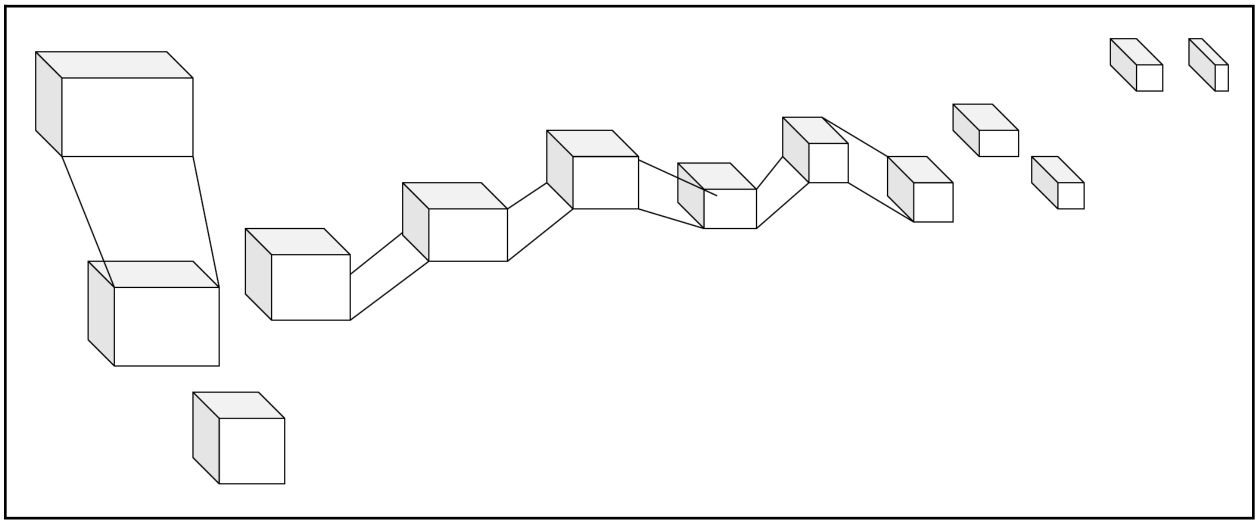
The higher order dimensions algorithm based on cube replication and extrapolation of different sets. The main idea is to compute in Scikit-python the high dimensions to recreate new mathematical and computational scenarios with python. In this example, I show how algorithmic logarithmic regression works to elucidate some important machine learning concepts for Scikit and to propose another level of computational analysis.



We have some functions with logarithms and vectors as exponents.
The logarithmic regression works across different higher dimensions that change through a constant t or time and the dimensions replicate each other in a non linear time.The constant time Is not uniform, is chaotic and it depends on regressive algorithms to commute with each other in different cube states or dimensional states.
Composition of higher dimensions:

Algebras from vector and hyperplane arrangements
There are several natural algebraic spaces related to the Tutte polynomial arising in commutative algebra, hyperplane arrangements, box splines, and index theory; we discuss a few. For each hyperplane H in a hyperplane arrangement A in k d let lH be a linear function such that H is given by the equation lH(x) = 0.

f(x)=1+logx6,logx6(2x+4)−log6P

Prop:

f(x) 1+logx6(Dimension1)=logx6(Dimension2)*

VECTORS (variable(exp3)<===   <===variable(exp3))

1+logx6(Dimension3) = $logx6\ (2\ dimension2)\ +\ logx6(dimension1)$

A matroid subdivision is a polyhedral subdivision P of a matroid polytope PM where every polytope P ∈ P is itself a matroid polytope. Equivalently, it is a subdivision of PM whose only edges are the edges of PM. In the most important case, M is the uniform matroid Ud,and PM is the hypersimplex Δ(d, n).

Matroid subdivisions arose in algebraic geometry [HKT06, Kap93, Laf03], in the theory of evaluated matroids [DW92, Mur96], and in tropical geometry [Spe08]. For instance, Lafforgue showed that if a matroid polytope PM has no nontrivial matroid subdivisions, then the matroid M has (up to trivial transformations) only finitely many realizations over a fixed field F. This is one of very few results about realizability of matroids over arbitrary fields.

