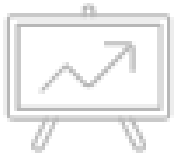


Regression Project Of Kaggle Using ML and DL

Regression Project
Kaggle : House Prices: Advanced Regression Techniques

Goal

미국 Iowa 주 Ames 시 주택의 측면을 설명하는 79가지 변수를 사용하여
각 주택의 판매 가격 변수인 “SalePrice”를 예측한다.

Metric

RMSE(Root-Mean-Squared-Error) = $\sqrt{\frac{1}{|\hat{R}|}\sum_{\hat{r}_{ui}\in\hat{R}}(r_{ui}-\hat{r}_{ui})^2}$

RMSE는 예측 오차(Error) 제곱(Squared)의 평균(Mean)한 값의 제곱근(Root)을
의미하며 이 RMSE의 값을 따라 예측 치를 평가한다.

목표 변수와 목표 변수를 설명하는 79가지 변수, So many...

*SalePrice : 주택의 달러 판매 가격. 목표 변수

* MSSubClass: 주택 구매의 유형
* MSZoning: 일반 구역 분류
* LotFrontage: 부동산과 연결된 거리의 선형 피트
* LotArea: 제곱 피트 Lot 사이즈
* Street: 경로 접근 유형
* Alley: 골목 접근 유형
* LotShape: 주택의 일반 형태
* LandContour: 주택의 평탄도
* Utilities: 사용 가능한 공익사업 유형
* LotConfig: Lot 구성
* LandSlope: 주택의 경사
* Neighborhood: Ames시 경계 내의 물리적 위치
* Condition1: 간선도로 또는 철도와의 근접성
* Condition2: 간선도로 또는 철도와의 근접성(초기가 있는 경우)
* BldgType: 주거 유형
* HouseStyle: 주거 양식
* OverallQual: 전체 소재 및 마감 품질
* OverallCond: 전체 상태 등급
* YearBuilt: 원 공사일
* YearRemodAdd: 리모델 날짜
* RoofStyle: 지붕의 종류
* RoofMatl: 지붕 재료
* Exterior1st: 외부 덮개
* Exterior2nd: 주택의 외부 덮개(두 개 이상의 재료인 경우)
* MasVnrType: 석조 베니어 유형
* MasVnrArea: 평방 피트 내 석조 베니어 영역
* ExterQual: 외장재질 수준
* ExterCond: 외장 재질의 현재 상태
* Foundation: 주택 기초 유형
* BsmtQual: 지하실 높이
* BsmtCond: 지하실의 일반 상태
* BsmtExposure: 워크아웃 또는 정원 수준의 지하 벽
* BsmtFinType1: 지하 마감면적의 품질
* BsmtFinSF1: 1유형의 마감 제곱 피트
* BsmtFinType2: 두 번째 마감된 영역의 품질(있는 경우)
* BsmtFinSF2: 2유형의 마감 제곱 피트
* BsmtUnfSF: 지하 공간의 미완성 된 평방 피트
* TotalBsmtSF: 지하실 면적의 총 평방 피트
* Heating: 난방 유형
* HeatingQC: 난방 품질 및 상태
* CentralAir: 중앙 공조 조절
* Electrical: 전기 시스템
* 1stFlrSF: 1층 평방 피트
* 2ndFlrSF: 2층 평방 피트
* LowQualFinSF: 저품질 마감 제곱 피트(모든 층)
* GrLivArea: 위 등급(지상) 생활 면적 제곱피트
* BsmtFullBath: 지하실 풀 욕실
* BsmtHalfBath: 지하 반 욕실
* FullBath: 등급 이상의 Full 욕실
* HalfBath: 등급 이상의 Half 욕실
* Bedroom: 지하층 이상의 침실 수
* Kitchen: 주방 수
* KitchenQual: 주방 품질
* TotRmsAbvGrd: 등급 이상의 총 객실(화장실은 포함하지 않음)
* Functional: 홈 기능 등급
* Fireplaces: 벽난로 수
* FireplaceQu: 벽난로 품질
* GarageType: 차고 위치
* GarageYrBlt: 차고가 지어진 연도
* GarageFinish: 고 내부 마감
* GarageCars: 차고의 자동차 수용량
* GarageArea: 차고의 평방 피트
* GarageQual: 차고 품질
* GarageCond: 차고의 상태
* PavedDrive: 포장된 진입로
* WoodDeckSF: 평방 피트 내 목재 데크 면적
* OpenPorchSF: 실외 현관의 평방 피트
* EnclosedPorch: 실내 현관의 평방 피트
* 3SsnPorch: Three season porch의 평방 피트
* ScreenPorch: 스크린 현관의 평방 피트
* PoolArea: 수영장의 평방 피트
* PoolQC: 수영장 품질
* Fence: 울타리 품질
* MiscFeature: 다른 범주에서 다루지 않는 기타 기능
* MiscVal: 기타 기능의 달러 가치
* MoSold: 판매 월
* YrSold: 판매 연
* SaleType: 판매 유형
* SaleCondition: 판매 조건.

Index

Chapter 1 : EDA

Step 1 : Data Load

- **Training data와 test data의 크기 확인**
  training data : (1460, 81)
  test data : (1459, 80) => exclude target data “SalePrice”

```
print(df_train.shape)
df_train.head()
```

(1460, 81)

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	Land
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	

```
print(df_test.shape)
df_test.head()
```

(1459, 80)

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	B
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Feedr	Norm	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	NAmes	Norm	Norm	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norm	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Norm	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	Inside	Gtl	StoneBr	Norm	Norm	

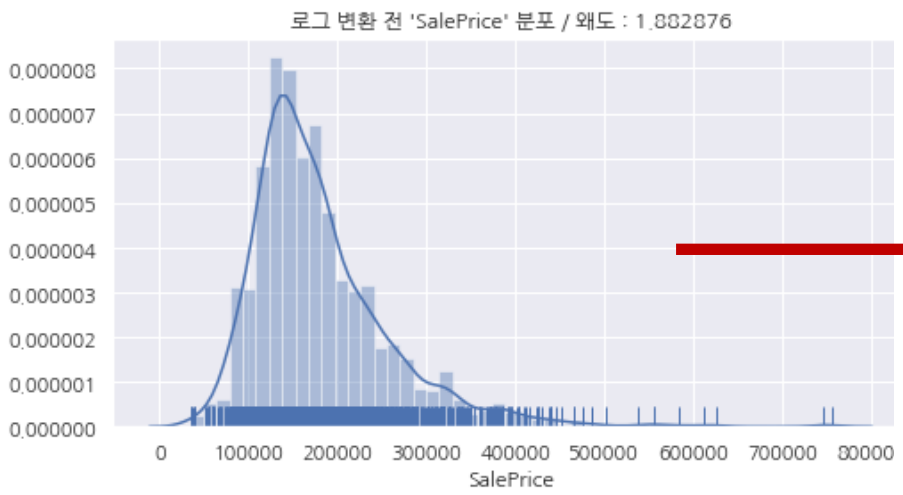
- **사용하지 않을 “Id” 변수 제거**

```
df_train.drop(['Id'], axis=1, inplace=True)
df_test.drop(['Id'], axis=1, inplace=True)
```

- **전처리를 동일하게 하기 위하여 training data 와 test data의 merge**

```
df_merge = pd.concat([df_train, df_test]).reset_index(drop=True)
```

