

환영합니다.



# Machine Learning 과 Scikit-learn 이해

한국 오라클 권철민

# 학습을 위한 중요한 세가지

- 질문
- 공유
- 프로젝트

# Confession

저는 머신러닝 전문가가 아닙니다. 저 역시 배워 나가는 중입니다.

서로가 알고 있는 것을 공유하면서 머신러닝/딥러닝에 대한 방향성을 잡는 시간이 되었으면 합니다.

# Machine Learning ? Deep Learning ?

- Deep Learning 은 Machine Learning의 한 분야 입니다.
- Deep Learning 기반은 심층 Neural Network (hidden layer가 2개 이상) 입니다.
- Deep Learning 은 현재 Machine Learning 분야에서 가장 잘나가는 영역입니다.
- 설명 중 Machine Learning 과 Deep Learning 을 통칭해야 할 때는 Machine Learning으로 , Machine Learning 과 Deep Learning 을 구분해야 할 때는 전통적인 Machine Learning , Deep Learning으로 언급하겠습니다.

# 시작하겠습니다.



먼저 python 기반의 Machine Learning 코딩 부터 살펴 보겠습니다.

처음에 개요만 , 나중에 다시 상세하게 리뷰하겠습니다.

# Titanic 생존자 Prediction




# Titanic 예제 구성

1. Pandas DataFrame으로 CSV 파일 로딩
2. 데이터의 이해를 위한 Visualization
3. 적합한 입력데이터를 위해 Data Cleanup 과 Transform
4. 알고리즘이 이해할 수 있도록 문자열등을 숫자형 Category로 Encoding
5. Training and Test Sets 로 Split
6. 머신러닝 알고리즘 파라미터 튜닝
7. 여러겹의 Training and Test Set 의 Cross Validate Evaluation



데이터 전처리(Preprocessing)



머신러닝 모델 학습과 생성 ,  
Evaluation

# Basic 머신러닝 프로그램 - 1

0. 데이터 전처리가 완료 되었다고 가정한다면

1. 학습을 위한 train set 와 평가를 위한 Test Set로 데이터를 나눔

```
from sklearn.model_selection import train_test_split  
...
```

```
num_test = 0.20
```

```
X_train , X_test , y_train , y_test = train_test_split(X_all , y_all , test_size = num_test , random_state=23)
```

전체 데이터의 80% 는 Train Set , 20%는 Test Set으로 나눔



# Basic 머신러닝 프로그램 - 2

## 2. 머신러닝 학습 및 예측 , 그리고 Evaluation

```
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy_score
```

```
clf = DecisionTreeClassifier()
```

머신러닝 알고리즘으로 DecisionTree 선택

```
clf.fit(X_train , y_train)
```

DecisionTree 알고리즘에 Train용 입력데이터 , Train용 결과 데이터를 사용하여 머신러닝 학습

```
predictions = clf.predict(X_test)
```

학습된 모델의 테스트 입력데이터를 이용하여 예측결과 도출

```
print(accuracy_score(y_test , predictions))
```

예측 결과와 실제 결과 데이터를 비교하여 정확도 산출

결과

0.837988826816

# 머신러닝?

## Machine Learning



what society thinks I do



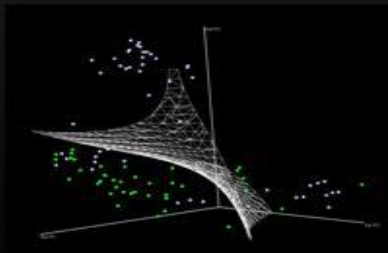
what my friends think I do



what my parents think I do

$$\begin{aligned} L_p &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_i \alpha_i \\ \alpha_i &\geq 0, \forall i \\ \mathbf{w} &= \sum_i \alpha_i y_i \mathbf{x}_i, \sum_i \alpha_i y_i = 0 \\ \nabla \hat{g}(\theta_t) &= \frac{1}{n} \sum_{i=1}^n \nabla \ell(x_i, y_i; \theta_t) + \nabla r(\theta_t) \\ \theta_{t+1} &= \theta_t - \eta_t \nabla \ell(x_{i(t)}, y_{i(t)}; \theta_t) - \eta_t \cdot \nabla r(\theta_t) \\ \mathbb{E}_{i(t)} [\ell(x_{i(t)}, y_{i(t)}; \theta_t)] &= \frac{1}{n} \sum_i \ell(x_i, y_i; \theta_t) \end{aligned}$$

what other programmers think I do



what I think I do

```
>>> from scipy import svm
```

what I really do

생각만큼 근사하지 않을 수 있습니다.

그림은 <http://sanghyukchun.github.io/57/> 발췌

# 머신러닝 ?

머신은 무엇을 학습하는가?



# 머신러닝, 왜 필요한가?

## 일반적인 개발 Application

```
def estimate_house_sales_price(num_of_bedrooms, sqft, neighborhood):
    price = 0

    # In my area, the average house costs $200 per sqft
    price_per_sqft = 200

    if neighborhood == "hipsterton":
        # but some areas cost a bit more
        price_per_sqft = 400

    elif neighborhood == "skid row":
        # and some areas cost less
        price_per_sqft = 100

    # start with a base price estimate based on how big the place is
    price = price_per_sqft * sqft

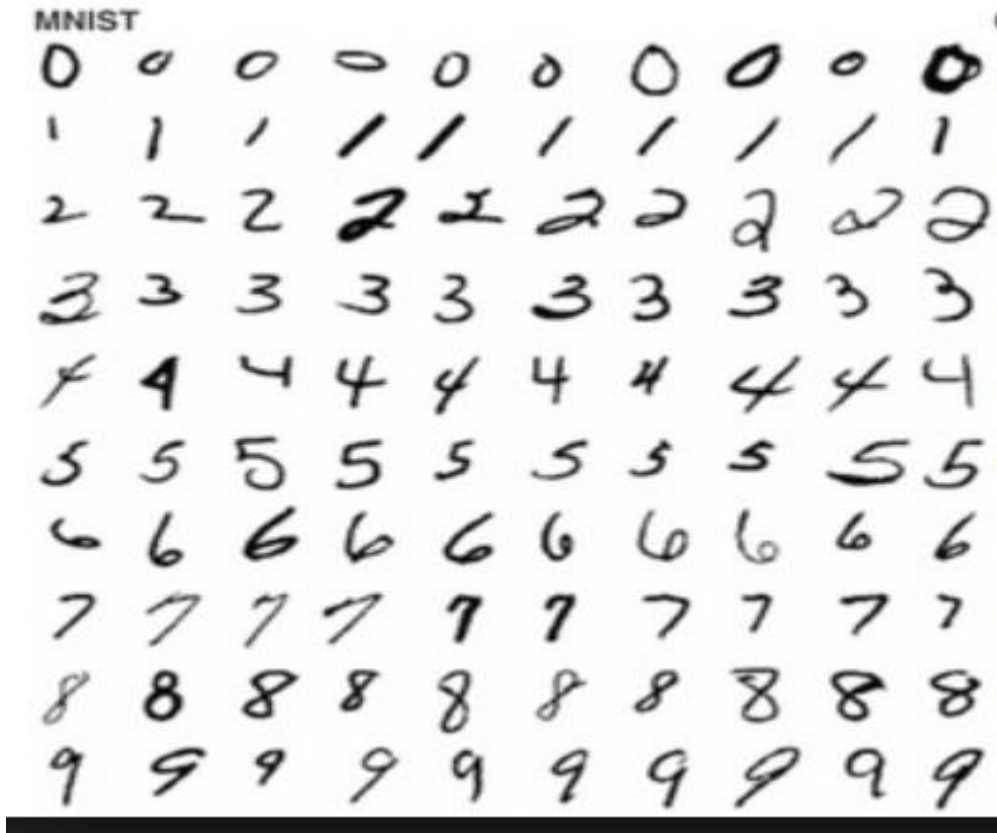
    # now adjust our estimate based on the number of bedrooms
    if num_of_bedrooms == 0:
        # Studio apartments are cheap
        price = price - 20000
    else:
        # places with more bedrooms are usually
        # more valuable
        price = price + (num_of_bedrooms * 1000)

    return price
```

예제는 <https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471>에서 발췌

- 주택 부동산 가격을 구성하는 많은 요소들. 평수, 방의 개수, 주변지역, 학군, 주변 아파트 가격등 다양한 가격이 결합되어 가격 결정. (대부분의 머신러닝 사례는 예제 코드보다 훨씬 복잡, 수십~수백개의 변수들이 서로 결합되어 만드는 복잡도)
- 많은 조건에 따라 if else 로 분기하면서 복잡도를 증가시키고 이러한 조건들은 다양한 원인으로 반드시 변하게 되어 있으므로 이에 대한 대응도 어렵고 예측 정확도가 떨어질수 밖에 없음.

# 머신러닝, 왜 필요한가?



- 동일한 숫자라 하더라도 여러 변형으로 인해 숫자 인식에 필요한 여러 특징 (feature) 들을 if else 와 같은 조건으로 구분하여 숫자를 인식하기 어렵다.

