**딥러닝 자연어\_RNN**

[파이썬 로드맵]

스크린샷이(가) 표시된 사진

자동 생성된 설명

\* 머신러닝 : 어느정도 데이터의 정제를 잘 해주면, 숫자 기반으로

딥러닝 : 랜덤으로 weight, bais 등을 주고, 알아서 학습시킴

ex)영상처리, 신호처리(음원 장르 분류), 자연어처리(다음 문장 예측 등) -> 머신러닝으로 힘들다(메모리 부족)

스크린샷이(가) 표시된 사진

자동 생성된 설명

- numpy : 2차원 배열 (중요!)

- pandas : dataframe (많이 사용)

- tensorflow (+ keras) : tensor 객체 = 크기와 방향(벡터) + 깊이(차원)

=> numpy의 "축"의 개념을 반드시 이해해야 사용할 수 있다.

딥러닝 : 데이터를 학습시켜 모델을 만드는 머신러닝 알고리즘의 집합 : "~ 네트워크"

비지도학습(군집분석, 연관분석)은 딥러닝 X

학습을 통해 변수들에서 특징을 찾아가는 것 -> 기계학습

MLP(Multi Layerd Perceptron) classifier => 정제한 데이터를 넣어주면 머신러닝, 데이터 원본을 넣어주면 딥러닝

# w와 b를 찾기 위해 처음 값은 임의의 값이 들어가도록 함

# weight와 bias를 -10 ~ 10 / -100 ~ 100 으로 준 이유 : 이미 답을 알고 있음

w = ft.Variable(tf.random\_normal([1], -10.0, 10.0))

b = ft.Variable(tf.random\_normal([1], -100.0, 100.0))

# 영상이면 2차원, 컬러 영상이면 3차원...차원만 달라진다.

# w와 b를 찾기 위한 식 y' = W\*x + b

hypothesis = W \* x\_data + b

# 처음에 계산 후, w와 b를 변경(역전파 알고리즘 많이 사용)

# 역전파 알고리즘 : 예측된 내용을 이전 layer에 전달하여 사용.

CNN (convolution layer) : random하게 filter 적용

- 랜덤한 값을 곱하여 pooling(size를 줄임) -> 새로운 네트워크를 정의.

\* 알파고는 강화학습 (딥러닝이 아닌데...시장 왜곡됨)

[] Text Mining

(데이터 마이닝 > 텍스트 마이닝 > 자연어처리)

- 한글이 전 세계에서 13번째로 많이 사용. 언어처리가 어렵다(조사가 너무 많다. 단어들의 순서가 바뀌어도 이해 가능.)

- 데이터 분석에서 가장 중요한 것 ?

HOW : 어떻게 찾는지

WHAT : 뭘 찾는지

|  |  |  |  |
| --- | --- | --- | --- |
|  | | WHAT | |
| Y | N |
| HOW | Y | Optimization (최적화) | Insight (통찰) |
| N | Solution | Discovery |

\* 언제/어떻게 쓰이는지 알고 있으면 더 잘 배울 수 있다. (목표설정)

- 순서 : 문서(비정형 데이터) -> corpus(저장된 문서. sample or 수집)

-> TermDocumnetMatrix(구조화된 문서. 형태소 분석. 어떤 단어가 많이 나왔는지 빈도수 도출

=>연관분석) -> 분석(분류,군집 분석-spam mail/연관, 감성 분석 - 선호도 평가, 단어 별 긍정적/부정적)

[] NLTK(Natural Language Toolkit) 패키지 사용

[] 자연어처리(Natural Language Processing, NLP)

- 처리 순서

1) 형태소 분석 (Morphological analysis) : 형태소로 분해하고 품사를 결정.

