CS376 Term Project

Team 1
2017XXXX 유재민 2017XXXX 조영인
2017XXXX 최재민 2018XXXX 이종서

January 16, 2020

1 Model Descriptions

1.1 Data Processing

1.1.1 Label Feature

먼저, label 데이터의 빈도 분포를 normal 분포를 따르도록 변형하였다. linear model이 normal 분포를 따르는 데이터에 대해 높은 성능을 보이기 때문에 label 데이터의 빈도 분포를 normal 분포에 가깝게 변형하고자 했다. 주어진 label 데이터의 빈도 분포가 log normal 분포와 비슷하다는 점을 확인하였고, 이에 따라 label 데이터를 $\ln(1+x)$으로 맵핑을 한 값을 가지고 model을 fitting 하였다.

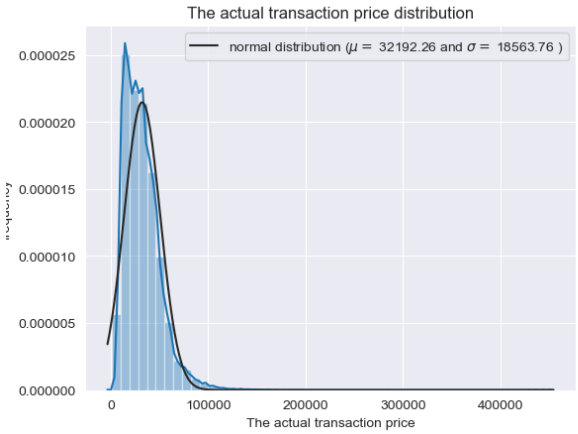


Figure 1: Before

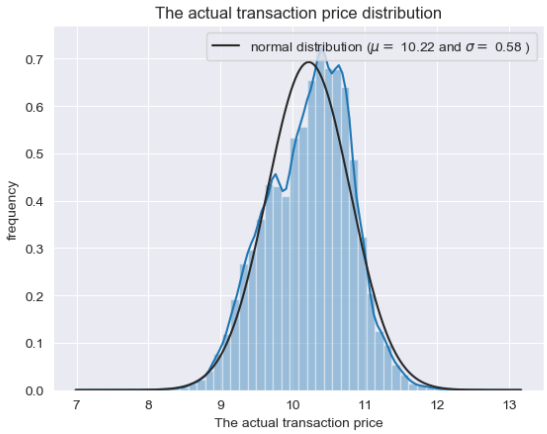


Figure 2: After

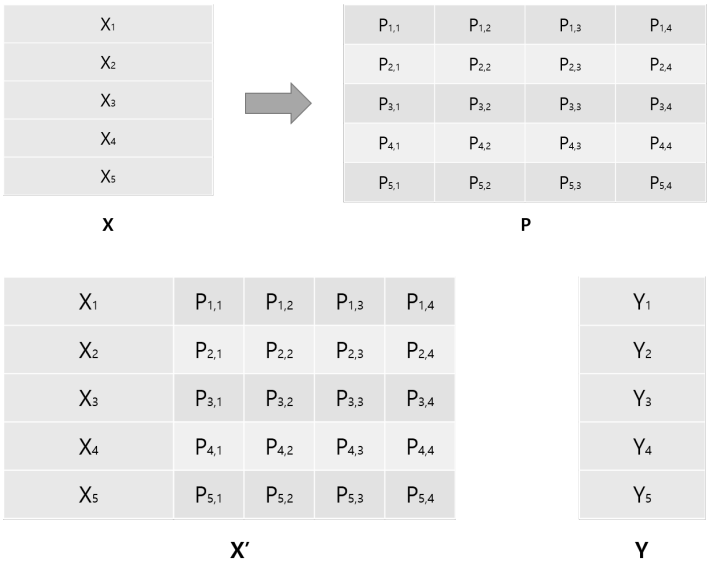
1.1.2 Missing Data & Type Converting

결실된 데이터는 모두 0으로 채웠다. 날짜형 feature (0,18,19번)의 데이터는 data_train.csv에서 가장 오래된 해의 1월 1일로부터 지난 날 수로 바꿨다.

