House Price Prediction with Machine Learning[*]

Keping Wang

April 24, 2017

Abstract

This paper introduces various machine learning methods for house prediction. Experiments are conducted to choose the best model for this task, and support vector regression is the winner.

1 Introduction

House Price Prediction, or Hedonic Regression, is an important topic in real estate economics. Once we can accurately predict the house price given various features of the house, the prediction model can be applied to other economics tasks such index constructing, house value assessment, and analyzing demand for various housing characteristics. Yet economists are more familiar with causal inference than prediction. At the same time, the machine learning field, where some statisticians and computer scientists work on, is dedicated at making predictions. So I decide to write this paper to introduce the machine learning methods for economists interested at house price prediction. The techniques introduced here can be applied to other prediction tasks as well.

This paper has two main themes:

1. Introducing the machine learning methods and explaining why they are better at making predictions than simple linear regressions.

2. Exploring by experiments the best machine learning method for house price prediction.

The paper is organized as follows: 1) this brief introduction, 2) section 2 briefly reviews the economics literature of house price prediction with machine learning methods; 3) section 3 introduces the supervised learning framework, with emphasis on gradient descent, non-linearity, prediction VS causality, regularization, and cross-validation; 4) section 4 shows the four supervised learning models I'm going to compare, linear regression with L2 regularization, Lasso regression, support vector regression, and random forest regression; 5) section 5 compares the prediction results of the various models; 6) section 6 introduces an unsupervised learning technique called principal component analysis; 7) section 7 compares different feature selection methods; and finally 8) section 8 is the conclusion.

2 Economics Literature

Many researchers have shown that certain machine learning techniques are more powerful at house price prediction than the traditional Ordinary Least Squares (OLS) model. Yoo et al. (2012) uses a Random Forest model for variable selection during hedonic regression and shows that it achieves an R^2 of 0.97, while an OLS of similar configuration only got R^2 of 0.76. Selim (2009) compares OLS and Artificial Neural Networks (ANN)[1] using the

[*] GitHub: `https://github.com/kepingwang/AmesHousing`

[1] I also tried ANN, but it performs worse than other methods. So I choose not to introduce it in this paper.

house price data of Turkey and finds out that the Root Mean Squared Error (RMSE, the smaller, the better) of OLS and ANN are 1.57 and 0.66 respectively.

Some researchers don't exploit the machine learning methods for hedonic regression, but trying to improve the prediction accuracy by manually engineering features. For example, during the attempt of wine price prediction, Costanigro et al. (2007) finds that fitting different kinds of wines differently greatly improves the model accuracy. Although machine learning nowadays still rely on human feature engineering for top performance, the kind of feature engineering in their paper could be automatically accomplished by a machine learning model.

Computer scientists and statisticians study machine learning for the single purpose of making predictions. It is understandable that machine learning methods are better at fitting values than the traditional econometric approaches. Just as Athey (2016) said on Quora[2], machine learning will have enormous impact on economics. Her work is concentrated on applying machine learning to causal inference, where some tweaks of the machine learning models have to be made. However, for tasks like hedonic house price prediction, the only goal is prediction accuracy, then many off-the-shelf machine learning methods would be sufficient.

3 Supervised Learning Framework

In this section I will briefly introduce the machine learning methods that I will try to apply to the hedonic house price regression problem. All these methods fall into the category of supervised learning, where the true data y is provided, and the model is trained to take input X and generate an output that is as similar to y as possible.

Under supervised learning, there are two general categories of problems, classification and regression. The difference is that the truth variables y in classification task only take a limited number of discrete values, while the y in prediction take continuous values. Hedonic house prediction fits exactly into the supervised regression task.

From now on, let's take an OLS model and I'll use it to illustrate the machine learning techniques. Here is the OLS model:

$$\hat{y} = f(x) = w \cdot x + b, \tag{1}$$

where x is a vector of features (like number of bedrooms, floor size, etc...), w is a vector denoting the weight of these features, and b is a bias term to "shift" the predicted values.

A supervised learning task is to choose the w and b variables so that y and $yhat$ is as similar as possible. We need an exact measure of how similar y and $yhat$ are, and we call that the loss function. For regression tasks, it usually takes the following form, Mean Squared Error (MSE):

$$loss = \frac{1}{n}\sum_{i}^{n}(y_i - f(x_i))^2. \tag{2}$$

Then the our OLS task becomes choosing w and b variables so as to minimize the loss. There are three possible approaches for this optimization task:

1. Randomly try all w and b from the range of all real numbers and keep the variables that lead to the smallest loss. This is stupid and I am just kidding.

2. Solve the quadratic optimization problem mathematically and calculate the value of w and b from the math solution. In OLS, we would have $w = (X'X)^{-1}X'Y$. The solution looks neat, but there are problems with this approach:

   (a) Not general enough. For each different kind of model $\hat{y} = f(x)$, the math has to be redone to calculate the best weights. The calculation is simple for OLS, but for more complex models, it becomes hardly

[2]https://www.quora.com/How-will-machine-learning-impact-economics?redirected_qid=6706789

possible to derive a straight math solution. For example, think of a non-linear regression model: $f(x) = w_2 \cdot \frac{1}{1+e^{-w_1 \cdot x}}$, where x is of dimension k, w_1 is of dimension $k \times h$, and w_2 is of dimension h.

(b) Computationally very expensive both in terms of time and memory (Cook, 2010). It takes $O(n^3)$ time and $O(n^2)$ memory. When n becomes as large as one million, it would require terabytes of memory.

3. In the machine learning area, people use a technique called gradient descent to find the optimal weight. The only requirement for gradient descent is that the loss function is differentiable. The loss function could be formulated as $loss(y, X, w)$ where y and X are input values and are treated as constant during optimization, while w denotes all the weight variables (including the bias terms). Gradient of the loss with respect to the weight is:

$$\nabla_w loss(y, X, w), \tag{3}$$

and the gradient descent algorithm is the following:

Algorithm 1 Gradient Descent

```
1: Choose hyperparameter learning rate $\alpha$
2: Initialize $w$ with some initial value
3: repeat
4:     $w = w - \alpha \nabla_w loss(y, X, w)$
5: until in an iteration, the change of the loss is small enough
```

Gradient descent is a general algorithm, in that as long as the loss function is differentiable (don't even need to be globally differentiable), the gradient of weight variables w could be calculated using chain rules. And for constrained optimization problems, they can first be converted to unconstrained optimization problems using Lagrange multiplier method, and then solved by applying gradient descent.

To summarize, our supervised learning regression task has become: picking a model $f(x; w)$, with w as parameters, and train the model to minimize loss using gradient descent. The researcher's task is more or less mainly about picking the right model $f(x; w)$.