# 머신러닝 , 왜 필요한가?

머신러닝 기반 Application

```
def estimate_house_sales_price(num_of_bedrooms, sqft, neighborhood):  
    price = <computer, plz do some math for me> 머신러닝 알고리즘  
  
    return price
```

- 만약 컴퓨터가 스스로 주택 가격 데이터를 참조하여 주택가격을 구성하는 다양한 요소들이 가격 영향 계수들을 계산하여 정확한 주택가격을 계산한다면 ?
- 또한 가격 요소들이 변하거나 이에 따른 가격 영향 계수들이 변하더라도 프로그램을 수정하지 않고 대상 데이터만을 분석하여 정확한 주택가격을 반영할 수 있다면?

# 머신러닝, 왜 필요한가?

머신러닝이 찾아낸 최적 가격 영향 계수

```
def estimate_house_sales_price(num_of_bedrooms, sqft, neighborhood):  
    price = 0  
  
    # a little pinch of this  
    price += num_of_bedrooms * .841231951398213  
  
    # and a big pinch of that  
    price += sqft * 1231.1231231  
  
    # maybe a handful of this  
    price += neighborhood * 2.3242341421  
  
    # and finally, just a little extra salt for good measure  
    price += 201.23432095  
  
    return price
```

머신러닝이 찾아낸 주택가격  
구성 요소 별 영향 계수

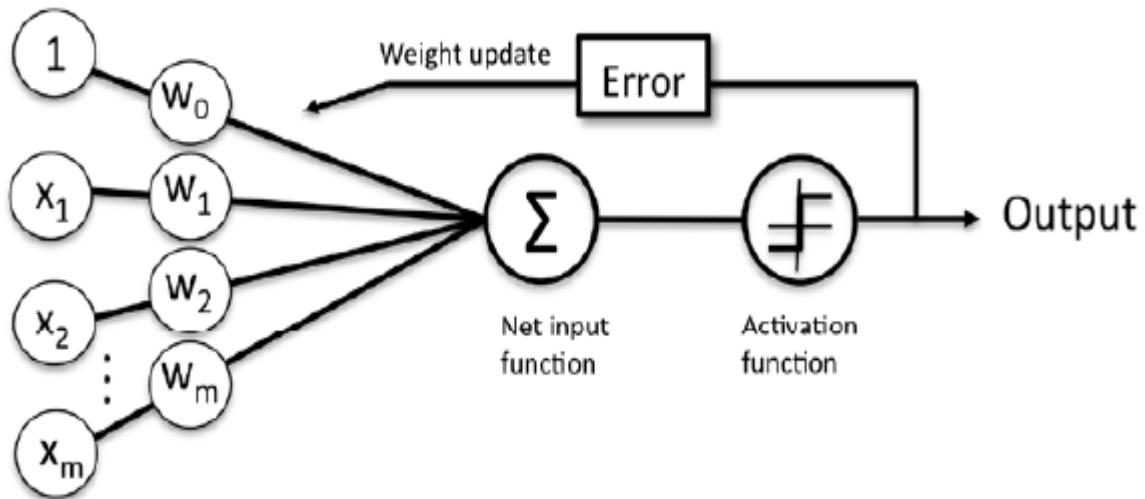
- 컴퓨터가 주택가격 데이터를 기반으로 주택가격 구성요소 별로 최적의 영향계수를 찾아냄
- 예제에서 주택가격이란 방의 개수 \* 0.84.. + sqft(평방크기) \* 1231 + 주변지역 \* 2.32 + 추가적으로 201을 더한 결과
- 머신러닝은 각 입력변수(주택가격 구성요소)에 최적화된 개별 영향계수  $W$  (weight, 가중치) 를 찾는것이다.
- 서울의 주택가격과 지방의 주택가격의 구성요소별 최적 영향 계수가 다르더라도 프로그램을 수정할 필요 없이 데이터만 바꾸면 됨.
- 서울의 주택가격 최적 영향 계수를 구하기 위해서 전국 주택가격 데이터를 사용한다면 정확도가 떨어질 것임. **이처럼 머신러닝은 데이터 기반의 알고리즘이므로 최적의 데이터를 사용하는 것이 알고리즘 자체보다 중요함.**



# 머신은 무엇을 학습하는가?

머신러닝에서 학습이란 해당 데이터를 기반으로 반복적인 연산등을 통해 머신러닝 알고리즘에 필요한 파라미터들이 최적화 되는 과정을 의미. 이렇게 학습된 머신러닝 모델을 기반으로 최적의 해를 구할 수 있다.

예를 들어 Neural Network 에서 학습이란 ?



$W$  를 학습

입력변수  $x_1, x_2, \dots, x_m$  별로 최적화된 weight 값  $w_0, w_1, w_2, \dots, w_m$  값을 학습을 통해 찾아냄.

여기서 학습이란 데이터를 기반으로 최적의 weight값을 찾을 때 까지 반복적인 연산등을 통해 점차적으로  $W$  를 보정해 나가는 것임



# 머신러닝이란?



프로그램을 수정하지 않고도 데이터를 기반으로 자동으로 분류/예측등의 결과를 도출할 수 있는 알고리즘 기법을 통칭

개발자가 데이터나 업무로직의 특성을 직접 감안한 프로그램을 만들 경우 난이도와 개발 복잡도가 너무 높아 질수 밖에 없는 다양한 IT 영역에 솔루션을 제공하게 됨.

데이터 마이닝, 영상인식, 음성 인식, 자연어 처리에서 머신러닝을 적용하면서 급속하게 발전을 이루게됨.

# 머신러닝 세계의 주요 논제

세상의 복잡한 로직을 어떻게 데이터 기반의 확률로 일반화 시킬수 있을까?

# 머신러닝 알고리즘 프로세스 개론

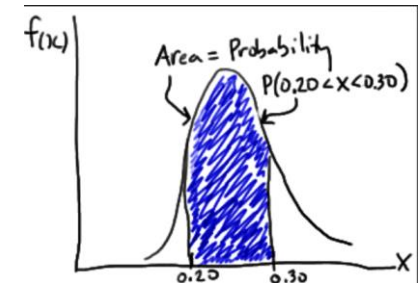
매우 많은 변수들로 복잡한 방정식

$$\begin{cases} y_1 = a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ y_2 = a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \end{cases}$$

선형대수로 단순화

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

확률통계로 결정값 도출



가장 높은 확률을 가지는 값이 결정값

# 머신 러닝 유형

머신 러닝 유형  
(By 마스터 알고리즘 책에서)

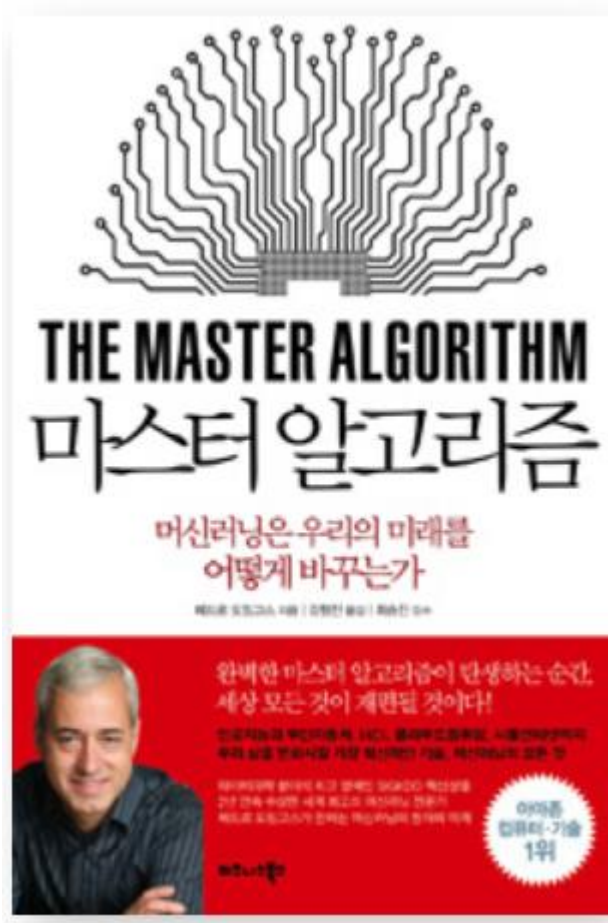
기호주의 : Decision Tree 등

연결주의 : Neural Network / Deep Learning

유전 알고리즘

베이지안 통계

유추주의 : KNN, Support Vector Machine



# Machine Learning의 일반적인 분류

- Supervised Learning
  - Classification(분류)
  - Regression(회귀)
  - Anomaly detection ( 이상 탐지)
  - Recommendations( 추천 시스템)
  - Vision/Audio detection/recognition
  - Text Analysis , NLP
- Unsupervised Learning
  - Clustering
- Reinforcement Learning

# 머신러닝 요소

다음중 머신러닝에서 **가장 중요한** 요소는 ?

?

