



# Code\_Review

데이터: <https://www.kaggle.com/c/house-prices-advanced-regression-techniques/>

코드: <https://dining-developer.tistory.com/23>

수업시간에도 다루었고, 자주 쓰이는 모델인 LightGBM, XGBoost, RandomForest, 그리고 GradientBoost과 앙상블 기법을 사용하였습니다.

모델의 일반성을 위해 모델을 돌릴때, cvfold를 사용하였습니다.

```
#Validation function
n_folds = 5

def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(X_train.values)
    rmse= np.sqrt(-cross_val_score(model, X_train.values, y_train, scoring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import xgboost as xgb
import lightgbm as lgb

model_gb = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05,
                                     max_depth=4, max_features='sqrt',
                                     min_samples_leaf=15, min_samples_split=10,
                                     loss='huber')
model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                             learning_rate=0.05, max_depth=3,
                             min_child_weight=1.7817, n_estimators=2200,
                             reg_alpha=0.4640, reg_lambda=0.8571,
                             subsample=0.5213, silent=1,
                             random_state =7, nthread = -1)

model_lgb = lgb.LGBMRegressor(objective='regression',num_leaves=5,
                              learning_rate=0.05, n_estimators=720,
                              max_bin = 55, bagging_fraction = 0.8,
                              bagging_freq = 5, feature_fraction = 0.2319,
                              feature_fraction_seed=9, bagging_seed=9,
                              min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)

model_rf = RandomForestRegressor(n_estimators=3000,
                                max_depth=4, max_features='sqrt',
                                min_samples_leaf=15, min_samples_split=10)
```

결과적으로 xgboost 만을 사용하였을 때 상위 22% 정도에 랭크되었음을 확인할 수 있습니다.


Randomforest의 경우 성능이 다른 모델들에 비하여 현저히 떨어짐을 볼 수 있었습니다.

Note. 해당 파라미터들은 모두 파라미터 튜닝은 되지 않은 상황입니다. 해당 분석 및 모델을 개발을 진행한 분은 파라미터 튜닝을 적극적으로는 하지 않은 듯 한데, optuna로서 해결할 수 있을 것입니다.

기본적으로 parameter tuning에 있어서는 gridsearchcv, randomsearchcv 등을 많이 쓰고, 최근에는 optuna 기반으로 자동화된 parameter tuning을 많이 시도합니다. 이를 통해 성능 향상을 도모해볼 수 있습니다.

## Optuna

optuna.trial.Trial — Optuna 3.1.0 documentation

 <https://optuna.readthedocs.io/en/stable/reference/generated/optuna.trial.Trial.html>

효율적인 최적화 알고리즘을 탑재하고 있으며, GridSampling, RandomSampling 또한 모두 지원합니다.

효율적인 작업을 위한 병렬화가 가능하며, 시각화 툴 또한 제공합니다.

4가지 정도의 모델에 대하여 평가를 해보았을 때, 이러한 boosting 기반의 ensemble 모델들을 또한 ensemble 시킨 모델을 통해 결과 향상을 꾀하였습니다.

```
model_gb.fit(X_train, y_train)
model_xgb.fit(X_train, y_train)
model_lgb.fit(X_train, y_train)

pred_gb = model_gb.predict(X_test)
pred_xgb = model_xgb.predict(X_test)
pred_lgb = model_lgb.predict(X_test)

total_weight = (1. / gb_score) + (1. / xgb_score) + (1. / lgb_score)
pred = (pred_gb * (1. / gb_score) + pred_xgb * (1. / xgb_score) + pred_lgb * (1. / lgb_score)) / total_weight
pd_test_pred = pd.DataFrame({'Id': Id_test, 'SalePrice': np.expml(pred)})
pd_test_pred.to_csv('submission.csv', index=False)
```

총합이 1이 되도록 모델마다 weight를 정해준 모습을 볼 수 있습니다. 이 때 가장 잘 나온 gradient boosting, xgboost, lightgbm의 점수의 역수만큼을 total weight로 정해주고, 각 모델별 prediction의 가중치에 해당 모델의 prediction 값 자체를 곱해주어 sum 해줌으로서 최종 prediction을 만들어 submit한 모습을 볼 수 있습니다.

여기까지는 상위 13% 정도에 랭크되게 됩니다.

Boosting, Bagging은 기본적으로 같은 알고리즘끼리의 앙상블입니다.

하지만 다른 알고리즘끼리의 앙상블 또한 시도해볼 수 있습니다. 해당 방법에는 stacking과 blending이 있습니다.