3.1 Beyond Linearity

One advantage of this supervised learning framework over traditional linear regression is that the model $f(x; w)$ can be flexible and incorporates all kinds of non-linearity. Because we have the magic tool of gradient descent, our model $f(x; w)$ could take various forms and the calculation will be the same.

Economists have been playing a lot with various flavors of linear regression and inference of estimation error. Linear regression is good because its simplicity allows economists to easily derive estimation errors for the weights, understand the model, and convey the understanding to the general audience.

Most non-linear models cannot really be easily summarized in one or two sentences to convey the finding, but they can involve complicated logics and are well-suited for making predictions. Non-linearity really brings magic to prediction. Think of AlphaGo and self-driving cars, they are all enabled by various kinds of non-linearity in the machine learning model.

Let me use house price prediction as an example to show how non-linearity makes the prediction model more powerful. Imagine there to be a feature (one dimension of the input X), describing whether there is a swimming pool. If you include the swimming pool feature into a simple linear regression, you find that it increases house price by 20000 dollars on average. However, likely a more accurate estimation is that for great neighborhoods the a swimming pool adds the price by 40000, while for not-so-good neighborhoods it only adds the prices by 10000.

Then the classic econometric way of handling this is to include the interaction term of neighborhood quality and swimming pool ($neighborhood \times pool$). That is acceptable when we have a very limited number of variables.

But when we are predicting using many variables, the interactions between variables are usually to complicated to specify manually.

Using non-linear models, we could handle complex interactions automatically. In this example, the model specification could be $f = sign(w_1 neighborhood + b_1) \cdot w_2 pool$, which can serve the same functionality as the interaction term. We do still have to manually choose from a range of supervised learning models, just as we have to pick interaction terms manually. But some supervised learning models have the power to allow enormously different kinds of interactions between variables, and they can be trained very easily, then only those interactions that do help with the prediction will remain with a large enough weight (coefficient).

3.2 Feature Selection

Feature selection, or variable selection is very important for a prediction model. Sometimes the original data contains lots of duplicate or useless features, and including these features will consume some degrees of freedom[3]. If we had an unlimited number of observations, then for some flexible models, more features will strictly improve performance. With a limited number of observations, we would still want to do feature selection. However, feature selection has to be done automatically, instead of manually, since there could be a huge amount of features and we are trying to find a general approach for house price prediction.

3.3 Prediction instead of Causality

Another difference between the supervised learning method and the econometrics method is that in supervised learning, prediction is the only goal and it doesn't care anything about causal inference. Athey (2016) has been working on modifying machine learning methods for causal inference, but that is not the focus of this paper.

Prediction doesn't care about omitted variables. For prediction, the more variables, the better, but we can hardly exhaust all the variables. Also prediction here only provides information about expected values, but not about confidence.

3.4 Regularization and Cross-Validation

To show how accurate is our model for prediction, we cannot only see the R^2 of the fitted values for our training data set. With a complex model specification, it is always possible to perfectly fit a very small dataset, but then this model won't be able to generalize well to unseen data.

So the model needs to be evaluated using unseen data. This is called test set. At the very beginning I randomly choose part of the data as the test data, keep them away from the training process, and only use the test data finally for evaluating accuracy.

A good prediction model, should on the one hand be complex enough to incorporate the interactions of variables, but on the other hand be limited so as to generalize well to unseen data. Generally speaking, the methods used to limit the "power" of the model is called regularization.

One regularization method I'm going to use here is the explicit L2 regularization. That is, for each weight (coefficient) w, a regularization term $reg * w^2$ is added to the total loss of the model, where reg is a hyperparameter to be chosen before training. This regularization means that there is a cost for using each additional weight, i.e. it is better to fit the model with as fewer weight nodes.

Another technique for finding better model is cross-validation. For each category of models, there are hyperparameters to determine before training, for example, what variables to include. These hyperparameters has to be decided on another separate dataset called validation data. The validation data is usually held out apart from the original training data, and is also separate from the test data.

[3]Degree of freedom only applies to linear regression. But for non-linear models there are similar mechanisms where including useless features make the model worse.

The last technique I'll be using to make the model better at generalization is called stochastic gradient descent (SGD). SGD still iterates to optimize the weights but in each iteration it uses a random mini-batch (could be 50 samples) of training data, instead of the whole training data. Zhang et al. (2016) has shown that SGD is implicitly regularizing the solution.

In the next section I'll briefly introduce the supervised learning models that I'll use for house price prediction: support vector regression, random forest regression, and neural networks.

4 Supervised Learning Models

4.1 Linear Regression with L2 Regularization

In linear regression with L2 regularization, our model for prediction is still

$$f(X) = XW + b, \tag{4}$$

but our loss function to minimize becomes:

$$loss = \|y - (XW + b)\|_2^2 + \lambda \|W\|_2^2, \tag{5}$$

where $\|a\|_2$ means the L2 norm of a, that is $\|a\|_2 = \sqrt{\sum_i a_i^2}$.

Here $\|W\|_2^2$ is the regularization term, which will try to make the model parameters close to zeros. The term $\|y - (XW + b)\|_2$ term on the left will try to make the model fit as much as it could on the training data. Thus the fitting term and regularization term work together to make give the model good performance on the test set. $\lambda > 0$ denotes the strength of regularization, and it is a hyperparameter of the model, to be determined using the validation set.

4.2 Lasso Regression

Lasso (least absolute shrinkage and selection operator) regression also uses the same linear fitting term, but uses L1 norm for regularization and performs automatic feature selection. The loss function is:

$$loss = \|y - (XW + b)\|_2^2 + \lambda \|W\|_1, \tag{6}$$

where $\|a\|_1$ means the L1 norm of a, that is $\|a\|_1 = \sum_i |a_i|$.

It can be helpful to think of the two problems in constrained form (CMU, CMU):

$$\min \|y - (XW + b)\|_2^2 \quad \text{subject to} \quad \|W\|_2^2 \le t \tag{7}$$
$$\min \|y - (XW + b)\|_2^2 \quad \text{subject to} \quad \|W\|_1 \le t \tag{8}$$

Now t is the hyperparameter. For any λ and corresponding solution in the previous formulation, there is a value of t such that the above constrained form has the same solution.

