MEMPUTE V1

PATTERN NETWORK DATABASE

www.memonode.com

Bayesian Pattern Probability Inference Engine

- Pattern Database Green Technology –

Pattern Database는 빈도수에 기반하여 패턴을 발견하고 베이지안 확률 추론을 수행하는 베이지안 패턴 확률 추론 머신이다. 학습과정은 주어진 데이터에 대하여 에포크 한번에 완결된다. 자주 반복되는 패턴들에서 가지가 계속 뻗어 나가며 상위 레벨로 올라 가면서 패턴은 점점 더 추상화 되고 압축된다.

Pattern Database는 메모노드라는 분산 퓨전 관계형 데이터베이스위에서 구성된다. 모든 발견 패턴은 베이터베이스에 저장되고 추론에 사용된다. 메모노드 데이터베이스에 관한 내용은 www.memonode.com을 참고하고 언급된 모든 예제들은 깃허브https://github.com/mempute 에 있으며 Pattern Database에 관한 예제는 https://github.com/mempute/anormal 위치에 mempute_pattern_chain_database_example.ipynb 이 있다.

Pattern Database는 데이터에서 빈도수에 의해 규칙을 발견하고 이를 패턴화한다. 패턴들은 서로 경쟁하고 가장 높은 강도의 것이 invoke되어 추론에 사용된다.

Pattern Database는 목표값을 타겟팅 하기위해 목표값과의 오차로부터 역전파로 파라미터 오차를 조정하는 연역방식이 아니다. 데이터로 부터 라벨없이 빈도수에 의해 데이터를 가장 잘 설명하는 패턴을 발견하게 되며 이 과정은 귀납적이다. 목표값과의 연결 역시 가장 연결 강도가 높은 연결을 스스로 발견하고 사전 학습하므로써 추론시에 사후 예측한다. 이 모든 과정은 귀납식이다.

Pattern Database는 시간대 혹은 이벤트 등의 범주 단위 정도의 연관성이면 된다. 우리는 많은 경우에 입력에 잘 정합되는 목표값을 알지 못하며 이럴때 Pattern Database는 주어진 입력에 가장 가능성 높은 목표를 사후확률로 예측해 낸다. 연역방식인 신경망과 달리 귀납방식으로 추론함으로써 과적합의 위험없이 정확한 추론이 가능하다.

또한 멀티 소스대 타겟을 연결 학습하여 소스중 타겟에 정상성이 가장 높은 입력 요소를 추론하여 발견해 낸다.

이러한 기능성으로 Pattern Database는 예측, 분류, 타겟 라벨의 자동생성, 데이터 정상성 판별, 오류 데이터 필터링 등 빅데이터 처리의 모든 광범위한 영역에서 신경망뿐 아니라 이제까지 발명되어진 어떠한 데이터 처리 기술보다 다변적 기능을 강력하고도 범용적으로 수행할 수 있다.

또한 신경망과의 하이브리드 학습을 수행하여 신경망을 더 광범위한 영역에서 부스팅하여 기존에 신경망으로 해결하지 못했던 난제들을 해결할 수 있다.

타겟 라벨 생성은 높은 레벨의 패턴 추상화로 이루어 진다. 신경망 이나 기타 기술은 이러한 기능이 없으며 신경망은 잠재코드의 차원을 작게 줄이면 정보는 붕괴되어 의미없어 진다. 언어 모델에서 발전된 트랜스포머는 압축을 수행하지 않으며 컨볼루션 망은 압축을 위해 최대값만을 선택함으로써 많은 정보가 손실된다. 깃허브에 이상데이터 판별 예제에서 **Pattern Database**로 생성한 타겟 라벨으로 신경망과 하이브리드 학습을 진행한 결과를 보면 높은 수준의 제한된 라벨은 학습되어 판별 효과가 있었으나 더 넓은 범위의 낮은 레벨의 라벨은 목표값으로 학습되지 못하였다. 신경망은 매우 제한된 높은 정합성을 갖은 타겟 범주의 학습만 가능할 뿐이다. 언어의 경우 **one hot**으로 분류되어야 할 어휘 사이즈가 **10000**개도 쉽게 넘을 수 있고 신경망의 언어 모델은 이를 달성하나 언어라는 것은 문법도 있고 의미적으로도 우리는 정형된 패턴으로 사용하기에 가능한 것이다. 개나 고양이 분류에서 보듯 이러한 분류들도 또한 정상성이 매우 높은 데이터이다.

