Python - P57

super ([type.[, object-or-type]])

return proxy object that delegate method call to parent / sibling class of type

useful for accessing inherited method that have been overriden in class

object-or-type : determine method resolution order to be searched

base class searched for member during lookup in the order

search start from class right after the <type>

__mro__() attribute of object-or-type list method resolution search order

used by both getattr() and super()

dynamic and can change whenever inheritance hierarchy updated

ex. object-or-type.__mro__() : D -> B -> C -> A -> object

type : B

super(type, object-or-type) search C -> A -> object

if object-or-type omitted, then super object returned unbound

if object-or-type is object

then isinstance(object-or-type, type) must be TRUE

if object-or-type is type (useful for classmethod)

then issubclass(object-or-type, type) must be TRUE

super() used to refer to parent class without naming explicitly, make code maintainable

use closely parallel use of super() in other programming language

or to support cooperative multiple inheritance in dynamic execution environment

unique to Python and not found in statically compiled language or only support single inheritance

possible to implement "diamond diagram", multiple base classes implement same method

good design dicate this method have same calling signature in every case

because order of calls determined at runtime

order adapt to change in class hierarchy

can include sibling class unknown prior to runtime

typical superclass call :

```
class C(B):
    def method(self, *args, **kw):
        super().method(*args, **kw)
```

等价于 super(C, self).method(*args, **kw)

Python - P58

super ([type[, object-or-type]])

super() also work for attribute lookup

possible use case called descriptor in parent / sibling class

super() implemented as part of binding process for explicit dotted attribute lookup

ex. super().__getItem__(name)

by implementing own __getattribute__() method

for searching class in predictable order support cooperative multiple inheritance

undefined for implicit lookup using statement/operator, ex. super()[name]

super() not limited to use inside method aside from zero argument form

two argument form specify argument exactly and make appropriate reference

zero argument form only work inside class definition

as compiler file in necessary detail to correctly retrieve class being defined

as well as accessing current instance for ordinary method

class type(object)

class type(name, bases, dict)

with one argument, return type of <object>

returned value is type object, generally same object as object.__class__

isinstance() built-in function recommended for testing type of object

since take subclass into account

with three argument, return new type object

essentially dynamic form of class statement

name: become __name__ attribute

bases: tuple itemize base classes, become __bases__ attribute

dict: dictionary as namespace containing definition for class body

and copied to standard dictionary to become __dict__ attribute

ex. C = type('C', (object,), dict(a=1, b=2))

類似于 class C(object):

a = 1

b = 2

有 c = C()

c.a ⟶ 1

c.b ⟶ 2

Python - P59

zip (*iterables), return iterator of tuple
where i-th tuple contain i-th element from each of argument sequence/iterable
iterator stop when shortest input iterable exhausted
with single iterable argument, return iterator of 1-tuple
with no argument, return empty iterator
left-to-right evaluation order of iterable guaranteed
possible idiom for clustering data series into n-length group
using zip(*[iter(s)]*n)
repeat same iterator n time, each output tuple has result of n calls to iterator
has effect of dividing input into n-length chunk
zip() should only be used with unequal length input
when not care about trailing, unmatched value from longer iterable
or use itertools.zip_longest() instead

```
def zip(*iterables):
    sentinel = object()
    iterators = [iter(it) for it in iterables]
    while iterators:
        result = []
        for it in iterators:
            elem = next(it, sentinel)
            if elem is sentinel:
                return
            result.append(elem)
        yield tuple(result)
```

zip() in conjunction with * operator can be used to unzip list

```
x, y = [1,2,3], [4,5,6]
list(zip(x,y)) ⟶ [(1,4),(2,5),(3,6)]
x1, y1 = zip(*zip(x,y))
list(x1) ⟶ [1,2,3]
list(y1) ⟶ [4,5,6]
```

Python - P60

__import__(name, globals = None, locals = None, fromlist = (), level = 0)

invoked by import statement.

direct use discouraged in favor of importlib.import_module()

function import module <name>

potentially using given <globals> and <locals>

to determine how to interpret name in package context

<fromlist> give name of object [redacted] or submodule

should be imported from module given by <name>

standard implementation not use <locals> argument at all

use <globals> only to determine package context of import statement

<level> specify whether to use absolute / relative import

default 0 only perform absolute import

positive value for <level> indicate number of parent directory

to search relative to directory of module calling __import__()

≠ <name> of form package.module

top-level package returned, not module named by <name>

when non-empty <fromlist> given

module named by <name> returned

to import spam result in bytecode resembling code

spam = __import__('spam', globals(), locals(), [], 0)

import spam.ham result in

spam = __import__('spam.ham', globals(), locals(), [], 0)

return toplevel module [redacted] because object bound to name by impore statement

from spam.ham [redacted] import eggs, sausage as saus , result in

_temp = __import__('spam.ham', globals(), locals(), ['eggs', 'sausage'], 0)

eggs = _temp.eggs

saus = _temp.sausage

spam.ham module returned from __import__()

name to import retrieved and assigned to respective name

importlib.import_module() simply import module by name

Python - P61

cluster analysis, preprocessing : summarization : preprocess for regression, PCA, classification
compression : image processing / vector quantization
search : finding K-nearest neighbour
localizing search to small number of cluster
detection : outlier

quality : high intra-class similarity : cohesive within cluster
low inter-class similarity : distinct between cluster
clustering method : similarity measure / implementation
ultimately ability to discover hidden pattern

issues : scalability : cluster all data instead of sample
ability to deal with different type of attribute
constraint from user / domain
interpretability and usability
ability to deal with noisy data / high dimensionality