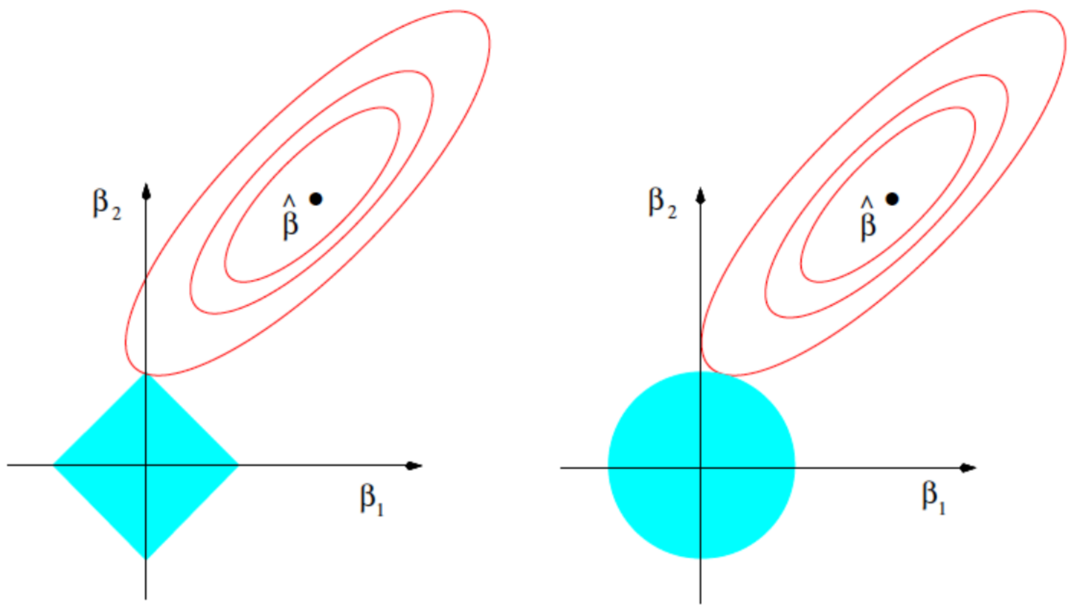

Figure 1: Linear Regression: L1 vs L2 regularization

The graph[4] illustrates the difference between L1 norm and L2 norm weight regularization. The figure shows the case where the weight W contains only two dimensions, β_1, and β_2. $\hat{\beta}$ in the graph denotes the weight W that best fits the linear prediction part of the model. The light blue shapes (square or circle) denotes the constraints applied by the regularization term, and the size of the constraint area will change with regularization strength λ.

The main difference here is that L1 regularization allows weight parameters to be exactly zero, which means the solution falls on some corner cases, which for L2 regularization, the weight can hardly be exactly zero.

Because L1 regularization renders zero weights for some variables, Lasso regression utilizes this fact to do feature selection. There are some specific approaches for how features are selected, but the one I use here goes as follows:

Algorithm 2 Feature Selection for Lasso

```
1: Initialize $K$ to be the set of all features.
2: repeat
3:     Run Lasso with feature set $K$, get feature weights.
4:     $K$ = features in $K$ with non-zero weights.
5: until all features in $K$ has non-zero weights.
```

By applying feature selection, lasso increase the robustness of the model by removing the random influences of useless features.

Linear regression with regularization could be viewed as either 1) unconstrained optimization, which could be solved in general by gradient descent, or 2) constrained quadratic (and convex) optimization, which could be solved by quadratic programming.

[4]Picture taken from CMU (CMU)

4.3 Support Vector Regression

Support vector regression (Basak et al., 2007) has the following optimization objective:

$$\text{minimize} \quad \frac{1}{2}\|W\| + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{9}$$

$$\text{subject to} \quad -\varepsilon - \xi_i^* \leq y_i - W \cdot x_i - b \leq \varepsilon + \xi_i \tag{10}$$

$$\xi_i, \xi_i^2 \geq 0 \tag{11}$$

This seems intimidating, but let me introduce the general ideas with the help of the graph[5].

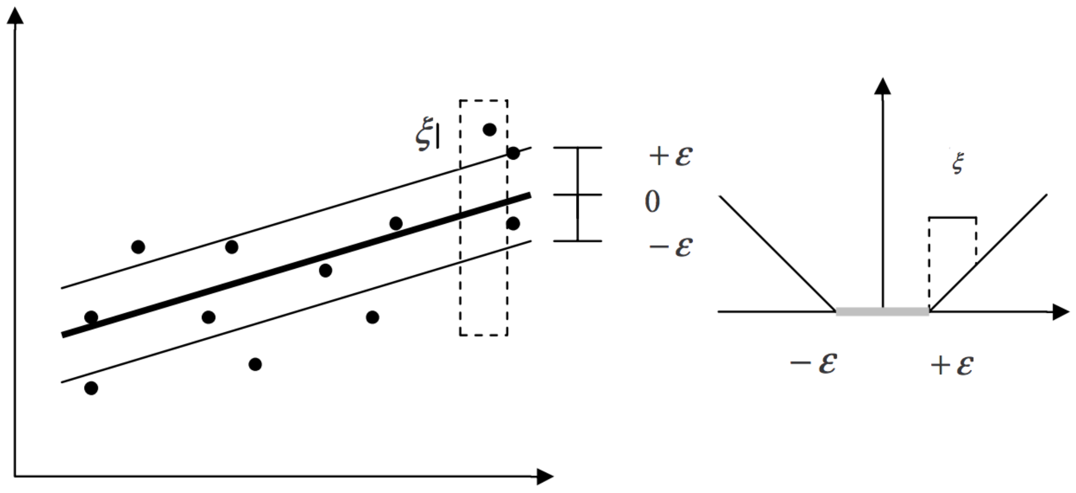

Figure 2: Soft Margin Linear SVR

For now, the prediction term is still $WX + b$. In the constraints of the objective, we can see that the error of prediction is actually constrained to be within the interval $(-\varepsilon - \xi^*, \varepsilon + \xi)$. The ε part is the hard margin constraint, and the ξ part is the soft margin constraint. ε is a hyperparameter of the model, rather than a parameter to be learned.

If we ignore all ξ terms, the model becomes: you must fit the data so that all fitting errors fall within $(-\varepsilon, \varepsilon)$. When the hard condition is met, try to minimize the L2 norm of weight. If the data cannot be fitted with margin ε, then the margin SVR problem has no solution.

Adding ξ terms softens the margin and makes sure there is always a feasible solution. The data point that falls out of $(-\varepsilon, \varepsilon)$ thus adds penalty to the objective function. $C > 0$ is the strength of penalty for data points outside of hard margin.