다만 이문서의 두번째 파트인 **mempute** 신경망 버전의 강력한 시계열망인 제너릭 망으로는 생성된 라벨로 하이브리드 학습을 진행할수록 오차가 줄어들어 학습이 됨을 확인할 수 있다. 그러나 규칙이 희박한 자연데이터로부터 넓은 범위의 이산 라벨을 수렴시키는 것은 신경망에게는 매우 버거운 일로써 학습시간이 오래 걸린다. 사전학습은 인코더를 생성하는 과정인데 **Pattern Database**의 추상화된 라벨은 단일 혹은 몇 개 차원 정도로 입력에 대한 분류 태그를 제공하여 이를 타겟으로 인코더 학습시킬 경우 더 높은 사전학습 결과를 얻을 수 있을 것이다.

Pattern Database는 비지도 학습, 타겟 연결 학습, 사전학습, 전이학습, 증강학습 및 보상에 따른 패턴 강도를 조정으로 강화학습등 다양한 학습을 수행 할 수 있다.

Pattern Database는 학습의 결과로 발견된 패턴을 데이버데이스에 저장하고 계속적인 추가학습으로 인해 패턴이 데이터베이스에 누적됨에 따라 점점 더 정확하고 오류에 강한 추론이 가능하다. 이는 통계표본이 클수록 모집단과 유사해져 예측 결과가 정확해 지는 것과 같은 이유이다

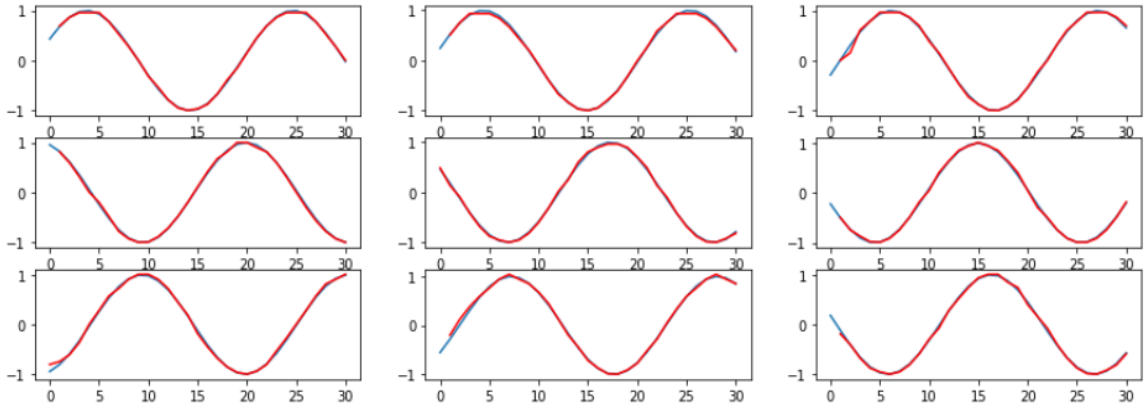
Pattern Database는 고성능의 대용량 분산 관계형 데이터베이스위에서 구동되어 대량의 패턴 학습 및 추론이 가능하며 분산 분할 학습과 패턴 병합을 하여 대량의 패턴 처리를 빠른 시간에 수행할 수 있다.

Pattern Database는 신경망의 강력한 전처리기로서도 사용될 수 있다. 입력데이터에서 타겟을 가장 잘 설명하는 요소를 정확하게 분리하내는 과정은 사람의 판단이나 개입이 필요없어 데이터 학습의 모든 과정을 자동화시킬 수 있다. 이와 같은 기능성으로 **Pattern Database**는 다양한 분야에서 범용적으로 사용될 수 있을뿐만 아니라 이상 탐지나 개인화 추천 시스템등 일반적으로 기존

기계학습 방법으로 효과가 적거나 결과가 애매한 분야에 탁월한 성능을 기대 할 수 있고 추론 결과에 대한 확률 해석 설명을 제공할 수 있다.