1. Data



2. 알고리즘

# 머신러닝의 단점

- 데이터에 너무 의존적이다. (Garbage In , Garbage Out )
- Training 시에 최적의 결과를 도출하기 위해 수립된 머신러닝 모델은 실제 Test 데이터 적용 시 Overfitting 되기 쉽다.
- 복잡한 머신러닝 알고리즘으로 인해 도출된 결과에 대한 논리적인 이해가 어려울 수 있다. ( 머신러닝은 블랙 박스)
- 데이터만 집어 넣으면 자동으로 최적화된 결과를 도출할 것이라는 것은 환상이다.(특정 경우에는 개발자가 직접 만든 코드보다 정확도가 더 떨어질수 있다) 끊임없이 모델을 개선하기 위한 노력이 필요하기 때문에 데이터의 특성을 파악하고 최적의 알고리즘과 파라미터를 구성할 수 있는 고급 능력이 필요하다.

# 왜 데이터 수집에 열광하는가?



구글과 페이스북에서 보유하고 있는 데이터로 최적화 된 머신러닝 모델을 다른 회사가 이길 수 있을까? ( 더 좋은 알고리즘을 가지고 있더라도?)



다양하고 광대한 데이터를 기반으로 만들어진 머신러닝 모델은 더 좋은 품질을 약속한다.  
앞으로 많은 회사의 경쟁력은 어떠한 품질의 머신러닝 모델을 가지고 있느냐에 결정 될 수 있다.



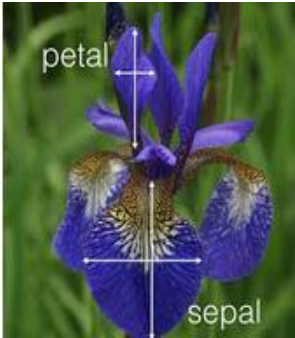


# Classification 의 기본 로직 – Iris 사례

← Feature → Class labels

Training Data	번호	꽃받침 길이	꽃받침 너비	꽃잎 길이	꽃잎 너비	Iris 꽃 종류는?
	1	5.1	3.5	1.4	0.2	Setosa
	2	4.9	3.0	1.4	0.2	Setosa
	....					....
	50	6.4	3.5	4.5	1.2	Versicolor
	....					....
	150	5.9	3.0	5.0	1.8	Virginica

Training 데이터로 모델 학습  
( Feature와 Class 다 포함)

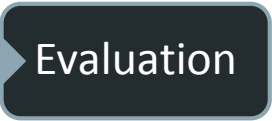


Test Data	번호	꽃받침 길이	꽃받침 너비	꽃잎 길이	꽃잎 너비	Iris 꽃 종류는?
	1	5.1	3.5	1.4	0.2	?
	....					?
	50	6.4	3.5	4.5	1.2	?

Test 데이터(Feature)로 학습  
모델 통해 유추된 class 값  
Predict



Predict 된 class값과  
실제 Test 데이터  
class 값 평가



# 다시 한번 Basic 머신러닝 프로그램 리뷰

## 2. 머신러닝 학습 및 예측, 그리고 Evaluation

```
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy_score
```

```
clf = DecisionTreeClassifier()
```

머신러닝 알고리즘으로 DecisionTree 선택

```
clf.fit(X_train, y_train)
```

DecisionTree 알고리즘에 Train용 입력데이터, Train용 결과 데이터를 사용하여 머신러닝 학습

```
predictions = clf.predict(X_test)
```

학습된 모델의 테스트 입력데이터를 이용하여 예측결과 도출

```
print(accuracy_score(y_test, predictions))
```

예측 결과와 실제 결과 데이터를 비교하여 정확도 산출

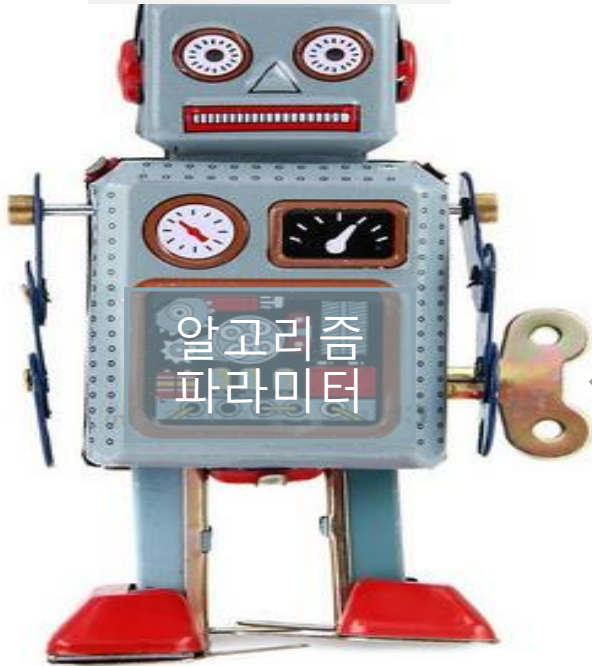
결과

0.837988826816

# 머신러닝 모델과 파라미터

Data

머신러닝 모델

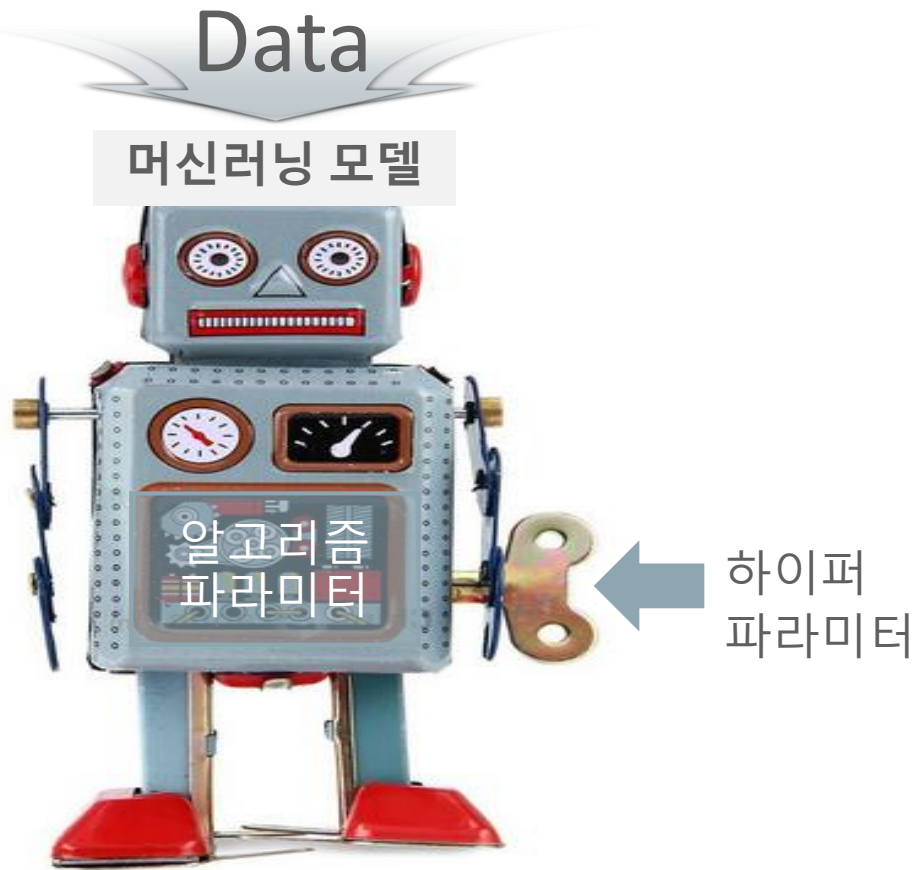


알고리즘  
파라미터

하이퍼  
파라미터

- 머신러닝은 주어진 데이터를 기반으로 사용자가 지정한 알고리즘으로 최고의 결과를 도출할 수 있도록 입력 변수(Input variables/Features)를 선별하고 알고리즘의 다양한 Parameter들을 최적화하여 모델링 하는 것
- 머신이 학습하는 것은 알고리즘의 다양한 Parameter를 주어진 데이터 기반에서 지속적으로 갱신하면서 최적을 찾는 것임.

# 머신러닝 모델과 파라미터



- Parameter 는 데이터와 머신러닝 알고리즘에 의해서 자동으로 최적화 된다. 따라서 최고의 머신러닝 결과를 도출하기 위해서는 사용자가 데이터를 최적으로 제공하는 것이 가장 중요하다.
  - 적합한 입력 변수 선택, Outlier 및 NA 처리, Normalization/Scaling, 적절한 데이터셋 제공 등
- 머신러닝 알고리즘의 파라미터에 영향을 줄 수 있는 파라미터를 하이퍼 파라미터라고 한다. 하이퍼 파라미터는 사용자가 튜닝을 위해 설정가능하다. 파라미터는 데이터와 알고리즘에 의해 자동으로 설정되지만 하이퍼 파라미터를 통해 영향을 줄 수 있다. 예를 들어 Neural Network에서 parameter 는 W(weight) 와 B(bias) 이며 하이퍼 파라미터는 L(learning rate) 임. ( 때로는 두가지가 서로 혼용되어서 사용되기도 함. 오라클 RDBMS 로 얘기하면 하이퍼 파라미터는 SQL Hint ?)

