**МІНІСТЕРСТВО ОСВІТИ І НАУКИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп'ютерних наук та інформаційних технологій**

**Кафедра систем штучного інтелекту**



**Звіт до лабораторної роботи №2**

**з дисципліни “ Машинне навчання ”**

**Виконав**:

ст. гр. КН-311

Ткачук Орест

**Викладач:**

Бойчук А. Р.

Львів – 2021

**Мета:** засвоїти основні відомості про роботу з алгоритмом метод головних компонент(PCA) та дерева рішень

Завдання

1. Частина 1. Підготовка даних
   1. Завантажити дані згідно варіанту.
   2. Розділити на тренувальну, валідаційну та тестові вибірки в пропорції 70/15/15
   3. Обробити NaN значення - або видаленням, або заповненням середнім, або якимось іншим чином. Обраний варіант обгрунтувати.
   4. Нормалізувати дані - для тренувальних даних обрахувати значення mean(середнього) та standard deviation(стандартного відхилення) для кожної з ознак і перетворити дані віднявши обраховане середнє та поділивши на стандартне відхилення. Зауважте, що на відміну від Z-score тут немає модуля.
   5. Нормалізувати валідаційні та тестові даних на основі обрахованих mean та std для тренувальних даних
2. Частина 2. Побудова дерева рішень
   1. Побудувати 3 різні дерева рішень(Decision tree) з різною глибиною на основі даних згідно варіанту.
   2. Візуалізувати отримані дерева рішень. Порівняти точність отриманих дерев рішень на валідаційній та тренувальній вибірках.
3. Частина 3. Зашумлення.
   1. Добавити до тренувальних даних шум. Зробити це додаючи випадкове(uniform(рівномірний) розподіл в межах [-0.1,0.1]) зміщення до кожного елементу тренувальної вибірки датасету по кожній з ознак.
   2. Побудувати 3 різні дерева рішень(Decision tree) з різною глибиною на основі даних згідно варіанту.
   3. Візуалізувати отримані дерева рішень. Порівняти точність отриманих дерев рішень на валідаційній та тренувальній вибірках.
4. Частина 4. Знешумлення
   1. Обрахувати на оригінальній тренувальній вибірці PCA. Візуалізувати отримані компоненти.
   2. Обрахувати на зашумленій тренувальній вибірці PCA. Візуалізувати отримані компоненти.
   3. Реконструювати дані використовуючи PCA обрахований на зашумлених даних для знешумлення. Для цього використати ознаки що пояснюють 95% variance.
   4. Візуалізувати оригінальні, зашумлені і знешумлені дані
5. Частина 5
   1. Побудувати на основі зашумлених і знешумлених даних тренувальної вибірки дерева рішень. Порівняти їхню точність на валідаційній та тренувальних вибірках.
   2. Візуалізувати отримані дерева рішень

**Виконання:**

**Варіант 21**

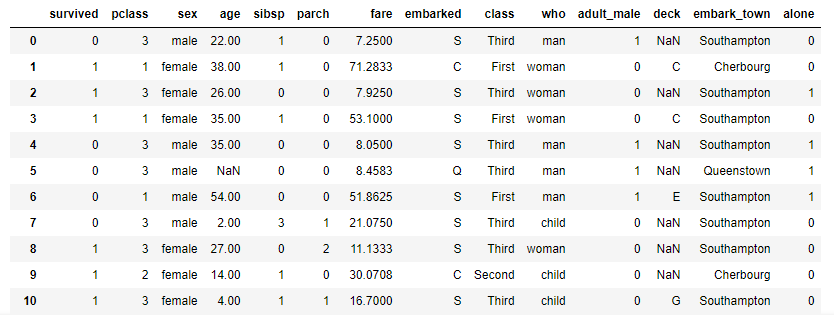
|  |  |  |
| --- | --- | --- |
| Варіант # | Датасет | Назва колонки яку потрібно передбачити |
| 1 | titanic | 'survived', |

1. Завантажуємо дані згідно варіанту, приводимо True i False до 1 і 0, а також видаляємо колонку alive (оскільки вона надалі буде однозначно визначати колонку, яку нам потрібно передбачити за допомогою дерева рішень)

titanic = sns.load\_dataset('titanic')

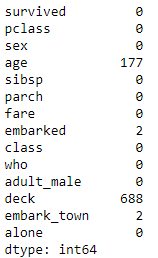
titanic.drop('alive', inplace=True, axis=1)

titanic



Перевіряємо значення на NaN

titanic.isnull().sum()



Одна з колонок є числовою і має невелику кількість відсутніх значень, тому їх можна замінити середнім. Друга колонка має категоріальні дані як відсутні в великій кількості (майже 80%) тому її можна видалити. Третя має відсутні лише 2 значення, які ніяк не вплинуть на наш датасет. Також перетворюємо категоріальні дані за допомогою one-hot encoding.

titanic = titanic.dropna(subset=['embarked','embark\_town'])

titanic.drop('deck', inplace=True, axis=1)

titanic['age'] = titanic['age'].fillna(int(np.mean(titanic['age'])))

for i in columns\_str:

if i != 'adult\_male' and i != 'alone':

new = pd.get\_dummies(titanic[i], prefix=i)

titanic.drop(i, inplace=True, axis=1)

titanic = pd.concat([titanic, new], axis=1)

Розділяємо дані на тренувальну, тестову та валідаційну вибірки в відношенні 70/15/15

train, validate, test = np.split(titanic.sample(frac=1, random\_state=42), [int(.7\*len(titanic)), int(.85\*len(titanic))])

Нормалізовуємо числові тренувальні дані обраховуючи для них mean i std. Нормалізовуємо числові тестові та валідаційні дані за допомогою обрахованих для тренувальних mean i std.

