1) 실전 코딩

- 전처리 직접 코드 구현 + 모델(수상자 코드 참고)

1-1. 발전소 코드 시간 오래 걸리는 주요 요인1(13분 무시할 만함)

```
train = data_loader_all_v2(data_loader_v2, train_list,
folder=train_folder, train_label=train_label, event_time=10, nrows=60)

test = data_loader_all_v2(data_loader_v2, test_list, folder=test_folder,
train_label=None, event_time=20, nrows=60)

y = train['label']
train.drop('label',axis=1,inplace=True)
```

```
CPU times: user 7.31 s, sys: 6.17 s, total: 13.5 s
Wall time: 13min 7s
```

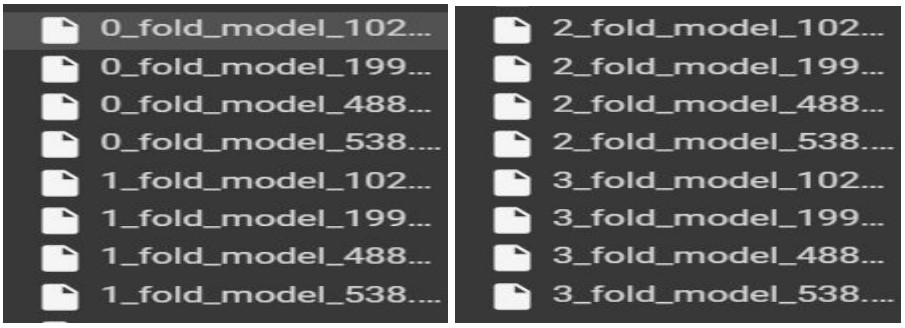
1-2. 발전소 코드 시간 오래 걸리는 주요 요인2(13시간 너무 오래 걸림)

```
# 4FOLD, 3SEED ENSEMBLE
# 총 12개의 모델을 평균내어 예측한다
lucky_seed=[538] # 3등 seed 538 사용 (결과가 더 좋음) 4885,1992,1022

for num,rs in enumerate(tqdm(lucky_seed)):

    kfold = KFold(n_splits=4, random_state = rs, shuffle = True)

    # dacon code
    cv=np.zeros((train.shape[0],198))

    for n, (train_idx, validation_idx) in enumerate(kfold.split(train)):

        x_train, x_validation = train.iloc[train_idx],
train.iloc[validation_idx]
        y_train, y_validation = y.iloc[train_idx],
y.iloc[validation_idx]

        model = lightgbm.LGBMClassifier(**parms, random_state=rs)

        model.fit(x_train, y_train, eval_set=[(x_validation,
y_validation)], early_stopping_rounds= 30, verbose=2) # 30
        joblib.dump(model, '%s_fold_model_%s.pkl'%(n,rs))
      # joblib.dump(model, '../2_Code_pred/%s_fold_model_%s.pkl'%(n,rs))
```

```
        # CROSS-VALIDATION , EVALUATE CV
        cv[validation_idx,:] = model.predict_proba(x_validation)
```

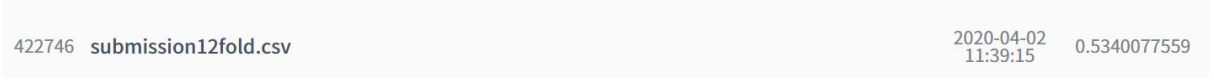
```
14289.45s/it]CPU times: user 15h 37min 40s, sys: 51 s, total: 15h 38min 31s
```

2. 해결책
   - 기존 4FOLD X 3SEED = 12가지 경우의 수를 4FOLD X 1SEED X 3명 으로 하여, 같은 코드에 SEED만 다르게 하여 각자 3명이 코드 돌린다.



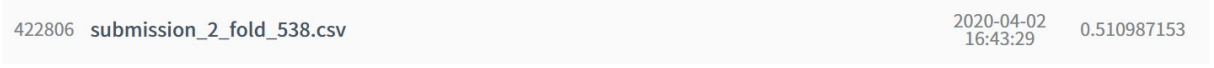
   - 앙상블 코드의 결과로 pkl 파일이 생성됨
   - pkl 파일을 업로드하여 이를 통해, 시간을 잡아먹는 요인2를 실행하지 않고 submission 제출 파일을 생성 가능
   - 결과 : (전) 16시간 → (후) 1시간 미만

3. 제출 관련 유의 사항
   -



   - 1등 코드와 동일 하게 4FOLD X 3SEED(4885,1992,1022) → 0.53
   -

   - 1FOLD X 1SEED(1992) → 0.55
   -



   - 1FOLD X 1SEED(538) → 0.51 (맨 위 앙상블보다 단일 모델이 성능이 더 좋게 나옴)