- **종속 변수 “SalePrice”의 정규분포화 : Log 적용**
  - 회귀 분석에서는 종속 변수의 분포가 정규 분포를 따른다고 가정한다.
  - 데이터의 분포가 정규 분포의 형태가 아닌 경우 로그 변환, 제곱근 변환, Box-Cox 변환 등을 통하여 정규 분포를 따르도록 변환한다.
  - 여기에서는 Log 변환 적용



- 로그 변환 전 “SalePrice” 분포
  왜도 : 1.88287
- 로그 변환 후 “SalePrice” 분포
  왜도 : 0.12133

Step 2 : Data Preprocessing

- **Null 값이 존재하는 변수의 type과 총 null 값 확인(34개 변수)**

```
MSZoning (object) : 4
LotFrontage (float64) : 486
Alley (object) : 2721
Utilities (object) : 2
Exterior1st (object) : 1
Exterior2nd (object) : 1
MasVnrType (object) : 24
MasVnrArea (float64) : 23
BsmtQual (object) : 81
BsmtCond (object) : 82
BsmtExposure (object) : 82
BsmtFinType1 (object) : 79
BsmtFinSF1 (float64) : 1
BsmtFinType2 (object) : 80
BsmtFinSF2 (float64) : 1
BsmtUnfSF (float64) : 1
TotalBsmtSF (float64) : 1
Electrical (object) : 1
BsmtFullBath (float64) : 2
BsmtHalfBath (float64) : 2
KitchenQual (object) : 1
Functional (object) : 2
FireplaceQu (object) : 1420
GarageType (object) : 157
GarageYrBlt (float64) : 159
GarageFinish (object) : 159
GarageCars (float64) : 1
GarageArea (float64) : 1
GarageQual (object) : 159
GarageCond (object) : 159
PoolQC (object) : 2909
Fence (object) : 2348
MiscFeature (object) : 2814
SaleType (object) : 1
```

- **일부 변수들의 null 값을 "None" / "NA" / 0 으로 대체**
  - null 값이 의미하는 것이 시설이 존재하지 않음을 나타내는 것으로 판단되는 변수에 한해서
  - 골목, 지하실, 차고, 수영장, 석조 베니어, 벽난로, 담장, 기타 기능과 관련된 변수들

  - 골목 : Alley
  - 지하실(categorical) : BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2
  - 지하실(numeric) : BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath
  - 차고(categorical) : GarageType, GarageFinish, GarageQual, GarageCond
  - 차고(numeric) : GarageYrBlt, GarageCars, GarageArea
  - 수영장 : PoolQC
  - 석조 베니어 : MasVnrType, MasVnrArea
  - 벽난로 : FireplaceQu
  - 울타리 : Fence
  - 기타 기능 : MiscFeature

- **상대적 우위를 가지는 Categorical 변수를 상대적 우위에 따라 숫자로 변환**
  - kaggle competition으로부터 제공받은 "data_description" 파일 참고
  - 시설 존재 여부 가능성에 따라 0부터 시작, 1부터 시작으로 구분(ex: "NA" 값이 있는 지)
  - ex => 'Ex' : Excellent, 'Gd' : Good, 'TA' : Average/Typical, 'Fa' : Fair, 'Po' : Poor

```
# 최저 등급이 1인 columns
df_merge["LandSlope"].astype("category",    categories=['Sev', 'Mod', 'Gtl'],ordered=True).cat.codes + 1
df_merge["ExterQual"].astype("category",    categories=['Po', 'Fa', 'TA','Gd', 'Ex'],ordered=True).cat.codes +
df_merge["ExterCond"].astype("category",    categories=['Po', 'Fa', 'TA','Gd', 'Ex'],ordered=True).cat.codes +
df_merge["HeatingQC"].astype("category",    categories=['Po', 'Fa', 'TA','Gd', 'Ex'],ordered=True).cat.codes +
df_merge["KitchenQual"].astype("category", categories=['Po', 'Fa', 'TA','Gd', 'Ex'],ordered=True).cat.codes +
df_merge["Electrical"].astype("category",   categories=['Mix', 'FuseP', 'FuseF', 'FuseA', 'SBrkr'],ordered=True
df_merge["Functional"].astype("category",   categories=['Sal', 'Sev', 'Maj2', 'Maj1', 'Mod', 'Min2', 'Min1', 'T
df_merge["LotShape"].astype("category",     categories=['IR3', 'IR2', 'IR1', 'Reg'],ordered=True).cat.codes + 1
df_merge["LandContour"].astype("category", categories=['Low', 'HLS', 'Bnk', 'Lvl'],ordered=True).cat.codes + 1
df_merge["Utilities"].astype("category",    categories=['ELO', 'NoSeWa', 'NoSewr', 'AllPub'],ordered=True).cat.

# 최저 등급이 0인 columns : 존재하지 않을 수 있음
df_merge["BsmtQual"].astype("category",     categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'],ordered=True).cat.
df_merge["BsmtCond"].astype("category",     categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'],ordered=True).cat.
df_merge["FireplaceQu"].astype("category",  categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'],ordered=True).cat.
df_merge["GarageQual"].astype("category",   categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'],ordered=True).cat.
df_merge["GarageCond"].astype("category",   categories=['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'],ordered=True).cat.
df_merge["BsmtExposure"].astype("category", categories=['NA', 'No', 'Mn', 'Av', 'Gd' ],ordered=True).cat.codes
df_merge["BsmtFinType1"].astype("category", categories=['NA', 'Unf', 'LwQ', 'Rec', 'BLQ', 'ALQ', 'GLQ'],ordere
df_merge["BsmtFinType2"].astype("category", categories=['NA', 'Unf', 'LwQ', 'Rec', 'BLQ', 'ALQ', 'GLQ'],ordere
df_merge["CentralAir"].astype("category",   categories=['N', 'Y'],ordered=True).cat.codes
df_merge["GarageFinish"].astype("category", categories=['NA', 'Unf', 'RFn', 'Fin'],ordered=True).cat.codes
df_merge["PoolQC"].astype("category",       categories=['NA', 'Fa', 'TA', 'Gd', 'Ex'],ordered=True).cat.codes
df_merge["Fence"].astype("category",        categories=['NA', 'MnWw', 'GdWo', 'MnPrv', 'GdPrv'],ordered=True).
```

- **Null 값이 존재하는 나머지 categorical 변수들은 최빈값(MODE)로 대체**

```
MSZoning (object) : 4
Exterior1st (object) : 1
Exterior2nd (object) : 1
SaleType (object) : 1
```

- **Null 값이 존재하는 나머지 numeric 변수들은 중앙값(MEDIAN)로 대체**

```
LotFrontage (float64) : 486
```

- **변수 값이 숫자이지만 값이 상대적 우위를 나타내지 않는 경우, 카테고리화**
  - MSSubClass, MoSold, YrSold

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

MSSubClass: 주택 구매의 유형

20	1-STORY 1946 & NEWER ALL STYLES
30	1-STORY 1945 & OLDER
40	1-STORY W/FINISHED ATTIC ALL AGES
45	1-1/2 STORY - UNFINISHED ALL AGES
50	1-1/2 STORY FINISHED ALL AGES
60	2-STORY 1946 & NEWER
70	2-STORY 1945 & OLDER
75	2-1/2 STORY ALL AGES
80	SPLIT OR MULTI-LEVEL
85	SPLIT FOYER
90	DUPLEX - ALL STYLES AND AGES
120	1-STORY PUD (Planned Unit Development) - 1946 & NEWER
150	1-1/2 STORY PUD - ALL AGES
160	2-STORY PUD - 1946 & NEWER
180	PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
190	2 FAMILY CONVERSION - ALL STYLES AND AGES