# 머신 러닝 파라미터 (or hyper parameter) 튜닝



<https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/>

- **max\_features**
- **N\_estimators**
- **Min\_sample\_leaf**

Parameter 또는 hyper parameter 튜닝보다 중요한 것은 머신러닝 알고리즘이 최적으로 수행 될 수 있게 데이터 셋을 가공/선별/보강 하는 것이다.

# ML Process Flow

모델 목표 설정



데이터  
가공/선별/보강



Training



Evaluation



2% 정확도 향상등 모델의  
목표 설정.



설정된 목표에 따라 최적의  
결과 도출을 위한 데이터  
가공/선별/보강



가공된 데이터 기반에서  
최적 알고리즘 및  
파라미터 튜닝등으로  
머신러닝 모델 학습 시킴



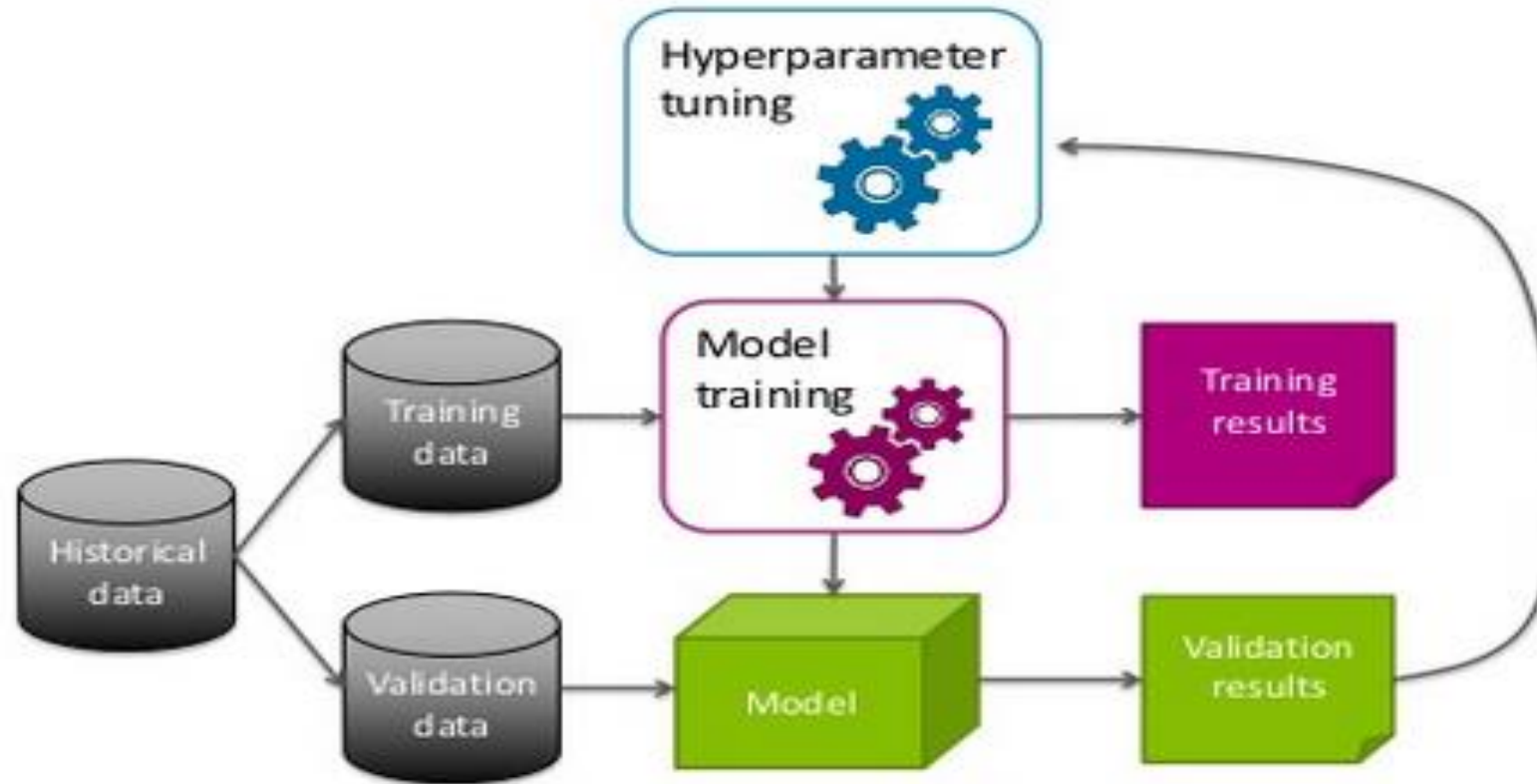
Test Set / Validation Set  
에서 Training 모델의  
성적을 평가. 평가가 나쁠  
경우 다시 Preprocessing  
및 Training 시킴





# ML Process Flow

<https://www.slideshare.net/AliceZheng3/evaluating-machine-learning-models-a-beginners-guide>



# ML 최적화

튜닝이라 쓰고 노가다라고 읽는다.

데이타를 재처리하고 파라미터를 학습시키는 작업  
데이터와 알고리즘을 이해하고 , 다양한 확률적  
방법들을 동원하여 최적화 시킴.





# Parameter 튜닝 – GridSearchCV 이용

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, accuracy_score
from sklearn.model_selection import GridSearchCV
```

```
clf = RandomForestClassifier()
```

```
parameters = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt', 'auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1, 5, 8]}
```

반복적으로 테스트를 수행할 파라미터들과 그 값들을 Python Dictionary 형태로 설정한다.

```
acc_scorer = make_scorer(accuracy_score)
```

최적 파라미터 기준을 설정할 Score 함수를 설정(여기서는 정확도)

```
grid_obj = GridSearchCV(clf, parameters, scoring=acc_scorer)
grid_obj = grid_obj.fit(X_train, y_train)
```

GridSearchCV 클래스에 파라미터와 평가 기준 Score 함수를 등록한 뒤 파라미터들을 반복적으로 대입하여 최적의 파라미터들을 도출한다.

```
clf = grid_obj.best_estimator_
```

최적 파라미터가 셋팅된 classifier 를 반환한뒤 해당 classifier 로 학습

