



Getting Started Prediction Competition

House Prices - Advanced Regression Techniques

Predict sales prices and practice feature engineering, RFs, and gradient boosting



Kaggle · 5,305 teams · Ongoing


Goal

It is your job to predict the sales price for each house. For each Id in the test set, you must predict the value of the SalePrice variable.

Metric

Submissions are evaluated on [Root-Mean-Squared-Error \(RMSE\)](#) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.)

File descriptions

- train.csv - the training set
- test.csv - the test set 
- data_description.txt - full description of each column, originally prepared by Dean De Cock but lightly edited to match the column names used here
- sample_submission.csv - a benchmark submission from a linear regression on year and month of sale, lot square footage, and number of bedrooms

Data fields

80 + 1 (Target)

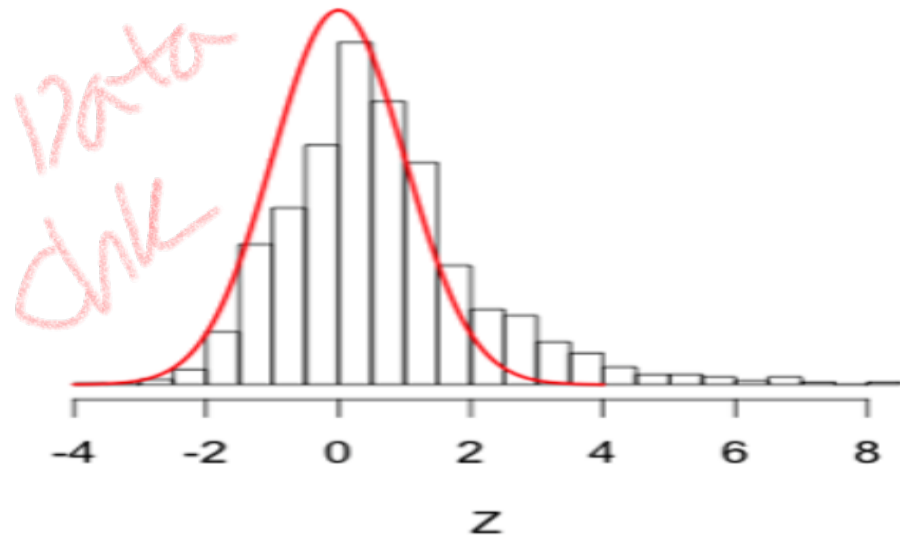
Here's a brief version of what you'll find in the data description file.

- **SalePrice** - the property's sale price in dollars. This is the target variable that you're trying to predict.
- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration

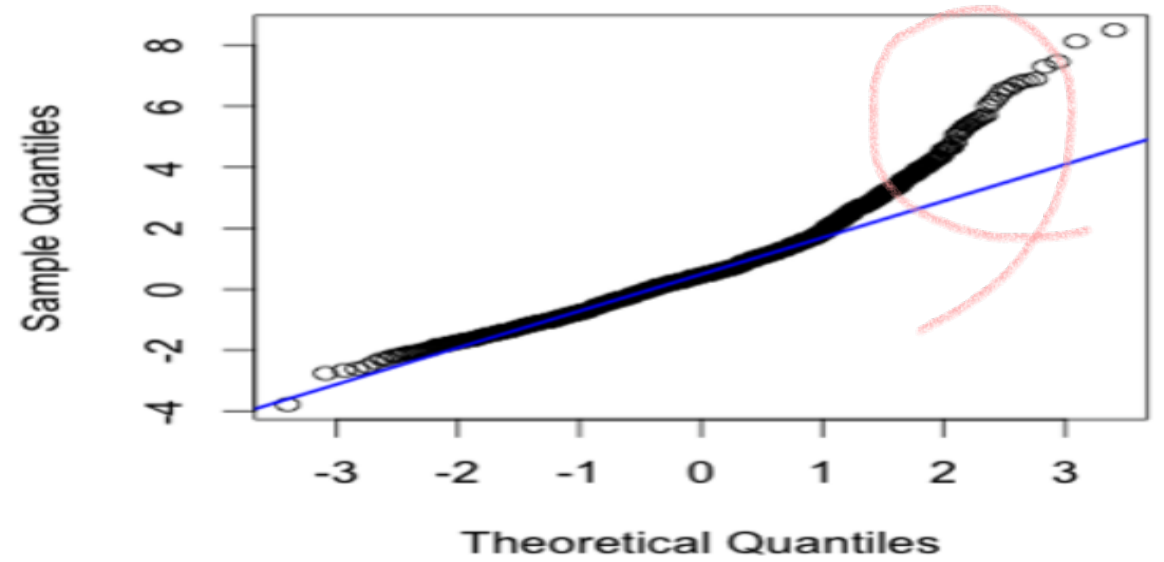
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality
- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation

- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: \$Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold
- SaleType: Type of sale
- SaleCondition: Condition of sale

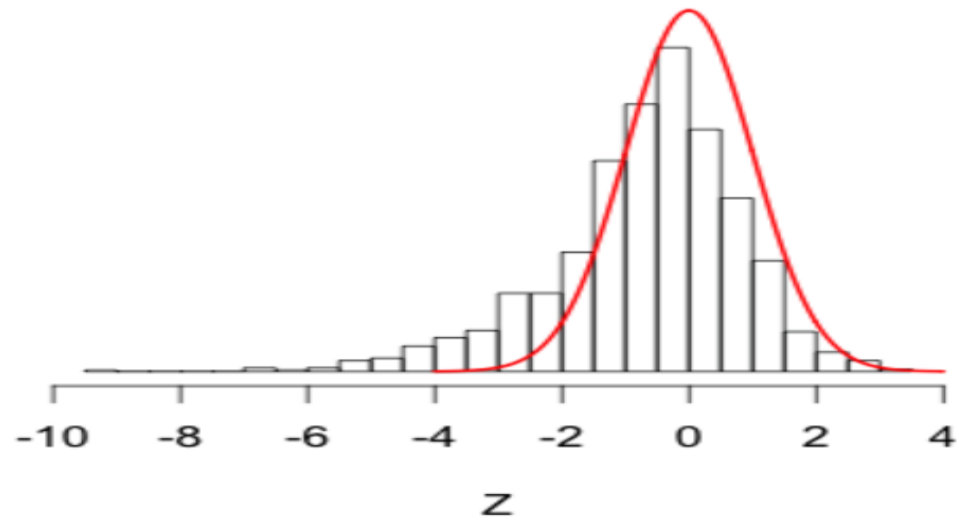
Skewed Right



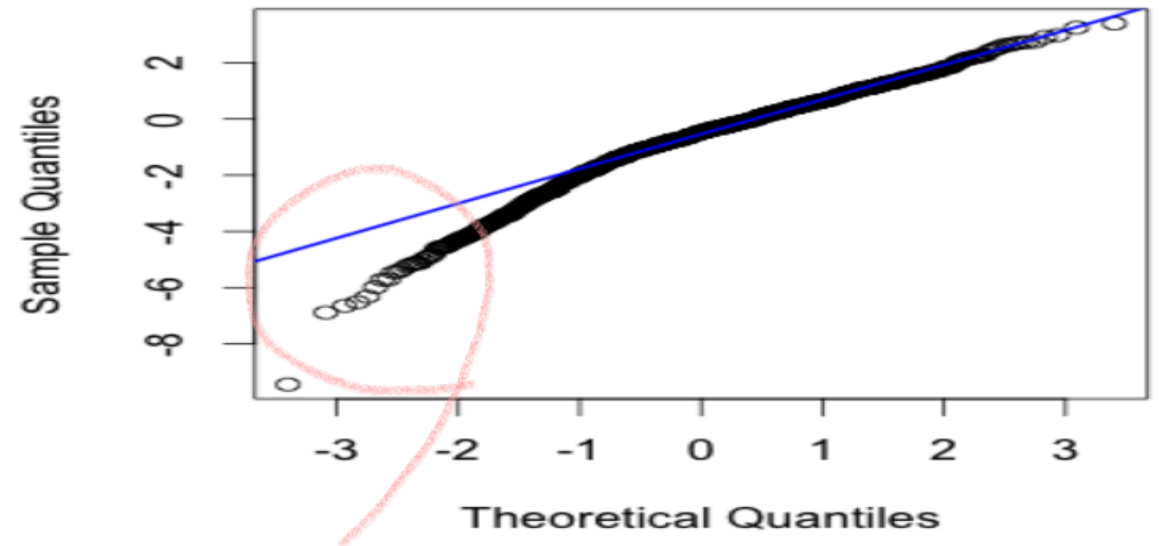
Normal Q-Q Plot



Skewed Left



Normal Q-Q Plot



5. 데이터 단위 변환

- 데이터의 스케일(측정단위)이 다른 경우 특히 KNN과 같은 거리 기반 모델에 영향이 큼

1) Scaling: 평균이 0, 분산이 1인 분포로 변환

2) MinMax Scaling: 특정 범위 (예, 0~1)로 모든 데이터를 변환

3) Box-Cox: 여러 k 값 중 가장 작은 SSE 선택

4) Robust_scale: median, interquartile range 사용 (outlier 영향 최소화)

Standard

```
from scipy.stats import boxcox
```

변수별 scaling 적용

```
df['X_scale'] = preprocessing.scale(df['X'])
```

```
df['X_minmax_scale'] = preprocessing.MinMaxScaler(df['X'])
```

```
df['X_boxcox'] = preprocessing.scale(boxcox(df['X']+1)[0])
```

```
df['X_robust_scale'] = preprocessing.robust_scale(df['X'])
```