Step 3 : Feature engineering

- **기존 변수의 조합/변형으로 새로운 변수 생성**
  - 기존 변수와 만든(create한) 변수를 구별하기 위해 생성 변수 앞에 'cr_'을 붙여 구분
  - 종류별 현관의 넓이 나타내는 변수의 조합으로 총 현관의 넓이 변수 생성
  - 지상, 지하실, 현관, 수영장의 넓이를 나타내는 변수의 조합으로 총 넓이 변수 성성
  - 울타리 등급, 수영장 넓이, 지하실 넓이, 차고 넓이, 벽난로 개수를 나타내는 변수들로 각 시설의 존재여부를 나타내는 변수 생성

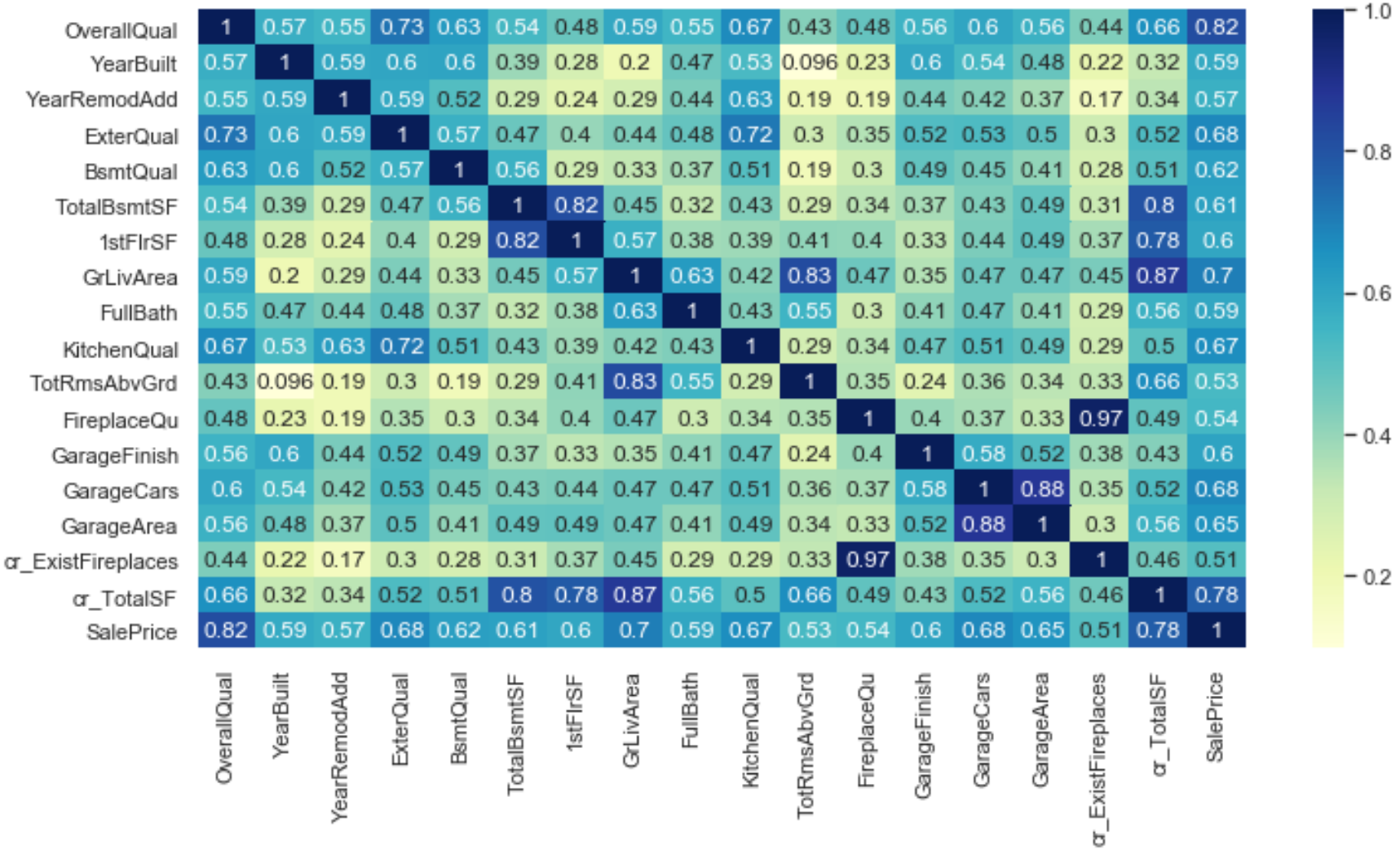
```
df['cr_TotalPorchSF'] = df['WoodDeckSF'] + df['OpenPorchSF'] + df['EnclosedPorch'] + df['3SsnPorch'] + df['ScreenPorch']
df['cr_TotalSF'] = df['GrLivArea'] + df['TotalBsmtSF'] + df['cr_TotalPorchSF'] + df['PoolArea']
```

```
df['cr_ExistFence'] = df['Fence'].apply(lambda x : 0 if x == 'NA' else 1)
df['cr_ExistPool'] = df['PoolArea'].apply(lambda x : 1 if x > 0 else 0)
df['cr_ExistBsmt'] = df['TotalBsmtSF'].apply(lambda x : 1 if x > 0 else 0)
df['cr_ExistGarage'] = df['GarageArea'].apply(lambda x : 1 if x > 0 else 0)
df['cr_ExistFireplaces'] = df['Fireplaces'].apply(lambda x : 1 if x > 0 else 0)
```

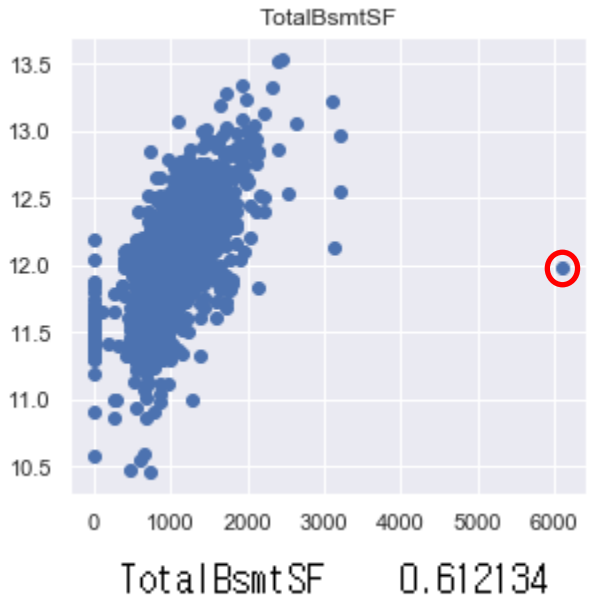
- **Outlier 제거**

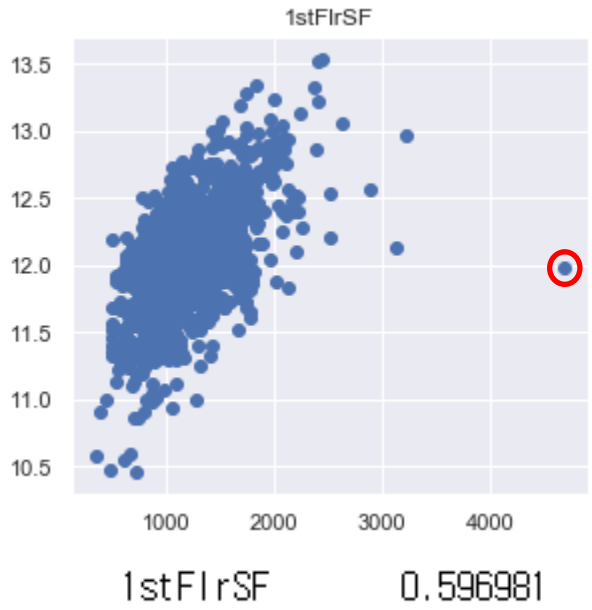
  - 종속변수 “SalePrice”와 상관계수의 절댓값이 0.5가 넘는 변수들 중에 scatter plot을 통해 확인