1.2 Modeling

4개의 base-model을 이용하여 데이터를 학습하고, 각 모델의 예측결과와 데이터를 가지고 최종적으로 결과를 예측하는 방법을 사용했다. 최종적으로 결과를 예측할 때에는 앞의 base-model과 다른 모델(meta-model)을 사용하였다. Base-model로 ElasticNet, GradientBoostingRegressor, XGBoost, LightGBM을 사용했고, meta-model로 Lasso를 사용했다.

Base-model에서 XGBoost, LightGBM과 scikit-learn의 GradientBoostingRegressor는 gradient boosting을 사용하는 regressor이다. scikit-learn의 ElasticNet은 L_1, L_2를 regularizer로 사용하는 선형 regressor이다. scikit-learn의 lasso는 L_1을 regularizer로 사용하는 선형 regressor이다. lasso regressor는 over-fitting을 줄여주고 feature 선택에 도움을 준다.



위 그림에서 행렬 X의 각 행은 하나의 아파트 데이터와 대응되고 열은 feature와 대응된다. 그리고 5-fold cross validation을 하기 위해 X의 행을 5개의 묶음으로 묶었으며, 이를 $X_1, \cdots, X_5$라고 두었다. 행렬 Y는 X에 대응되는 label 데이터이다.

학습 단계에서 P는 각각의 base-model에 5-fold validation을 적용하여 만들어진다. $P_{i,j}$는 j번 base-model을 X_i를 제외한 데이터와 그에 대응되는 Y로 학습한 뒤 X_i를 이용하여 예측한 결과이다. 이때 한 base-model마다 5개의 서로 다른 모델이 만들어지는데, 이를 모두 저장해 놓는다. 다시 말해, 5-fold validation으로 만들어진 5개의 모델 전부를 사용하는 것이다. $P_{i,j}$를 생성할 때 만들어진 모델을 $M_{i,j}$라고 하겠다. 앞에서 구한 P행렬을 X에 붙여서 새로운 데이터셋 X'을 만들고 X'을 feature, Y를 label 데이터로 사용하여 meta-model을 학습한다.

예측 단계에서 x(편의상 하나의 아파트 데이터라고 가정)가 주어지면 각 $i = 1, \cdots, 4$에 대해 5개의 모델 $M_{1,i}, \cdots, M_{5,i}$이 x에 대해 예측한 값의 평균을 구한다. 모든 i에 대해 실행하면 4개의 값이 나올 것이고, 이 4개의 값과 x에 대해 meta-model이 예측을 한다. 각 예측값 y를 $e^y - 1$에 맵핑한 값이 최종 결과가 된다.

2 Unique Methods

우리는 모델의 성능을 높이기 위해 unique method로써 **특별한 ensemble 기법**을 적용하였다. 여러가지 base-model이 주어졌을 때, 각 base-model은 5-fold validation을 적용하여 앞에 설명한 P 행렬을 만든다. 그 다음, 데이터(X)와 예측한 결과(P)를 가지고 meta-model을 학습시킨다. 이렇게 meta-model을 학습시키는 과정이 우리의 unique method이다. 이 방법을 사용하면 meta-model이 각 base-model이 예측한 값을 보고 최종결과를 예측할 수 있을 뿐만 아니라 주어진 데이터의 형태도 같이 보고 예측을 할 수 있어서 더 정확한 예측이 가능하다. 예를 들어서, 특정 데이터 분포에서 1번 모델이 더 예측을 잘 한다면 1번 모델에 가중치를 더 부여하는 식의 학습이 가능한 것이다.

위에서 사용된 Ensemble 기법을 unique method로 정했을 때, unique method를 제거했을 경우 여러 개의 base-model 중 어떤 모델을 기본 모델로 설정할 지에는 모호한 부분이 있다. 따라서, 기본 모델에서는 여러 예측값을 단순히 평균을 내기로 하였다. 즉, 기본 모델에서는 4개의 base-model이 예측한 값을 평균내서 아파트 가격을 예측한다.

3 Libraries

datetime, time 등 파이썬의 내장모듈은 따로 기술하지 않았으며, 파이썬의 버전은 3.7.1이다.