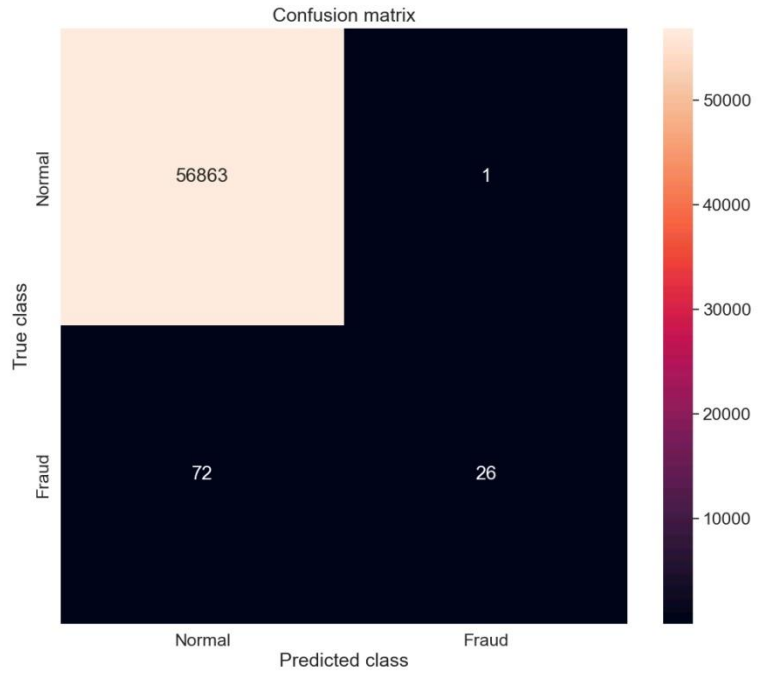
간단한 Pattern Database적용 예

사인 곡선 예측

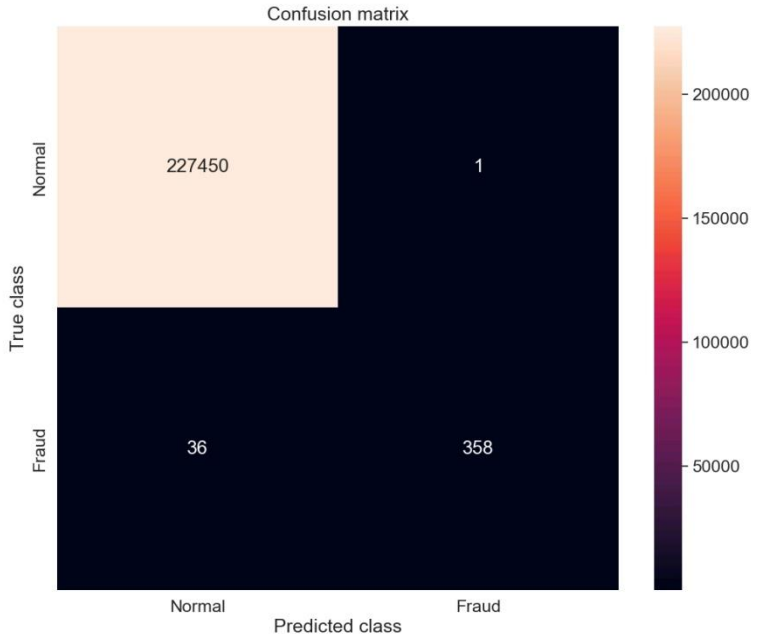


아마존 주가 예측

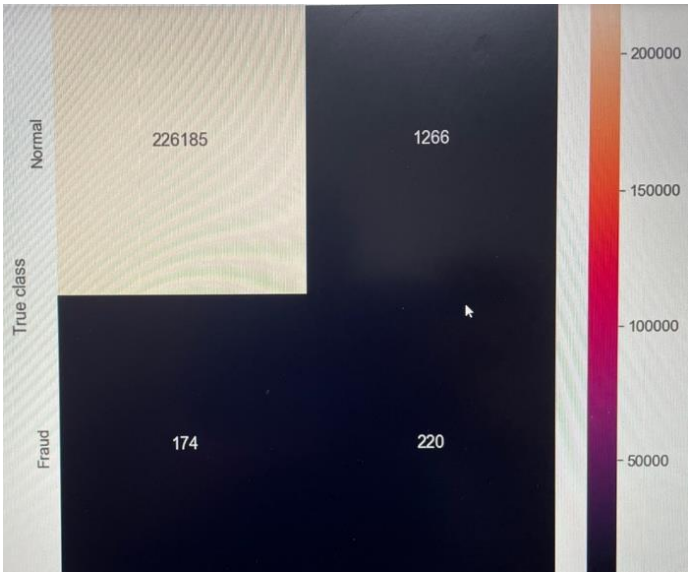
이상 탐지 데이터 예측



이상 탐지 학습 데이터 예측



신경망 학습 데이터 예측



Anormal detection(이상 탐지)는 비정상 데이터가 적어 신경망으로는 지도 학습을 하지 못하고 오토인코더로 학습하는데 테스트 데이터 예측도 정확도가 낮을뿐 아니라 학습데이터를 예측한 결과도 마찬가지이다. 신경망의 목적이 일반화에 있어 당연한 결과인데 이상 탐지의 경우 정상과 비정상의 데이터 비율이 비대칭이어서 학습 데이터가 축적되어도 최총 정확도가 향상될 수 없다. 학습 데이터를 예측한다는 것은 학습데이터에서 가능도 높은 패턴을 발견해 내는 일반화 과정뿐만 아니라 이러한 고 가능도 패턴을 데이터베이스에 직접적으로 계속 누적함으로써 경험치를 높혀 정확도가 지속적으로 향상되는 것을 의미한다.

위 이상탐지의 예를 살펴보면 차원 축소 압축이 2만분의 1로 된 것으로 압축은 일반화의 한 과정이고 학습데이터 예측이 이 정도 높은 일반화로도 높은 정확도로 나온 결과이고 테스트 데이터 예측도 같은 압축으로 나온 결과로서 예측되지 못한 비정상 케이스는 학습 데이터에는 없는 패턴이다. 또한 패턴 데이터베이스와 신경망을 하이브리드 형태로 증강 학습법을 수행한다면 state of the art를 달성 할 수 있을 것이다.

다음은 이상 탐지 예제의 패턴 추론과정의 확률 계산을 출력한 것으로서 이상인데 정상으로 판별하여 추론이 틀린 경우를 설명한다. prior_v는 목표값이고 0은 정상, 1은 이상 임을 나타내고 이상으로 예측해야하나 posterior(사후확률)을 보면 정상 출력이 -10.810163, 이상 출력이 -18.120785 으로서 정상 목표값의 확률이 더 높아 정상으로 판별한 것을 나타낸다. 여기서 pid는 목표값의 아이디, prior st는 사전확률, likely는 event(사건)하에서 발생되는 prior확률인 가능도이다.

event-prior probable result: pid(4249817) prior_v(0.000000) posterior(-10.810163) likely(-10.809829) prior(-0.000334) prior st(2993) prior ast(2994)