```
clf.fit(X_train, y_train)
```

grid\_obj.cv\_results\_ 를 사용하면 모든 파라미터를 순차적으로 대입한 결과를 볼수 있다

아, 봐 나 overfitting  
되었다고



너, 되게 싸가지  
없구나

# Classification 은 현재가 아닌 미래를 예측하는 것

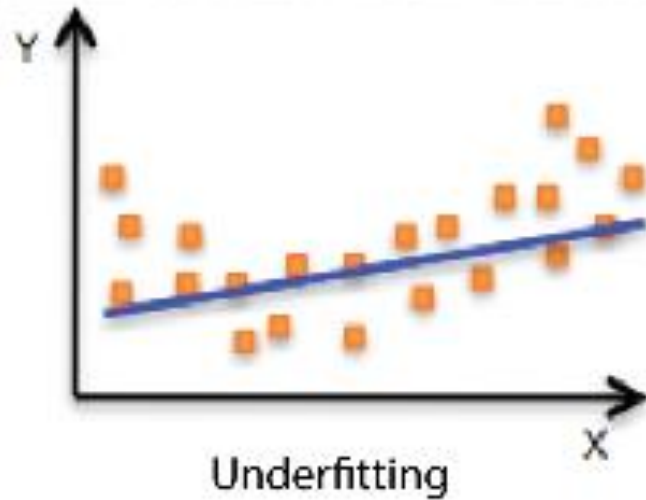
분류/예측은 앞으로의 데이터를 기준으로 얼마나 정확한 결과를 제공하는 것이 중요. 과거 데이터(학습 데이터)에서 좋은 결과를 내어도 실제 테스트 데이터에서 좋은 결과를 내지 못하면 아무런 소용이 없음.



- 모의고사에서 아무리 좋은 성적을 내어도 실제 수능에서 성적을 못내면 말짱 헛것임.
- 좋은 모의고사의 조건은 수능에서 나올만한 문제를 얼마나 다양하게 수험생에게 제공하는가에 달려있음.

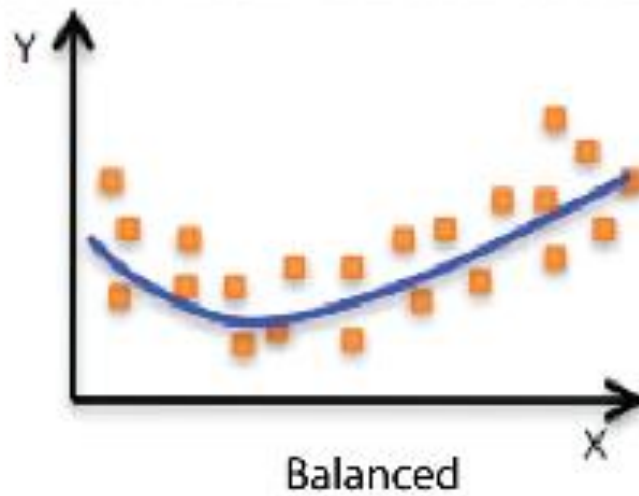
# Underfitting , Overfitting

지나치게 단순화함



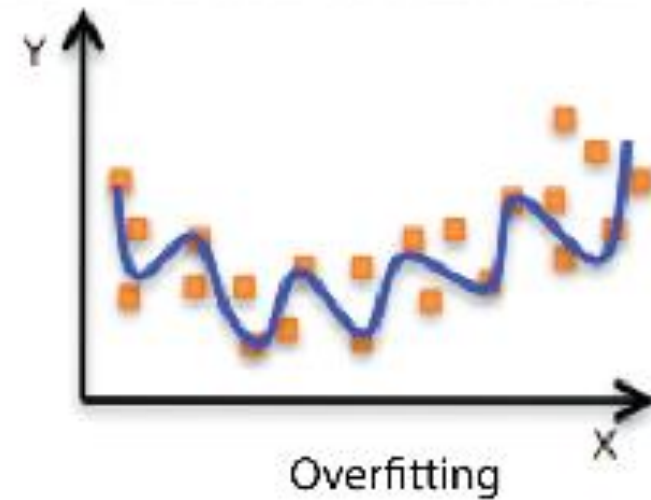
Overfitting 을 너무 두려워한 모델

이정도면 괜찮은데?



Training Set에서 정확도가 떨어지더라도 Test Set에서 상대적인 정확도가 높을 것으로 추정되는 모델

너무 상세화함



Training Set에서 정확도가 높지만 Test Set에서 상대적인 정확도가 매우 낮을 것으로 추정되는 모델

# 데이터의 특성에 따른 Overfitting



- 머신러닝 알고리즘은 주어진 Training 데이터에서 가장 최고의 정확성을 가진 결과를 제공하기 위해 만들어짐. 따라서 Training 데이터의 특성에 좌우 될 수 밖에 없음.
- Training 데이터 기반에서 정확도를 지나치게 강화한 모델은 Test 나 실제 데이터에서 오히려 정확도가 떨어질 확률이 높다.

# Overfitting 을 극복하기 위한 방안

- 많은 데이터셋을 준비한다.
  - 많은 량의 Training 데이터 준비. 여기서 많은 량이란 Feature 들의 다양성을 키워줄 수 있는 량을 뜻함. 반복적이거나 중복이 많은 데이터는 소용 없음.
- 최적의 Feature 를 선택하거나 가공한다.
  - Feature Selection
  - Feature Extraction(Extraction은 추후 설명 )
  - 데이터 가공 , NA 처리
- Training 과 Test Set 를 효율적으로 섞어서 구성
  - Random 구성
  - Stratified 구성
  - Cross-validation (교차 검증) Training 세트와 Test 세트를 여러벌 만들어서 반복적으로 교차 테스트. ( 진정한 성적은 모의고사 1번 잘 맞은 점수가 3년간 모의고사 성적의 평균 )
- 속성값 Scaling , Normalization
- Regularization : 머신러닝 모델이 세부사항에 너무 최적화되어 복잡한 결과를 도출하지 않도록 제어(벌칙 부과)
- Overfitting 에 강한 알고리즘을 선택한다.

# Titanic 데이터 가공

- Titanic 코드의 데이터 가공 부분을 참조하겠습니다.

# 문자열 값을 숫자형 카테고리 값으로 변환

대부분의 머신러닝 알고리즘은 숫자형 값을 선호. LabelEncoder 클래스를 이용하면 문자열들을 숫자형 카테고리 값으로 변환.

```
from sklearn import preprocessing
```

```
def encode_features(df_train, df_test):
```

```
    features = ['Fare', 'Cabin', 'Age', 'Sex', 'Lname', 'NamePrefix']  
    df_combined = pd.concat([df_train[features], df_test[features]])
```

변환하려는 컬럼(feature)를 선택. Training과 Test 셋 모두에 들어있는 데이터 값들을 기준으로 숫자형 카테고리 값으로 변환

```
    for feature in features:
```

```
        le = preprocessing.LabelEncoder()
```

```
        le = le.fit(df_combined[feature])
```

```
        df_train[feature] = le.transform(df_train[feature])
```

```
        df_test[feature] = le.transform(df_test[feature])
```

```
    return df_train, df_test
```

컬럼(feature) 별로 Loop 를 돌면서 해당 feature들을 숫자형 카테고리값으로 label encoding 수행

```
data_train, data_test = encode_features(data_train, data_test)
```



# 문자열 값등을 숫자형 카테고리 값으로 변환

```
data_train.head()
```

출력

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Lname	NamePrefix
0	1	0	3	1	4	1	0	0	7	100	19
1	2	1	1	0	0	1	0	3	2	182	20
2	3	1	3	0	7	0	0	0	7	329	16
3	4	1	1	0	7	1	0	3	2	267	20
4	5	0	3	1	7	0	0	1	7	15	19

# Cross-Validation



- Test Set 과 Training Set 를 반복적으로 다르게 추출하여 모델 학습과 평가를 반복적으로 수행.
- 왼쪽 예에서는 10번에 걸쳐서 학습과 평가를 반복함. 최종 평가는 10번의 평균값으로 평가

# Cross Validation – KFold class 이해

Kfold 클래스를 사용하면 쉽게 Cross validation 할 수 있는 **인덱싱 데이터를** 제공한다.

```
def run_kfold_test():
```

```
    kf = KFold(10, n_folds=5) 10개의 Indexing 데이터를 5겹의 Training / Test Set를 만듦
```

```
    for train_index , test_index in kf: 만들어진 5겹 Set 만큼 Loop 를 돌면서 train data용 index와 test data용 index를 출력
```

```
        print('train_index:',train_index , 'test_index:', test_index)
```

```
run_kfold_test()
```

출력

```
train_index: [2 3 4 5 6 7 8 9] test_index: [0 1]
```

```
train_index: [0 1 4 5 6 7 8 9] test_index: [2 3]
```

```
train_index: [0 1 2 3 6 7 8 9] test_index: [4 5]
```

```
train_index: [0 1 2 3 4 5 8 9] test_index: [6 7]
```

```
train_index: [0 1 2 3 4 5 6 7] test_index: [8 9]
```

# Cross Validation 예제

```
def run_kfold(clf):
```

```
    kf = KFold(891, n_folds=10)
```

891개의 Indexing 데이터를 10겹의 Training / Test Set를 만듦

```
    outcomes = []
```

```
    fold = 0
```

```
    for train_index, test_index in kf:
```

10 번 ( 10겹 Training / Test Set) Loop 를 돌면서 Cross validation 수행

```
        fold += 1
```

```
        X_train, X_test = X_all.values[train_index], X_all.values[test_index]
```

```
        y_train, y_test = y_all.values[train_index], y_all.values[test_index]
```

Kfold 로 설정된 train index 와 test index 를 통해 Train 데이터셋과 Test 데이터 셋을 반복적으로 추출

```
        clf.fit(X_train, y_train)
```

```
        predictions = clf.predict(X_test)
```

추출된 데이터셋으로 10번 Loop 마다 재학습 및 정확도 산출

```
        accuracy = accuracy_score(y_test, predictions)
```

```
        outcomes.append(accuracy)
```

```
        print("Fold {0} accuracy: {1}".format(fold, accuracy))
```

```
    mean_outcome = np.mean(outcomes)
```

```
    print("Mean Accuracy: {0}".format(mean_outcome))
```

10번의 정확도를 가지고 평균 정확도 산출

# Cross Validation 예제

