Report for Mini Project 1: House Price Regression

MoeKid101

Questions (a) → Sec.1, (b) → Sec.2, (c) → Sec.3, (e)(f) → Sec.4, (g) → Sec.5, (h) → Sec.6.

1. Understanding The Dataset

It is directly visible from the CSV file that there are in total 4 features (*BldgType, OverallQual, GrLivArea, GarageArea*) provided to fit the labels *SalePrice*. Of these features and labels, *BldgType* is different from others in that this is an attribute representing category, and we have no knowledge about its relationship with *SalePrice*. Therefore, to determine a proper meaning of the attribute, we plot the relationship between *BldgType* and *SalePrice* (shown in Fig.1).

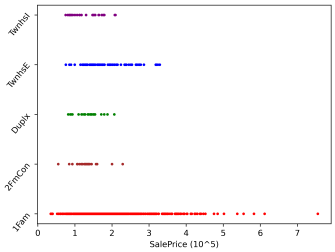

Figure 1. *SalePrice* distributions under different *BldgType* values.

There being no significant increasing relationship between the target and the attribute indicates that *BldgType* is a **nominal attribute** instead of an ordinal attribute. Consequently, in the subsequent solution, this attribute is encoded using one-hot encoding.

Detailed statistical information (including number of samples, mean and variance of each attribute) is presented in Table.1 (training set) and Table.2 (test set), from which we can see that most of the data sample belongs to *1Fam* type, and thus the mean and the variance are dominated by the samples coming from *1Fam* class.

2. Model Selection

Among the popular regression models, I chose *support vector regression* [4]. The rationale behind this choice is that the given dataset is very small in scale, and intuitively, there should be a strong linear relationship between *SalePrice* and *GrLivArea*, *GarageArea*. Furthermore, taking into consideration the dominance of *1Fam* class, a probabilistic model might suffer from class imbalance problem and fail to perform well on the minority classes. In contrast, support vector methods is robust to an uneven distribution across different classes.

Having opted for SVR, the other details regarding the models becomes apparent. As is mentioned in Sec.1, the nominal attribute *BldgType* is encoded using one-hot encoding, leading to an input of 8-dimension vector and an output of 1 real value. Then the regression problem is mathematically expressed as:

Given training set $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right) | i \in [m]\right\}$ where $\boldsymbol{x} \in \mathbb{R}^8$ and $y \in \mathbb{R}$, we solve the optimization problem

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ &\text{s.t. } y^{(i)} - \left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) \leq \epsilon + \xi_i \\ &\text{and } \left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) - y^{(i)} \leq \epsilon + \xi_i^* \\ &\text{and } \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{1}$$

To train and evaluate the model, we directly import the SVR module in sklearn. As shown in Listing.1, the model performance is evaluated through mean squared error, and the remaining problem is to solve for the optimal hyperparameters including the choice of kernel, the punishment constant C, *etc*.

```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from numpy import random

# shuffle dataset and split into training and validation set
data, label = torch.load('data_trans')
shuffle_idx = np.arange(data.shape[0])
random.shuffle(shuffle_idx)
data, label = data[shuffle_idx], label[shuffle_idx]
sepBound = int(0.7 * data.shape[0])
trainData, trainLabel = data[:sepBound], label[:sepBound]
valData, valLabel = data[sepBound:], label[sepBound:]

# train and evaluate model
model = SVR()
model.fit(trainData, trainLabel)
valPred = model.predict(valData)
valErr = mean_squared_error(valLabel, valPred)
print(f'validation: {valErr}')
```

Listing 1. SVR training and evaluation

Table 1. Statistics of the training set.