approach : partitioning : construct and evalute (K-means, K-medoids, CLARANS)
hierarchical : decompose set of data object (Diana, Agnes, BIRCH, CAMELEON)
density-based : use connectivity / density function (DBSACN, OPTICS, DenClue)
grid-based : build multi-level granularity structure (STING, WaveCluster, CLIQUE)

distinction : exclusive / non-exclusive : belong to multiple clusters or not
fuzzy / non-fuzzy : belong to every cluster with weight $0 \leq w_i \leq 1$, $\sum_i w_i = 1$
probabilistic clustering : partial data or all data (complete)
heterogenous / homogeneous : cluster of widely different size / shape / density

type : separated : closer to all point in own cluster
centered : closer to center of own cluster
centroid : average of all point
medoid : most "representative" point in cluster
contiguous : closer to ≥ 1 point in own cluster
dense : closer to higher density than lower

Python - P62

partitioning, partitioning database D of n objects into set of k clusters

minimize $\sum_{i=1}^{k} \sum_{p \in C_i} [d(p, c_i)]^2$

global optimal : exhaustively enumerate all partition

heuristic method : K-means / K-medoids

K-means : each cluster represented by center of cluster

K-medoid / PAM (partition around medoids) : represented by one object in cluster

K-means

select K points as initial centroids

REPEAT

form K clusters by assigning all points to closest centroid

recompute centroid of each cluster

UNTIL centroids don't change in the iteration

intial centroid often chosen randomly

cluster produced vary from one iteration to another

centroid typically mean of point in the cluster

'closeness' measured by Euclidean distance, cosine similarity, correlation

converge for common similarity measure

cluster

most of convergence happen in first few iteration, until relatively few point change

evaluation : usual measure is SSE (sum of squared error)

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} [dist(m_i, x)]^2$$

easy way to reduce SSE is to increase K

good clustering with smaller K have lower SSE than poor clustering with higher K

centroid : multiple runs : probability not useful

sample and use hierarchical clustering to determine initial centroid

select > k initial centroids and then select among >k initial centroids

select most widely separated

emptiness : choose point from cluster with highest SSE

typically split cluster and reduce overall SSE for clustering

repeat several times for several empty clusters

Python - P63

K-means

weakness : applicable only when mean defined
need to specify number of cluster K in advance
unable to handle noisy data and outlier
not suitable to discover cluster with non-convex shape

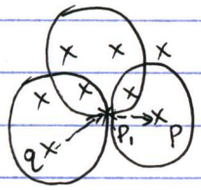
sensitivity: extreme value may substantially distort distribution of data
K-medoids take most centrally located object in cluster

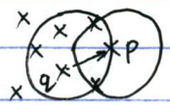
hierarchical clustering use distance matrix as clustering criteria
not require number of cluster K as input, but need termination condition

AGNES agglomerative nesting use single-link method and dissimilarity matrix
single-link method : smallest distance between element in one cluster and one in other
merge node with least dissimilarity
go on in non-descending fashion
eventually all nodes belong to same cluster

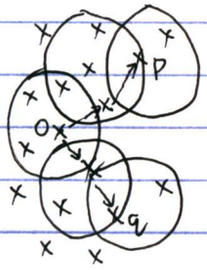
DIANA divisive analysis, inverse order of AGNES
eventually each node form cluster on own

density clustering, clustering based on density (local cluster criterion)
such as density-connected point
major feature : discover cluster of arbitrary shape / handle noise / one scan
need density parameter as termination condition
Eps : maximum radius of neighbourhood
MinPts : minimum number of point in Eps-neighbourhood of point
$N_{Eps}(p)$: {q belong to D | dist(p,q) ≤ Eps}





directly density-reachable : point p directly density-reachable from point q
p belong to $N_{Eps}(q)$ and core point condition : $|N_{Eps}(q)| \geq MinPts$



density-reachable : point p density-reachable from point q
chain of point $p_1 = q, p_2, \dots, p_n = p$, p_{i+1} directly density-reachable from p_i
density-connected : point p density-connected to point q
point o : both p and q density-reachable from o

Python - P64

DBSCAN density based spatial clustering of application with noise
- rely on density-based notion of cluster : maximal set of density-connected point
- discover cluster of arbitrary shape in spatial database with noise

OPTICS ordering point to identify clustering structure
- produce special order of database w.r.t. density-based clustering structure
- cluster-ordering contain info equivelant to density-based clustering
  - corresponding to broad range of parameter setting
- good for both automatic and interactive cluster analysis
  - including finding intrinsic clustering structure
- can be represented graphically or using visualization technique

assessing tendency, assess if non-random structure exist in data
- by measuring probability data generated by uniform data distribution
- Hopkins Static : test spatial randomness by statistic test

extrinsic quality, $Q(C, C_g)$ for clustering C given ground truth C_g,
- cluster homogeneity : the purer, the better
- cluster completeness : should assign object belong to same category
- rag bag : putting heterogeneous object into pure cluster more penalized than into rag bag
- small cluster preservation : splitting small category into pieces is more harmful
  - scatter coefficient = avg intra cluster distance / avg inter cluster distance

intrinsic quality, cluster cohesion : measure how closely object related in cluster
- measured by within cluster sum of square

cluster separation : measure how distinct / separated one cluster from other
- measured by between cluster sum of square

silhouette coeffient : combine both cohesion and separation, but for individual point

validity aspect : determining clustering tendency of set of data
- comparing result of cluster analysis to externally known result
- evaluating how well result of cluster analysis fit data without reference to external information
- comparing result of different set of cluster analysis to determine the best