2) 구문 분석 : 문법 규칙 등으로 문장을 해석

3) 의미 분석 (Semantic Analysis) : 문자에서 형태소의 의미를 해석

문서(Document) -> 문단(Paragraph) -> 문장(Sentence) -> 어절(Word phrase) -> 형태소(Morpheme)

-> 음절(Syllable) -> 음소(Phoneme)

\* 불용어(Stopword) : 불필요한 단어 (한글 불용어는 일일이 사전으로 만들어야 한다-말뭉치)

- (형태소)체언 중에서도 명사를 많이 뽑아낸다.

- 형태소 분석

원형 복원 (영어는 이부분이 중요 / 한글은 '조사'가 중요)

-- 사용 패키지

R : KoNLP (https://konlpy.readthedocs.io/en/latest/)

Python : KoNLPy

Java : KOMORAN, HanNanum ...

\* KoNLPy

# Kkma : 꼬꼬마(한글 형태소 분석기)

from konlpy.tag import Kkma

# JPype1 패키지 필요! -> 한글이 안되서 설치 안될 때가 있다. (정 안되면 Visual Studio 설치)

-- 품사 태깅이 필요하다. (http://coderby.com/pds/14) : 내가 원하는 형태소 찾을 때..

ex) 가방에 들어가신다 => 가방/NNG + 에/JKM + 들어가/VV + 시/EPH + ㄴ다/EFN

\* Komoran

pos(pharase, flatten=True) : phrase(문장) 넣어주면 분석. flatten은 형태소 분석기마다 조금식 다를 수 있다.

-- 말뭉치 : '말'들의 집합 (\*.txt)

kolaw = 한국 법률 말뭉치(constitution.txt)

kobill = 대한민국 국회 의안 말뭉치. id:의안 번호 (1809890.txt - 1809899.txt)

[] 워드 클라우드

>pip install wordcloud

>pip install konlpy

\* JPytpe1이 설치가 안될 때

1) jpype whl file 설치 (KoNLPy.org -> User guide -> Installation -> Windows -> JPype1(>=0.5.7)

-> JPype1-0.7.0-cp37-cp37m-win\_amd64.whl //cp+버전(37) 다운로드

-> jupyter Notebook 에서 pip install로 설치!!

2) visual studio 설치 (c++ MFC / c++CLI 체크 후 설치) 둘 중 하나 선택하여 jpype 설치 후 konlpy 설치

>pip install matplotlib

>print(komoran.nouns("%r"%data[0:1000]))

# "%r"% : rawtext로 변환

- 불용어는 반드시 set 객체로 지정!

\* jupyter Notebook은 한글을 지원하지 않는다. (python 기본 font가 영어)

-> wordcloud 객체 생성 시 폰트 지정 필요!

\* matplotlib - figure : 도화지(matplotlib가 그림을 그려줄 객체. size : inch)

[] 연관분석

- 데이터들 사이에서 자주 발생하는 속성을 찾는, 그리고 그 속성들 사이에 연관성이 어느 정도 있는지 분석

ex) 미국의 모 마트 - 금요일에 맥주와 기저귀를 같이 사는 사람들이 많더라.

- 알고리즘 : apriori(연역적), FP-growth, DHP 알고리즘 등.

apriori : item/거래의 수가 많아지면 메모리가 많이 필요하다는 단점이 있으나, 매우 쉬움

>pip install apyori

- 지지도(Support)와 신뢰도(confidence)

지지도 : A -> B / 전체 비율

신뢰도 : B / A

-- 지지도와 신뢰도를 모두 생각해야 한다.

- 향상도(Lift) : A -> B의 신뢰도 / B의 지지도

- transaction data : 서로 관련있는 데이터의 모음 (python에서 list 안에 list가 들어있는, 2차원 list 형태로!)