BldgType	Overall	1Fam	2FmCon	Duplx	TwnhsE	TwnhsI
Number of Samples	1000	837	23	36	75	29
OverallQual	6.125 / 1.382	6.146 / 1.400	5.000 / 0.6594	5.028 / 0.8656	6.800 / 1.095	6.034 / 1.129
GrLivArea	1510 / 512.0	1537 / 528.0	1470 / 586.6	1543 / 411.2	1308 / 288.1	1259 / 322.5
GarageArea	473.4 / 208.8	485.2 / 210.0	288.9 / 239.6	415.5 / 267.1	468.1 / 105.3	363.7 / 144.3
SalePrice	1.823 / 8.025	1.873 / 8.364	1.296 / 3.546	1.352 / 2.628	1.844 / 5.786	1.326 / 3.879

Each table item except those in "Number of Samples" row contain two parts: the average and the standard deviation value of samples belonging to the specific *BldgType*. Specially, in "SalePrice" row, average value is divided by 10^5 and standard deviation is divided by 10^4.

Table 2. Statistics of the training set.

BldgType	Overall	1Fam	2FmCon	Duplx	TwnhsE	TwnhsI
Number of Samples	259	218	3	11	21	6
OverallQual	6.081 / 1.377	6.101 / 1.420	5.667 / 1.247	5.000 / 0.603	6.571 / 1.003	5.833 / 0.6872
GrLivArea	1548 / 559.8	1563 / 578.2	2030 / 183.9	1762 / 497.9	1307 / 281.4	1223 / 341.2
GarageArea	475.5 / 218.8	478.6 / 219.5	513.3 / 152.2	475.2 / 274.4	504.9 / 142.7	240.3 / 187.6

$*$ To help the model converge faster, all the features and labels are rescaled to have similar mean values (*GrLivArea* divided by 500, *GarageArea* divided by 100, and *SalePrice* divided by 10^5).

3. Solving for the Optimal Hyperparameter

The choice of hyperparameters are simply based on the model performance on validation set. To be more specific, 50 models are trained for each combination of hyperparameter, and the average MSE is used to evaluate this set of parameters.

With well-defined evaluation for hyperparameters, grid search is carried out on three kernels: linear, polynomial and radial basis function kernel.

Linear kernel: $K\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = {\boldsymbol{x}^{(i)}}^{\top} \boldsymbol{x}^{(j)}$, therefore the only parameter is the punishment constant C. According to Fig.2, $C^*_{\text{lin}} = 0.1$.

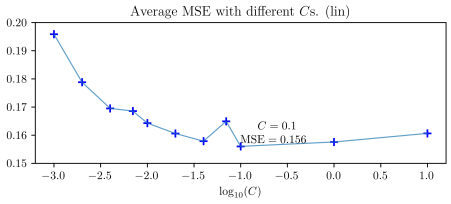

Figure 2. Performance comparison under linear kernel.

RBF kernel: $K\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \exp\left(-\gamma \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|^2\right)$. Inspired by the SMO algorithm, we can perform separate optimizations for different parameters. According to Fig.3 and Fig.4, $(\gamma^*, C^*_{\text{rbf}}) = (0.02, 1.0)$ is a local optimum for both γ and C. Consequently, $(0.02, 1.0)$ achieves local optimum for the 2-dimension optimization problem with high probability.

Poly Kernel: $K\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right) = \left({\boldsymbol{x}^{(i)}}^{\top} \boldsymbol{x}^{(j)} + \text{coef0}\right)^d$.

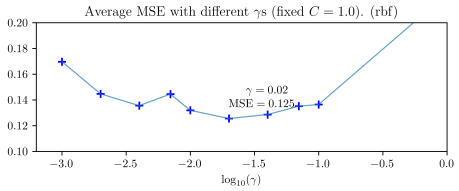

Figure 3. Performance comparison under RBF kernel.

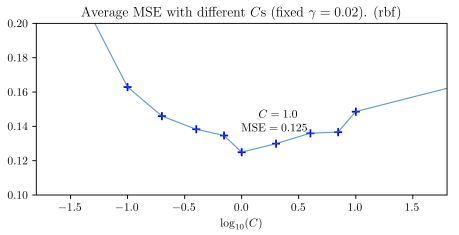

Figure 4. Performance comparison under RBF kernel.