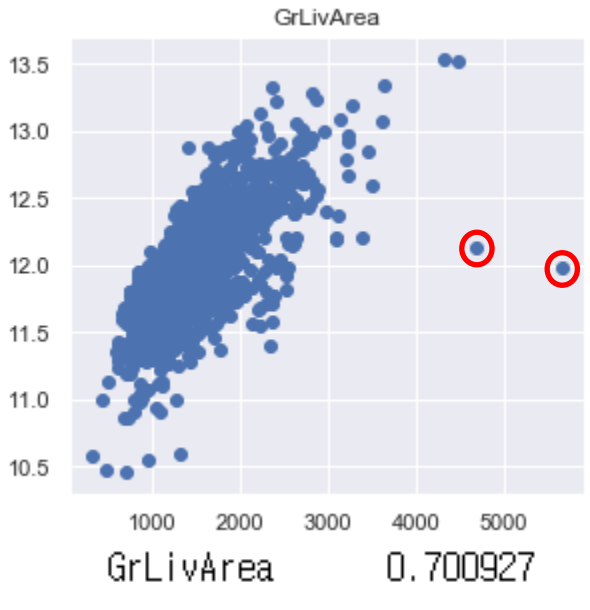
SalePrice와 상관계수의 절댓값이 0.5 이상인 변수들의 상관계수



- scatter plot으로 outlier 예상이 가능한 변수들의 scatter plot과 상관계수







- 판별된 outlier들의 row index 확인 및 제거

```
TotalBsmtSF : [1298]
1stFlrSF : [1298]
GrLivArea : [523, 1298]
```

```
outlier = [523, 1298]
for frame in (df_merge, dfy):
    frame.drop(outlier, inplace=True)
```

- 어떤 변수도 제거하지 않는 version1과 일부 변수들을 제거하는 version2로 구분하여 modeling 진행

- Version1 : categorical 변수들의 one-hot-encoding, **전처리 종료**

```
1  df_all = pd.get_dummies(df_merge)
2  print(df_all.shape)
```

```
(2917, 255)
```

- Version2 : 상관계수의 절댓값이 0.1 이하인 변수 제거
  - 직접 생성한 변수는 제거에서 제외

```
LandContour    -0.057796
Utilities       0.012630
LandSlope      -0.038552
OverallCond    -0.036821
ExterCond       0.049342
BsmtFinType2    0.014065
BsmtFinSF2      0.004863
LowQualFinSF   -0.037951
BsmtHalfBath   -0.005124
3SsnPorch       0.054914
PoolArea        0.074338
PoolQC          0.084999
MiscVal        -0.020012
```

- **Version2 : 단일 값이 2,900개 이상인 변수 제거(총 row는 2,917개)**

  - 직접 생성한 변수는 제거에서 제외

  Street의 가장 많은 값의 갯수 : 2905

- **Version2 : 상관계수의 절댓값이 0.9가 넘으면 다중공선성으로 판단하여 제거, 전처리 종료**

  - SalePrice와 다중공선성을 보이는 변수는 확인되지 않음
  - 독립변수간 다중공선성이 나타날 경우, 종속변수와의 상관계수가 낮은 변수 제거

```
Fireplaces(0.491998) : ['cr_ExistFireplaces']
FireplaceQu(0.543095) : ['cr_ExistFireplaces']
GarageYrBlt(0.349013) : ['GarageQual' 'GarageCond' 'cr_ExistGarage']
GarageQual(0.363258) : ['GarageYrBlt' 'GarageCond' 'cr_ExistGarage']
GarageCond(0.356766) : ['GarageYrBlt' 'GarageQual' 'cr_ExistGarage']
Fence(-0.145701) : ['cr_ExistFence']
cr_ExistFence(-0.177199) : ['Fence']
cr_ExistGarage(0.322994) : ['GarageYrBlt' 'GarageQual' 'GarageCond']
cr_ExistFireplaces(0.510253) : ['Fireplaces' 'FireplaceQu']
```

  - Fireplaces, cr_ExistFireplaces, GarageYrBlt, GarageCond, cr_ExistGarage, Fence 변수 제거

Chapter 2 : ML Modeling

Step 1 : Set Criteria

- Cross Validation은 10 K-fold로 한다.(XGBoost Regression은 5 K-fold)
- K-fold 교차 검증

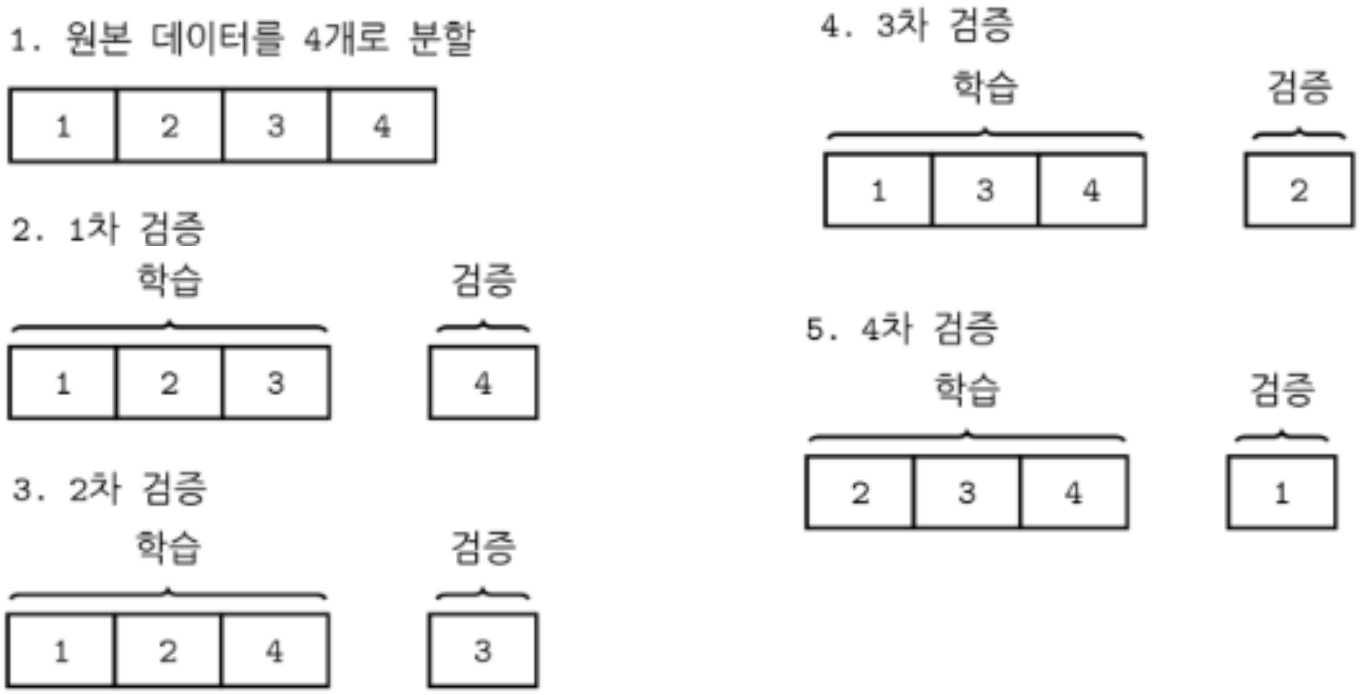
(1). 전체 데이터를 K개의 부분 집합 ($\{D_1, D_2, \cdots, D_k\}$)로 나눈다.