event-prior probable result: pid(4249833) prior_v(1.000000) posterior(-18.120785) likely(-10.116419) prior(-8.004366) prior st(1) prior ast(2994)

- 매뉴얼 -

패턴 네트워크 데이터베이스의 명령은 파이썬 MEMPUTE 크라이언트 API 와 서버측의 SQL 명령어의 결합으로 구성된다.

파이썬 클라이언트 API

driver

- Pattern network database 접속 드라이버를 로드한다.

connect

- Pattern network database 접속 세션을 개설한다.

statement

- Pattern network database sql 문장을 실행할 문장 객제를 생성한다..

direct

- 패턴 네트워크 데이터베이스 sql 문장을 실행한다.

mempute

- 패턴 네트워크 데이터베이스 sql 문장을 실행하고 결과로 파이썬 배열을 리턴받는다..

array

- 패턴 네트워크 데이터베이스와 데이터를 통신하기위한 배열을 생성한다.

inarray

- 서버측 배열에 클라이언트측 배열 데이터를 로드한다.

focus

- 배열을 포커싱하여 후행 작업의 대상이 되게 한다.

Ex)

```
drv = mp.driver(3)
con = mp.connect(drv, "loopback", "", "")
stmt = mp.statement(con)
mp.focus(stmt, "execute mempute('array', 'eval_input')")
mp.inarray(stmt, x_test, 1)
```

스키마 구성

Perception

- 학습 단위

퍼셉션 생성

- 퍼셉션 이름과 퍼셉션이 위치할 로케일 이름을 명시한다. 로케일 이름이 주어지지 않으면 기본 위치에 퍼셉션이 생성된다.

syntax

execute mempute('perception', 'perception_name locale locale_name positional')

execute : sql 실행 명령

mempute : 패턴 네트워크 실행 명령

perception : 퍼셉션 생성 명령

perception_name : 퍼셉션 이름

locale : 퍼셉션 생성 위치, 생략 하면 기본 위치에 생성

locale_name : 퍼셉션 생성 위치 이름

positional : [unfix|0|1|2]