Grid search is first performed for d and coef0 under $C_{\text{poly}} = 1.0$, as is shown in Fig.5. Having selected $(d^*, \text{coef0}^*) = (2, 0.1)$, searching for C_{poly} suggests that $C^*_{\text{poly}} = 0.1$ (Fig.6).

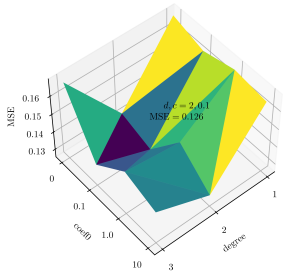

Figure 5. Performance comparison under Poly kernel.

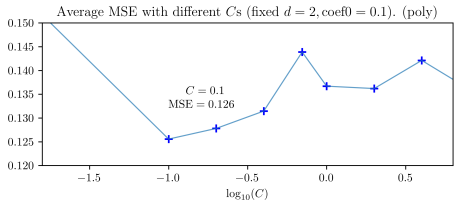

Figure 6. Performance comparison under Poly kernel.

With the experiments above, it is concluded that the optimal model is achieved by RBF kernel with $\gamma^* = 0.02$, $C^* = 1.0$. I took these parameters and the whole training set to build a model for test submission.

4. Test Set Prediction Analysis

Of the 250 test examples, my model predicted 34 of them to be more expensive than $250,000$. Of these 34 items, 10 have *OverallQual* > 8, 17 have *GrLivArea* > 2000, and 26 have *GarageAre* > 700.

In terms of dominant factors, I would model it as those factors A with high $\mathbf{Pr}[\text{high price}|A] = \frac{\mathbf{Pr}[\text{high price}, A]}{\mathbf{Pr}[A]}$. Using such a definition, it is easily observed through Eq.2 that having *OverallQual* > 8 is the most dominant factor of high price.

$$\begin{aligned}
\mathbf{Pr}[\text{high price}|\textit{OverallQual} > 8] &= \frac{10}{10} = 1.0 \\
\mathbf{Pr}[\text{high price}|\textit{GrLivArea} > 2000] &= \frac{17}{40} = 0.425 \\
\mathbf{Pr}[\text{high price}|\textit{GarageArea} > 700] &= \frac{26}{35} \approx 0.743 \qquad (2)
\end{aligned}$$

5. Another Regressor: MLP

In this section I tried neural networks as the second regressor. Neural networks helps us determine whether we are making wrong assumption that there exists linear relationship between input and output. For example, if neural networks with more than one layer (MLP) significantly outperforms the support vector machine, linear models is likely underfitting and should be abandoned.

To be applied to the dataset, all the ANN networks have 8 neurons at the input layer and 1 neuron at the output layer. 16 samples are grouped into one batch, and an SGD optimizer is utilized to optimize the model. *ReLU* is adopted as the activation function, and optimizer parameters include: `lr=1e-4`, `momentum=0.9` and `weight_decay=1e-4`.

Table.3 demonstrates the model structures I have tried and the average MSE of 5 models trained independently. It is obvious according to the table that depth in network helps very little on this dataset as not even one single model achieves similar performance to the SVR model. Consequently, I'm inclined to concluded that SVR is one of the most suitable models for this dataset.

Table 3. Neural network performances

Structure	Average MSE
$8 \to 6 \to 1$	0.188
$8 \to 16 \to 1$	0.194
$8 \to 16 \to 6 \to 1$	0.177
$8 \to 16 \to 16 \to 6 \to 1$	0.191

6. CIFAR-10 Using CNN

CIFAR-10 is a dataset significantly larger than our given housing price regression dataset. It consists of $50,000$ training images uniformly distributed among 10 different classes and $10,000$ test samples. All samples have 3 color channels and a size of 32×32, leading to an input vector dimension $3 \times 32 \times 32 = 3072$.