for i in columns\_num:

mean = np.mean(train[i])

std = np.std(train[i], ddof=1)

train[i] = (train[i]-mean)/std

validate[i] = (validate[i]-mean)/std

test[i] = (test[i]-mean)/std

1. Побудова дерева рішень. Будуємо 3 різні дерева рішень(Decision tree) з різною глибиною та візуалізовуємо їх.

train\_x = train.loc[:, train.columns != 'survived']

train\_y = train.loc[:, train.columns == 'survived']

for i in range(3,8,2):

clf = tree.DecisionTreeClassifier(max\_depth = i)

clf.fit(train\_x, train\_y)

test\_x = deepcopy(test.loc[:, test.columns != 'survived'])

test\_y = deepcopy(test.loc[:, test.columns == 'survived'])

validate\_x = deepcopy(validate.loc[:, validate.columns != 'survived'])

validate\_y = deepcopy(validate.loc[:, validate.columns == 'survived'])

print('first train score ('+str(i)+f' depth) = {clf.score(train\_x, train\_y)}')

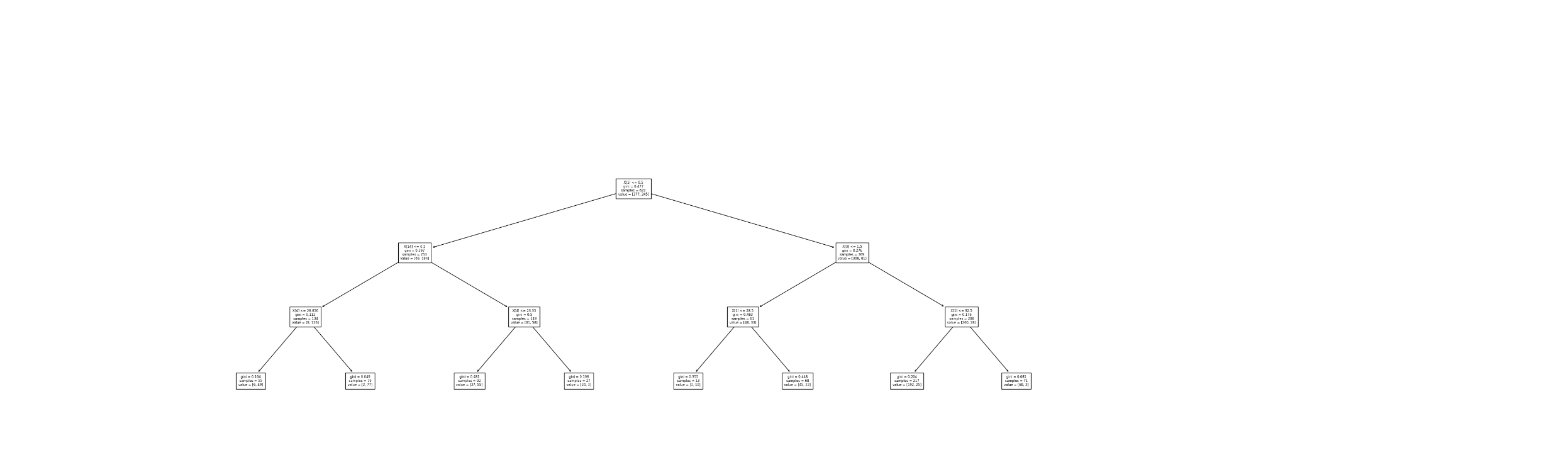
print('first test score ('+str(i)+f' depth) = {clf.score(test\_x, test\_y)}')

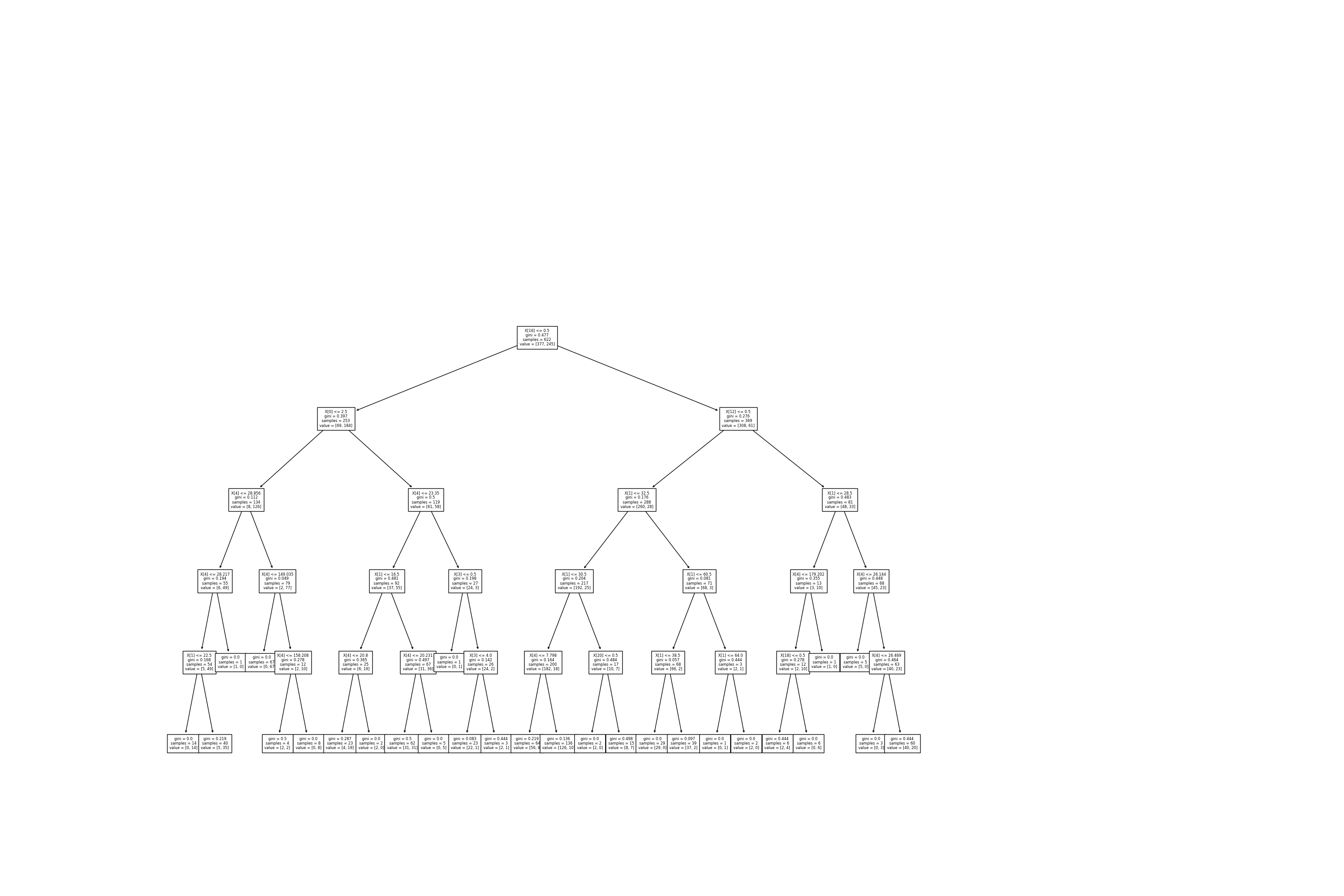
print('first validate score ('+str(i)+f' depth) = {clf.score(validate\_x, validate\_y)}\n')

