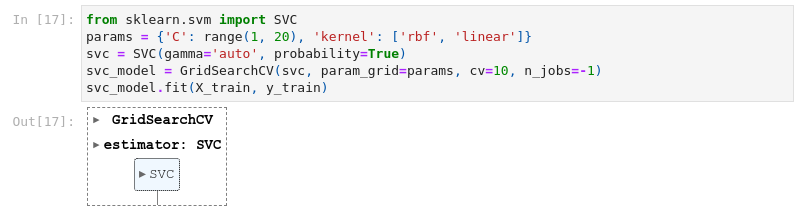
Ініціалізуємо список results, у який будемо додавати результати моделей.

  
  
Рисунок 3.13 - Список результатів моделей

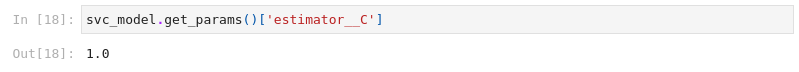
Імпортуємо клас GridSearch для знаходження оптимальних гіперпараметрів для кожної моделі.

  
  
Рисунок 3.14 - Імпортування GridSearchCV

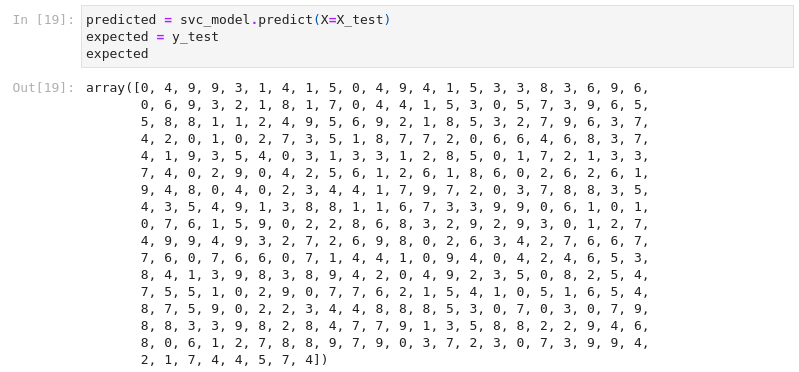
Імпортуємо SVC з модуля sklearn.svm. Ядрами для SVC будуть "rbf", "linear". Задамо параметр C, який контролює трейд-офф між гладкістю вирішальної границі та правильною класифікацією точок. Де чим більше С, тим краще класифікуються точки. cv — генератор перехресної перевірки або ітерація, у цьому випадку є 10-кратна перехресна перевірка. n\_jobs — максимальна кількість одночасно запущених воркерів; у цьому випадку встановлено значення -1, що означає, що використовуються всі ЦП.

  
  
Рисунок 3.15 - Тренування моделі SVM

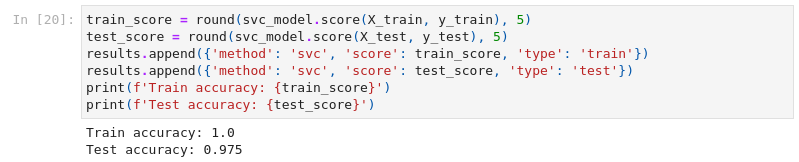
Покажемо оптимальні параметри моделі.

  
  
Рисунок 3.16 - Оптимальний параметр C

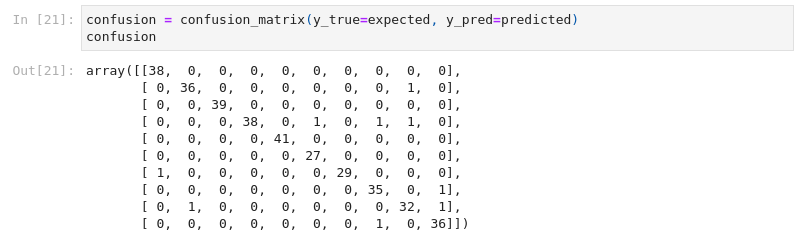
Викличемо метод predict в об'єкта svc\_model, передавши в нього значення для пронозування, виведемо їх.