RelationRecord (namedtuple : 이름이 있는 tuple을 상속받음) type

-> name = value 형태 (items,support, ordered\_statistics[])

-- items : 연관 규칙 A (Left Hand Side) -> B (Right Hand Side), frozenset : 읽기전용

-- support : 지지도

-- ordered\_statistics = [] // list

item\_base : lhs

item\_add : rhs

lift : 향상도

\* 웹 크롤링 : requests, selenium / 파싱 : beautifulSoup

[] python 함수 읽기! (시각화 함수 등 많은 함수에서 사용)

# \* : tuple 매개변수 \*\* : dict 매개변수

# a,b : 값이 반드시 들어가야 함. 필수 매개변수

# d : 값이 들어가지 않아도 됨. 선택 매개변수

def some(a,b,\*c,d=0,\*\*e):

print(a,b,c,d,e)

# 순서 인수, 튜플인수, 선택 인수, 키워드 인수

some(1,2,3,4,5,d=6,hello=10,world=20)

pandas의 2차원 데이터를 가져올 때,

for문 중첩시키는 것이 아니라, apply / applyMap 함수등을 사용

ex) DataFrame.apply(func, axis=0, broadcast=None, raw=False, reduce=None, result\_type=None, args=(), \*\*kwds)

[] 축

a = np.arange(12).reshape(3,4) # 행 렬

np.sum(a,axis=0) # 열을 고정시키고 행들을 합쳤다.

b=np.arange(24).reshape(2,3,4) # 행(x) 렬(y) 면(depth)

np.sum(b,axis=0)

[] DataFrame

pandas -> 집계함수, apply 함수 꼭 해야 함

(반복문으로 할 수는 있으나, 해당 함수들이 메모리도 적게 사용하고 속도도 더 빠르다)

[] 인공신경망

- 오버피팅(과적합)의 문제

지역 최저값 (local minima)의 문제

- 알파고(강화학습) <-> 머신러닝 딥러닝 : 그 둘은 상관 없다.

- 인간의 뉴런(neuron) : 자극(전기적 신호)의 강약에 의해 전달.

학습을 하면서 weight들을 계손 바꿔간다.

x0\*w0 + x1\*w1 + x2\*w2 [+1\*b] -> f //뉴런의 갯수만큼의 w값들이 필요

입력값(x) \* 가중치(weight) + (bias) -> f (활성화 함수)를 통해 다음 뉴런으로 전달

-- bias는 연구자 재량

\* 선형 결합 = 다 더한다 = 시그마

- 구성

Input layer(neuron/perceptron)의 갯수 = 입력하는 변수의 수(컬럼의 수) or +1

Hidden layer : 은닉층 이전 input layer의 출력을 모두 받아서 사용.

(은닉층의 갯수가 3개 이상 : Deep Neuron Network) - 연산 수가 많다.

-- 너무 많으면 과적합(overfitting)이 발생 할 수 있다 : 학습한 데이터셋에만 잘 맞음.

-- hidden layer의 shape : input layer보다 작고, output layer보다 크게 하는 경우가 보통.

Output layer의 갯수 = 분류하고자 하는 class의 수 : 값이 무엇인지에 따라 선택하는데, 가장 큰 값을 선택하면 그것이 분류한 결과의 값.

- 활성화 함수

-- ReLU : 영상처리, 신호처리에서 사용 학습이 반복되어도 원본 값을 잃지 않음

(0 이하면 0 / 0보다 크면 원본 값)

-- Softmax : 그래프의 값을 모두 합치면 1이 되어야 한다. 분류 문제에서 사용

(출력한 값을 가지고, 가장 큰 값을 1로 바꾸고 나머지를 0으로 바꾼다면 = one hot encoding)

-- Sigmoid : Hidden layer에서 많이 사용

- 종류

-- 전방 전달 신경망(Feedforward Neural Network) : 순전파 알고리즘

-> 간단하지만 학습 횟수가 많아야 한다.(오차 발생에 대한 가중치 변경을 위해)

-- 방사 신경망(Radial Basis Function network) : 출력되어 다음 layer의 입력값으로 들어가는 값들이 모두

조금씩 변경되도록.