(2). $\{D_1, D_2, \cdots, D_{k-1}\}$를 사용하여 회귀분석 모형을 만들고 $\{D_k\}$로 교차검증 한다.

(3). $\{D_1, D_2, \cdots, D_{k-2}, D_k\}$를 사용하여 회귀분석 모형을 만들고 $\{D_{k-1}\}$로 교차검증 한다.

⋮

(4). $\{D_2, \cdots, D_k\}$를 사용하여 회귀분석 모형을 만들고 $\{D_1\}$로 교차검증 한다.



- **가장 기본적이고 간단한 회귀 알고리즘인 Linear Regression으로 결과를 확인한다.**

  - Linear Regression은 복잡도를 제어할 수 없는 간단한 모형이기 때문에 Linear Regression으로 구한 값보다 좋으면 양호한 것으로 판단한다.

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D = w_0 + w^T x$$

[Linear Regression Cross Validation 점수]

- Score : validation 평균 점수(validation 편차)
- Linear Regression(all columns) : 0.12620 (0.01200)
- Linear Regression(removed) RMSE : 0.12353 (0.01218)

[Linear Regression Submission 제출 점수]

- Score : submission 점수 (validation과 오차)
- Linear Regression(all columns) : 0.13514 (0.00894)
- Linear Regression(removed) RMSE : 0.14042 (0.01689)

Step 2 : Single Algorithm Learning

- **각 Single Algorithm의 Modeling 과정**

  (1). 각 algorithm의 최적 parameter들을 GridSearchCV 통해 구한다.
  (2). GridSearchCV로 구한 복수개의 모형 중 최고 점수를 낸 parameter를 가진 모형은 best_estimator_ 속성에 저장된다.
  (3). best_estimator_에 저장된 모형을 cross_val_score를 사용하여 10 K-fold로 한번 더 교차검증을 반복한다.
  (4). 10개의 cross_val_score들의 평균과 편차를 구한다.
  (5). best_estimator_에 저장된 모형으로 예측 값을 출력하여 제출 점수를 확인한다.
  (6). 제출 점수와 cross_val_score 평균값의 오차를 구하고 오차가 cross_val_score의 편차에서 크게 벗어나지 않는 지 확인한다.
  (7). (1)~(6)의 과정을 version1 data(all columns)와 version2 data(removed data)에 동일하게 적용한다.

- **Ridge 회귀 모형**

  - 선형회귀 계수(weight)에 대한 제약 조건을 추가함으로써 과최적화를 막는 정규화 선형회귀 방법 중 하나이다.
  - 가중치들의 제곱 합을 최소화 하는 것을 추가적인 제약 조건으로 한다.

$$w = \arg min_w \left(\sum_{i=1}^{N} {e_i}^2 + \lambda \sum_{j_i}^{M} {w_j}^2 \right)$$

  - $\lambda \sum_{j_i}^{M} {w_j}^2$이 추가된 규제 항이 된다.
  - alpha : 하이퍼모수 λ. 정규화 정도를 조절하며 크면 정규화 정도가 커지고 가중치의 값들이 작아진다. 0이 되면 일반적인 선형 회귀 모형이 된다.
  - max_iter : gradient solver의 최대 반복 횟수이다.
  - 규제 항은 훈련하는 동안에만 비용함수에 추가되며 성능을 평가하거나 예측할 때는 포함하지 않고 규제가 없는 성능 지표로 평가한다.
  - 단점은 모든 예측 변수를 중요도에 따라 가중 값만 축소시킬 뿐, 0값을 부여하지 않기 때문에 불필요한 변수가 제거되지 않고 항상 남아있게 된다.

[Ridge Cross Validation 점수]

- Score : validation 평균 점수(validation 편차)
- Ridge (all columns) RMSE : 0.11286 (0.01269)
- Ridge (removed) RMSE : 0.11593 (0.01321)

[Ridge Submission Score]

- Score : submission 점수 (validation과 오차)
- Ridge (all columns) : 0.11983 (0.00697)
- Ridge (removed) RMSE : 0.12751 (0.01158)

[Ridge(all columns) parameters]

- Ridge(alpha=19.5, copy_X=True, fit_intercept=True, max_iter=None, normalize=False, random_state=None, solver='auto', tol=0.001)

[Ridge(removed columns) parameters]

- Ridge(alpha=11.0, copy_X=True, fit_intercept=True, max_iter=None, normalize=False, random_state=None, solver='auto', tol=0.001)

- **Lasso 회귀 모형**

  - 선형회귀 계수(weight)에 대한 제약 조건을 추가함으로써 과최적화를 막는 정규화 선형회귀 방법 중 하나이다.
  - 가중치들의 절댓값의 합을 최소화 하는 것을 추가적인 제약 조건으로 한다.

$$w = \arg min_w \left(\sum_{i=1}^{N} {e_i}^2 + \lambda \sum_{j_i}^{M} |w_j| \right)$$

  - $\lambda \sum_{j_i}^{M} |w_j|$이 추가된 규제 항이 된다.
  - alpha : 하이퍼모수 λ. 정규화 정도를 조절하며 크면 정규화 정도가 커지고 가중치의 값들이 작아진다. 0이 되면 일반적인 선형 회귀 모형이 된다.
  - Lasso의 중요한 특징은 덜 중요한 특성의 가중치를 0으로 만들어 완전히 제거하려고 한다는 점이다. 다시 말해, Lasso는 자동으로 특성 선택을 하고 희소 모델(sparse model)을 만든다. 즉, 0이 아닌 특성의 가중치가 적다.