  
  
Рисунок 3.17 - Прогнозовані значення моделі SVC

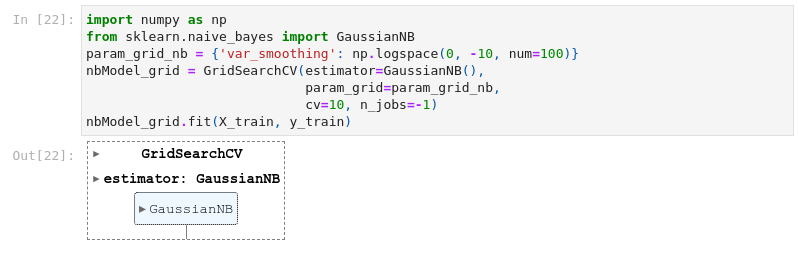
Визначимо точність моделі на тренувальних та тестових даних.

  
  
Рисунок 3.18 - Точність моделі SVC

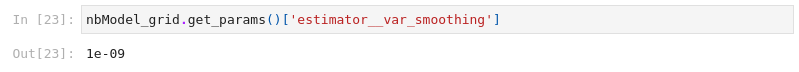
Виведемо інформацію про правильно і неправильно прогнозованих значеннях за допомогою матриці невідповідностей для заданого класу.

  
  
Рисунок 3.19 - Матриця невідповідностей моделі SVC

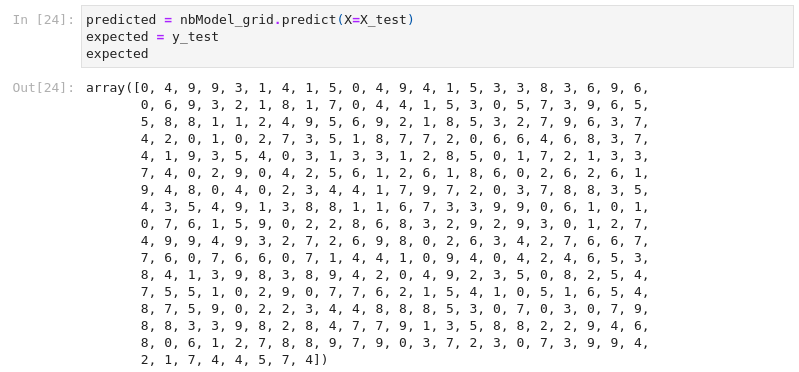
Імпортуємо GaussianNB з модуля sklearn.svm. Налаштуємо гіперпараметр var\_smoothing, який є обчисленням стабільності, щоб розширити (або згладити) криву і, отже, врахувати більше вибірок, які знаходяться далі від середнього розподілу. У цьому випадку np.logspace повертає числа, рівномірно розподілені в логарифмічній шкалі, починаючи з 0, закінчуючи -9, і генерує 100 зразків.

  
  
Рисунок 3.20 - Тренування моделі GaussianNB

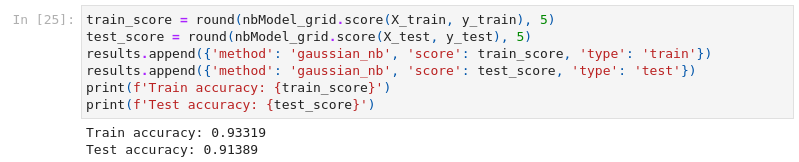
Покажемо оптимальні параметри моделі.

  
  
Рисунок 3.21 - Оптимальне значення параметру var\_smoothing

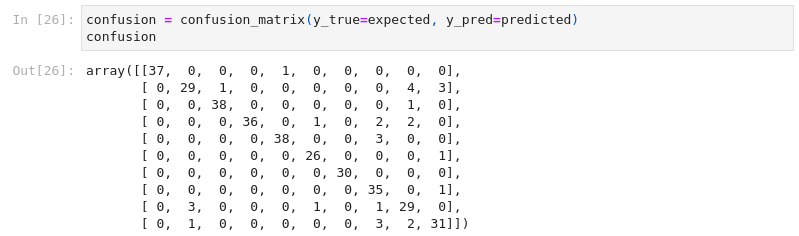
Викличемо метод predict в об'єкта nbModel\_grid, передавши в нього значення для пронозування, виведемо їх.

