Report Title: **House Price Prediction**

Author Name: **王小慧**

Student ID: **21307110415**

Institute: **信息科学与工程学院**

Report Date: **2024/11/25**

Contents

1 Introduction

The prediction of house prices based on various attributes is a critical task in real estate analytics and decision-making. This study aims to construct and evaluate machine learning models for predicting house prices using a dataset from Kaggle, containing diverse features. The objective is to implement and analyze different preprocessing strategies and model architectures to achieve accurate predictions. We apply several advanced machine learning and deep learning techniques such as Multi-Layer Perceptron (MLP), Principal Component Analysis (PCA) and model stacking to improve prediction accuracy and computational efficiency.

1.1 Principal Component Analysis (PCA)

PCA is a dimensionality reduction technique that transforms a dataset with correlated features into a set of linearly uncorrelated components, known as principal components. The transformation is achieved by projecting the original data onto a new coordinate system where the first principal component explains the maximum variance, followed by subsequent components with decreasing variance. Given a set of data X, the covariance matrix of the data X is computed as:

$$\mathbf{C} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X} \tag{1}$$

The eigenvectors V and eigenvalues λ of C are determined, where V represents the directions of maximum variance. The data is projected onto the principal components:

$$\mathbf{Z} = \mathbf{XV} \tag{2}$$

Here, Z represents the transformed data in the lower-dimensional space. By retaining only the top k components, PCA reduces the dimensionality while preserving the most critical information.

1.2 Model Stacking

Model stacking is an ensemble learning technique that combines multiple base models (level-0 learners) and a meta-model (level-1 learner) to improve prediction accuracy. The base models are trained independently on the dataset, and their predictions serve as input features for the meta-model, which learns to optimize the final prediction.

Let $\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \ldots, \hat{y}_i^{(m)}$ denote predictions from m base models for a sample i.These predictions are used as input to train the meta-model:

$$\hat{y}_i = f_{\text{meta}}(\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \ldots, \hat{y}_i^{(m)}) \tag{3}$$

where f_{meta} represents the meta-model, such as a linear regressor or neural network.

The final prediction $\hat{y}_i$ is a weighted combination of the base models' predictions, as learned by the meta-model.

By leveraging PCA for dimensionality reduction and model stacking for ensemble learning, this study builds a robust and efficient framework for house price prediction.

2 Experimental Setup

Before formally commencing the experiment, we explored the house price prediction task using a basic MLP model. The performance of this model was evaluated using the root mean squared error (RMSE) metric, which yielded a score of **0.17327** on the test dataset.

2.1 Dataset

- **Training Set:** Includes numerical and categorical features describing houses and their corresponding prices as target labels.
- **Testing Set:** Contains the same features but lacks price labels, used for final prediction evaluation.

2.2 Data Preprocessing

- **Feature Selection:** Excluded the Id column as it is not predictive. Retained only numerical features for simplicity and efficiency.
- **Handling Missing Values:** Imputed missing numerical data using interpolation and removed any remaining rows with missing values.
- **Feature Scaling:** Standardized all numerical features using StandardScaler to normalize their distributions, ensuring consistent feature magnitudes.

2.3 Parameter Configuration

- **Loss Function:** Mean Squared Error Loss (MSELoss)
- **Optimizer:** Adam optimizer (lr=0.004)
- **Other Hyperparameters:** Batch Size = 32, Epochs = 200

2.4 Model Architecture

A Multilayer Perceptron (MLP) was used as the initial model for predicting house prices.

```
class HousePriceModel(nn.Module):
    def __init__(self, input_dim, drop_rate=0.5):
        super(HousePriceModel, self).__init__()
        self.fc1 = nn.Linear(input_dim, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128,64)
        self.fc5 = nn.Linear(64,32)
        self.dropout = nn.Dropout(drop_rate)
        self.fc6 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = torch.relu(self.fc5(x))
        x = self.dropout(x)
        x = self.fc6(x)
        return x
```

While this result demonstrates the model's ability to capture certain underlying patterns in the data, it has notable limitations. One significant drawback of this approach is its reliance on numerical features, neglecting categorical features that are likely critical to the prediction task. So, it still needs improvement, including incorporating categorical features, enhancing feature engineering, exploring dimensionality reduction methods, and employing ensemble techniques, to boost predictive performance.

