# 실전 스터디 소개

KU-BIG 학술부

# 실전 스터디 방안 1

## 2주 Term: 데이터 분석 스터디

- 매회 **특정 주제**(분류, 예측 모델, 이미지, 자연어 처리… )
- 해당 주제에 맞는 **동일한 dataset**을 선택해, **데이터 분석 프로젝트** 진행
- 각자 어떤 식으로 분석을 진행했는지, 구성원들에게 프레젠테이션

**1주차 - 전처리**: 데이터의 특징, 분석 방법 등 의논
**2주차 - 모델링**: 해당 모델 선택 이유, 성과 등 설명

KU-BIG

# 실전 스터디 방안 1 – 예시

kaggle

DACON
DATA TO VALUE

CodaLab
Accelerating reproducible computational research.

COMPAS

**주제: 분류 모델 – Titanic dataset**


**1주차 - 전처리**: 데이터 특징, 시각화, null 값 제거 방법 등 인사이트 공유
**2주차 - 모델링**: 해당 모델 선택 이유, accuracy 비교

KU-BIG

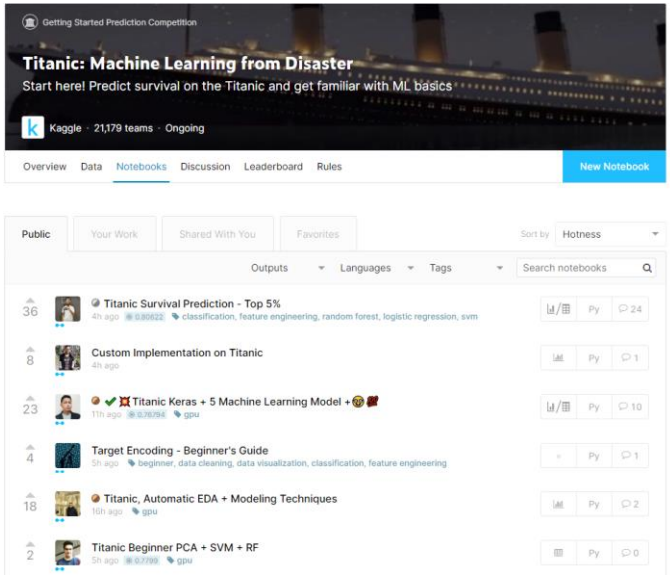
# 실전 스터디 방안 2

## 1주 term 코드 리뷰 스터디

- 매회 **특정 주제**(분류, 예측 모델, 이미지, 자연어 처리… )
- 주제에 적합한 **각자 다른 코드** (전문가 github, 캐글 상위 랭커) 선택
- 각자 kernel에 코드 한 줄 한 줄마다 주석을 달아와, 구성원들에게 설명
- 전부 돌아가며 코드 리뷰를 끝낸 후, 문제 풀이 및 코드 필사

# 실전 스터디 방안 2 – 예시

## 주제: 분류 모델




In [31]:

```
#Encoding using OneHot Encoder

#Train_set
ohe = OneHotEncoder(sparse=False) #ohe = OnehotEncoder
Enco = ohe.fit_transform(train[['Embarked']]) #Enco = Encoding
final_encode = pd.DataFrame(Enco, columns = ['C', 'Q', 'S']) # I create a data frame to concat w
ith train
train = pd.concat([train,final_encode] , axis = 1) # concat both data frame

#Test_Set
ohe_t = OneHotEncoder(sparse=False) #ohe = OnehotEncoder
Enco_t = ohe_t.fit_transform(test[['Embarked']]) #Enco = Encoding
final_encode_t = pd.DataFrame(Enco_t, columns = ['C', 'Q', 'S']) # I create a data frame to conc
at with train
test = pd.concat([test,final_encode_t] , axis = 1) # concat both data frame

#Dropiing addiotinal Columns
train.drop(columns =['PassengerId','Embarked','Title'], axis = 1 , inplace = True)
test.drop(columns =['PassengerId','Embarked','Title'], axis = 1 , inplace = True)
```

In [32]:

```
train.head(2)
```

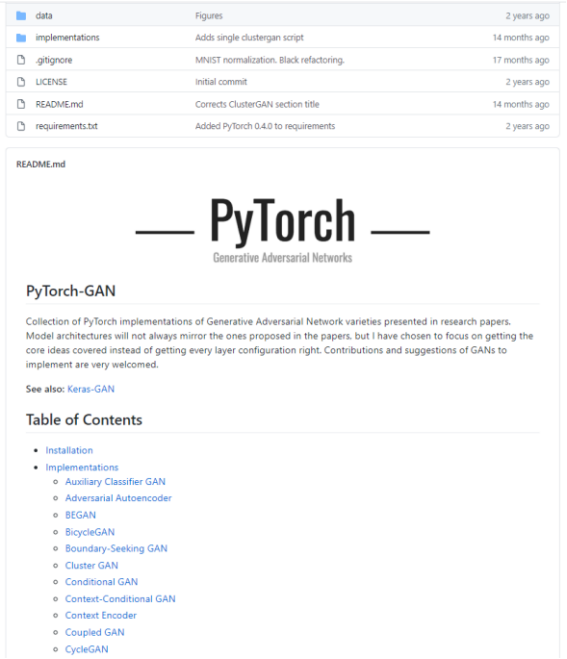
Out[32]:

| | Survived | Pclass | Sex | Age | SibSp | Parch | Fare | C | Q | S |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 1 | 22.0 | 1 | 0 | 7.2500 | 0.0 | 0.0 | 1.0 |
| 1 | 1 | 1 | 0 | 38.0 | 1 | 0 | 71.2833 | 1.0 | 0.0 | 0.0 |