  
  
Рисунок 3.22 - Прогнозовані значення моделі GaussianNB

Визначимо точність моделі на тренувальних та тестових даних.

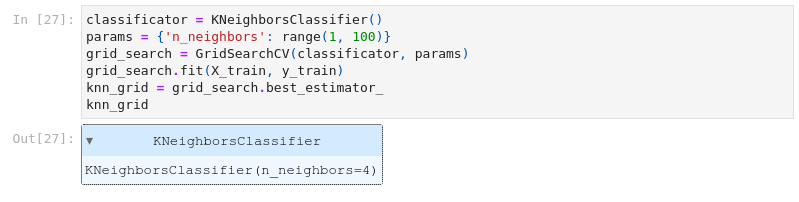
  
  
Рисунок 3.23 - Точність моделі GaussianNB

Виведемо інформацію про правильно і неправильно прогнозованих значеннях за допомогою матриці невідповідностей для заданого класу.

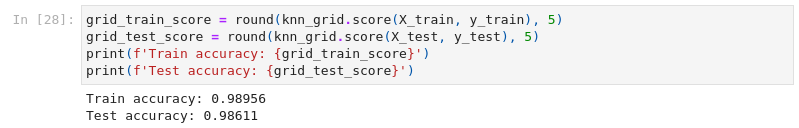
  
  
Рисунок 3.24 - Матриця невідповідностей моделі GaussianNB

## Налаштуйте гіперпараметр K в KNeighborsClassifier

Цього разу налаштуємо гіперпараметри за допомогою Gridsearh та звичайного циклу. Нехай будемо мати відрізок від 1 до 99 включно та визначимо найкращі параметри та порівняємо результати, перебираючи кількість сусідів.

  
  
Рисунок 3.25 - Визначення найкращого параметра за допомогою GridSearchCV

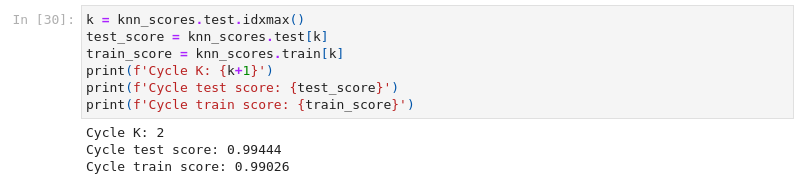
Визначимо точність моделі на тренувальних та тестових даних

  
  
Рисунок 3.26 - Точність моделі K-Nearest Neighbors GridSearchCV

Натренуємо моделі в циклі та визначимо найкращу.

  
  
Рисунок 3.27 - Визначення найкращого параметра за допомогою циклу

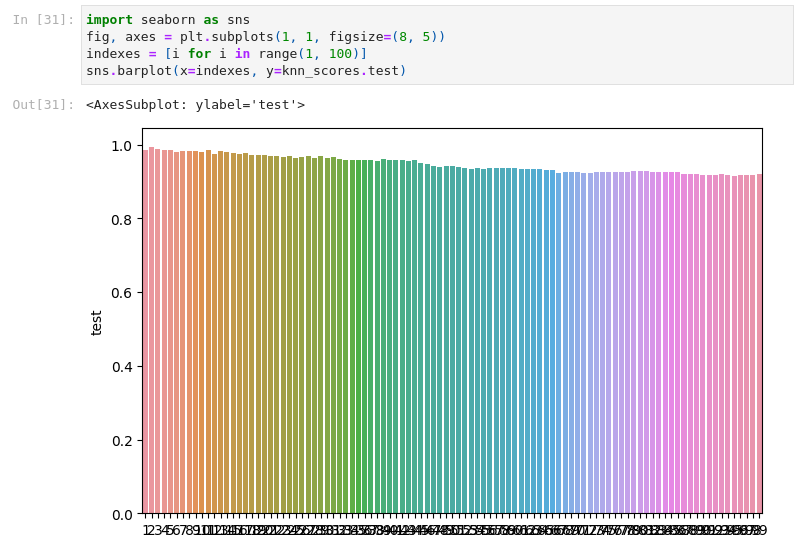
Визначимо найкращий параметр для KNN у циклі.