3.1 Pandas

version: 0.23.4
Purpose: 효율적인 데이터 관리 및 저장

3.2 NumPy

Version: 1.15.4
Purpose: Numerical Computation

3.3 XGBoost

Version: 0.81
Purpose: base-model 3

3.4 LightGBM

Version: 2.2.2
Purpose: base-model 4

3.5 Scikit-learn

Version: 0.20.1
Purpose: ElasticNet(base-model 1), GradientBoostingRegressor(base-model 2), Lasso(meta-model)

4 Source codes

4.1 Model Class

AverageModel은 기본 모델에서 예측 값의 평균을 구해주는 클래스이고 StackedModel은 unique method를 적용해주는 클래스이다. Model 클래스는 이 둘을 종합하여 사용자가 쉽게 두 모델을 사용할 수 있게 해주는 클래스이다.

Model 클래스에서 생성자의 파라미터로는 unique, load_dir이 있다. load_dir로 문자열을 넘겨주면 해당 문자열 위치에 있는 모델을 불러온다. 만약 load_dir이 None이면 unique의 값에 따라 모델을 생성하게 된다. unique가 True이면 unique method를 적용한 모델을 사용하게 되고 False이면 기본 모델을 사용하게 된다.

Model 클래스가 제공하는 함수와 기능은 다음과 같다.

- fit(X,y): feature 데이터 X와 label 데이터 y를 이용해 학습한다.
- predict(X): feature 데이터 X를 이용해 예측한다.
- evaluate(test_x,test_y): feature 데이터 test_X와 label 데이터 test_y를 이용해 performance를 계산하고 출력한다.
- save(save_dir): save_dir 위치에 모델을 저장한다.

Reference : Stacked Regressions to predict House Prices from Kaggle

```
class AverageModel(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models

    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]
        i = 0
        for model in self.models_:
            print(i, 'fit...')
            i += 1
            model.fit(X, y)
        return self

    def predict(self, X):
        print('predict...')
        predictions = np.column_stack([
            model.predict(X) for model in self.models_])
        return np.mean(predictions, axis=1)
```

```
class StackedModel(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, shuffle=True, random_state=42)

        out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            print(i,'fit...')
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                pred_y = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = pred_y

        new_X = np.concatenate((X[:,SECOND_LAYER_INDEX],out_of_fold_predictions),axis=1)
        self.meta_model_.fit(new_X, y)
        return self

    def predict(self, X):
        print('predict...')
        meta_features = np.column_stack([
            np.column_stack([model.predict(X) for model in base_models]).mean(axis=1)
            for base_models in self.base_models_])
        new_X = np.concatenate((X[:,SECOND_LAYER_INDEX],meta_features),axis=1)
        return self.meta_model_.predict(new_X)
```

```
class Model(object):
    def __init__(self, unique = True, load_dir = None):
        if load_dir:
            self.model = joblib.load(load_dir)
            return

        # base models
        ENet = ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
                          max_iter=1000, normalize=False, positive=False, precompute=False,
                          random_state=42, selection='cyclic', tol=0.0001, warm_start=False)
        GBoost = GradientBoostingRegressor(n_estimators=500, learning_rate=0.05,
                                           max_depth=4, max_features='sqrt',
                                           min_samples_leaf=15, min_samples_split=10,
                                           loss='huber', random_state=5)
        XGB = xgb.XGBRegressor(colsample_bytree=0.4400, gamma=0.05,
                               learning_rate=0.3, max_depth=4,
                               min_child_weight=2, n_estimators=2200,
                               reg_alpha=0.48, reg_lambda=0.85,
                               subsample=0.5213, silent=True,
                               random_state=7, nthread=-1)
        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=0.22,
```

```
                                feature_fraction_seed=9, bagging_seed=9,
                                min_data_in_leaf=6, min_sum_hessian_in_leaf=11)
        if unique:
            # meta model
            lasso = make_pipeline(RobustScaler(), Lasso(alpha=0.0005, random_state=42))
            self.model = StackedModel(base_models=(ENet, GBoost, XGB, LGB), meta_model=lasso)
        else:
            self.model = AverageModel(models = (ENet,GBoost,XGB,LGB))

    def fit(self, X, y):
        return self.model.fit(X,np.log1p(y))

    def predict(self, X):
        return np.exp(self.model.predict(X)) - 1.

    def evaluate(self, test_x, test_y):
        pred_y = self.predict(test_x)
        performance = 1. - np.mean(np.absolute(test_y - pred_y) / test_y)
        print("performance:", performance)

    def save(self, save_dir):
        joblib.dump(self.model, save_dir)
```

4.2 train & test function

쉽게 train과 test를 할 수 있도록 세 개의 함수를 제공하였다.