KU-BIG

# 실전 스터디 방안 2 – 예시

## 주제: 생성 모델 - GAN



| | | |
|---|---|---|
| data | Figures | 2 years ago |
| implementations | Adds single clustergan script | 14 months ago |
| .gitignore | MNIST normalization. Black refactoring. | 17 months ago |
| LICENSE | Initial commit | 2 years ago |
| README.md | Corrects ClusterGAN section title | 14 months ago |
| requirements.txt | Added PyTorch 0.4.0 to requirements | 2 years ago |

README.md

PyTorch
Generative Adversarial Networks

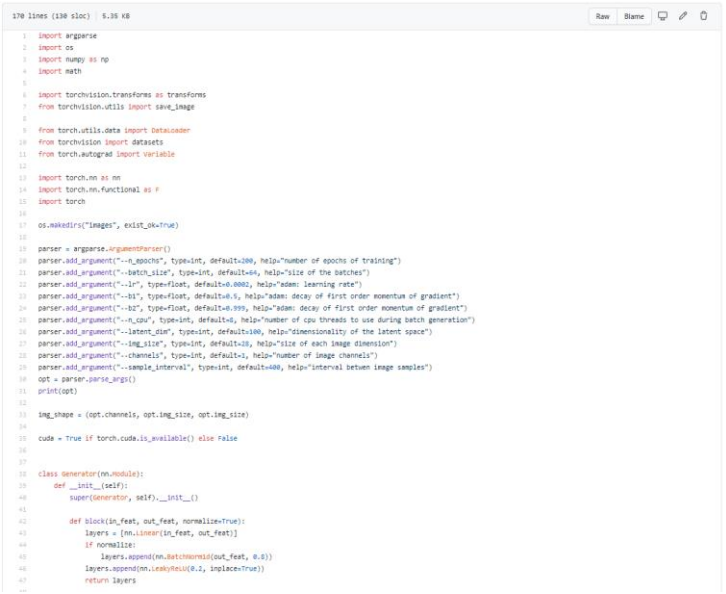
### PyTorch-GAN

Collection of PyTorch implementations of Generative Adversarial Network varieties presented in research papers. Model architectures will not always mirror the ones proposed in the papers. but I have chosen to focus on getting the core ideas covered instead of getting every layer configuration right. Contributions and suggestions of GANs to implement are very welcomed.

See also: Keras-GAN

### Table of Contents

- Installation
- Implementations
  - Auxiliary Classifier GAN
  - Adversarial Autoencoder
  - BEGAN
  - BicycleGAN
  - Boundary-Seeking GAN
  - Cluster GAN
  - Conditional GAN
  - Context-Conditional GAN
  - Context Encoder
  - Coupled GAN
  - CycleGAN



170 lines (130 sloc) | 6.35 KB — Raw | Blame

```python
import argparse
import os
import numpy as np
import math

import torchvision.transforms as transforms
from torchvision.utils import save_image

from torch.utils.data import DataLoader
from torchvision import datasets
from torch.autograd import Variable

import torch.nn as nn
import torch.nn.functional as F
import torch

os.makedirs("images", exist_ok=True)

parser = argparse.ArgumentParser()
parser.add_argument("--n_epochs", type=int, default=200, help="number of epochs of training")
parser.add_argument("--batch_size", type=int, default=64, help="size of the batches")
parser.add_argument("--lr", type=float, default=0.0002, help="adam: learning rate")
parser.add_argument("--b1", type=float, default=0.5, help="adam: decay of first order momentum of gradient")
parser.add_argument("--b2", type=float, default=0.999, help="adam: decay of first order momentum of gradient")
parser.add_argument("--n_cpu", type=int, default=8, help="number of cpu threads to use during batch generation")
parser.add_argument("--latent_dim", type=int, default=100, help="dimensionality of the latent space")
parser.add_argument("--img_size", type=int, default=28, help="size of each image dimension")
parser.add_argument("--channels", type=int, default=1, help="number of image channels")
parser.add_argument("--sample_interval", type=int, default=400, help="interval betwen image samples")
opt = parser.parse_args()
print(opt)

img_shape = (opt.channels, opt.img_size, opt.img_size)

cuda = True if torch.cuda.is_available() else False


class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()

        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]
            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers
```

# 커리큘럼

## 생각해둔 주제들

- 분류 모델 – Titanic, Heart Disease
- 회귀, 예측 모델 – House Prices, Adult Census Income, Covid-19
- imbalanced data – SMOTE 기법, 불량품 감지, 사기 거래 탐지
- 이미지 데이터 처리 – Intel Image classification, cat vs dog
- 자연어 데이터 처리 – Fake and real news, Tweet sentiment extraction
- 시계열 데이터 처리 – 주가, 환율 분석