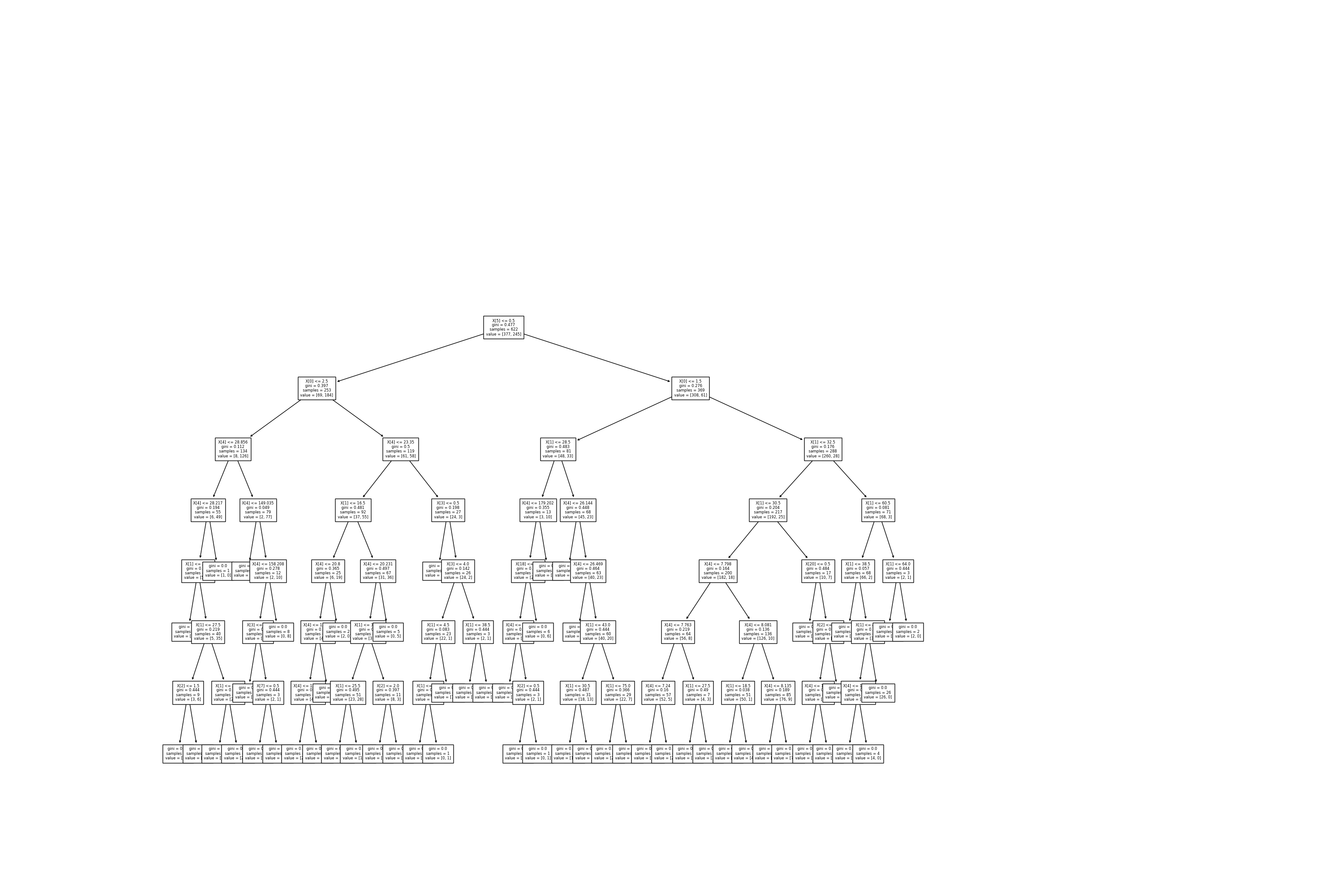
plt.figure(figsize=(40,12))

tree.plot\_tree(clf, fontsize=6)

plt.savefig('tree\_high\_dpi'+str(i), dpi=100)







train score (3 depth) = 0.8360128617363344

test score (3 depth) = 0.7910447761194029

validate score (3 depth) = 0.8345864661654135

train score (5 depth) = 0.8504823151125402

test score (5 depth) = 0.7835820895522388

validate score (5 depth) = 0.7969924812030075

train score (7 depth) = 0.8906752411575563

test score (7 depth) = 0.753731343283582

validate score (7 depth) = 0.7894736842105263

1. Додаємо рівномірний шум в межах [-0.1; 0.1] до числових даних.