데이터 프레임 전체에 scaling 적용

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import MinMaxScaler
```


1. XGBoost(eXtra Gradient Boost)의 개요

트리 기반의 알고리즘의 앙상블 학습에서 각광받는 알고리즘 중 하나입니다.

GBM에 기반하고 있지만, GBM의 단점인 느린 수행시간, 과적합 규제 등을 해결한 알고리즘입니다.

XGBoost의 주요장점

- (1) 뛰어난 예측 성능
- (2) GBM 대비 빠른 수행 시간
- (3) 과적합 규제(Overfitting Regularization)
- (4) Tree pruning(트리 가지치기) : 긍정 이득이 없는 분할을 가지치기해서 분할 수를 줄임
- (5) 자체 내장된 교차 검증
 - 반복 수행시마다 내부적으로 교차검증을 수행해 최적화된 반복 수행횟수를 가질 수 있음
 - 지정된 반복횟수가 아니라 교차검증을 통해 평가 데이터셋의 평가 값이 최적화되면 반복을 중간에 멈출 수 있는 기능이 있음
- (6) 결손값 자체 처리

XGBoost는 독자적인 XGBoost 모듈과 사이킷런 프레임워크 기반의 모듈이 존재합니다.

독자적인 모듈은 고유의 API와 하이퍼파라미터를 사용하지만, 사이킷런 기반 모듈에서는 다른 Estimator와 동일한 사용법을 가지고 있습니다.

2. XGBoost의 하이퍼 파라미터

일반 파라미터

: 일반적으로 실행 시 스레드의 개수나 silent 모드 등의 선택을 위한 파라미터, default 값을 바꾸는 일은 거의 없음

파라미터 명	설명
booster	- gbtrees(tree based model) 또는 gblinear(linear model) 중 선택 - Default = 'gbtree'
silent	- Default = 1 - 출력 메시지를 나타내고 싶지 않을 경우 1로 설정
nthread	- CPU 실행 스레드 개수 조정 - Default는 전체 다 사용하는 것 - 멀티코어/스레드 CPU 시스템에서 일부CPU만 사용할 때 변경

주요 부스터 파라미터

: 트리 최적화, 부스팅, regularization 등과 관련된 파라미터를 지칭

파라미터 명 (파이썬 래퍼)	파라미터명 (사이킷런 래퍼)	설명
eta (0.3)	learning rate (0.1)	0.05 ~ 0.1 - GBM의 learning rate와 같은 파라미터 - 범위: 0 ~ 1
num_boost_round (10)	n_estimators (100)	- 생성할 weak learner의 수
min_child_weight (1)	min_child_weight (1)	- GBM의 min_samples_leaf와 유사 - 관측치에 대한 가중치 합의 최소를 말하지만 GBM에서는 관측치 수에 대한 최소를 의미 - 과적합 조절 용도 - 범위: 0 ~ ∞
gamma (0)	min_split_loss (0)	- 리프노드의 추가분할을 결정할 최소손실 감소값 - 해당값보다 손실이 크게 감소할 때 분리 - 값이 클수록 과적합 감소효과 - 범위: 0 ~ ∞
max_depth (6)	max_depth (3)	- 트리 기반 알고리즘의 max_depth와 동일 - 0을 지정하면 깊이의 제한이 없음 - 너무 크면 과적합(통상 3~10정도 적용) - 범위: 0 ~ ∞

~~sub_sample~~
(1)

subsample
(1)

- GBM의 subsample과 동일
- 데이터 샘플링 비율 지정(과적합 제어)
- 일반적으로 0.5~1 사이의 값을 사용
- 범위: 0 ~ 1

colsample_bytree
(1)

colsample_bytree
(1)

- GBM의 max_features와 유사
- 트리 생성에 필요한 피처의 샘플링에 사용
- 피처가 많을 때 과적합 조절에 사용
- 범위: 0 ~ 1

lambda
(1)

reg_lambda
(1)

- L2 Regularization 적용 값
- 피처 개수가 많을 때 적용을 검토
- 클수록 과적합 감소 효과

alpha
(0)

reg_alpha
(0)