The main characteristic of Support Vector Regression is that only data points within the margin $(-\varepsilon, \varepsilon)$ are involved in calculating the weight (especially when C is close to *zero*), and these data points are called support vectors. Therefore the model is very resilient to outliers. The power of SVR lies more in its dual form:

$$\text{maximize} \quad -\frac{1}{2}\sum_{i,j}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j\rangle - \varepsilon\sum_i(\alpha_i + \alpha_i^*) + \sum_i y_i(\alpha_i - \alpha_i^*) \tag{12}$$

$$\text{subject to} \quad \sum_i(\alpha_i - \alpha_i^*) = 0 \text{ and } \alpha_i, \alpha_i^* \in [0, C] \tag{13}$$

[5]Graph taken from Basak et al. (2007)

The relationship between the solution of the primal problem and the dual problem is:

$$W = \sum_i (\alpha_i - \alpha_i^*) x_i \quad \text{and therefor, } f(x) = \sum_i (\alpha_i - \alpha_i^*) \langle x_i, x \rangle + b \tag{14}$$

This is called support vector expansion, i.e., W can be completely described as a linear combination of the training patterns x_i. Even for evaluating $f(x)$, we don't need to compute W explicitly.

This dual problem can be solved with 1) quadratic programming, 2) Lagrange multipliers and gradient descent, or 3) some faster specific algorithms.

The advantage of the dual that we can than apply kernel trick. To understand kernel trick, let's pause and think about feature space transformation. If we want to use three features to fit a linear model, it would be like $f(x) = w_1x_1 + w_2x_2 + w_3x_3 + b$. If we want to allow for more flexibility, we probably would add square terms of features to the model $f(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_1^2 + w_5x_2^2 + w_6x_3^3$. Then we probably want to add interaction terms too: x_1x_2, x_1x_3, and x_2x_3. Then it starts to make us economists uncomfortable when we are dealing with many more variables and want to manually add higher order terms and higher order interaction terms. This kind of feature space transformation can be done automatically with kernel trick.

If we have two data points $x^{(1)}$ and $x^{(2)}$. We can have the following equation (Konstantin, Konstantin):

$$\begin{aligned} k(x^{(1)}, x^{(2)}) &= (x^{(1)} \cdot x^{(2)})^2 = (\sum_i x_i^{(1)x_i^{(2)}})^2 = \sum_{i,j} x_i^{(1)} x_j^{(1)} x_i^{(2)} x_j^{(2)} \qquad (15) \\ &= (x_1^{(1)}x_2^{(1)} + x_1^{(1)}x_3^{(1)} + x_2^{(1)}x_3^{(1)}) \cdot (x_1^{(2)}x_2^{(2)} + x_1^{(2)}x_3^{(2)} + x_2^{(2)}x_3^{(2)}) \qquad (16) \end{aligned}$$

so the squared inner products of two data points can be used to represent the inner products all their interaction terms. This $k(x^{(1)}, x^{(2)})$ is called a kernel function, which can represent a feature space transformation by some calculations between two data points. Adding kernel functions will concatenate the feature space they represent. $k(x^{(1)}, x^{(2)}) = x^{(1)} \cdot x^{(2)} + (x^{(1)} \cdot x^{(2)})^2$ represents the inner product of two data points in the following feature space:

$$[x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3] . \tag{17}$$

By plugging different forms of kernel function $k(x_i, x_j)$[6] back into our dual SVR problem to replace $\langle x_i, x_j \rangle$, we can fit the model with different kinds of feature space transformation. Sometimes the feature space is very huge because of all the higher order terms and interaction terms, but dual SVR with kernel trick could solve the problem easily.

A kernel function often used is the RBF (Gaussian) kernel:

$$k(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) = \exp\left(-\gamma \|x-y\|^2\right), \tag{18}$$

which represents a normalized polynomial infinite degree feature space transformation:

$$\phi(x) = e^{-\gamma x^2} \left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \sqrt{\frac{(2\gamma)^3}{3!}}x^3, \cdots\right] \tag{19}$$

To sum up, support vector machine can estimate weights ignoring far away data points and can use kernel functions to perform very complicated feature space transformation.

[6] The notation $k(x^{(1)}, x^{(2)})$ and $k(x_i, x_j)$ both represent the kernel function, only I write the parameters in different forms. Later also $k(x, y)$.

4.4 Random Forest Regression

Random Forest Regression(Liaw and Wiener, 2002) is a method that's said to be a panacea for all data science problems. It differs from previously introduced models in two ways: 1) it is a tree-based model, which cannot be trained with the general gradient descent method; and 2) it is an ensemble of many models. Random Forest is an ensemble of multiple decision trees.

The following graph[7] illustrates a decision tree[8] for house price prediction. The decision tree is trained by recursively split the data by a threshold of a feature until some ending condition is met. The numbers within parentheses are sample size. In the first step, a sample of size 100 is split to a 54 observations with living area < 1800 sqft, and 46 observations with living area ≥ 1800 sqft. The feature used for splitting could be continuous, where a threshold value is taken, or categorical (or dummy), where true of false is used. The feature and its threshold used for splitting is chosen by some algorithm, so as to minimize the prediction loss (recursively at each step).

At each step, the algorithm either goes on with splitting, or ends. The ending conditions could be things like maximum depth of a tree, or minimum size of subsample at a node. At a leaf node, we get a subsample meeting all previous conditions. For example, in the graph, we have 31 observations with living area ≥ 1800 sqft and number of bedrooms at least 2, and the mean house sale price of these 31 observations is \$140k.

This decision tree can then be used for prediction. If we have a house with living area 1900 sqft, 2 bedrooms, and one garage, this model will predict its house price to be \$190k.

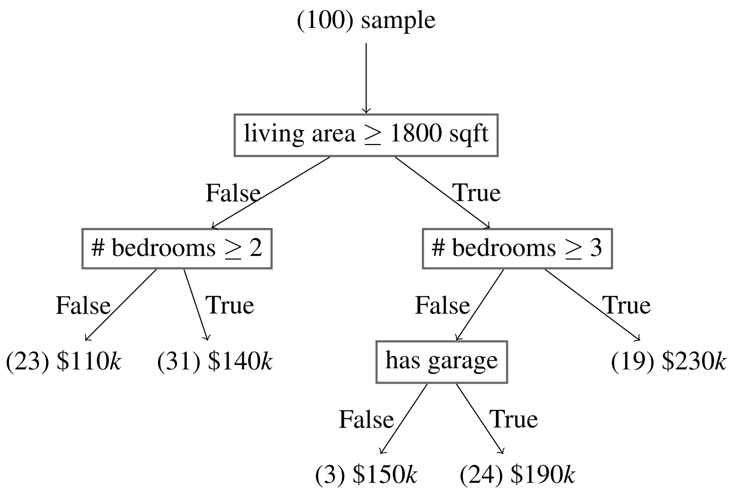

Figure 3: A Decision Tree predictor for house price

A decision tree is powerful in that it can provide the best possible fit for the training data if no ending condition is applied. In machine learning terms, we say this model has very low bias. The tree structure allows for a high level of non-linearity, and also does feature selection automatically in the sense that only features which result in the most fitted model will get involved in the tree building process. It can be viewed as automatically specifying many interaction terms, and only the ones that fit the data best.