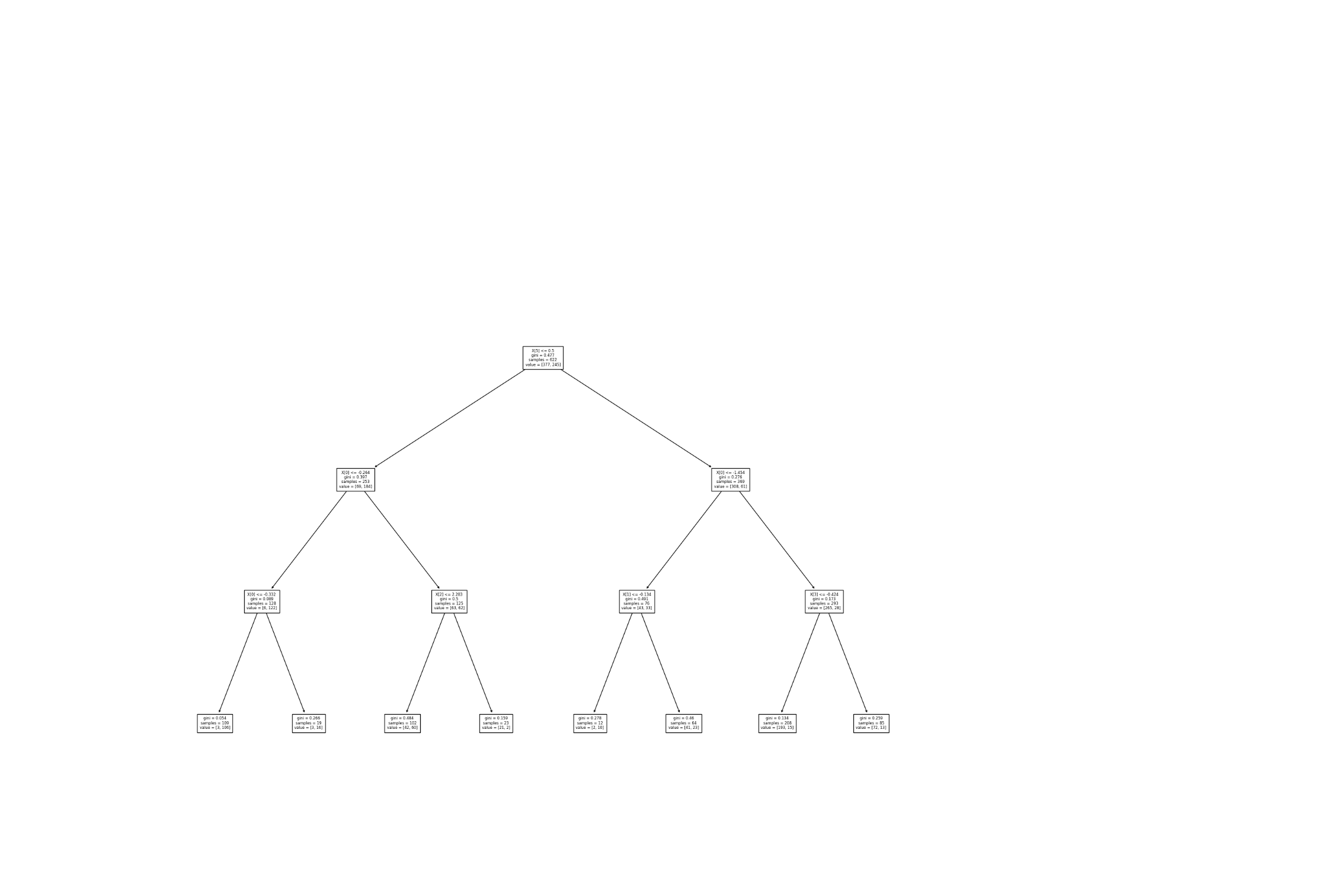
noise\_train = deepcopy(train)