[Lasso Cross Validation 점수]

- Score : validation 평균 점수(validation 편차)
- Lasso (all columns) RMSE : 0.16040 (0.01670)
- Lasso (removed) RMSE : 0.16104 (0.01657)

[Lasso Submission Score]

- Score : submission 점수 (validation과 오차)
- Lasso (all columns) : 0.17650 (0.0161)
- Lasso (removed) RMSE : 0.17847 (0.01743)

[Lasso (all columns) parameters]

- Lasso(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=1000, normalize=False, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)

[Lasso (removed columns) parameters]

- Lasso(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=1000, normalize=False, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)

- **ElasticNet 회귀 모형**

  - 선형회귀 계수(weight)에 대한 제약 조건을 추가함으로써 과최적화를 막는 정규화 선형회귀 방법 중 하나이다.
  - 가중치들의 절댓값의 합과 제곱 합을 동시에 제약 조건으로 가지는 것으로, Ridge 회귀와 Lasso 회귀를 절충한 모델이다.

$$w = \arg min_w \left(\sum_{i=1}^{N} {e_i}^2 + \lambda_1 \sum_{j_i}^{M} |w_j| + \lambda_2 \sum_{j_i}^{M} {w_j}^2 \right)$$

  - λ_1과 λ_2 두 개의 하이퍼 모수를 가진다. 혼합 정도는 혼합 비율 r을 사용해 조절한다. r=0이면 Ridge 회귀와 같고 r=1이면 Lasso 회귀와 같다.
  - alpha : 하이퍼모수 λ. 정규화 정도를 조절하며 크면 정규화 정도가 커지고 가중치의 값들이 작아진다. 0이 되면 일반적인 선형 회귀 모형이 된다.
  - l1_ratio : λ_1과 λ_2의 혼합 비율

[ElasticNet Cross Validation 점수]

- Score : validation 평균 점수(validation 편차)
- ElasticNet (all columns) RMSE : 0.12943 (0.01566)
- ElasticNet (removed) RMSE : 0.13380 (0.01696)

[ElasticNet Submission Score]

- Score : submission 점수 (validation과 오차)
- ElasticNet (all columns) : 0.14259 (0.01316)
- ElasticNet (removed) RMSE : 0.15012 (0.01632)

[ElasticNet (all columns) parameters]

- ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.0, max_iter=1000, normalize=False, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)

[ElasticNet (removed columns) parameters]

- ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.0, max_iter=1000, normalize=False, positive=False, precompute=False, random_state=None, selection='cyclic', tol=0.0001, warm_start=False)

- **Gradient Boosting Regression모형**

  - 약한 학습기(Weak Learner)를 결합하여 강한 학습기(Strong Learner)를 만드는 Boosting 방법 중 하나로 Gradient descent를 사용하여 최적의 parameter를 찾는 Boosting 방법이다.
  - Ensemble에 이전까지의 오차를 보정하도록 예측기를 순차적으로 추가한다.
  - loss : 손실 함수(loss function)을 지정하는 parameter이다.
    'ls'는 least squares로 최소자승법(residual의 제곱의 합을 최소화)을 의미한다.
    'lad'는 least absolute deviation으로 오차의 절댓값의 합계를 의미한다.
    'huber'는 huber loss로 ls와 lad를 절충한 것이다. 일정한 범위를 정해서 그 안에 있으면 오차를 구하고, 그 밖에 있으면 오차의 절댓값을 구한다.
  - max_features : 분할할 최적의 숫자를 지정한다.

[Gradient Boosting Regression Cross Validation 점수]

- Score : validation 평균 점수(validation 편차)
- Gradient Boosting Regression (all columns) RMSE : 0.12943 (0.01566)
- Gradient Boosting Regression (removed) RMSE : 0.13380 (0.01696)

[Gradient Boosting Regression Submission Score]

- Score : submission 점수 (validation과 오차)
- Gradient Boosting Regression (all columns) : 0.14259 (0.01316)
- Gradient Boosting Regression (removed) RMSE : 0.15012 (0.01632)

[Gradient Boosting Regression (all columns) parameters]

- GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=1600, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)

[Gradient Boosting Regression (removed columns) parameters]

- GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.1, loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=3100, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)

- **Support Vector Regression모형**

  - SVM의 maximal margin과 같은 특징들을 유지하여 Regression을 사용할 수 있다.
  - SVM 회귀는 SVM 분로의 목표와는 반대로 한다. 일정한 마진 오류 안에서 두 클래스 간의 도로 폭이 가능한 최대가 되도록 하면서 제한된 마진 오류 안에서 가능한 많은 샘플이 들어가도록 학습한다.
  - kernel : 알고리즘에 사용할 kernel 유형을 지정한다. 'rbf'는 가우시안 방사 기저 함수 (Radial Basis Function)을 의미하며 비선형 특성을 다루는 함수 중 하나이다. 여기에서는 이 'rbf'만 사용한다. (kernel 인수의 종류로는 'linear', 'poly', 'rbf', 'sigmoid'가 있다.
  - gamma : 하나의 훈련 샘플이 영향을 미치는 범위를 결정한다. 작은 값은 넓은 범위를 의미하며 큰 값은 영향을 미치는 범위가 제한적이다.
  - C : error의 규제 parameter이다. C를 줄이면 margin의 폭이 넓어지지만 margin 오류도 커진다. C가 커지면 margin이 좁아지지만 margin 오류가 적다.
  - epsilon : margin의 폭을 지정한다. (허용 오차와는 다르다.)
  - tol : 허용 오차

[Support Vector Regression Cross Validation 점수]

- Score : validation 평균 점수(validation 편차)
- Support Vector Regression (all columns) RMSE : 0.39571 (0.02713)
- Support Vector Regression (removed) RMSE : 0.39548 (0.02703)

[Support Vector Regression Submission Score]

- Score : submission 점수 (validation과 오차)
- Support Vector Regression (all columns) : 0.41716 (0.02145)
- Support Vector Regression (removed) RMSE : 0.41656 (0.02108)

[Support Vector Regression (all columns) parameters]

- SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.0001, kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

[Support Vector Regression (removed columns) parameters]

- SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.0001, kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

- **XGBoost Regression모형**

  - XGBoost는 Extreme Gradient Boosting의 약자로 Gradient Boosting 알고리즘을 핵심으로 한다.
  - 병렬 처리를 사용하기 때문에 학습과 계산이 빠르고 Greedy-algorithm을 사용한 자동 가지치기로 과적합이 잘 발생하지 않는다.

[XGBoost Regression Cross Validation 점수]

- Score : validation 평균 점수(validation 편차)
- XGBoost Regression (all columns) RMSE : 0.11742 (0.00971)
- XGBoost Regression (removed) RMSE : 0.12090 (0.01061)

[XGBoost Regression Submission Score]

- Score : submission 점수 (validation과 오차)
- XGBoost Regression (all columns) : 0.13020 (0.01278)
- XGBoost Regression (removed) RMSE : 0.13660 (0.0157)

[XGBoost Regression (all columns) parameters]

- XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, importance_type='gain', learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=1000, n_jobs=1, nthread=None, objective='reg:linear', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1)

[XGBoost Regression (removed columns) parameters]

- XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, importance_type='gain', learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=700, n_jobs=1, nthread=None, objective='reg:linear', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1)

[Regression of All columns]

- Linear Regression (all) RMSE : 0.13514
- Ridge (all) RMSE : 0.11983
- Lasso (all) RMSE : 0.1765
- ElasticNet (all) RMSE : 0.14259
- Gradient Boosting Regression (all) RMSE : 0.13461
- Support Vector Regression (all) RMSE : 0.41716
- XGBoost Regression (all) RMSE : 0.1302

[Regression of Removed columns]

- Linear Regression (removed) RMSE : 0.14042
- Ridge (removed) RMSE : 0.12751
- Lasso (removed) RMSE : 0.17847
- ElasticNet (removed) RMSE : 0.15012
- Gradient Boosting Regression (removed) RMSE : 0.13569
- Support Vector Regression (removed) RMSE : 0.41656
- XGBoost Regression (removed) RMSE : 0.1366

Chapter 3 : DL Modeling

Step 1 : Preprocessing

- DL Modeling의 과정은 Google Colab을 사용하여 진행

- #Chaper1 단계에서 진행한 전처리 중 Remove Outlier을 제외한 과정 진행
  - Missing values 대체
  - 상대적 우위를 갖는 categorical 변수들을 상대적 우위에 따라 숫자로 변환
  - numeric value가 상대적 우위를 갖지 않는 numeric 변수들의 카테고리화
  - 기존 변수의 조합/변형으로 새로운 변수 생성
  - 모든 categorical 변수의 one-hot-encoding
  - 여기서는 version1과 version2로 구분하지 않고 모든 변수 사용

Step 2 : Set Neural Network

- Neural Network는 Keras를 사용하여 구현하며 구현 순서는 아래와 같다.