- 시퀀스에 포지션 지정 여부 지시
- Unfix: 포시션 없음
- 0 : 포지션 없음
- 1 : 목표값만 포지션 설정
- 2 : 입력값과 목표값 모두 포지션 설정

ex)

mp.direct(stmt, "execute mempute('perception', 'anormal locale percep_loc')")

sequence

- 입력과 타겟 데이터의 길이 설정

syntax

execute mempute('sequene', input_size, target_size, 'sequence_reduce_idx', scale_figure)

sequence : 시퀀스 설정 명령

input_size : 입력값 길이 설정

taget_size : 목표값 길이 설정

sequence_reduce_idx : 시퀀스 아이템별 차원축소할 인덱스 나열

scale_figure : 양수이면 스케일 업 계수, 소수점 이하 지정 계수까지 취득, 음수이면 스케일 다운 계수, 소수점 이상 지정 계수까지 취득, 0 이면 스케일 조정 없

음, 생략이면 기본값으로 스케일 업 2

ex)

```
mp.direct(stmt, "execute mempute('sequence', 5, 3, 2)")
```

channel

- **데이터 구조 기술**

syntax

execute mempute('channel', division, 'attribute')

channel : 데이터 구조 설정 명령

division : 0 – 입력 구조 기술, 1 – 목표 구조 기술, 2 – 채널 기술 시작, 3 – 채널 기술 마감, 한번에 구성할 경우 2 와 3 생략 가능

attribute : { description_enum }

description_enum : description_enum | description

description : prefix figure type

prefix : scale | reduction | union | narrow

scale : l | L

l : scale up

L : scale down

reduction : r | R #차원 축소

union : union_notation scale_down_or_not #차원축소와 스케일 조정 함께 적용

union_notation : u | U

scale_down_or_not : L | #스케일 다운 혹은 지정하지 않으면 스케일 업

narrow : narrow_notation scale_down_or_not #채널별로 차원축소와 스케일 조정 함께 적용

narrow_notation: n | N

scale_down_or_not: L | # 스케일 다운 혹은 지정하지 않으면 스케일 업

figure : scale_digit . reduction_digit

scale_digit : #스케일 조정 계수

reduction_digit: #차원 축소 계수, 축소할 차원수의 역수

type : b | s | f | i | l | d #b(byte), s(short), f(float), i(32bit int), l(64bit int), d(double)

ex)

```
mp.direct(stmt, "execute mempute('channel', 1, '{u0.02d}')")
```

array

- **Pattern network database 와 데이터 입출력 인터페이스**

syntax

execute mempute('array', 'array_name locale locale_name division mode id_gen seq_save batch_size num_devide')

array : 배열 생성 및 참조 명령

array_name : 배열 이름

locale : 퍼셉션 생성 위치, 배열을 생성할 때 생략 하면 메모리 배열 생성, 참조 할때 로케일 이하 생략하면 기존에 배열 이름으로 등록된 배열 리턴

locale_name : 퍼셉션 생성 위치 이름

division : 1 – 입력배열, 0 – 출력배열, 2 – 입력 & 마인드 구성

mode : 1 – 바이너리, 0 – 문자형

id_gen : -1 – 시퀀스 아이디 엔진에서 생성하지 않고 입력에서 주어짐, 0 – 기본값으로 아이디 생성, 1 – 아이디 생성

seq_save : 0 – 시퀀스 저장 안함, 1 – 시퀀스 저장

size_batch : 일괄 실행 단위 설정, -1 - 32 양수 최대값, 0 – 5000,

num_devide : 1 - 컨볼루션 연산 윈도우 커널 스트라이드할때 하나의 원 이미지에서 윈도우 스트라이드 갯수만큼 설정하여 파생 시퀀스들을 하나로 묶는다, 0 – 안함

ex)

```
mp.array(stmt, "execute mempute('array', 'anormal_input locale array_loc 1 1 1 0 0 0')")
```

load

- **저장된 배열에서 메모리 로드**

syntax

execute mempute('load', 'array_name', start, end)

load : load command

array_name : array name

start : load 시작 시퀀스 아이디

end : load 끝 시퀀스 아이디, -1 이면 끝까지 로드

ex)

mp.direct(stmt, "execute mempute('load', 'anormal_target', 0, -1)")

cognit

- **학습 명령**

syntax

execute mempute('cognit', 'input_array', 'target_array')

ex)

mp.direct(stmt, "execute mempute('cognit', 'anormal_input', 'anormal_target')")

predict

- **추론 명령**

syntax

execute mempute('predict', 'input_array', 'output_array')