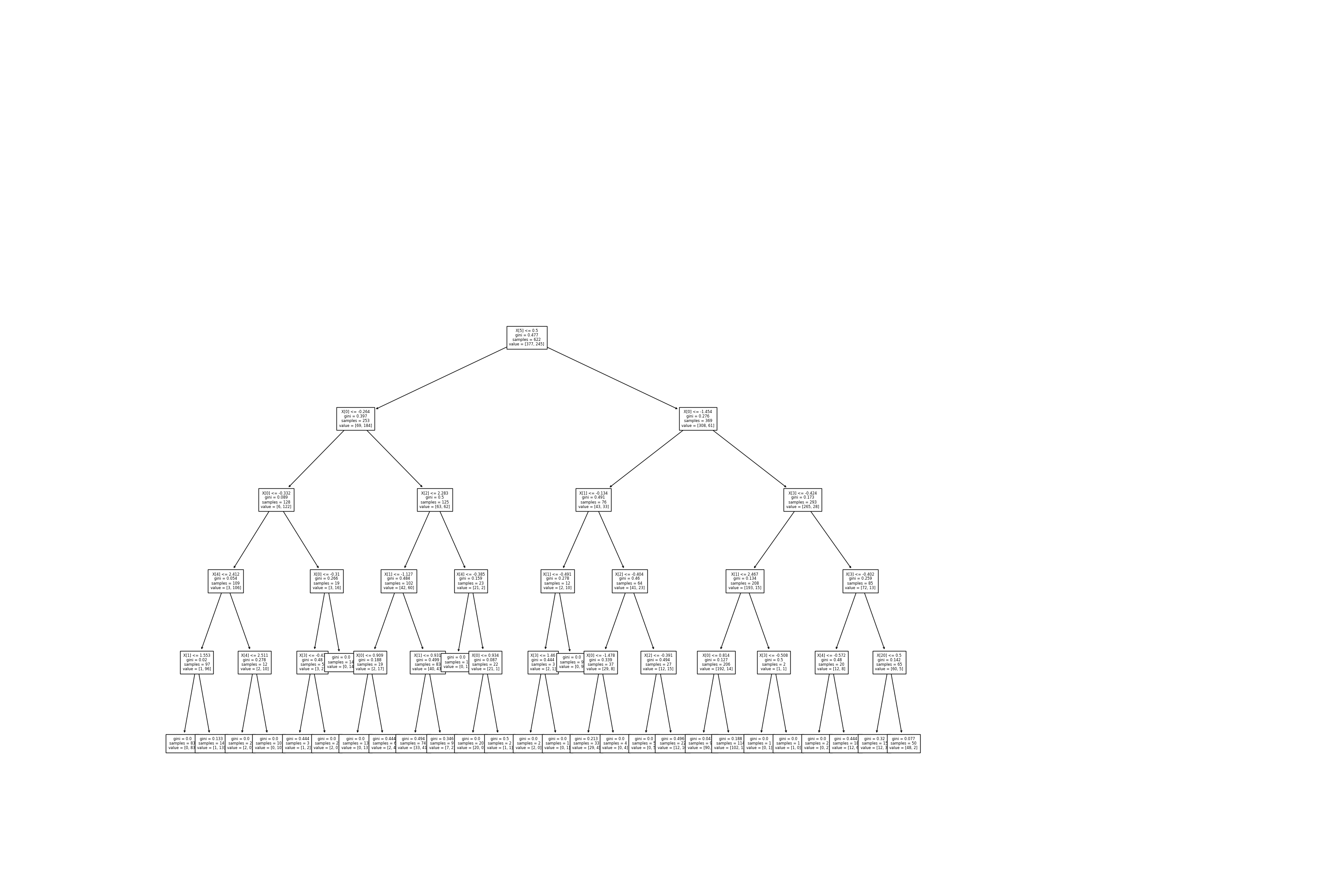
for i in columns\_num:

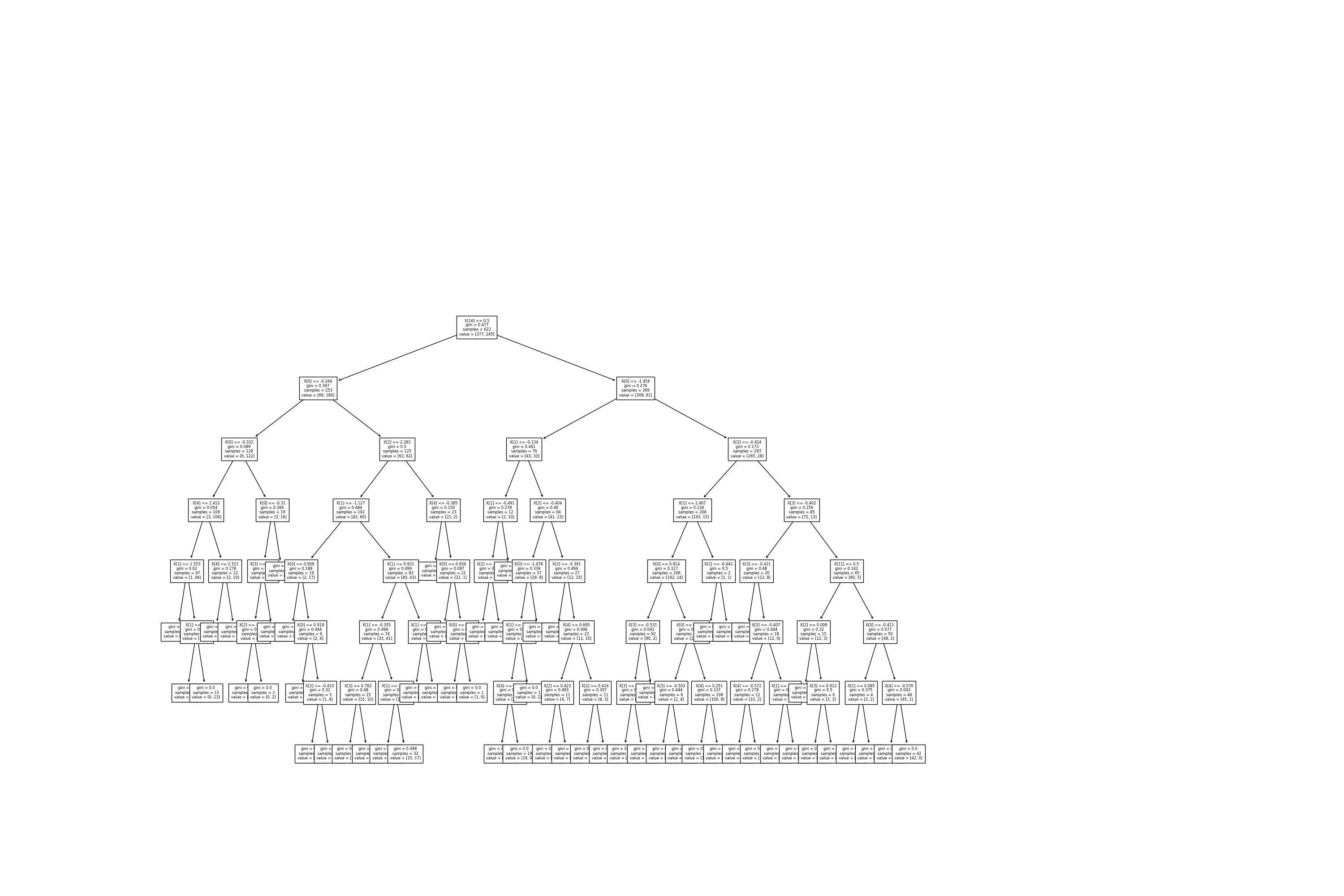
noise = np.random.uniform(-0.1,0.1, len(train[i]))