4. 결론
   1. SEED 별 각자 pkl 파일을 돌리고, 이를 모아 업로드하여 사용한다면 코드 실행 시간 단축 가능

   2. 앙상블과 유사하게 다중 seed 적용이 꼭 좋은 결과를 도출하지는 않음(단일 seed가 더 좋을 수 있음), 따라서 초기에는 다중 seed를 실행하여 좋은 성능을 내는 seed를 찾는데 주력하고, 후반부에는 단일 seed와 비교하여 가장 좋은 성능이 나오는 seed를 채택하는 것이 바람직해보임

2) 전처리 심화

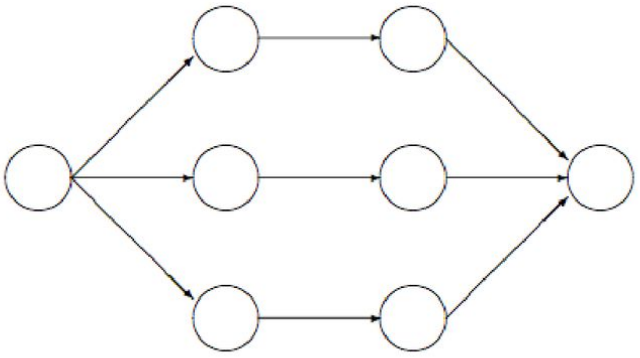
출처 : https://m.blog.naver.com/hancury/220396495672

1. 결측값대체(imputation) : 결측값을 추정한 대체값을 넣어 분석

2. 결측값 대체의 종류
    a. ‘변수 전체’ 를 대체 : unit imputation
    b. ‘변수 일부’ 를 대체 : item imputation
    c. 방법론적으로는 크게 single imputation, multiple imputation 으로 나뉜다.

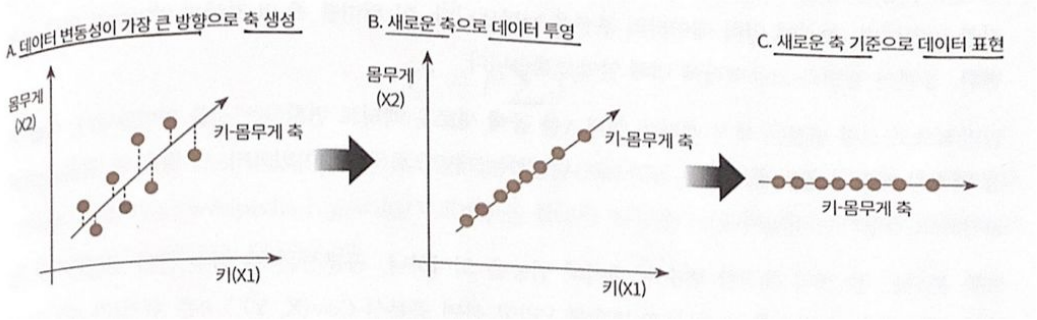
3. Single Imputation
    a. hot deck
    b. mean imputation
        i. 평균값으로 결측값 대체
    c. regression imputation
        i. 다른 값들을 회귀추정 한 값으로 결측값 대체
        ii. 결측대체값에 변동이 전혀 없는 fitted value 를 넣다보니 계수추정치에 대한 신뢰도가 과대평가되는 경향
        iii. 이를 방지하기 위해 residual(잔차)을 추가해준 것을 ‘stochastic regression imputation’이라 하며, single imputation 중에서는 편향이 제일 적은 결과를 가져온다고 알려져 있다. (하지만 여전히 과대 평가되는 경향이 있음)

4. Multiple imputation(Single Imputation 문제를 해결하기 위해 나옴)
    a. single imputation 을 거친 여러개의 데이터 셋을 만들어 평가하므로 관측값은 변함이 없어 동일값들이 많이나온 것처럼 평가되어 높은 가중치를 갖는데 비해, 결측대체값 들은 만들 때마다 추가해주는 residual로 인한 변동 때문에 관측값보다 상대적으로 '약한 계수 추정 신뢰도' 를 갖게 된다