However, a tree based model has very high variation, which means high generalization error. In the above graph, there are three data points that have living area ≥ 1800 sqft, less than 3 bedrooms, and no garage. They show that conditional on the living area ≥ 1800 sqft and less than 3 bedrooms, the expected value of a garage is \$40k. It is possible that these 3 observations all happen to have particular poor conditions for some unobserved characteristics, and that the real value of a garage is only about \$10k. To reduce the model variance while maintaining the high fitting power of the decision tree, the random forest model use an ensemble of decision trees.

[7]The graph is made up for illustration, not representing real data.
[8]Here it could be more precisely called a regression tree.

Model ensemble means grouping multiple models together for prediction. A random forest with 20 decision trees will get train these different trees from 20 different subsamples randomly drawn from the whole training dataset, and final predicted value is the mean of the predicted values of the 20 decision trees.

A random forest model can have many hyperparameters, but a useful rule is to build each tree as deep as possible and to ensemble as many trees as possible. With this rule, we don't really need to tune any hyperparameters for a random forest. However, model performance does gets improved when the maximum number of features in trees is limited. Usually, thie maximum number of features is set to the square root of the total number of features.

To summarize, a random forest regression model is an ensemble of decision (regression) trees. It has low bias and low variance, and is very easy to tune.

5 Experiments

In this section, I'll introduce the data and preprocessing, evaluation metric of accuracy, and then show the results of different prediction models. All experiments are conducted in Python, with the help of the scikit-learn package (Pedregosa et al., 2011).

5.1 Data and Preprocessing

The data comes from De Cock (2011). It describes the sale of individual residential property in Ames, Iowa from 2006 to 2010. The data set contains 2930 observations and a large number of explanatory variables (23 nominal, 23 ordinal, 14 discrete, and 20 continuous). Please refer to De Cock (2011) for a complete description of all variables.

In my models, I treat discrete variables the same as continuous variables, encode ordinal variables with integer values starting from 0, and transform nominal variables using one-hot encoding. For continuous variables with skewed distributions, including the final sale price to predict, I make log transformations ($\phi(x) = \log(1+x)$) to make them closer to normal distribution. Finally, there are 271 features, meaning that the input matrix X has 271 columns.

Here are the histograms of House Sale Price - the variable to predict, before and after log transformation.

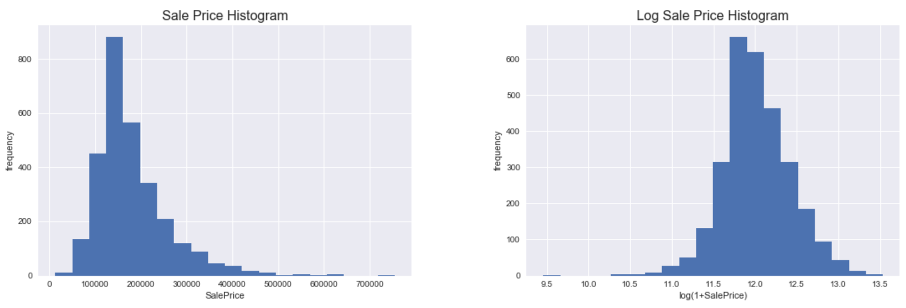

Figure 4: Sale Price distribution before and after log transformation.

I split the whole 2930 observations into a training set of 2200 observations, a validation set of 230 observations, and a test set of 500 observations. Here I don't bother to use k-fold cross validation, but only hold out the validation set.

Before the training of the models, each input feature is standardized with mean and standard deviation from training set. Feature standardization allows gradient descent method and other training methods to work better.

5.2 Evaluation of Accuracy

The loss we use here is the mean squared loss as mentioned above. And we use two measures of prediction accuracy, R^2 and RMSE (Root Mean Squared Error):

$$R^2 = 1 - \text{SSE}/\text{SST}, \tag{20}$$

$$RMSE = \sqrt{\text{SSE}/n}, \tag{21}$$

$$\text{where} \quad \text{SSE} = \sum_i^n (y_i - f(x_i))^2, \tag{22}$$

$$\text{SST} = \sum_i^n (y_i - \bar{y}). \tag{23}$$

RMSE is more usually used in the machine learning literature, and it more directly measures the prediction error of a single data point. While R^2 is usually used in the econometrics literature to see how much total variation is explained by the fitted values. RMSE is a more direct measure of accuracy for prediction. Here I also include R^2 mostly for the convenience of the audience from an economics background.

5.3 Prediction Results

The models are trained on the training set. The hyperparameters are determined by running the model repeatedly with different hyperparameters, and choosing the hyperparameters that result in the best results in validation set. The tuned models are then evaluated using the test set, and these results show the final performance of the models.

Experiments are done with all four previously mentioned models: 1) linear regression with L2 loss, 2) Lasso regression, 3) support vector regression, and 4) random forest regression. For the first two models, the only hyperparameter is λ, the strength of regularization. SVR has two hyperparameters, C, error penalty strength, and ε, specifying the tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. Here I use an SVR with RBF kernel. For random forest, the only hyperparameter is the number of trees it ensembles. All these models have more hyperparameters, but I only choose to tune on the most important ones mentioned above.

Table 1: Model Prediction Results

	Results: (RMSE, R^2)		
Model	Training	Validation	Test
Linear Regression without Regularization (OLS)	0.0992, 0.940	1e7, -1e15	6e7, -2e16
Linear Regression with L2 Regularization	0.1143, 0.921	0.1162,0.922	0.1177, 0.918
Lasso Regression	0.1049, 0.933	0.1127, 0.927	0.1139, 0.923
Support Vector Regression	0.1199, 0.912	0.1041, 0.937	0.1133, 0.924
Random Forest Regression	0.0504, 0.985	0.1312, 0.900	0.1315, 0.898

In the table, results are shown in (RMSE, R^2) pairs, and the test column is the final result. A model is good if it has a small RMSE and a large R^2. To note again, the training set has 2200 observations, the validation set has 230 observations, and the test set has 500 observations. Here for comparison, I also add a linear regression without regularization model, which is what economists refer to as OLS. We can see that the best model is Support Vector Regression, and the worst is Linear Regression without Regularization.

An OLS without regularization fits the model pretty good. But without regularization, there are no constraints on the weights (coefficients), and thus the weight can be chosen whatever value as long as it fits the model. For example, if two houses have identical features except that the basement of one house is larger by 1 sqft, and that the house with the larger basement happen to have a sale price of $\$20k$ higher. Then a model without regularization could conclude that the value of basement area is $\$20k$ per sqft. Below are plots of predicted values against actual values for OLS without regularization. We can see that it fits training data well but fails to generalize at all.