#print(noise\_train[i] - (noise\_train[i]-noise))

noise\_train[i] += noise







train score (3 depth) = 0.8295819935691319

test score (3 depth) = 0.7835820895522388

validate score (3 depth) = 0.8345864661654135

train score (5 depth) = 0.8215434083601286

test score (5 depth) = 0.753731343283582

validate score (5 depth) = 0.8195488721804511

train score (7 depth) = 0.8167202572347267

test score (7 depth) = 0.7611940298507462

validate score (7 depth) = 0.7593984962406015

1. Обраховуємо і візуалізовуємо PCA для звичайних і зашумлених даних. Знешумлюємо зашумлені дані.

norm = copy(train[['age', 'fare']])

pca = PCA(0.95).fit(noisy)

plot\_pca(norm, 'age', 'fare', pca)

components = pca.transform(noisy)

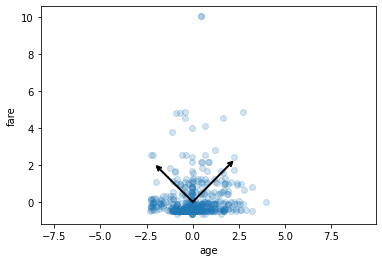
norm\_filtered = pca.inverse\_transform(components)

norm\_restored = copy(train)

norm\_restored['age'] = norm\_filtered.T[0]

norm\_restored['fare'] = norm\_filtered.T[1]

PCA(0.95) для оригінальних даних



Explained variance = [1.11029201 0.90083368]

Components = [[ 0.70339493 0.71079925][-0.71079925 0.70339493]]

Mean = [-0.00359274 -0.003597 ]

PCA(0.95) для зашумлених даних

noisy = copy(noise\_train[['age', 'fare']])

pca = PCA(0.95).fit(noisy)

plot\_pca(noisy, 'age', 'fare', pca)

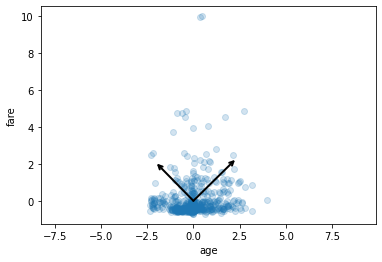
components = pca.transform(noisy)

noisy\_filtered = pca.inverse\_transform(components)

noisy\_restored = copy(train)

noisy\_restored['age'] = noisy\_filtered.T[0]

noisy\_restored['fare'] = noisy\_filtered.T[1]

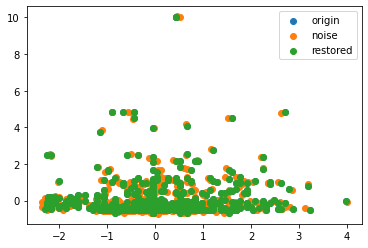


Explained variance = [1.11092597 0.88614646]

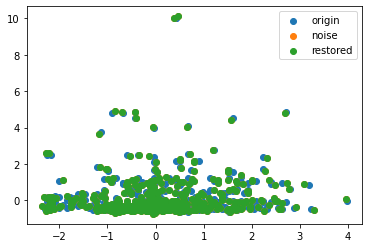
Components = [[ 0.71317933 0.70098162][-0.70098162 0.71317933]]

Mean = [-0.00115508 -0.00025016]