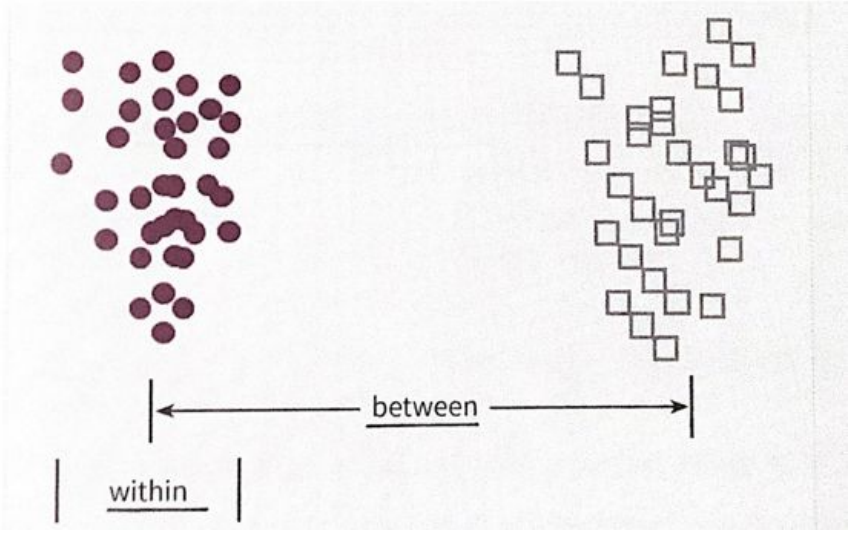


b. Incomplete data Imputed data Analysis results Pooled results

5. 차원 축소 : 많은 피처로 구성된 다차원 데이터 세트의 차원을 축소해 새로운 차원의 데이터 세트를 생성
    a. PCA(Principal component analysis)
        i. 여러 변수 간 상관관계를 이용해 이를 대표하는 주성분을 추출해 차원 축소
        ii. 방법 : 가장 높은 분산을 가지는 데이터 축을 찾아 이 축으로 차원 축소
        iii.

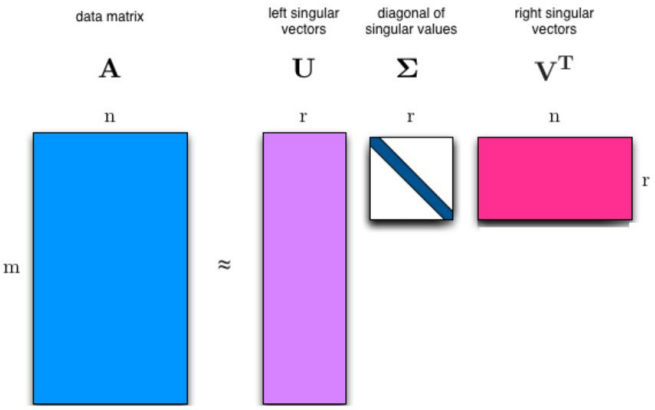


    b. LDA(Linear discriminant analysis) = 선형 판별 분석법
        i. PCA 유사한 점: 입력 데이터 세트를 저차원 공간에 투영해 차원 축소
        ii. PCA 차이점: 분류에서 개별 클래스를 분별할 수 있는 기준을 최대한 유지하며 차원 축소 (클래스 간 분산 ↑, 클래스 내부 분산 ↓)



    c. SVD(Singular value decomposition) = 특이값분해

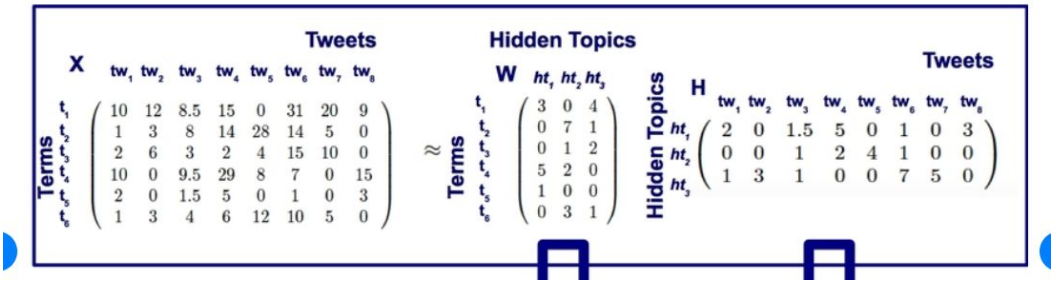
i. PCA와 유사, 행과 열의 크기 다른 행렬도 적용 가능

ii. 

d. NMF(Non-negative matrix factorization)

i. 낮은 Rank를 통한 행렬 근사 방식 변형

ii. 원본 행렬의 모든 원소 값이 0이상 이면 두개의 양수 행렬로 분해

iii. 

3) 4fold X 3seed ≠ 4fold X 1seed X 3번 인데 이유 고민해보기

a) 코드 실수인듯;; 맨위에서 볼 수 있듯이 실제 결과 동일하게 나옴