  
  
Рисунок 3.28 - Точність моделі K-Nearest Neighbors у циклі

Бачимо, що GridSearch дуже близько підібрався до оптимального параметра. У циклі модель виявилася точнішою і k = 2 із тестовою точністю в 0.99444, де gridSearch показав, що найкращим є k = 4 із тестовою точністю в 0.98611. Додамо knn з циклу у список results.

  
  
Рисунок 3.29 - Додавання найкращого результату KNN до результатів

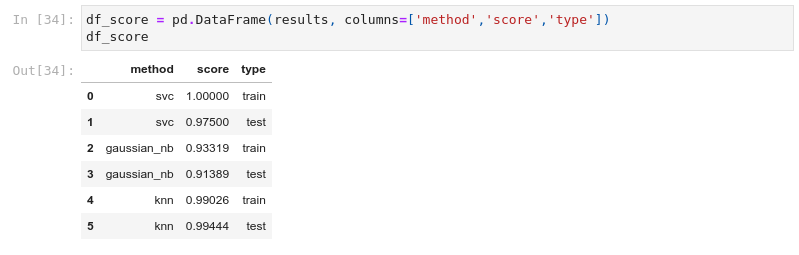
Подивимося залежність точності KNN від параметру k у викгляді гістограми.

  
  
Рисунок 3.30 - Зображення залежності точності KNN від значення k

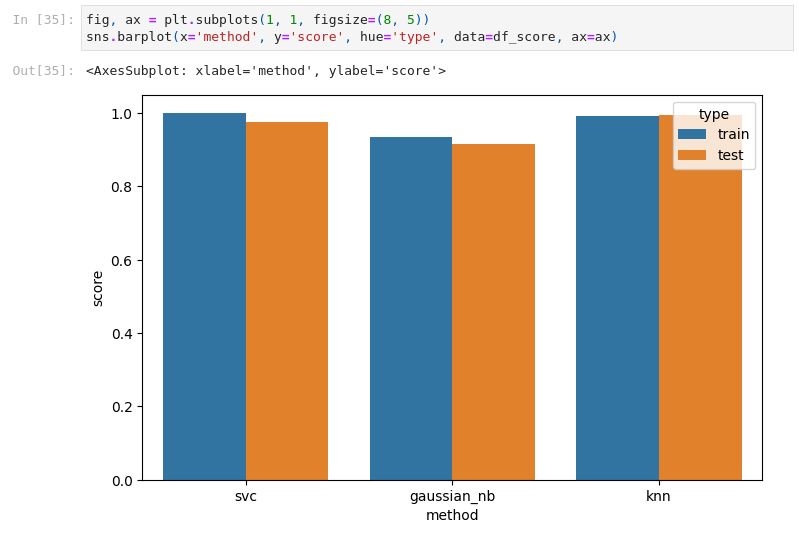
Бачимо, що зі збільшенням k точність спадає.

## Порівняння результатів моделей KNN, GaussianNB, SVC

Порівняємо отримані результати моделей. Утворимо датафрейм результатів.

  
  
Рисунок 3.31 - Датафрейм результатів

Для наочності побудуємо гістограму.

  
  
Рисунок 3.32 - Результати моделей

З огляду бачимо, що як на тренувальних, так і на тестових найгірше себе показав GaussianNB. Краще за нього відпрацював SVC. Однак з-поміж них усіх найкраще спрогнозував результати KNN.