- L1 Regularization 적용 값
- 피처 개수가 많을 때 적용을 검토
- 클수록 과적합 감소 효과

scale_pos_weight
(1)

scale_pos_weight
(1)

- 불균형 데이터셋의 균형을 유지

max leaves, rate_drop,
XGB default mode(0)

default값

학습 태스크 파라미터

: 학습 수행 시의 객체함수, 평가를 위한 지표 등을 설정하는 파라미터

파라미터 명	설명
objective	<ul style="list-style-type: none">- 'reg:linear' : 회귀- binary:logistic : 이진분류- multi:softmax : 다중분류, 클래스 반환- multi:softprob : 다중분류, 확률반환
eval_metric	<ul style="list-style-type: none">- 검증에 사용되는 함수정의- 회귀 분석인 경우 'rmse'를, 클래스 분류 문제인 경우 'error' <hr/> <ul style="list-style-type: none">- rmse : Root Mean Squared Error- mae : mean absolute error- logloss : Negative log-likelihood- error : binary classification error rate- merror : multiclass classification error rate- mlogloss: Multiclass logloss- auc: Area Under Curve

과적합 제어

- eta 값을 낮춥니다.(0.01 ~ 0.1) → eta 값을 낮추면 num_boost_round(n_estimator)를 반대로 높여주어야 합니다.
- max_depth 값을 낮춥니다.
- min_child_weight 값을 높입니다.
- gamma 값을 높입니다.
- subsample과 colsample_bytree를 낮춥니다.

※ Early Stopping 기능 :

GBM의 경우 n_estimators에 지정된 횟수만큼 학습을 끝까지 수행하지만, XGB의 경우 오류가 더 이상 개선되지 않으면 수행을 중지

n_estimators 를 200으로 설정하고, 조기 중단 파라미터 값을 50으로 설정하면, 1부터 200회까지 부스팅을 반복하다가

50회를 반복하는 동안 학습오류가 감소하지 않으면 더 이상 부스팅을 진행하지 않고 종료합니다.

(가령 100회에서 학습오류 값이 0.8인데 101~150회 반복하는 동안 예측 오류가 0.8보다 작은 값이 하나도 없으면 부스팅을 종료)

validation set 있어야

파이썬 래퍼의 교차 검증 수행 및 최적 파라미터 구하기

: xgboost는 사이킷런의 GridSearchCV와 유사하게 **cv()**를 API로 제공합니다.

```
xgb.cv(params, dtrain, num_boost_round=10, nfold=3, stratified=False,
        folds=None, metrics=(), obj=None, feval=None, maximize=False,
        early_stopping_rounds=None, fpreproc=None, as_pandas=True,
        verbose_eval=None, show_stdv=True, seed=0, callbacks=None, shuffle=True)
```

- **params(dict)**: 부스터 파라미터
- **dtrain(DMatrix)**: 학습 데이터
- **num_boost_round(int)**: 부스팅 반복횟수
- **nfold(int)**: CV폴드 개수
- **stratified(bool)**: CV수행시 샘플을 균등하게 추출할지 여부
- **metrics(string or list of strings)**: CV 수행시 모니터링할 성능 평가 지표
- **early_stopping_rounds(int)**: 조기중단을 활성화시킴. 반복횟수 지정

xgb.cv의 반환 값은 데이터프레임 형태입니다.