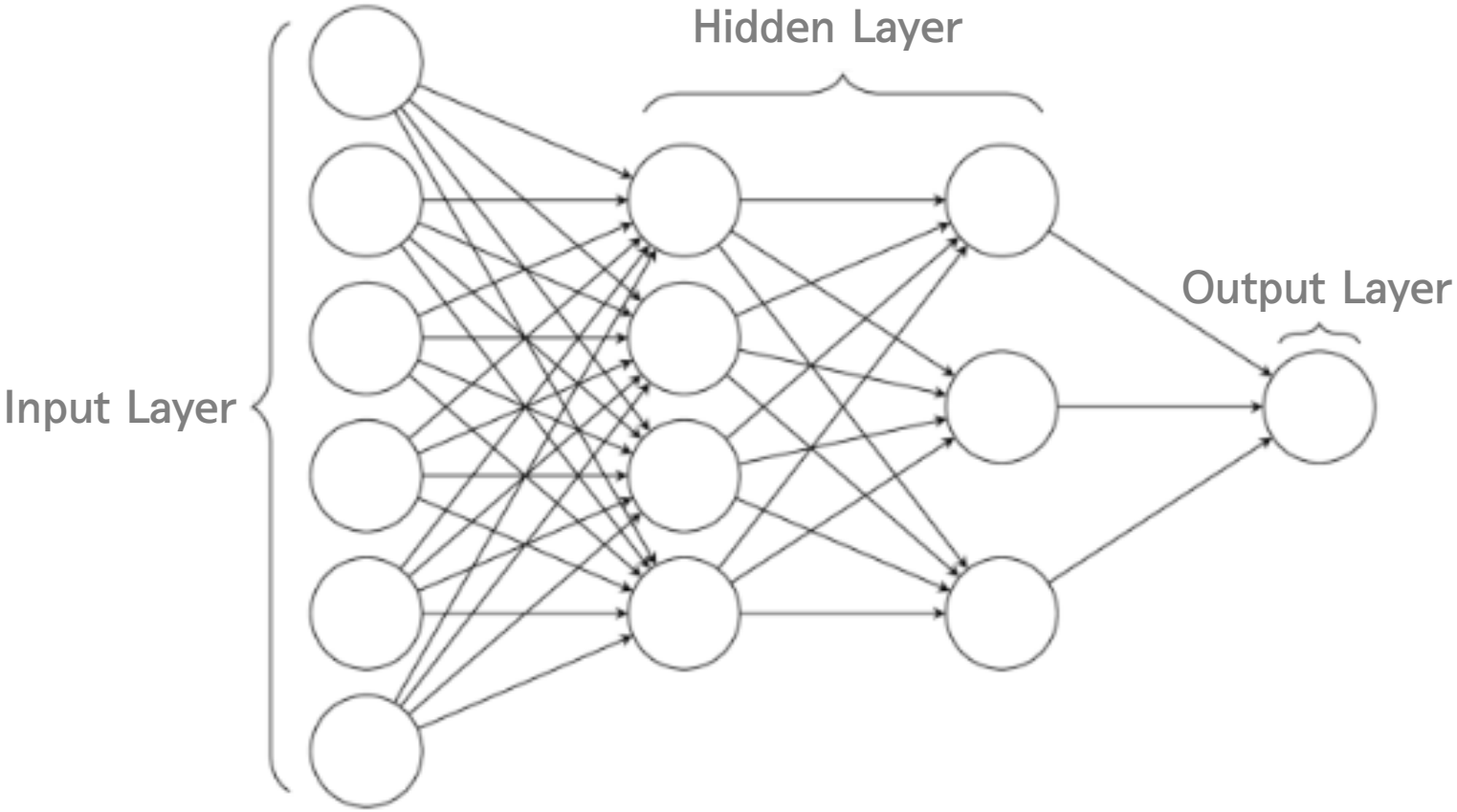
(1). 'Sequential'로 모형 클래스 객체를 생성한다.

(2). 'add' method로 layer를 추가한다.
- 입력 layer부터 순차적으로 추가한다.
- 첫 번째 layer는 'input_dim' parameter를 사용하여 입력 크기를 설정한다.
- 각 layer는 첫 번째 parameter로 출력 뉴런의 수를 입력한다.
- 각 layer는 'activation' parameter로 activation function을 지정한다.
  (activation function은 linear한 단순 합을 입력 받아 non-linearity를 제공한다.)
- 'kernel_initializer' parameter로 각 layer의 초기 가중치를 설정하는 방법을 지정한다.
- 마지막 layer의 출력 뉴런의 수는 output의 종류에 맞게 설정한다.(여기서는 1)

(3). 'compile' method로 모형을 완성한다.
- 'loss' parameter로 loss function을 지정한다.
- 'optimizer' parameter로 최적화 알고리즘을 지정한다.
- 'metrics' parameter로 training단계에서 기록할 성능 기준을 설정한다.

(4). 'fit' method로 training을 진행한다.
- 'epoch' parameter로 반복될 epoch의 수를 지정한다.
- 'validation_split' parameter로 training data 중 교차 검증에 사용될 부분을 지정한다.

- **최적 Network를 탐색**
  - hidden layer의 개수 : 3개~5개
  - 각 layer 출력 뉴런의 개수 : 128개, 256개, 512개
  - activation function : ReLU, Leaky ReLU
  - optimizer : SGD, RMSProp, Adam

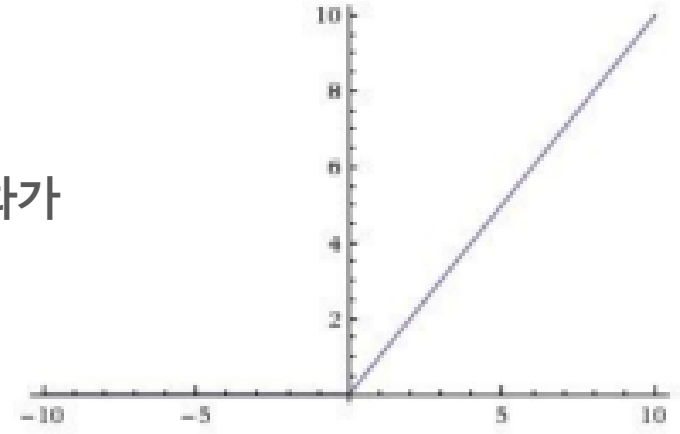


• Used Activation Function

- Neural Net에서 activation function이 없으면 몇 layer를 가지든지 단일의 linear한 function으로 표현이 가능해진다.