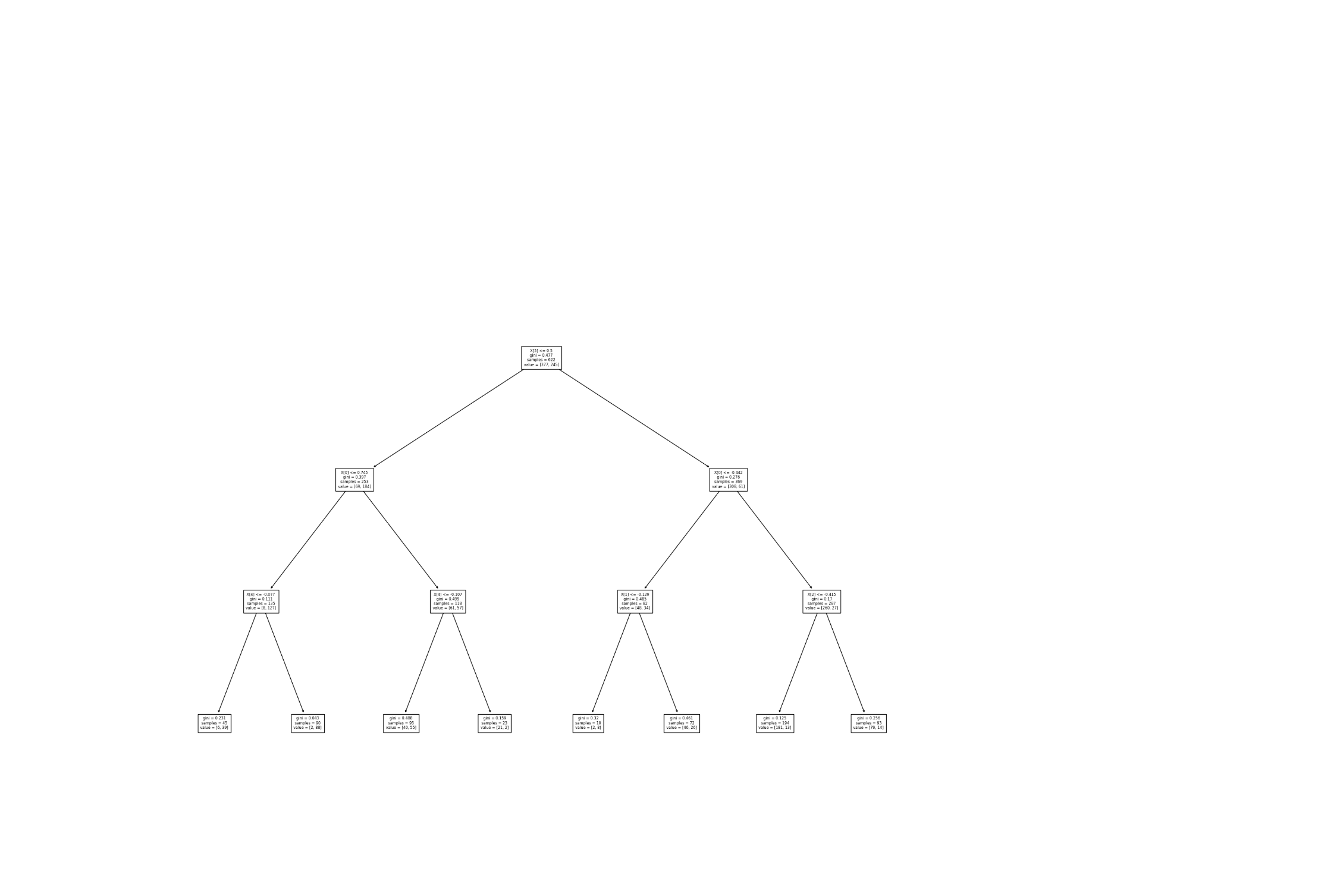
Порівняння оригінальних, зашумлених, знешумлених і відновлених оригінальних даних