## Stacking

```
from mlxtend.regressor import StackingCVRegressor
# 6개 모델 위에 xgboost 모델을 하나 더올려 stacking을 한다.
stack = StackingCVRegressor(regressors=(gb, xgboost, lightgb, lasso, ridge),#svr
                             meta_regressor=xgboost,
                             use_features_in_secondary=True,
                             n_jobs=-1)

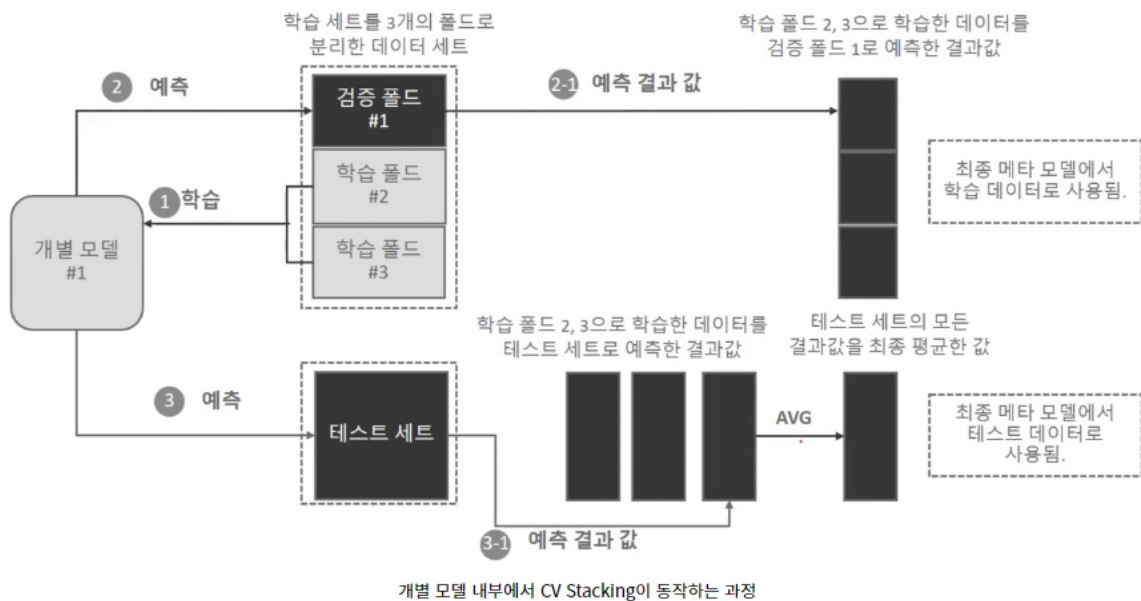
score = rmsle_cv(stack)
print("stack score: {:.4f} ({:.4f})".format(score.mean(), score.std()))
```

출력 결과)

```
stack score: 0.1115 (0.0158)
```

위에서 meta\_regressor은 가장 좋았던 알고리즘을 기반으로 다른 모델을 stacking하고자 하여 xgboost를 옵션으로 준 것입니다.

Stacking은 보팅처럼 간단한 함수를 사용하는 대신 이러한 여러 예측기의 예측을 입력으로 받아들여 이들을 취합하는 모델을 훈련하는 것이 기본적인 아이디어로 차용하고 있습니다.



kfoldcv의 기본적인 아이디어는 차용을 하되, 각 fold가 한번은 validation set, k-1번은 training set으로 작동합니다.

즉 하나의 fold에 대해서 완벽하게 iteration은 돌아갑니다.

각각의 검증 폴드에 나온 예측결과값들끼리 stacking하고, 원본 test set을 예측했을 때 나온 k번의 결과값들을 average를 취해줌으로서 stacking을 수행하는 셈입니다.

## Blending

```
[22]:
gb_model = gb.fit(X_train, y_train)
xgboost_model = xgboost.fit(X_train, y_train)
lightgb_model = lightgb.fit(X_train, y_train)
svr_model = svr.fit(X_train, y_train)
lasso_model = lasso.fit(X_train, y_train)
ridge_model = ridge.fit(X_train, y_train)

def blended_predictions(X):
    return ((0.05 * lasso_model.predict(X)) + \
            (0.05 * ridge_model.predict(X)) + \
            (0.05 * svr_model.predict(X)) + \
            (0.1 * gb_model.predict(X)) + \
            (0.15 * xgboost_model.predict(X)) + \
            (0.1 * lightgb_model.predict(X)) + \
            (0.5 * stack_model.predict(np.array(X))))

pred = blended_predictions(X_test)
pd_test_pred = pd.DataFrame({'Id': Id_test, 'SalePrice': np.expml(pred)})
pd_test_pred.to_csv('submission.csv', index=False)
```

해당 블로그를 쓴 분의 코드를 보고 가장 이해가 가지 않았던 부분입니다.

해당 모델 별로 가중치를 정해서 결국 return을 할 때, 각 모델의 return 값에 정한 가중치를 곱해줌으로서 최종적인 값을 prediction 해주는 과정입니다. 즉 stacking에서는  $cv = k$  에서  $k$  번의 iteration을 돌린 후  $k$ 로 나눠주어 평균값을 사용하였다면, 위 코드에서 사용한 코드의 경우에는 가중치를 곱해주어 가중평균을 사용하였다는 것입니다.

하지만 해당 코드를 리뷰하면서, 어떠한 근거로서 해당 가중치를 선정하였는지는 의문이었습니다.

kaggle에 코드를 제출을 하고, 점수를 높이는 방향으로 가중치를 조정하여 하이퍼파라미터 튜닝을 진행한다면 이 또한 data leakage에 해당하는 부분이고,  $cv$ 를 사용하였음에도 불구하고 generalized model로서의 타당성을 잃게 됩니다.