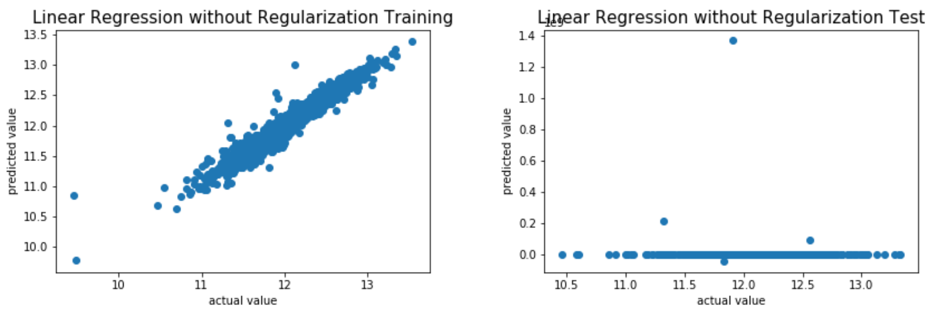

Figure 5: How OLS without Regularization Fails to Generalize

Regularization (either L1 or L2) will add penalty to the final loss function for weights. If the weight for basement area is very huge, say $w_{\text{basementArea}} = 20000$, then this will add $\lambda 20000^2$ to the final loss.[9] It could happen that when we decrease $w_{\text{basementArea}}$ from 20000 to 200, the fitting error (MSE) term is only increased by 1e-5. Then even with a very small (but not zero) regularization strength λ, the model will pick a much smaller $w_{\text{basementArea}}$ and thus makes more sense. The thing to remember: never make prediction without regularization.

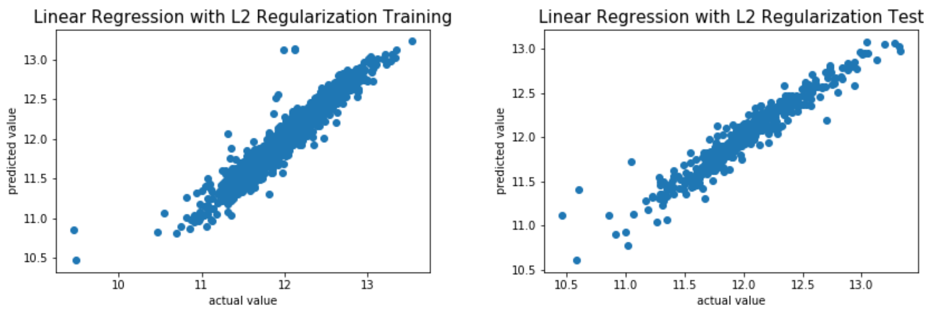

Figure 6: Predicted Values for Linear Regression with L2 Regularization

The two graphs above shows the result of linear regression with L2 regularization. You can see how the predicted values change magically with the simple regularization term $\lambda \|W\|_2^2$ added to the loss function.

The predicted values for the remaining models won't look much different, so I won't plot them. From the table, we can see that SVR is the best choice for the house price prediction task, and Lasso is slightly worse. Random Forest, although fits the training set extremely well, is not a very good model, probably because it is hard for a tree-based model to capture a simple linear relationship between x and y. If the real data is $y = 2x + 3$, then the decision tree is forced to split x into small intervals, and then calculate the mean of y within each interval.

6 Unsupervised Learning

So we've seen how supervised learning methods work for house price prediction. In this section, I'll introduce a completely different machine learning framework, unsupervised learning.

In an unsupervised learning setup, we don't know the actual house prices at all, and we are not asked to predict house prices. The goals are rather ambiguous for unsupervised learning, and different approaches of unsupervised

[9]The number is only made up. Also in the real model, I do log transformation for Sale Price and various features.

learning have different goals. In general, the goal is to explore patterns in the existing data. Here I'll introduce one specific unsupervised learning method, Principal Component Analysis.

6.1 Principal Component Analysis

Principal Component Analysis (PCA) (Shlens, 2014) is a technique to reduce the dimensionality of the data and at the same time still enable the dimensionality reduced data to well represent the original data.

First this dimensionality reduction is presented as a linear transformation P ($m \times r$) of the original data X ($n \times m$). We denote the transformed data as Z ($n \times r$), with $Z = XP$. P reduces the data dimensionality from m to r, with $r \leq m$.

Then we choose the desired property of transformed data Z to be removing feature redundancy and explaining data variance. To formalize things, we introduce the notation of covariance matrix. The covariance matrix of Z, C_Z is defined as:

$$C_Z \equiv \frac{1}{n} Z^T Z. \tag{24}$$

C_X is a square symmetric $r \times r$ matrix. The diagonal terms measure the variance explained by each feature, and the off-diagonal terms are the covariance between features. The desired property of Z can thus be stated as: 1) C_Z is diagonal, and 2) the diagonal values of C_Z are the highest possible ones. We can derive the following:

$$C_Z = \frac{1}{n} Z^T Z = \frac{1}{n} (XP)^T (XP) = \frac{1}{n} P^T X^T X P \tag{25}$$

$$= P^T (\frac{1}{n} X^T X) P = P^T C_X P \tag{26}$$

A theorem in linear algebra states that for a symmetric matrix A, we can decompose it as $A = EDE^T$ where D is a diagonal matrix, and E is a matrix of eigenvectors of A arranged as columns. Thus we can select matrix P to be a matrix where each column $P_{[:,j]}$ is an eigenvector of C_X, and by this selection, $P \equiv E$. We also know that for matrix P consisting of orthonormal vectors, we have $P^T P = I$.

$$C_Z = P^T C_X P = P^T (EDE^T) P = P^T (PDP^T) P \tag{27}$$

$$= (P^T P) D (P^T P) = D \tag{28}$$

Now we have finished the derivation of PCA. The principal components of X are the eigenvectors of $C_X = \frac{1}{n} X^T X$. The j^{th} diagonal value of C_Z is the variance of X along $P_{[:,j]}$. The principal components (PC) are ordered so that the first principal component (PC1) corresponds to the normalized eigenvector of C_X with the largest (absolute) eigenvalue, which means PC1 explains the most variance in the data. And if we remove the dimensionality of PC1, then the second principal component explains the most variance for the remaining data, etc.

This graph below[10] shows the first two principal components of a Gaussian random data. The first principal component is pointing the the upper left. If we project data onto this PC1, the projected data will have high variance.