-- 코헨 자기조직 신경망(Kohonen self-organizing Network) : 출력값을 자기 자신의 입력값으로 다시 사용

-- 순환 인공 신경망(Recurrent Neural Network) : 이전 layer의 입력이 다음 layer의 입력으로 어느정도

전달이 된다. 계산량이 많아진다.

\* 역전파 알고리즘 - 오차를 가지고 이전 layer의 weight를 재조정 : 회수가 줄어들고, 오차수행이 빠르다.

미분값 계산 : 기울기 // weight와 bias는 연속된 값이 입력되서 나온 연속된 값. 오차를 줄이기 위해 미분

(값들의 차이가 줄어드는 효과) -> 재조정된 값이 정답에 가까운 값이 아닐 수도 있다!

[] 딥러닝

- 문제를 해결하기 위한 머신러닝 알고리즘의 집합

- TensorFlow

-- Data Flow Graph 방식 : node와 node 사이의 edge(weight)

// tensor의 흐름. tensor 라는 data를 그래프의 흐름을 통해 학습을 시킨다.

-- tensor : 크기, 방향, 차원, 오류 -> numpy의 n차원 행렬

-- 수학 계산과 데이터의 흐름을 node와 edge를 사용한 방향 그래프(Directed Graph)

Operation(node) : 수식이다. tensor를 입력받아 tensor를 출력

Tensor(data) : n차원 배열 / n차원 리스트.

Session : operation에 tensor를 적용시켜 실행하기 위한 객체(실행 환경을 캡슐화)

Variables : 값을 저장할 변수(프로그램을 실행하는 동안 변하는 값들을 저장할 객체 - y는 값이 아니라 operation이기 때문에..)

\* y = w\*x + b : 수식은 operation. 값은 tensor(w,b :학습을 찾아야 할 값 - parameter. variable / x : 입력 값 - placeholder). y는 operation. 값이 아니다!!!

variable : 처음부터 주어진 입력값의 크기만큼 생성

placeholder : 행 하나의 값만.

- 딥러닝 프레임워크 : 대부분 C++로 만들어져 있고, UI가 Python.

- 알렉스넷 : 이미지 분류를 하기 위한 알고리즘 - 여러 가지의 알고리즘을 조합.

>pip install tensorflow

- 선형방정식 (선형회귀식) : 행렬의 연산으로만 값을 구할 수 있어야 한다.

\* 0의 수가 압도적으로 많은 행렬 : 희소행렬

- kbig.kr (빅데이터 센터) -> 인프라 신청 // hadoop server, bigdata server 빌려줌(원격)

[] DNN

- Deep Neural Network : hidden layer가 많은 Network (보통 3개 이상)

- 평균 제곱 오차 : 제곱해서 평균 구하기 / 유클리드 제곱 거리 : A와 B 사이의 거리(삼각함수)

\* reduce : 축 별 집계

- train data로 학습시키고, train data로 확인하고, test data로도 확인한다. -> overfitting 확인하려고

- 1 세대 : "모든" 데이터가 input -> hidden -> output 된 결과

// 하나의 행이 들어가서 나온 것을 의미하는 것이 아님!

[] RNN

- Recurrent Neural Network (순환 신경망)

-- h : 각 레이어의 출력 -> 다음 x의 weight와 곱해질 입력 (이전의 입력값이 계속 유지가 된다.)

-- 은닉층의 메모리 셀(= node) : 이전 학습된 데이터

- overfitting의 문제를 보완 : drop out - 특정 갯수의 perceptron 출력을 0으로. / layer를 건너띄어서.

- Bidirectional Recurrent Neural Network (양방향 순환 신경망)

-- 다음 단어를 예측 (뒤에 있는 데이터를 가지고 문맥 예측)

- RNN의 단점(학습이 진행될수록 오래된 기억이 희미해짐)을 보완 : LSTM (장 단기 기억법)

LSTM (Long Short Term Memory) : node 내부에서 많은 일이 이루어진다. 시간이 많이 걸리는 단점!