return value : 예측값 파이썬 배열

ex)

predictions = mp.mempute(stmt, "execute mempute('predict', 'eval_input', 'eval_output')")

oblivion

- **학습 완료 망각 명령**

syntax

execute mempute('oblivion', 0)

ex)

mp.direct(stmt, "execute mempute('oblivion', 0)")

phyper

- **Hyper parameter**
- **Sensitivity #민감도 설정**

ex)

mp.direct(stmt, "execute mempute('phyper', 'sensitivity 2.7')")

discriminate

- **판별 데이터를 생성한다.**

Syntax 1

execute mempute('discriminate', 'operate', 'parameter')

discriminate : 판별 명령

operate : 0 | 1 | 2 | 3 | 4

0 : 입력으로부터 추출되는 최종레벨 패턴 아이디 시퀀스를 출력하기위한 출력 시퀀스 사이즈 설정

Parameter : output_sequence_size

Ex) mp.mempute(stmt, execute mempute('discriminate', 0, 5)")

1 : revise inference 하기위해 패턴아이디 시퀀스를 입력받기위한 입력시퀀스 사이즈 설정

Parameter : input_sequence_size

Ex) mp.mempute(stmt, execute mempute('discriminate', 1, 5)")

2 : syntax 2 의 operate 1 의 강도순 연속 아이디 공간 사이즈 리턴

Ex) vocabulary_size = mp.mempute(stmt, "execute mempute('discriminate', 2, -1)")

3 : syntax 2 의 operate 1 의 수행과정에서 연속 아이디를 빌드하는 범위의 하한 레벨 설정, 이를 수행안하면 탑베렐만 대상으로 아이디 빌드

Ex) mp.mempute(stmt, "execute mempute('discriminate', 3, 8)")

4 : syntax 2 의 operate 103 의 입력의 특징 feature 를 추출하는데 있어서 parmeter 값이 1 이면 특징과 일치하는 입력 데이트를 그대로 출력 옵션, 0 이면 채널설정의 차원축소 옵션에 따라 차원 축소된 데이터를 출력

Ex) mp.mempute(stmt, "execute mempute('discriminate', 4, 1)")

syntax 2

execute mempute('discriminate', 'input_array', 'operate', 'parameter', 'output_array',

'target_array')

discriminate : 판별 명령

input_array : 입력 데이터 배열 이름

output_array : 출력 데이터 배열 이름

target_array : 목표 데이터 배열 이름

operate : 0 | 1 | 100 | 101 | 102 | 103

0 : 입력 데이터에 대한 사전 학습이 선행된 후에 시퀀스 강도 오더링, input_array 의 각 시퀀스별로 phyper test_margin 으로 설정된 레벨이상의 패턴들만 총합을 구하여 정렬

Parameter : 0 | 1 | 2 | 3

0 : 강도만 오더링 & descending

1 : 강도만 오더링 & ascending,

2 : 강도+탑레렙 넘버 오더링 & descending

3 : 강도+탑레렙 넘버 오더링 & ascending

Output_array format

[Sequence_id order_index strength_sum top_level_number]

Ex) mp.mempute(stmt, "execute mempute('discriminate', 'anormal_input', 0, 3, 'disc_output')")

1 : 패턴을 강도순으로 연속 아이디로 정의, input_array 의 각 시퀀스별로 탑레벨 패턴 혹은 phyper all_lev_build 가 설정되면 모든 레벨 패턴을 강도 기준으로 연속 아이디로 정의하고 각 패턴을 생성 아이디로 대체하여 output_array 에 출력한다.

Parameter: 0 | 1 | 2 | 3

0 : 로드 & 결과리턴

1 : 빌드 & 결과리턴

2 : 빌드 & 저장

3 : 빌드 & 저장 & 결과리턴

Ex)

mp.mempute(stmt, execute mempute('discriminate', 0, 5)")

mp.mempute(stmt, "execute mempute('discriminate', 'anormal_input', 1, 3, 'disc_output')")

100 : 외부 데이터에 대한 차원 축소 빌드 서비스

Parameter : 차원축소 사이즈

Ex)

mp.mempute(stmt, execute mempute('discriminate', 100, 20)")

101 : 외부 데이터에 대한 차원 축소 서비스

Parameter : 없음

Return value : eval_input 배열 데이터에 대한 차원 축소된 아이디를 desc_out 배열에 적재하여 리턴

Ex)

r = mp.mempute(stmt, "execute mempute('discriminate', 'eval_input', 101, 0, 'disc_output')")