[10]Taken from https://en.wikipedia.org/wiki/Principal_component_analysis#/media/File:GaussianScatterPCA.svg

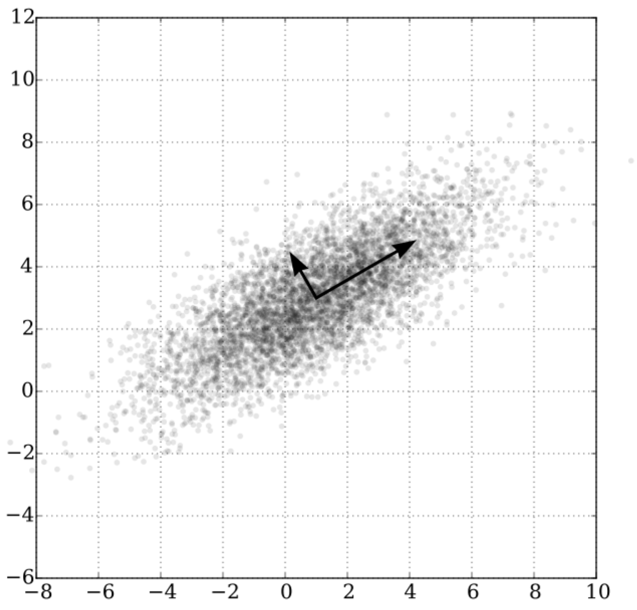

Figure 7: First two Principal Components of a Gaussian Data

6.2 Principal Component Analysis on House Features

We've already introduced PCA. Then let's try the principal component analysis on our house features. The following analysis is conducted on the training set.

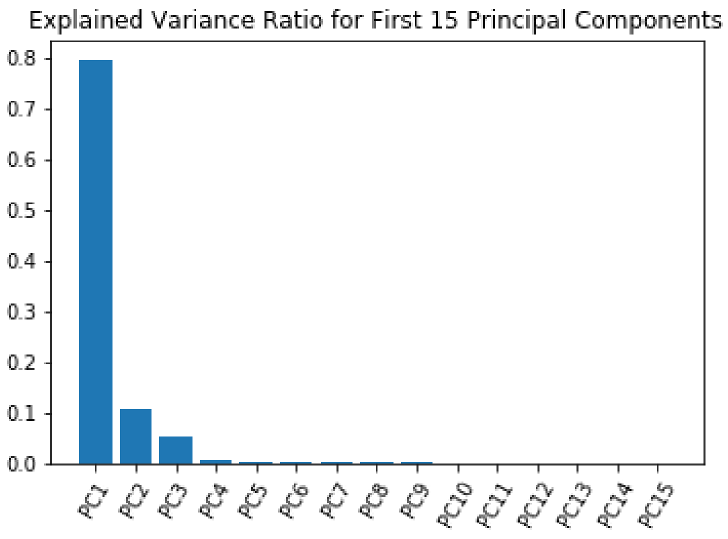

Figure 8: Explained Variance Ratio for First 15 Principal Components

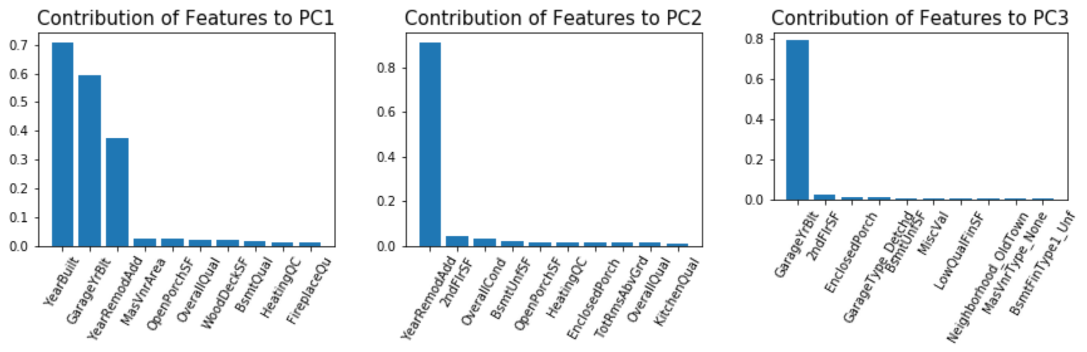

Figure 9: Contribution of Features to first 3 Principal Components

Although PCA is a widely adopted technique[11] for extracting useful information from data, the result here doesn't really make much sense. The first three principals are mostly composed of three year related features: 1) the year the house was built, 2) the year the garage is built, 3) the year the house was most recently remodeled.

This could be partly my bad because I forgot to do log transformation on these year variables. The variables like living area, after log transformation, won't remain much variance. Thus the untransformed year variables take on to explain most of the variance in data.

Still, the main one to blame is PCA itself. Explaining the most variance in data is really a limited standard of choosing important features. Imagine that the phone number of the house owner is accidentally included to the dataset as a continuous number variable, then phone number will probably contribute a lot to some principal component(s) because it brings a lot of variance and is not redundant at all. The computer has no idea that a phone number as a number is totally irrelevant for house price prediction. This is a problem inherent to almost all unsupervised learning methods.

Yet on the other hand, PCA captures the patterns in the data, and the abnormal result of PCA gives us the information that probably I should also do log transformation on the year variables, or keep all variables untransformed.

[11] Adopted more in classification than regression.

