Final Project of Quantecon

Yalin Liu

June 8, 2023

Contents

Dataset

Introduction

- I spilt the dataset into 10 subsets by the city.
- This is because different cities tend to have different factors which are invisible.
- Create new variables through 'amenities' feature.
- Using ColumnTransformer to handle categorical variables.
- Through Google map, maybe we can define more categorical variables.

Introduction

In [54]:

```
# Encoding categorical features

transformer = ColumnTransformer([
    ('One Hot', OneHotEncoder(drop='first'), ['neighbourhood_group', 'room_type']),
    ('Binary', ce.BinaryEncoder(), ['neighbourhood'])
], remainder='passthrough')
```

Example:Hong Kong

In [32]:

```
HK.copy=HK
X=HK.drop(['amenities','longitude','latitude','city','price','room_type'],axis=1)
y=HK['price']
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=111)
```

In [33]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3772 entries, 9244 to 268762
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   host_is_superhost            3772 non-null   float64
 1   host_has_profile_pic         3772 non-null   float64
 2   host_identity_verified       3772 non-null   float64
 3   accommodates                 3772 non-null   int64
 4   bedrooms                     3772 non-null   float64
 5   review_scores_rating         3772 non-null   float64
 6   review_scores_accuracy       3772 non-null   float64
 7   review_scores_cleanliness    3772 non-null   float64
 8   review_scores_checkin        3772 non-null   float64
 9   review_scores_communication  3772 non-null   float64
 10  review_scores_location       3772 non-null   float64
 11  review_scores_value          3772 non-null   float64
 12  instant_bookable             3772 non-null   int64
 13  Entire place                 3772 non-null   uint8
 14  Hotel room                   3772 non-null   uint8
 15  Private room                 3772 non-null   uint8
 16  Shared room                  3772 non-null   uint8
dtypes: float64(11), int64(2), uint8(4)
memory usage: 427.3 KB
```

Models

OLS

- Serves as the baseline model.
- It may exists the problem of overfit.

OLS

```
#baseline 需要进一步对数据进行处理
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(X_train,y_train)
y_pred=reg.predict(X_test)
print("Coefficients: \n", reg.coef_)
print("Mean squared error: %.2f\n" % mean_squared_error(y_test, y_pred))
y_pred=reg.predict(X_train)
print("Mean squared error: %.2f\n" % mean_squared_error(y_train, y_pred))
```

```
Coefficients:
 [-8.58703951e+00  1.40810313e+02 -9.50472837e+00  6.21455917e+01
  3.52025946e+02  1.40227260e+02 -7.49295317e+01  4.86927775e+01
  8.87018198e+01 -7.83583805e+01  8.66776198e+00 -6.10255374e+01
  5.69284038e+01 -2.03156397e+15 -2.03156397e+15 -2.03156397e+15
 -2.03156397e+15]
Mean squared error: 1581285.09

Mean squared error: 3698527.64
```

OLS

```
result = permutation_importance(reg, X_train, y_train, n_repeats=10, random_state=111)
print(result['importances_mean']/result['importances_mean'].sum())
print(result.importances_std)
```

```
[ 1.45472005e-29 -3.90821045e-30  5.45735952e-29  8.87483221e-27
  2.97142225e-26  1.15572685e-26  4.56234827e-27  1.45527443e-27
  3.77779131e-27  4.59452028e-27 -1.15220088e-29  3.01900193e-27
  2.17696319e-28  4.18617900e-01  6.77377087e-02  4.26251923e-01
  8.73924685e-02]
[5.07199345e-05 3.81612755e-05 7.90721919e-05 2.89723164e-03
 5.00467957e-03 2.51501569e-03 1.14501987e-03 8.24247360e-04
 1.27834817e-03 1.10881389e-03 9.79333368e-05 1.02571612e-03
 4.74785145e-04 8.71269686e+21 1.98690864e+21 6.04809251e+21
 1.33276730e+21]
```

LASSO

Using GridSearchCV to get the best hyperparameter, again we analyse the importance of different features.

LASSO

LASSO

In [38]:
```
clf = linear_model.Lasso(alpha=1.0)
clf.fit(X_train, y_train)
result = permutation_importance(clf, X_train, y_train, n_repeats=10,random_state=111)
result['importances_mean']/result['importances_mean'].sum()
```

Out[38]:
```
array([ 1.20404843e-04,  0.00000000e+00,  3.97131691e-04,  1.19904276e-01,
        4.02580948e-01,  1.47213475e-01,  5.70993151e-02,  1.77505647e-02,
        4.68008266e-02,  5.73138648e-02, -1.67339312e-04,  3.60617009e-02,
        2.89900293e-03,  5.66509240e-02,  4.22766031e-02,  4.74845962e-05,
        1.30508175e-02])
```

In [39]:
```
alpha_range = np.logspace(-6, 6, 13)
param_grid = {'alpha': alpha_range}
grid_search = GridSearchCV(clf, param_grid=param_grid, cv=6, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train, y_train)
best_alpha_lasso = grid_search.best_params_['alpha']
clf = linear_model.Lasso(alpha=best_alpha_lasso)
lasso_score = -1.0 * cross_val_score(clf, X_train, y_train, cv=6, scoring='neg_root_mean_squared_error').mean()
lasso_score
```

Out[39]: 1751.7979783357844

Random Forest

Using Optuna to get the best hyperparameter, again we analyse the importance of different features.

Random Forest:

```
import optuna
from sklearn.ensemble import RandomForestRegressor

def objective(trial):
    n_estimators = trial.suggest_int("n_estimators", 50, 400)
    clf = RandomForestRegressor(n_estimators=n_estimators, random_state=111)
    scores = cross_validate(clf, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
    rmse=scores['test_score'].mean()
    return rmse

if __name__ == "__main__":
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=100)
    print(study.best_trial)
```

XGBoost

Using BayesSearchCV to get the best hyperparameter.

```python
from xgboost import XGBRegressor
estimator = XGBRegressor()

fit_params = {
    'early_stopping_rounds': 10,
    'eval_set':[(X_train, y_train)],
    'verbose': False,
}

search_space = {
    'max_depth': (0, 50),
    'n_estimators': (0, 1000),
    'learning_rate': (0.01, 1.0, 'log-uniform'),
    'gamma': (1e-9, 0.5, 'log-uniform'),
    'scale_pos_weight': (1e-6, 500, 'log-uniform'),
}


opt = BayesSearchCV(
    estimator=estimator,
    search_spaces=search_space,
    fit_params=fit_params,
    cv=5,
    scoring="neg_mean_squared_error",
    random_state=111,
    n_iter=3,
    verbose=1,
)
```

ANN

```python
import keras

import optuna

# 1. Define an objective function to be maximized.
def objective(trial):
    model = Sequential()

    # 2. Suggest values of the hyperparameters using a trial object.
    model.add(
        Conv2D(filters=trial.suggest_categorical('filters', [32, 64]),
               kernel_size=trial.suggest_categorical('kernel_size', [3, 5]),
               strides=trial.suggest_categorical('strides', [1, 2]),
               activation=trial.suggest_categorical('activation', ['relu', 'linear']),
               input_shape=input_shape))
    model.add(Flatten())
    model.add(Dense(CLASSES, activation='softmax'))

    # We compile our model with a sampled learning rate.
    lr = trial.suggest_float('lr', 1e-5, 1e-1, log=True)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=RMSprop(lr=lr), metrics=['accuracy'])
    return accuracy

# 3. Create a study object and optimize the objective function.
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

Dimension Reduction

PCA

```python
from sklearn.decomposition import PCA
pca = PCA(n_components='mle',svd_solver= 'full')
pca.fit_transform(X)
```

```
array([[-1.86033700e+00,  1.29128088e+00, -3.77213121e-01, ...,
        -1.79817912e-01,  2.74769947e-02, -5.01513618e-03],
       [-8.17786060e-01,  1.24206669e+00,  5.52618142e-01, ...,
        -1.16381018e-01, -7.15575971e-02,  4.82734668e-03],
       [-4.95193573e-01,  1.48852959e-01,  1.25546679e+00, ...,
        -8.18328595e-03, -2.15117652e-02, -3.87388513e-04],
       ...,
       [-2.32611096e+00, -6.02576701e-01, -3.83352489e-01, ...,
        -1.37331844e-01, -1.27181554e-03, -4.88760937e-03],
       [ 2.67988881e-01,  5.41140443e+00,  1.89909331e-01, ...,
        -6.50847068e-02, -3.22408445e-02,  1.74959715e-03],
       [-4.01702240e-01, -1.07898766e+00,  1.73712161e-01, ...,
        -1.10682938e-01, -8.02502432e-03, -4.34183628e-03]])
```

autoencoder

```
latent_dim = 28

class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Autoencoder(latent_dim)
```

```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

```
autoencoder.fit(X_train, X_train,
                epochs=10,
                shuffle=True,
                validation_data=(X_test, X_test))
```

Prospect

Further Improvement

- Some further Improvements may include:
- Also add some error analysis in the following.

Further Improvement

You've implemented regularized linear regression on housing prices

$$J(\vec{\mathrm{w}}, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{\mathrm{w}},b}\left(\vec{\mathrm{x}}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

But it makes unacceptably large errors in predictions. What do you try next?

Get more training examples
Try smaller sets of features
Try getting additional features
Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$
Try decreasing λ
Try increasing λ

Thank you!