102 : 외부 데이터에 대한 차원 축소 사이즈 리턴

Parameter : 없음

Return value : 외부 데이터에 대한 차원 축소 사이즈 리턴

Ex)

Vocabulary_size = mp.mempute(stmt, "execute mempute('discriminate', 102, -1)")

103 : 입력 데이터로부터 추출된 패턴들중에서 목표값 페턴을 가장 특징짓는 입력 feature 추출(타겟배열이 주어질때) 또는 사후 추론 결과를 가장 특징짓는 입력 feature 추출(타겟배열이 주어지지않을 때)

Parameter : margin #입력추출 패턴 탑 레벨에서 margin 레벨 개수만큼 하위 범위의 입력 패턴들을 대상으로 operation 수행

Return value : 입력 데이터에서 특징 추출이된 입력 feature 이외 나머지 원소들은 nan 값으로 재워진 입력배열과 동일 사이즈의 클라이언트 배열

Ex)

pickout = mp.mempute(stmt, "execute mempute('discriminate', 'anormal_input', 103, 1, 'anormal_output', 'anormal_target')")

associate - 입력 데이터의 마인드를 추출해 연상되는 타겟 장면을 리턴하는 연상 명령.

associate scene_id, scene_id2 - 두번째 장면이 첫번째 장면에 부합되는 총 마인

드 강도 합계 리턴

예) execute mempute('associate', 200, 201) - 선언된 타겟 게이트의 장면에서 200 번째와 201 번째의 부합 강도 리턴

associate scene_id
- 명시된 장면이 선언된 타겟 게이트의 장면에서 마인드 부합되는 총 강도 합계가 큰 순으로 장면 아이디를 result set 으로 반환.
- 이때 타겟 게이트는 mind 지지 명령으로 마인드 생성되있어야 한다.

예) execute mempute('associate', 200)

associate accord_string - 선언된 타겟 게이트에 대한 집합 연상 명령, 이때 타겟 게이트는 mind 지지 명령으로 마인드 생성되있어야 한다.

accord_string - accord/against inner/outer scene_id..

accord - scene_id 와 부합되는 장면 마인드
against - scene_id 에 반하는 장면 마인드
inner - 언급되는 장면들의 공통 마인드
outer - 언급되는 장면들의 union 마인드

예) execute mempute('associate', 'accord inner 200 300 against inner 201 301')
- 200, 300 장면의 공통 마인드에는 부합하고 201, 301 의 공통 마인드에는 반하는 장면들울 추출.

merge
- execute mempute('merge', suffix_1_or_0, 'merge_suffix', seperate_1_or_0, oblivion_1_or_0);
- 두개 인지망 학습 결과를 하나로 합치는 명령
- oblivion_1_or_0 - merge 후에 망각 루틴을 수행 할것인지 스텝 설정, 0 이면 망각 루틴 수행 안함

- seperate_1_or_0 - 1 이면 합쳐질 두개 인지망을 독립 인지망, 즉 워드 및 장면 일력이 별개로 입력.0 이면 워드 및 장면을 공유하고 학습만 서로 다른 범위를 수행한 결과를 merge
- suffix_1_or_0 가 1 이면 perception network 명령으로 합칠 인지망(호스트)을 지정한후 합쳐질 인지망게스트)을 merge_suffix 에 suffix 이름으로 설정, 호스트 인지망 이름에 suffix 가 추가된 인지망 이름의 게스트 인지망을 호스트 인지망에 merge

예)
execute mempute('percenption network', 'perception_name') - 합칠 호스트 인지망 설정.
execute mempute('merge', 1, 'guest_suffix_name', 0, 1/0) - 합쳐질 게스트 인지망 지시, merge 수행.

- 두개 독립적 인지망을 합치는 경우
예)
```
execute mempute('perception network', 'lang_pnet locale atom_db');

execute mempute('array', 'lang_array locale scene_db');

execute mempute('write', 1);

mempute load eng1_5000.txt kor1_5000.txt;

execute mempute('range', 0, 0);

execute mempute('percept');

execute mempute('perception network', 'lang_pnet2 locale atom_db');

execute mempute('array', 'lang_array2 locale scene_db');
```

execute mempute('write', 1);

mempute load eng5001_10000.txt kor5001_10000.txt;

execute mempute('range', 0, 0);

execute mempute('percept');

execute mempute('perception network', 'lang_pnet2');