- test_by_train_data(unique, using_ratio, training_ratio)
  data_train.csv에서 train과 test(evaluation)을 모두 진행하는 함수이다. 이 과정에서 별도의 파일이 저장되지 않는다. unique가 True이면 unique method를 적용한 모델을 사용하고 False이면 기본 모델을 사용한다. using_ratio와 training_ratio는 0과 1사이의 수를 가진다. using_ratio는 전체 셋 중에서 train과 test에 사용될 데이터의 비율이고 training_ratio는 using_ratio만큼 뽑은 데이터 셋 중에서 train에 사용될 데이터의 비율이다.

- train(unique)
  data_train.csv의 데이터를 train하고 모델을 저장하는 함수이다. unique가 True인 경우, unique method를 적용한 모델을 사용하고 학습한 모델을 unique.pkl에 저장한다. False인 경우, 기본 모델을 사용하고 학습한 모델을 base.pkl에 저장한다.

- test(unique)
  저장된 모델을 불러오고 data_test.csv의 데이터로 test하는 함수이다. unique가 True인 경우, unique.pkl에서 모델을 불러오며 예측한 아파트 값을 unique.csv에 저장한다. False인 경우, base.pkl에서 모델을 불러오며 예측한 아파트 값을 base.csv에 저장한다.

```
def test_by_train_data(unique=True, using_ratio = 1.0, training_ratio = 0.9):
    data = shuffle(get_train_data())
    tot_sz = int(data.shape[0]*using_ratio)
    train_sz = int(tot_sz*training_ratio)
    data = data[:tot_sz]
    data = data[data[:, 0].argsort()]

    train = data[:train_sz]
    test = data[train_sz:]
    train_x, train_y = train[:,:-1], train[:,-1]
    test_x, test_y = test[:,:-1], test[:,-1]

    model = Model(unique)
    model.fit(train_x, train_y)
```

```
    model.evaluate(test_x, test_y)

def train(unique=True):
    if unique:
        filename = 'unique'
    else:
        filename = 'base'

    model = Model(unique)
    train = get_train_data()
    train_x, train_y = train[:,:-1], train[:,-1]

    # fit model
    model.fit(train_x, train_y)
    print('model fitted')

    # save model
    model.save(filename+'.pkl')
    print('model saved')

def test(unique=True):
    if unique:
        filename = 'unique'
    else:
        filename = 'base'

    model = Model(unique,filename+'.pkl')
    test = get_test_data()

    # predict
    res = model.predict(test)

    # save result
    with open(filename+'.csv', 'w') as fout:
        for it in res:
            fout.write(str(it) + '\n')
```

5 Performance

학습과 테스트는 다음과 같은 환경에서 진행했다.

- OS: Windows 10 64bit
- Processor: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz
- RAM: 16.0 GB

모델을 평가할 때, data_test.csv에서 주어진 모든 데이터를 contract date로 정렬한 뒤 앞의 90%를 training set, 뒤의 10%를 validation set으로 사용하였다. data_test.csv에서 data_train.csv보다 미래 데이터가 많기 때문에 이러한 방법으로 성능을 체크하였다.
data_train.csv에서 모델 평가시 기본 모델과 unique method를 적용한 모델의 performance와 계산 시간은 각각 다음과 같다.

- Base: 0.8975 (약 200초)
- Unique: 0.9452 (약 750초)

data_train.csv로 모델을 학습시킬 때 계산 시간은 다음과 같다.

- Base: 약 170초

- Unique: 약 740초

data_test.csv로 학습된 모델을 테스트할 때 계산 시간은 다음과 같다.

- Base: 약 0.5초
- Unique: 약 1.5초

위와 같이 unique method를 적용한 결과 performance가 0.9452로 기본 모델에 대해 약 5% 향상되었다.