(1). ReLU

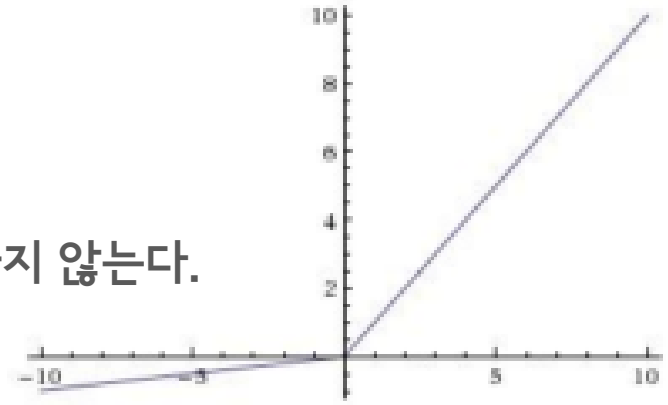
- max(0, x)
- x가 양수인 지점에서는 기울기가 1이기 때문에 포화가 발생하지 않는다.
- 연산이 매우 효율적이고 sigmoid나 tanh에 비해 6배 이상 빠르게 수렴한다.



- 모두 0 이상의 값을 가지고 있기 때문에 zero-centered가 아니다.
- x가 0보다 작을 때 gradient 값은 0이 되기 때문에 vanishing gradient가 발생한다.

(2). Leaky ReLU

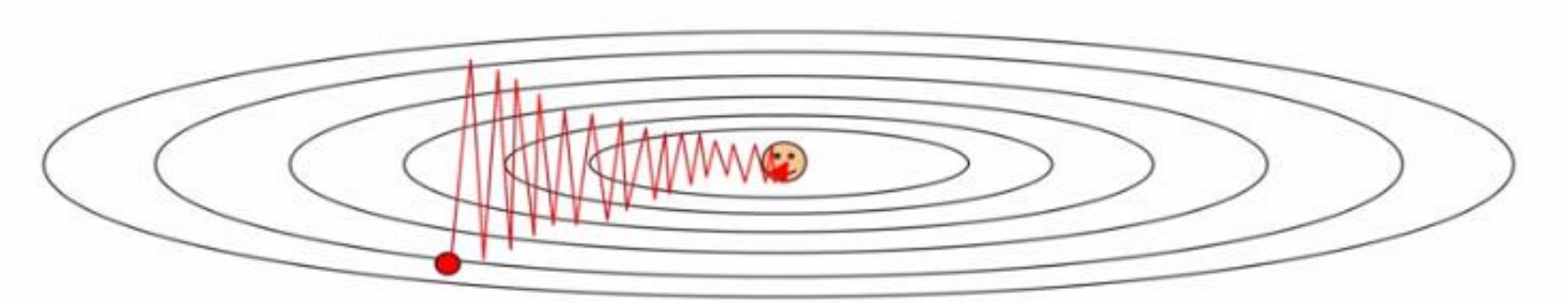
- max(0.1x, x)
- x가 0보다 작은 부분에서 0이 아닌 기울기를 가진다.
  => x가 0보다 크냐 작으냐에 상관없이 포화가 발생하지 않는다.
  => vanishing gradient 문제 또한 발생하지 않는다.



• Used Optimizer(weight parameter update)

(1). SGD(Stochastic Gradient Descent)

- 한번의 learning rate만큼 전진하기 위해 모든 데이터를 넣어주어야 했던 Gradient Descent에 비해서 조금(Mini batch)만 빠르게 보고 전진한다는 장점이 있다.
- 그러나 여전히 매우 느리기 때문에 실제 상황에서는 사용하기 꺼려진다.



- 위와 같은 loss function이 있다고 해보자. SGD에서 minimum으로 이동하기 위한 경로는, 수직으로는 경사가 급하기 때문에 빠르게 이동하고 수평으로는 경사가 얕기 때문에 천천히 이동한다.

결국 왼쪽의 파란선의 벡터와 같은 경로로 이동하게 된다. 그리고 위의 그림과 같이 지그재그로 이동을 하게 되고 그러다 보니 매우 느리게 진행을 하게 되는 현상이 나타난다.

(2). RMSProp

- AdaGrad의 식($G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$)에서 gradient의 제곱 값을 더해나가면서 구한 G_t의 부분을 합이 아니라 지수 평균으로 바꾸어서 대체한 방법이다. 이렇게 대체를 할 경우 AdaGrad처럼 G_t가 무한정 커지지는 않으면서 최근 변화 량의 변수간 상대적인 크기 차이는 유지할 수 있다. 식으로 나타내면 다음과 같다.

$$G = \gamma G + (1 - \gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G + \varepsilon}} \cdot \nabla_\theta J(\theta_t)$$

이 수식을 코드로 나타내면 다음과 같다.

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += -learning_rate * dx/(np.sqrt(cache) + 1e-7)
```

- 여기에서 도입된 decay_rate는 hyperparameter로 보통 0.9나 0.99의 값으로 설정이 된다. 이 decay_rate의 값으로 cache의 값이 서서히 감소한다. 따라서 AdaGrad의 장점인, 경사에 경도되지 않는 equalize 효과는 유지되면서 AdaGrad의 단점이었던 step_size가 0이 되어서 학습이 종료되는 문제점이 해결된다.

(3). Adam(Adaptive Moment Estimation)

- Adam은 RMSProp과 Momentum을 결합한 방식이다. 이 방식에서는 Momentum 방식과 유사하게 지금까지 계산해온 기울기의 지수 평균을 저장하며, RMSProp과 유사하게 기울기의 제곱 값의 지수평균을 저장한다.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta))^2$$

다만, Adam에서는 m과 v가 처음에 0으로 초기화되어 있기 때문에 학습의 초반부에서는 m_t, v_t가 0에 가깝게 bias 되어있을 것이라고 판단하여 이를 unbiased하게 만들어주는 작업을 거친다. m_t와 v_t의 식을 Σ형태로 펼친 후 양변에 expectation을 씌워서 정리해 보면 다음과 같은 보정을 통해 unbiased된 expectation을 얻을 수 있다. 이 보정된 expectation들을 가지고 gradient가 들어갈 자리에 $\widehat{m_t}$, G_t가 들어갈 자리에 v_t를 넣어 계산을 진행한다.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^{\,t}}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^{\,t}}$$
$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}}\hat{m}_t$$

- **Optimizer 별 최적의 Network와 parameter**

(1). SGD

- learning_rate의 조절(0.00001 ~ 1)에도 prediction 시 nan 값 발생

(2). Adam

- best score : 0.15321
- first layer : 출력 뉴런 128, leaky relu
- hidden layer : 4개, 출력 뉴런 128, relu

(3). RMSProp

- best score : 0.17380
- first layer : 출력 뉴런 128, leaky relu
- hidden layer : 4개, 출력 뉴런 128, leaky relu

Chapter 4 : Final Result

[All Scores]

- Ridge (all) RMSE : 0.11983 → **Best Score : 1099th**
- Ridge (removed) RMSE : 0.12751 **(all ranker is 4409)**
- XGBoost Regression (all) RMSE : 0.1302
- Gradient Boosting Regression (all) RMSE : 0.13461
- Linear Regression (all) RMSE : 0.13514
- Gradient Boosting Regression (removed) RMSE : 0.13569
- XGBoost Regression (removed) RMSE : 0.13660
- Linear Regression (removed) RMSE : 0.14042
- ElasticNet (all) RMSE : 0.14259
- ElasticNet (removed) RMSE : 0.15012
- DL Adam : 0.15321
- DL RMSProp : 0.17380
- Lasso (all) RMSE : 0.17650
- Lasso (removed) RMSE : 0.17847
- Support Vector Regression (removed) RMSE : 0.41656
- Support Vector Regression (all) RMSE : 0.41716
- DL SGD : nan

Thank you.