Given the large scale of the dataset, it becomes almost impossible to construct a massive fully connected neural network on my own laptop. Consequently, a convolutional neural network is adopted to reduce the number of optimizable parameters. After referencing several well-known CNN architectures, I chose *VGG* [3] which features 3×3 filters as the foundation for my own network.

The reason for choosing VGG is simple and clear: to achieve the same receptive field, VGG with size-three filters requires the minimum number of parameters (compared to other architectures like *AlexNet*), thereby reducing computational demands. The efficiency in computing makes it practical for image classification on CIFAR-10.

6.1. Architecture and Regularization

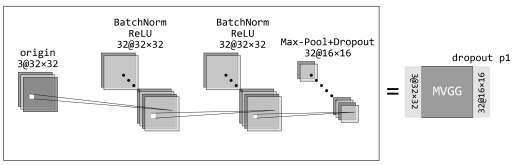

Figure 7. Defined *VGG Block* to simplify architecture.

As is shown in Fig.7, *VGG block* is defined to represent two consecutive convolution layers stacked together, followed by a pooling layer. Regularization techniques including *batch normalization* [1] and *dropout* [5] are introduced here to prevent the model from overfitting.

Using the *VGG block* defined above, my network takes the form shown in Fig.8, comprising 2 VGG blocks and thus resulting in a total of 4 convolution layers. Following the VGG blocks acting as the feature extractor are two dense layers considered as the classifier in the feature space.

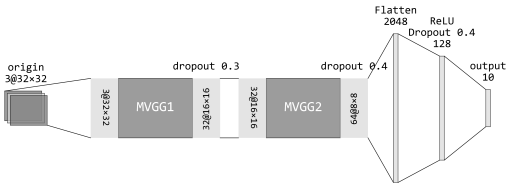

Figure 8. Network architecture.

Apart from the regularization techniques visible in Fig.8, *data augmentation* [2] as a data-level generalization is also utilized. Taking into account the difficulty of programming and computational complexity, I chose *horizontal flipping* and *random erasing* among all the popular augmentation methods. An example is given in Fig.9. To prevent random erasing from eliminating the critical information located in the center of pictures, I set the center of erased pixels to random values at off-center positions.

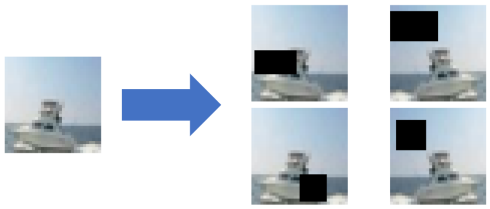

Figure 9. Exmaple of augmentation.

6.2. Performance and Analyses

I trained the model defined above using Adam optimizer with learning rate 0.001 and weight decay 0.0001 for 30 epoches, achieving an approximately 16.6% error rate. The confusion matrix obtained from test set is shown in Fig.10, from which it is intuitively demonstrated that the model achieves relatively satisfactory performance on the image classification task.

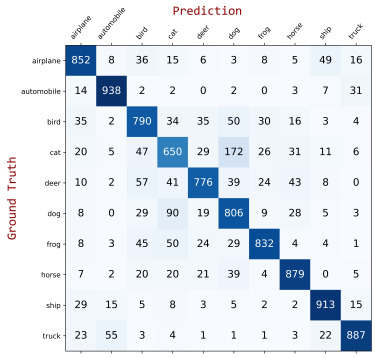

Figure 10. Confusion matrix plotted on test set.

However, it is noticeable that CIFAR-10 is a computer vision task conducted under very ideal assumptions, such as all images having the same size and the samples are uniformly distributed across 10 classes. In reality, people don't observe the world through 16×16 eyes, nor do they have equal opportunities to encounter all categories of objects. Before introducing LLM with powerful understanding capabilities, these issues require artificially introduced intervention mechanism to ensure an acceptable model performance.

p.s. The code for CIFAR-10 image classification is updated to my github repository.

References

[1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. 3

[2] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019. 4

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 3

[4] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14:199–222, 2004. 1

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014. 3