3 Experimental Procedure

This experiment was conducted using a publicly available implementation [1].

3.1 Data Cleaning

To improve model performance, significant outliers were removed from the dataset. For example, houses with a GrLivArea (above-ground living area) greater than 4000 but with a SalePrice below 300,000 were identified as anomalies and excluded. This ensured that the model would not be overly influenced by extreme values that did not follow the general trend of the data.

We also handled missing values based on data_description.txt:

- Features such as PoolQC, Alley, and Fence had missing values that were filled with "None", indicating the absence of these attributes.
- Numeric features like TotalBsmtSF and GarageArea were filled with zeros when the absence implied non-existent attributes.
- For features like LotFrontage, missing values were imputed using median values grouped by correlated features, such as Neighborhood.

3.2 Feature Engineering

3.2.1 Categorical Encoding

Categorical features were encoded to convert them into numerical values for model processing. Instead of using simple one-hot encoding, a more sophisticated approach was adopted by assigning numeric values based on the relationship between categorical feature values and the target variable (SalePrice). This method leverages the mean or median sale price for each category to assign values, improving model interpretability and performance.

Here an example of categorical encoding:

The MSSubClass feature, which represents the type of dwelling, originally had string values (like '180', '30', '45'). Then a mapping was created to assign numerical values to each unique category. This transformation is based on the mean, median and count of SalePrice for each category (count refers to how many times each category appears in the dataset).

```
full.groupby(['MSSubClass'])[['SalePrice']].agg(['mean','median','count'])

full["oMSSubClass"] = full.MSSubClass.map({
    '180': 1, '30': 2, '45': 2, '190': 3, '50': 3, '90': 3,
    '85': 4, '40': 4, '160': 4, '70': 5, '20': 5, '75': 5,
    '80': 5, '150': 5, '120': 6, '60': 6
})
```

After the mapping, the new encoded feature oMSSubClass contains numerical values that represent the same categories but in a form that can be used in machine learning models.

3.2.2 PCA

To enhance computational efficiency and address potential multi-collinearity among features, we applied Principal Component Analysis (PCA) to reduce the dimensional-

ity of the dataset. PCA transforms the original features into a new set of orthogonal components, known as principal components, which capture the maximum variance in the data. By reducing the number of features while preserving essential patterns, PCA not only simplified the dataset but also helped mitigate the risk of overfitting. This dimensionality reduction step was particularly beneficial given the large number of features generated after encoding categorical variables, as it allowed the model to focus on the most informative aspects of the data.

```
    pca = PCA(n_components=300)
    X_scaled = pca.fit_transform(X_scaled)
    test_X_scaled = pca.transform(test_X_scaled)
```

3.3 Grid Search

We implement grid search to find the best set of hyperparameters for different machine learning models. Grid search systematically evaluates multiple combinations of hyperparameter values to determine which configuration yields the best performance based on a chosen metric.

```
class grid():
    def __init__(self,model):
        self.model = model

    def grid_get(self,X,y,param_grid):
        grid_search = GridSearchCV(self.model,param_grid,cv=5, scoring="
            neg_mean_squared_error")
        grid_search.fit(X,y)
        print(grid_search.best_params_, np.sqrt(-grid_search.best_score_))
        grid_search.cv_results_['mean_test_score'] = np.sqrt(-grid_search.
            cv_results_['mean_test_score'])
        print(pd.DataFrame(grid_search.cv_results_)[['params', 'mean_test_score', '
            std_test_score']])
```

3.4 Ensemble Learning

A two-layer stacking framework was implemented:

- The first layer consisted of base models that generated new features.
- The second layer used a Kernel Ridge model to aggregate predictions from the first layer.

```python
class stacking(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self,mod,meta_model):
        self.mod = mod
        self.meta_model = meta_model
        self.kf = KFold(n_splits=5, random_state=42, shuffle=True)

    def fit(self,X,y):
        self.saved_model = [list() for i in self.mod]
        oof_train = np.zeros((X.shape[0], len(self.mod)))

        for i,model in enumerate(self.mod):
            for train_index, val_index in self.kf.split(X,y):
                renew_model = clone(model)
                renew_model.fit(X[train_index], y[train_index])
                self.saved_model[i].append(renew_model)
                oof_train[val_index,i] = renew_model.predict(X[val_index])

        self.meta_model.fit(oof_train,y)
        return self

    def predict(self,X):
        whole_test = np.column_stack([np.column_stack(model.predict(X) for model in
             single_model).mean(axis=1)
                                      for single_model in self.saved_model])
        return self.meta_model.predict(whole_test)

    def get_oof(self,X,y,test_X):
        oof = np.zeros((X.shape[0],len(self.mod)))
        test_single = np.zeros((test_X.shape[0],5))
        test_mean = np.zeros((test_X.shape[0],len(self.mod)))
        for i,model in enumerate(self.mod):
            for j, (train_index,val_index) in enumerate(self.kf.split(X,y)):
                clone_model = clone(model)
                clone_model.fit(X[train_index],y[train_index])
                oof[val_index,i] = clone_model.predict(X[val_index])
                test_single[:,j] = clone_model.predict(test_X)
            test_mean[:,i] = test_single.mean(axis=1)
        return oof, test_mean

stack_model = stacking(mod=[lasso,ridge,svr,ker,ela,bay],meta_model=ker)
```

4 Results and Analysis

4.1 Baseline

We evaluated the MLP model and achieved a score of **0.17327** on Kaggle.

4.2 Model using Stacking

The optimized model with feature engineering achieved a score of **0.1268** on the Kaggle competition leaderboard as shown in Figure 1, representing a significant improvement over the initial Multilayer Perceptron (MLP) model, which scored **0.17327**. This substantial progress highlights the effectiveness of the refined pipeline and advanced techniques employed throughout the experiment.

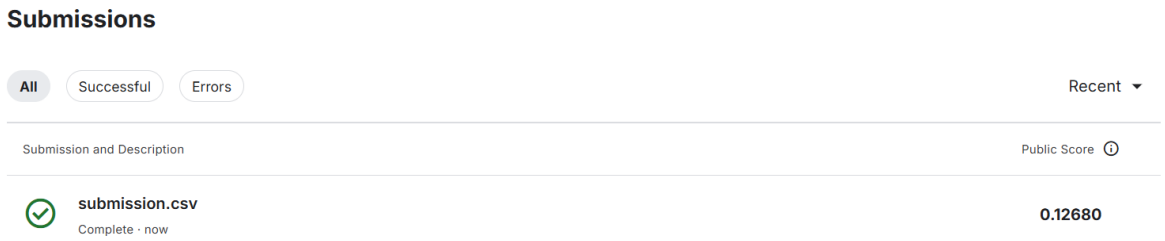

Figure 1 The Result in Kaggle

4.3 Impact of Encoding and Stacking

To further evaluate the effectiveness of categorical encoding and model stacking, we modified the model architecture to use a **MLP model** with the same architecture outlined in subsection 2.4. After applying the same preprocessing steps, including **categorical encoding and dimensionality reduction**, we achieved a score of **0.15795** on the Kaggle leaderboard. This result demonstrates the MLP model's capability when enhanced with categorical feature encoding, which allows the model to handle and learn from the non-numeric data in the dataset.

When compared to the initial model that excluded categorical features, which performed poorly, the inclusion of these features clearly led to an improvement in performance. However, the result still lags behind the performance of the model stacking approach, which achieved a lower RMSE score and placed higher on the leaderboard. The stacking model, which combines multiple base models through a meta-model, was able to capture complex relationships between the features more effectively than a single MLP model.

5 Conclusion

This experiment successfully implemented several machine learning pipelines for house price prediction. The initial MLP-based model demonstrated solid predictive capability, but it still left room for improvement, especially when it came to handling cat-

egorical features and capturing complex relationships within the data. To address these limitations, we further explored feature engineering and model stacking, which resulted in an improved score of **0.1268** on the Kaggle leaderboard.

Additionally, we conducted an experiment using feature engineering with the same MLP model to evaluate the impact of feature processing techniques. This experiment achieved a score of **0.15795**. The result underline the benefits of both feature engineering and model stacking.

In conclusion, this experiment highlights how a combination of feature engineering techniques and ensemble modeling strategies can significantly improve performance in regression tasks like house price prediction. By systematically refining the pipeline with these methods, we were able to achieve substantial improvements, underscoring their importance in tackling challenging machine learning problems.

References

[1] Massquantity. Kaggle-houseprices. https://github.com/massquantity/Kaggle-HousePrices.