```
run_kfold(clf)
```

Fold 1 accuracy: 0.7666666666666667

Fold 2 accuracy: 0.8539325842696629

Fold 3 accuracy: 0.797752808988764

Fold 4 accuracy: 0.8426966292134831

Fold 5 accuracy: 0.8651685393258427

Fold 6 accuracy: 0.8314606741573034

Fold 7 accuracy: 0.7640449438202247

Fold 8 accuracy: 0.797752808988764

Fold 9 accuracy: 0.8539325842696629

Fold 10 accuracy: 0.8089887640449438

**Mean Accuracy: 0.8182397003745319**

# Python ML 을 구성하는 여러 라이브러리

ML Library

Scikit-learn ( or  
tensorflow, Keras)

행렬, 선형대수, 통계등  
ML Library 기반 제공

Numpy , scify

데이터 Handling

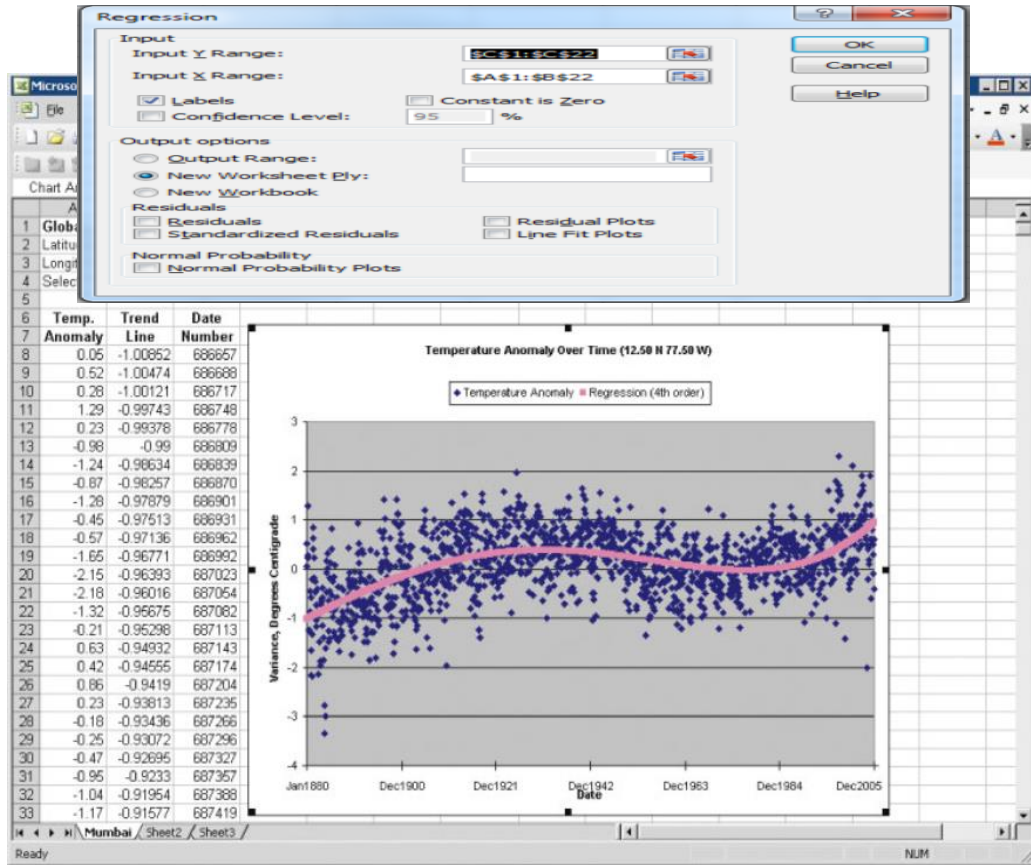
pandas

Visualization

Matplotlib ,  
seaborn

# Python 기반의 데이터 분석 요소

Excel 기반의 Data Science



Excel

sheet 기반에서 다양한  
데이터 가공

데이터 분석 알고리즘  
Function

차트 그리기

Python

Pandas

Scikit-learn

Matplotlib , Seaborn

# Pandas

*Agile Tools for Real World Data*

## Python for Data Analysis



O'REILLY\*

*Wes McKinney*

Python에서 데이터 처리를 위해 가장 인기있는 라이브러리

2차원 데이터를 효율적으로 가공/처리할 수 있는 다양한 훌륭한 기능 제공

Python 의 list , collection , numpy 쉽게 가공,처리,호환 가능.

CSV 등의 포맷 파일과의 인터페이스도 매우 쉬움.

API (SQL like API) 많아서 배우는 시간이 예상보다 많이 필요.

RDBMS SQL 익숙한 개발자들이 쉽게 적응하기 위해 SQL 인터페이스도 진행 중

Pandas 의 개발자인 Wes Mckinney는 월스트리트 금융회사의 분석 전문가임. 회사에서 사용하는 분석용 데이터 핸들링 툴이 맘에 안들어서 Pandas 를 개발하였다고 함. ( 이런 얘기를 들을때마다 IT 분야에 일하는 사람으로서 기가 죽습니다. )



# Pandas Dataframe

DataFrame 기반에서 다양한 데이터 처리 기능 제공

## Creating Pandas DataFrames from Python Lists and Dictionaries

Row Oriented

Dictionary

```
sales = [{ 'account': 'Jones LLC', 'Jan': 150, 'Feb': 200, 'Mar': 140 },
          { 'account': 'Alpha Co', 'Jan': 200, 'Feb': 210, 'Mar': 215 },
          { 'account': 'Blue Inc', 'Jan': 50, 'Feb': 90, 'Mar': 95 } ]
df = pd.DataFrame(sales)
```

List

```
sales = [ ('Jones LLC', 150, 200, 50),
          ('Alpha Co', 200, 210, 90),
          ('Blue Inc', 140, 215, 95) ]
labels = ['account', 'Jan', 'Feb', 'Mar']
df = pd.DataFrame.from_records(sales, columns=labels)
```

default

from\_records

	account	Jan	Feb	Mar
0	Jones LLC	150	200	140
1	Alpha Co	200	210	215
2	Blue Inc	50	90	95

Column Oriented

```
sales = { 'account': ['Jones LLC', 'Alpha Co', 'Blue Inc'],
          'Jan': [150, 200, 50],
          'Feb': [200, 210, 90],
          'Mar': [140, 215, 95] }
df = pd.DataFrame.from_dict(sales)
```

from\_dict