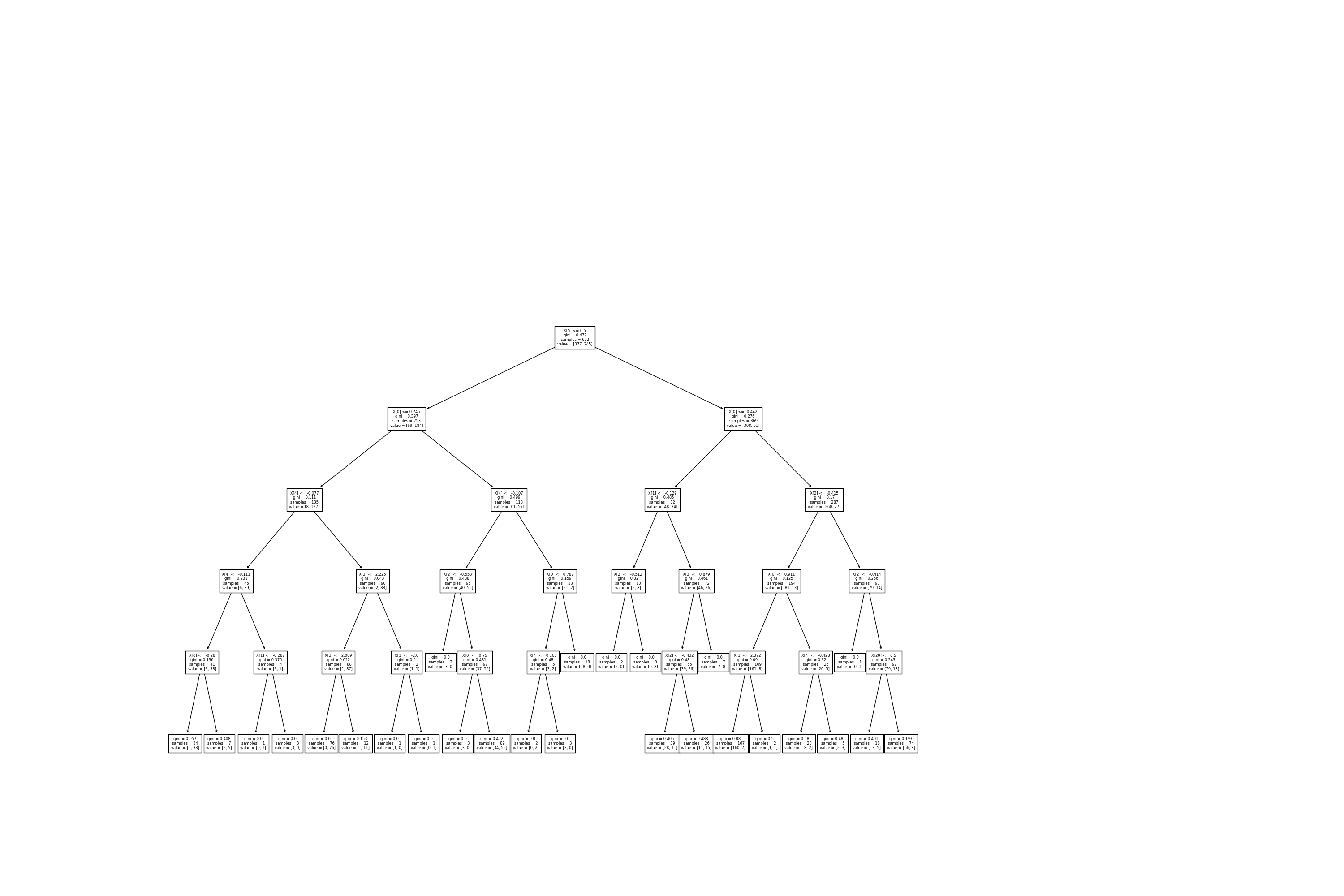


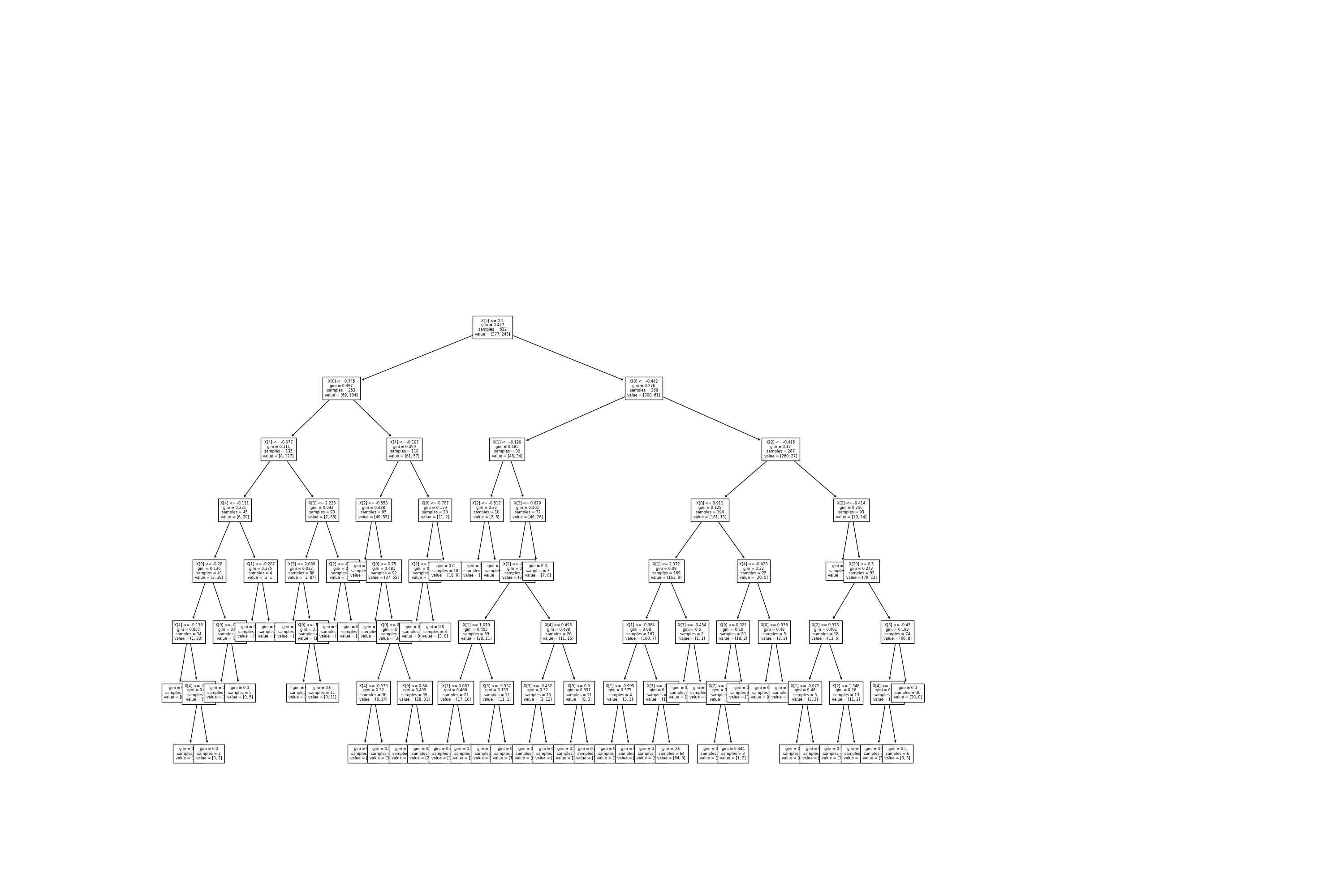
Порівняння оригінальних, зашумлених, знешумлених і відновлених зашумлених даних



1. Будуємо дерево рішень на основі знешумлених даних і порівнюємо точність з попередніми.







train score (3 depth) = 0.8311897106109325

test score (3 depth) = 0.7835820895522388

validate score (3 depth) = 0.8345864661654135

train score (5 depth) = 0.8585209003215434

test score (5 depth) = 0.7686567164179104

validate score (5 depth) = 0.8345864661654135

train score (7 depth) = 0.9292604501607717

test score (7 depth) = 0.746268656716418

validate score (7 depth) = 0.7894736842105263

|  |  |  |  |
| --- | --- | --- | --- |
| Depth | Original (train) | Noise (train) | Restored (train) |
| 3 | 0.836012861736 | 0.82958199356913 | 0.831189710610 |
| 5 | 0.85048231511254 | 0.82154340836012 | 0.858520900321 |
| 7 | 0.89067524115755 | 0.81672025723472 | 0.929260450160 |

|  |  |  |  |
| --- | --- | --- | --- |
| Depth | Original (test) | Noise (test) | Restored (test) |
| 3 | 0.79104477611940 | 0.78358208955223 | 0.783582089552 |
| 5 | 0.78358208955223 | 0.7537313432835 | 0.768656716417 |
| 7 | 0.75373134328358 | 0.76119402985074 | 0.746268656716 |

|  |  |  |  |
| --- | --- | --- | --- |
| Depth | Original (validate) | Noise (validate) | Restored (validate) |
| 3 | 0.83458646616541 | 0.83458646616541 | 0.834586466165 |
| 5 | 0.79699248120300 | 0.81954887218045 | 0.834586466165 |
| 7 | 0.79699248120300 | 0.75939849624060 | 0.789473684210 |