- GRU (Gated Recurrent Unit) : 기존 LSTM의 구조를 단순화 (= 시간이 더 빨라졌다.)

\* simple RNN = Vanilla RNN

[] Keras

- tensorflow 설치되어있어야 한다. (상위 레벨 인터페이스)

> pip install keras

- API

-- sequential : 직관적이고 편리함. 역전파 사용 불가. CNN에 주로 사용

-- functional : RNN에 주로 사용

https://keras.io/getting-started/functional-api-guide/

-- 선형 회귀 (Linear Regression)

loss='mse' : 평균 제곱 오차 / verbose=0 : log level (log level : 0~10)

-- 로지스틱 회귀 (Logistic Regression)

lr : learning rate

-- 다중 입력 모델 (model that accepts multiple inputs) :

입력이 많은 경우, 같은 입력을 A / B 등으로 나눠서 학습시킨 후, 합친다.

각각의 모델을 만든 후 concatenate

[] Word Embedding

- 단어를 벡터로 표현하는 대표적인 방법 (희소 표현 -> 밀집 표현으로 변환)

-- 희소 표현 : TDM or One Hot Encoding 처럼 1보다 0이 많은..(단어 간 유사성 표현 불가)

-- 밀집 표현 : 0과 1만 가지는 것이 아니라 실수값으로 => 단어와 단어 사이의 거리를 예상 할 수 있다.

(비슷한 위치 = 비슷한 의미)

- Word Embedding : 단어를 밀집 표현하는 방법 중 하나. (Word2Vec 사용)

-- CBOW (Continuous Bag of Words) : 주변 단어들을 가지고 중간 단어들을 예측

-- Skip-Gram : 중간 단어를 통해 주변 단어들을 예측

\* class Word2Vec(sentences=None, size=100, window=5, min\_count=5, workers=3)

-- size : 특징 벡터의 차원

-- window : 문장 내 현재 단어와 예측 단어 사이의 최대 거리

-- min\_count : 총 빈도가 이보다 낮은 모든 단어를 무시함

-- workers : 모델을 훈련시킬 쓰레드의 수(-1 : 내 컴퓨터의 모든 core로)

>pip install gensim

\* Term Document Matrix (= 레이블 인코딩) : 있으면 1 없으면 0

One Hot Encoding : 맞으면 1 나머지 0

\* 레이블 인코딩 안에 원 핫 인코딩이 들어있다.

CNN -> openCV (python : CV2) 먼저 공부해야...

- convolution : 합성곱

- convolutional layer : filter를 적용시켜서 새로운 그림을 만든다.

숫자값이 크다 = 특징이 세다. (경계선이 있는 영역에 대한 특징)

pooling하면 size가 작아진다.

- convolution layer와 pooling layer를 몇번 거쳐서 DNN의 입력으로 넣는다.

- 이 때, 이미지를 일부러 잘라니고 찌그러뜨려서 학습시킨다.

// 0 과 1 사이 vs 1 과 5 사이 : 1과 5 사이가 경계선이다.

0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1 1 0 0

0 1 5 5 5 5 5 5 1 0 0

0 1 1 1 1 1 1 5 1 0 0

0 1 1 1 1 1 1 5 1 0 0

0 1 5 5 5 5 5 5 1 0 0

0 1 5 1 1 1 1 1 1 0 0

0 1 5 1 1 1 1 1 1 0 0

0 1 5 5 5 5 5 5 1 0 0

0 1 1 1 1 1 1 1 1 0 0

0 0 0 0 0 0 0 0 0 0 0

\* 딥러닝 응용 : 라즈베리파이에 카메라모듈 붙이고, 모형 자동차에 연결 -> 물체인식 학습된 모델 올려서 자율주행 장난감. (이미 학습이 완료된 모델로..)