```
sales = [ ('account', ['Jones LLC', 'Alpha Co', 'Blue Inc']),
          ('Jan', [150, 200, 50]),
          ('Feb', [200, 210, 90]),
          ('Mar', [140, 215, 95]) ]
df = pd.DataFrame.from_items(sales)
```

from\_items

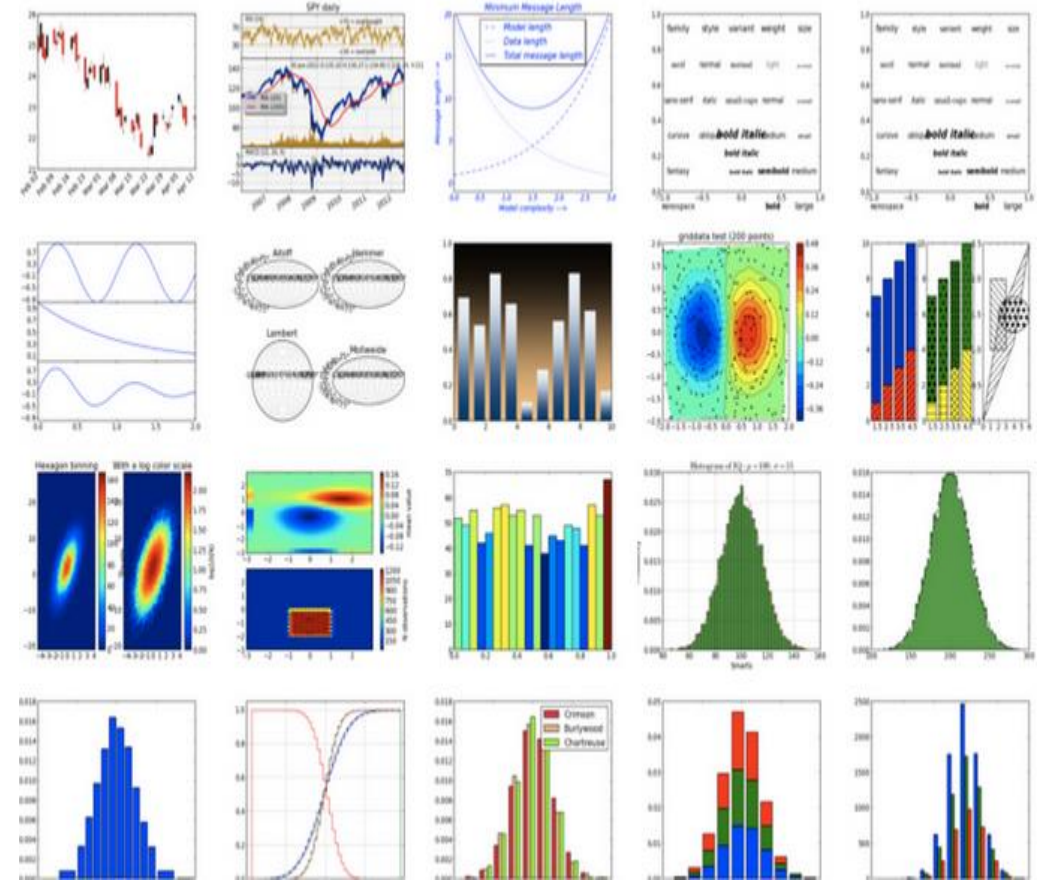
When using a dictionary, column order is not preserved.  
Explicitly order them:  
df = df[['account', 'Jan', 'Feb', 'Mar']]

# Pandas 장점 및 단점

- Python ML 기반의 데이터 가공 처리를 위해서 Numpy array 보다 쉽고, 다양한 기능 제공
- DataFrame 기반에서 다양한 API 제공하고 쉽고 빠르게 데이터 가공/분석/조회 가능. (Spark 2.0에서 Pandas의 DataFrame과 유사한 DataFrame 등장)
- 다양한 컬럼단위 연산등으로 SQL 보다 유연함.
- Join 등으로 다양한 데이터와 연결시에는 SQL 보다는 불편
- 작은 데이터에서는 매우 빠른 속도. 그러나 데이터가 커지면 SQL 보다 속도 지연
- API 기반으로 기존 SQL 사용자들은 새롭게 익숙해져야 하는 불편
- Python ML을 위해서는 자주 사용되는 기본 API 들은 반드시 학습할 필요 있음.

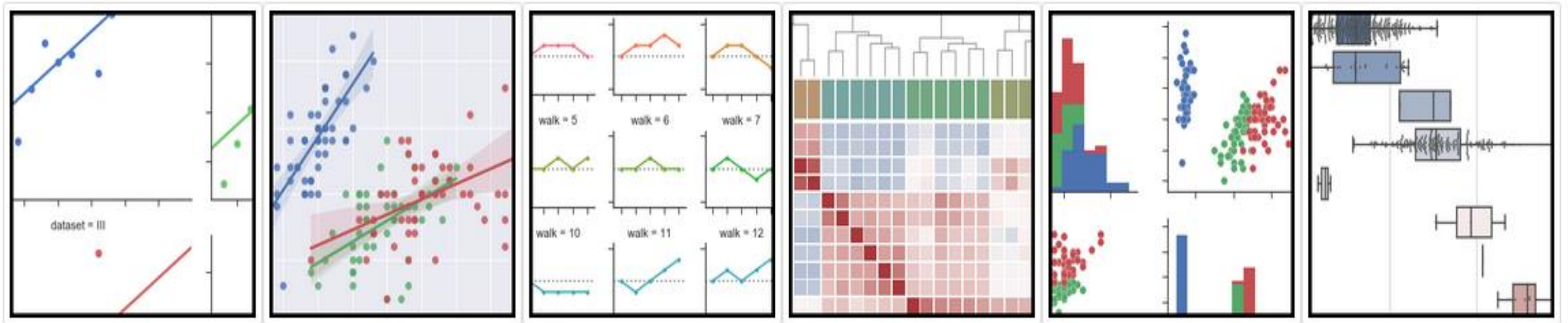
# Matplotlib

- Python Graph Visualization 으로 가장 많이 사용되는 라이브러리.
- Python Visualization 에 많은 공헌을 함.
- 그러나 직관적이지 못한 API 로 인해 개발에 익숙해 지는데 많은 시간 필요하며 현대적인 감각이 떨어지는 Visual 개선 필요.



# Seaborn

Matplotlib 보다 쉽고 , 예쁘게



Matplotlib 기반으로 쉽게 작성됨. Matplotlib 의 high level API 라고 이해하면 편함.

Default 설정만으로 Matplotlib 보다 수려한 Visual 제공

Pandas 와 쉽게 연동

그러나 Matplotlib 를 어느정도 알고 있어야 함.

# Scikit-Learn 소개



- Python 기반의 머신러닝을 위한 가장 쉽고 , 효율적인 개발 라이브러리.
- Python 기반의 머신러닝은 곧 scikit-learn 으로 개발하는 것을 의미할 정도로 오랜 기간 python 세계에서 인정 받아옴.
- R 에 비견 될 수 있는 Python 세계의 분석 라이브러리
- 그러나 최근에는 Deep Learning 의 강세와 Google에서 tensorflow 를 python 기반의 open source 로 오픈한 뒤로 부터 세간의 관심에서 조금씩 멀어짐.

# Scikit-Learn 특징



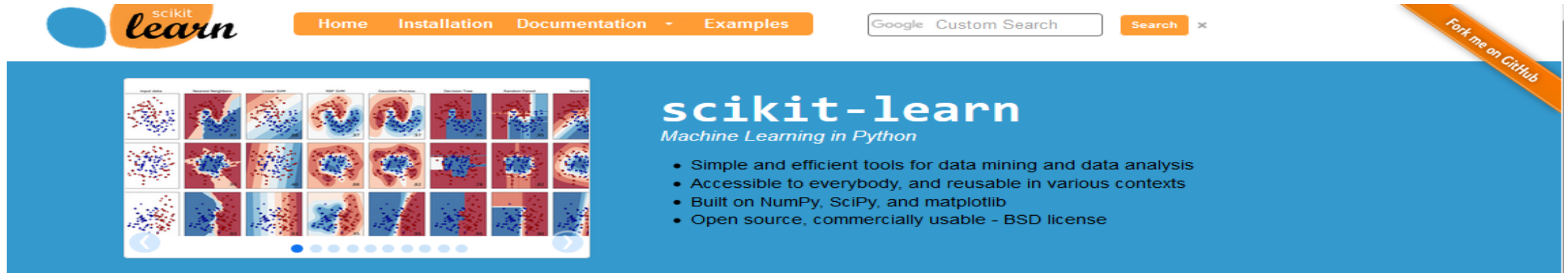
**scikit-learn**  
*Machine Learning in Python*

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

- 가장 쉽다 ( 적어도 Python 세계에서는.. )
- 가장 Python 스러운 API 를 제공한다.
- 머신러닝을 위한 매우 다양한 API 를 제공한다.
- Python 기반의 다른 머신러닝 package 들도 scikit-learn API 지향 ( 그러나 Scikit-learn 역시 R 의 쉽고 다양한 API 에서 영향을 받음. 그리고 R역시 SAS,MATLAB 의 영향을 받았음. )



# Scikit-learn : Machine Learning in Python



The header of the Scikit-learn website features the logo on the left, a navigation bar with links to Home, Installation, Documentation, and Examples, and a search bar on the right. Below the navigation bar is a large blue banner with a grid of 12 small plots showing various machine learning results. To the right of the grid, the text 'scikit-learn' is displayed in a large, bold font, followed by 'Machine Learning in Python' in a smaller font. Below this, a list of bullet points describes the library's features: 'Simple and efficient tools for data mining and data analysis', 'Accessible to everybody, and reusable in various contexts', 'Built on NumPy, SciPy, and matplotlib', and 'Open source, commercially usable - BSD license'. In the top right corner, there is an orange banner that says 'Fork me on GitHub'.

scikit-learn  
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

## Classification

Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, ... — Examples

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, ridge regression, Lasso, ... — Examples

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, ... — Examples

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** PCA, feature selection, non-negative matrix factorization. — Examples

## Model selection

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning

**Modules:** grid search, cross validation, metrics. — Examples

## Preprocessing

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.

**Modules:** preprocessing, feature extraction. — Examples

# Scikit-learn 의 주요 패키지

## Classification

Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, ... — Examples

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, ridge regression, Lasso, ... — Examples

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, ... — Examples

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** PCA, feature selection, non-negative matrix factorization. — Examples

## Model selection

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning

**Modules:** grid search, cross validation, metrics. — Examples

## Preprocessing

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.

**Modules:** preprocessing, feature extraction. — Examples



# Scikit-learn User guide : ML 이해를 위한 뛰어난 Tutorial 제공

- [http://scikit-learn.org/stable/user\\_guide.html](http://scikit-learn.org/stable/user_guide.html)

## User Guide

### 1. Supervised learning

- ▶ 1.1. Generalized Linear Models
- ▶ 1.2. Linear and Quadratic Discriminant Analysis
- 1.3. Kernel ridge regression
- ▶ 1.4. Support Vector Machines
- ▶ 1.5. Stochastic Gradient Descent
- ▶ 1.6. Nearest Neighbors
- ▶ 1.7. Gaussian Processes
- 1.8. Cross decomposition