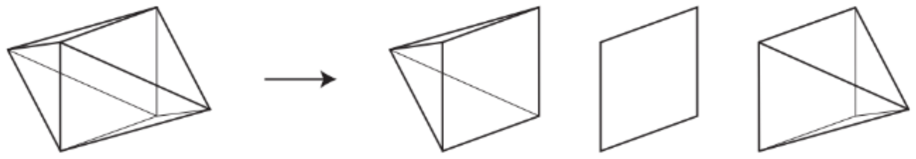


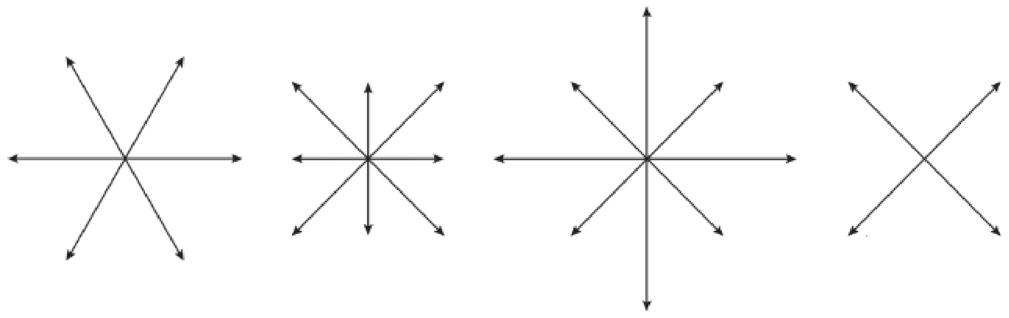
Figure 7.8: A matroid subdivision of $\Delta(2, 4)$.

This formula is straightforward in terms of coboundary polynomials: χM(k) (X, Y) = χM(X, Y K). For an extensive generalization, see Section 7.9.

• [Ard07a, Mph00] Root systems are arguably the most important vector configurations; these highly symmetric arrangements play a fundamental role in many branches of mathematics. For the general definition and properties, see for example [Hum90]; we focus on the four infinite families of classical root systems

An−1 = {ei − ej , : 1 ≤ i < j ≤ n}
Bn = {ei − ej , ei + ej : 1 ≤ i < j ≤ n} ∪ {ei
: 1 ≤ i ≤ n}
Cn = {ei − ej , ei + ej : 1 ≤ i < j ≤ n} ∪ {2ei
: 1 ≤ i ≤ n}
Dn = {ei − ej , ei + ej : 1 ≤ i < j ≤

illustrates the two-dimensional examples.



Regression equations in 562 scikit-learn

```
import numpy as np
from sklearn.datasets import load_boston
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
from sklearn.model_selection import cross_val_score
rng = np.random.RandomState(0)
dataset = load_boston()
X_full, y_full = dataset.data, dataset.target
n_samples = X_full.shape[0]
n_features = X_full.shape[1]
# Estimate the score on the entire dataset, with no missing
values
estimator = RandomForestRegressor(random_state=0,
n_estimators=100)
score = cross_val_score(estimator, X_full, y_full).mean()
print("Score with the entire dataset = %.2f" % score)
# Add missing values in 75% of the lines
missing_rate = 0.75
n_missing_samples = np.floor(n_samples * missing_rate)
missing_samples = np.hstack((np.zeros(n_samples -
n_missing_samples,
dtype=np.bool),
np.ones(n_missing_samples,
dtype=np.bool)))
rng.shuffle(missing_samples)
missing_features = rng.randint(0, n_features,
n_missing_samples)
# Estimate the score without the lines containing missing
```

```
values
X_filtered = X_full[~missing_samples, :]
y_filtered = y_full[~missing_samples]
estimator = RandomForestRegressor(random_state=0,
n_estimators=100)
score = cross_val_score(estimator, X_filtered,
y_filtered).mean()
print("Score without the samples containing missing values =
%.2f" % score)
# Estimate the score after imputation of the missing values
X_missing = X_full.copy()
X_missing[np.where(missing_samples)[0], missing_features] = 0
y_missing = y_full.copy()
estimator = Pipeline([("imputer", Imputer(missing_values=0,
strategy="mean",
axis=0)),
("forest", RandomForestRegressor(random_state=0,
n_estimators=100))])
score = cross_val_score(estimator, X_missing,
y_missing).mean()
print("Score after imputation of the missing values = %.2f" %
score)
```