execute mempute('array', 'lang_array2');

execute mempute('keep', 'lang_pnet2', 1);

execute mempute('perception network', 'lang_pnet');

execute mempute('array', 'lang_array');

execute mempute('merge', 0, 'lang_pnet2', 1, 1);

import - 원격의 인지망 학습 결과를 perception network 명령으로 포커스된 인지망으로 가져오는 명령

mempute import url [imp_percept_network_locale], [imp_a_percept_network_name], [imp_suffix_name]

url - 임포트할 원격지 설정 ip_address[:port_name] - port_name 을 생략하면 기본 포트로 접속

예)
execute mempute('import', 'loopback:45003')
- perception network 명령으로 설정된 호스트 인지망과 동일한 이름을 가진 원

격 인지망으로부터 학습 결과를 가져옴.

execute mempute('import', 'loopback:45003', 'imp_suffix_name')동
- perception network 명령으로 설정된 호스트 인지망과 동일한 이름과 imp_suffix_name 을 가진 원격 인지망으로부터 학습 결과를 가져옴.

execute mempute('import', 'loopback:45003', 'imp_percept_network_locale', 'imp_a_percept_network_name');
- 명시된 퍼셉션 이름과 로케일을 가진 원격 인지망으로 부터 학습 결과를 가져옴.

execute mempute('import', 'loopback:45003', 'imp_percept_network_locale', 'imp_a_percept_network_name', 'merge_suffix_name;');
- 명시된 퍼셉션 이름과 로케일, suffix 를 가진 원격 인지망으로 부터 학습 결과를 가져옴.

rebert - array 명령으로 설정된 포커스 게이트를 리셋 시킨다. 다른 array 명령을 수행하지 않고 리셋 시킬때 사용.

예) execute mempute('rebert')

reset - array 설정된 학습 포커스 리셋

reset step

step
learn - 학습 결과, 마인드 리셋
array - 배열, 마인드 리셋
all - 배열, 학습 결과, 마인드 리셋

예) execute mempute('reset', 'learn/array/all')

erase - array 설정된 학습 포커스 삭제

erase step

step
learn - 학습 결과, 마인드 삭제
array - 배열, 마인드 삭제
all - 배열, 학습 결과, 마인드 삭제

clear - 퍼셉션 단위 일괄 삭제 명령

execute mempute('clear', 'percept_name suffix_name percept_loc array_name array_loc step(learn/array/all) erase_api(erase/reset) rm_loc(erase/0)');

clear 명령 ex).

--execute mempute('clear', 'percept_name suffix_name percept_loc array_name array_loc step(learn/array/all) erase_api(erase/reset) rm_loc(erase/0)');

execute mempute('clear', 'atom_lotto 0 atom_db scene_lotto scene_db learn reset 0');

execute mempute('clear', 'atom_lotto 0 atom_db scene_lotto scene_db array reset 0');

execute mempute('clear', 'atom_lotto 0 atom_db scene_lotto scene_db all reset 0');

execute mempute('clear', 'atom_lotto 0 atom_db scene_lotto scene_db learn erase 0');

execute mempute('clear', 'atom_lotto 0 atom_db scene_lotto scene_db array erase 0');

-- 이 명령은 의미없다. scene_lotto 게이트만 선택되어 게이트 삭제는 위에서 됐으므로, all 은 퍼셉션을 삭제 하는 것인데 이럴려면 선택할때 scene_lotto 게이트 명을 주면 안된다.
execute mempute('clear', 'atom_lotto 0 atom_db scene_lotto scene_db all erase 0');

execute mempute('clear', 'atom_lotto 0 atom_db 0 0 learn erase 0');

execute mempute('clear', 'atom_lotto 0 atom_db 0 0 array erase 0');

execute mempute('clear', 'atom_lotto 0 atom_db 0 0 all erase 0');

-- 로케일 삭제는 루트 권한이 필요하므로 사용자가 루트 디비를 직접 로드한다.
load db root;

execute mempute('clear', 'atom_lotto 0 atom_db 0 0 array erase erase');
-- 위에서 all erase 로 퍼셉망의 모든 게이트를 삭제했으면 퍼셉망 이름 테이블에 해당 퍼셉망 정보가 없으므로 이럴경우네는 로케일 이름을 명시해야 한다.
execute mempute('clear', 'atom_lotto 0 atom_db 0 0 all erase erase');

학습 모니터링
Sqn> execute mempute('view', 'perception_name');