- ▶ 1.9. Naive Bayes
- ▶ 1.10. Decision Trees
- ▶ 1.11. Ensemble methods
- ▶ 1.12. Multiclass and multilabel algorithms
- ▶ 1.13. Feature selection
- ▶ 1.14. Semi-Supervised
- 1.15. Isotonic regression

## 1.1. Generalized Linear Models

The following are a set of methods intended for regression in which the target value is expected to be a linear combination of the input variables. In mathematical notion, if  $\hat{y}$  is the predicted value.

$$\hat{y}(w, x) = w_0 + w_1x_1 + \dots + w_px_p$$

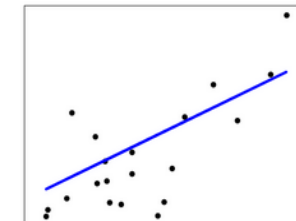
Across the module, we designate the vector  $w = (w_1, \dots, w_p)$  as `coef_` and  $w_0$  as `intercept_`.

To perform classification with generalized linear models, see [Logistic regression](#).

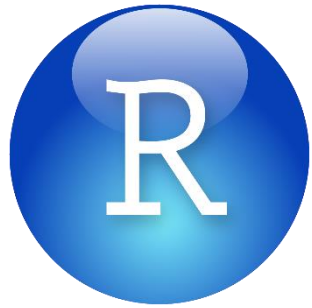
### 1.1.1. Ordinary Least Squares

`LinearRegression` fits a linear model with coefficients  $w = (w_1, \dots, w_p)$  to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w ||Xw - y||_2^2$$



# R 과 Python 비교 – 통계 분석 관점



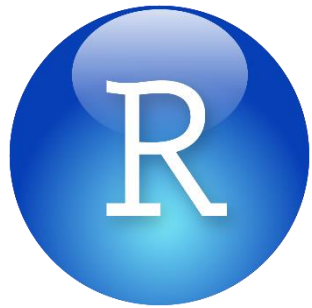
VS



- R 이용자들이 기본적으로 프로그램에 익숙하지 않다는 가정하에 개발됨.
- 매우 쉽고, 직관적으로 통계 분석 프로그램 작성 가능
- R Studio 를 통해 프로그램과 동시에 결과 Visualization 등 효율적인 통계 분석 개발 프레임워크 제공
- 전통적인 통계 분석 부터 머신러닝까지 커버

- 비교적 배우기 쉬운 python 기반으로 이용자들이 프로그램에 어느정도 익숙하다고 가정
- Pandas , numpy , matplotlib 등에 대한 API도 함께 익혀야 함.
- R Studio 와 같은 툴은 제공하지 않음( open source 기반으로 python을 지원하는 R Studio와 같은 툴도 존재하나 상대적인 기능 약함.)

# R 과 Python 비교 – 통계 분석 관점



VS



오픈 소스 통계분석 툴로서 R의 생산성은 Python 보다 높다.  
(일반 프로그램 Language인 Python 이 통계전용 Language인 R과 비교 될 수 있다는  
자체만으로 Python의 유연성이 매우 높다는 것을 증명함에 만족)

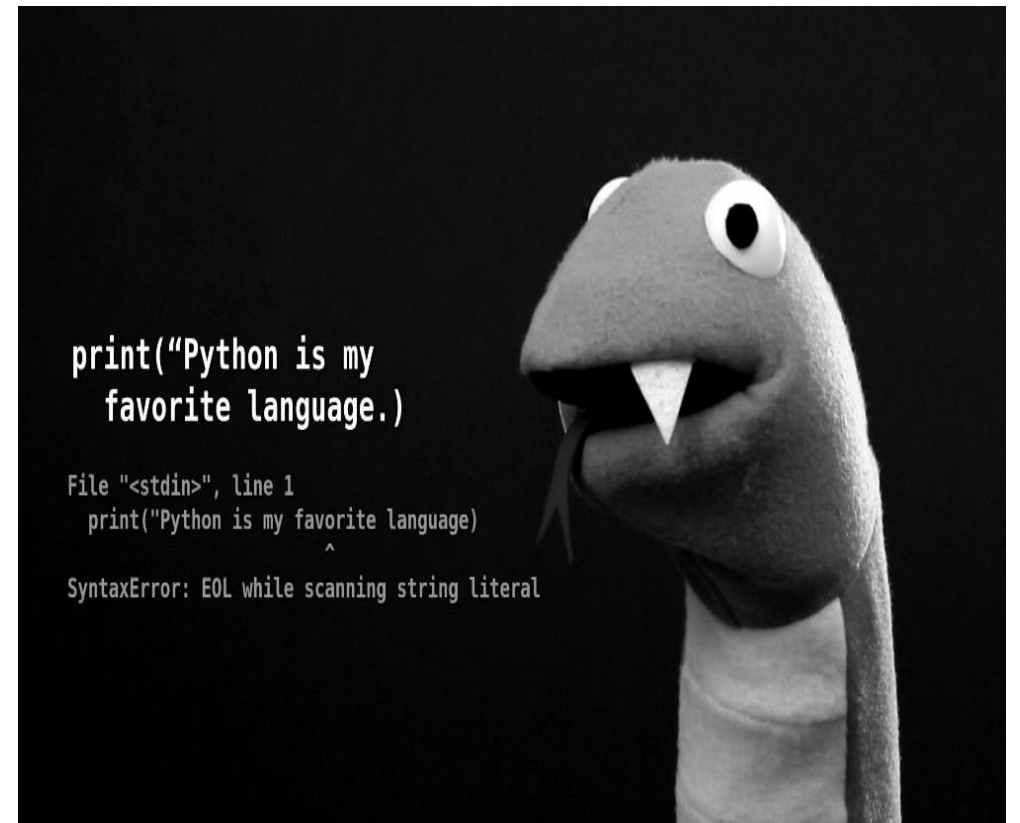
그럼에도 불구하고 Python 기반의 Machine Learning 에 주목해야 하는 이유는 무엇인가 ?

# ML + Python 강점 – Python의 놀라운 인기

Python 은 소리없이 프로그래밍 세계를 점령하고 있는 Language .

복잡한 세상 , 복잡한 코드는 가라

- 쉽고 뛰어난 개발 생산성으로 전 세계의 개발자들 뿐만 아니라 Academy 나 타 영역의 인재들도 Python 선호
- Google , Facebook 등 유수의 IT 업계에서도 Python 의 높은 생산성으로 인해 활용도가 매우 높음. ( 특히 Google)
- 오픈 소스 계열의 전폭적인 지원을 받고 있음.
- 놀라울 정도의 많은 라이브러리 지원은 어떠한 유형의 개발도 쉽게 가능 ( 역으로 선택의 자유가 많아서 오히려 머리가 아플 정도 )
- Interpreter Language의 특성상 속도는 느리지만 쉽고 유연한 특징으로 인해 데스크탑 , 서버 , 네트워크 , 시스템 , IOT 등 다양한 영역에서 사용되고 있음.



# ML + Python 강점 - 뛰어난 확장성 , 연계, 호환성

많은 라이브러리, 뛰어난 생산성을 가지는 Python 언어



# Machine Learning

- 분석 영역을 넘어서 ML 기반의 다양한 Application 개발이 쉽게 가능
- 기존 Application과의 연계도 쉬움 ( 서로 다른 언어로 개발된 Application 의 경우 Rest API )
- Enterprise 아키텍처에도 연계,확장 가능. Microservice 실시간 연계 등.

# ML + Python 강점 - Deep Learning 으로의 진격

- 유수의 Deep Learning Framework 이 Python 기반으로 작성( tensorflow Backend 는 성능때문에 C/C++ 로 작성) .
- 대부분의 Deep Learning 관련 Tutorial , 설명 자료들이 Python 으로 작성되어 제공.
- 현 시점에서 Deep Learning을 활용하기에 가장 좋은 시작점은 Python



주) R에서도 Keras와의 interface를 통해 tensorflow를 사용할 수 있도록 지원 가능

# Deep Learning 현실과 미래

*"You take the blue pill, the story ends. You wake up in your bed and believe whatever you want to believe. You take the red pill, you stay in Wonderland, and I show you how deep the rabbit hole goes."*

—[Morpheus](#), to [Neo](#)





Deep Learning 이 대세인데 , Scikit-Learn 대신 Tensorflow 부터 익히는게 맞지 않나요 ?

# 기존 전문 기술/지식에 따라 선택

- 이 영상인식, 자연어 처리, 음성인식 분야에서 어느정도 경험이 있는 개발자라면 Deep learning 기반의 tensorflow, keras 등 부터 시작하는 것이 Machine Learning에 더 빨리 적응할 수 있는 선택.
- 심층 신경망 Machine Learning 알고리즘 기반의 Deep Learning 은 위 분야에서 기존 ML 알고리즘 대비 우수성이 검증
- Data Science 영역에서 활동하고자 한다면 Deep Learning 기반의 tensorflow 보다는 scikit-learn 부터 시작하는 것이 나은 선택
- Scikit-learn 을 통해 Machine Learning의 구성요소와 이를 통한 주요 솔루션을 쉽게 이해 할 수 있다. Tensorflow api 는 상대적으로 프로그램에 익숙하지 않은 Data Scientist 들에게 쉽게 다가오지 않는다.
- Google은 tensorflow 를 보다 쉽게 접근하기 위해 Keras의 통합을 공식지원하고 있으며 scikit-learn에서 쉽게 tensorflow를 이용할 수 있도록 contribution 하고 있음.

# Deep Learning or Machine Learning ?

Machine Learning 으로 사용자에게 새로운 경험을 강화



- Deep Learning 은 Machine Learning의 한 방법
- Deep Learning 이 인기몰이를 하기 전부터 Machine Learning 은 많은 마법같은 일들을 해결해 옴.
- 앞으로도 전통적인 Machine Learning 으로 해결할 수 있는 멋진 일들이 산재해 있습니다. 그리고 Deep Learning 의 큰 인기는 확실히 지속될 것임. ( 전통적인 Machine Learning 에서 Deep Learning 기반 으로 빠르게 서비스를 변화시키려고 투자중입니다. )

# Deep Learning 하십시오

- 요약하자면 Deep Learning 은 반드시 경험해 보시는게 좋습니다.

저는 정통 통계 분석 전문가인데 Deep Learning을 배워야 하나요?

네,(부럽습니다. ^^;;)

ML 전문가가 된다는 것은 본잡한 논리를 데이터에 근거한 확률로 해석하는 감각을 키우는 긴 여정



- 틀은 틀일 뿐 Machine Learning에 대한 이해 없이는 Scikit-learn 이든 ,tensorflow 이든 학습하기가 어려움.  
( tensorflow는 Neural Network에 대한 이해 없이는 다가서기 어려움 )
- 자신의 가지고 있는 전문 지식과 능력 , ML 을 적용하려는 주요 분야에 따라 원하는 tool 을 선택
- ML의 이론적인 부분에 집중하는 것 보다 ML을 통한 여러 문제들을 해결하는 경험을 쌓는 것이 더 중요하고 tool은 쉽게 이를 지원할 수 있어야 함.

# 이제 뭘하지?

- 성능을 향상 시킬 수 있는 방법을 생각해 보자
  - 알고리즘 교체
  - 파라미터 튜닝
  - 데이터 재 가공
- 타이타닉 코드에 어느정도 익숙해 진뒤에 다양한 변형을 시도해 보자
- Kaggle 등에서 다양한 사례를 접해보자
- Deep Learning을 기본부터 시작하자