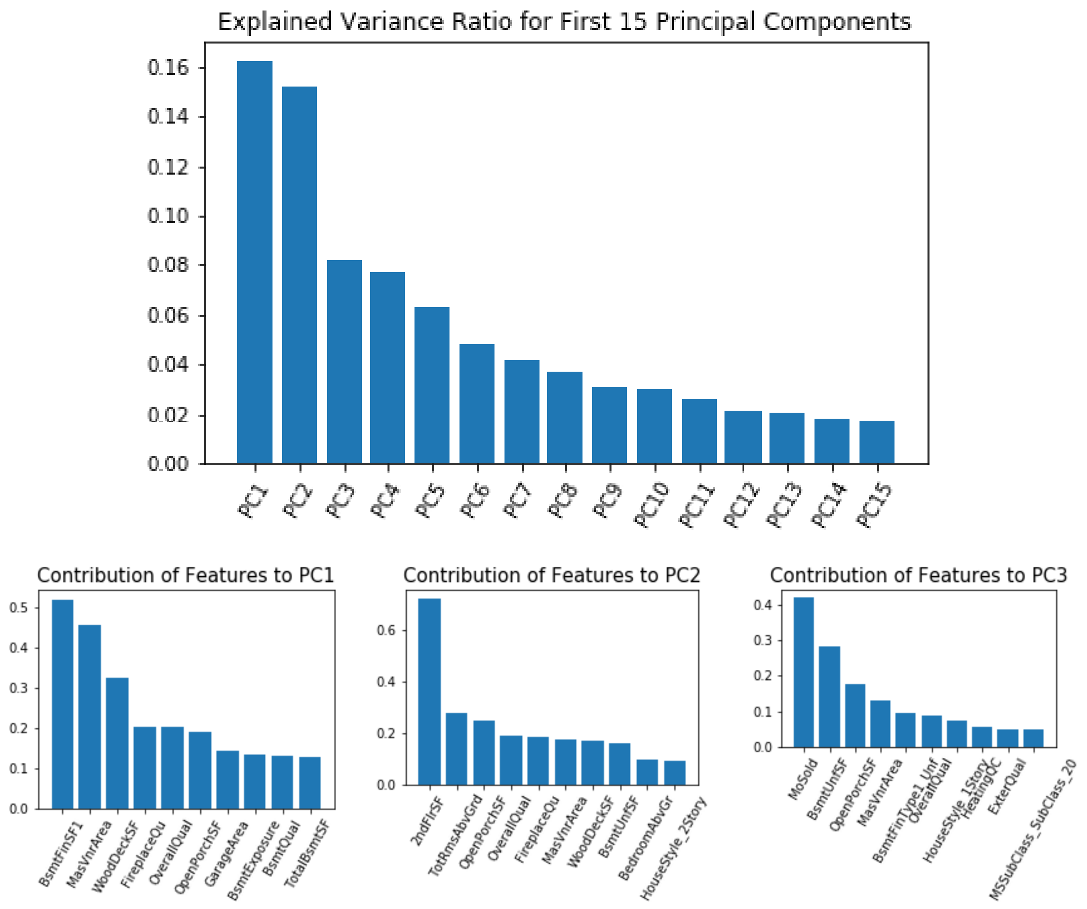

Figure 10: Results of PCA with Year Variables also Log Transformed

Then I log transformed the three year variables and redo the PCA. The results make more sense, with explained variance distributed more evenly, and area variables taking more weight in principal components. However, I don't yet know any reliable rules of variable transformation, except for trial and error, which is only available in a supervised learning setup.[12] In economics, we almost never log transform year variables, but PCA implies we should do that.

[12]Transformation of variables belong to another huge topic in machine learning, called feature engineering. There is fewer general approaches to introduce in feature engineering, so I decide not to go into details. Here taking log for the skewed continuous variables does increase model performance. Whether to take log on the three year variables don't really affect the model prediction result.

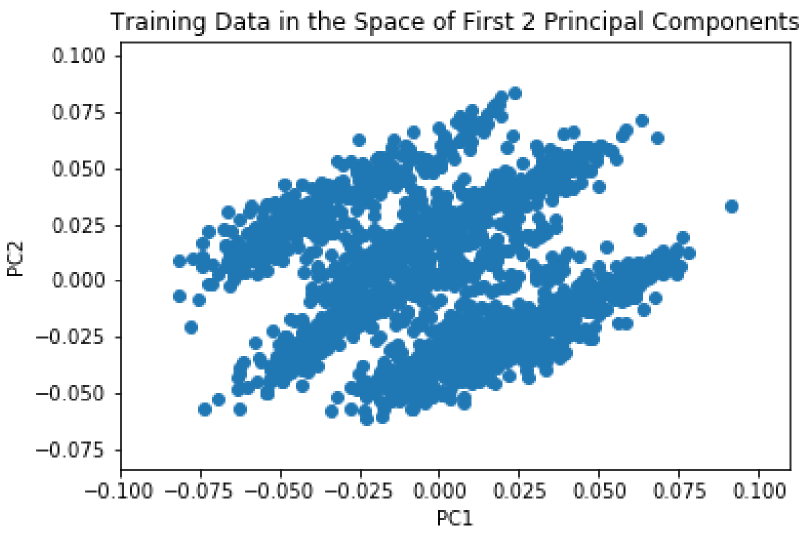

Figure 11: Data in the Space of First 2 Principal Components

Here I plot the training data onto a 2D plane of the first two components. The x-axis is the first principal component, and the y-axis is the second. Something interesting seem to be going on, but I'm not going to interpret here.

So the conclusion is that PCA is easily affected by variable transformation, and thus the information from PCA is not guaranteed to make sense. Still PCA can be performed for data visualization, and perhaps useful patterns can be observed by human eyes.

7 Feature Selection

In the previous section, we've seen that PCA could be used to reduce the dimensionality of our data. Dimensionality reduction is useful in helping human beings to understand the patterns in data. Yet here our goal is to make predictions. Is it possible that reduced data dimensionality also helps the model to make predictions? The answer is that it depends.

Below I'll show the model prediction results with four different set of features:

1. 5 hand picked features, GrLivArea (living area above ground), TotalBsmtSF (total sqft of basement area), LotArea (lot area), OverallCond (overall condition), and OverallQual (overall quality).
2. First 5 principal Components, with year variables also log transformed.
3. 146 features selected using Lasso regression. First tune a Lasso regression model on the full feature data, then drop the features with zero weight in the Lasso predictor.
4. First 146 principal Components, with year variables also log transformed. The number 146 is chosen to compare with the Lasso selected features.

Table 2: Results with 5 Hand Picked Features

	Results: (RMSE, R^2)		
Model	Training	Validation	Test
Linear Regression without Regularization (OLS)	0.1836, 0.795	0.1648, 0.843	0.1795, 0.810
Linear Regression with L2 Regularization	0.1836, 0.795	0.1648, 0.843	0.1795, 0.810
Lasso Regression	0.1836, 0.795	0.1648, 0.843	0.1795, 0.810
Support Vector Regression	0.1629, 0.839	0.1451, 0.878	0.1644, 0.841
Random Forest Regression	0.0622, 0.977	0.1522, 0.866	0.1713, 0.827

Table 3: Results with First 5 Principal Components

	Results: (RMSE, R^2)		
Model	Training	Validation	Test
Linear Regression without Regularization (OLS)	0.2346, 0.665	0.2230, 0.712	0.2391, 0.663
Linear Regression with L2 Regularization	0.2346, 0.665	0.2230, 0.712	0.2391, 0.663
Lasso Regression	0.2346, 0.665	0.2230, 0.712	0.2391, 0.663
Support Vector Regression	0.2350, 0.664	0.2258, 0.705	0.2404, 0.659
Random Forest Regression	0.0768, 0.964	0.1888, 0.794	0.2131, 0.732

Table 4: Results with 146 Lasso Features

	Results: (RMSE, R^2)		
Model	Training	Validation	Test
Linear Regression without Regularization (OLS)	0.1019, 0.937	0.1162, 0.922	0.1150, 0.922
Linear Regression with L2 Regularization	0.1150, 0.920	0.1162, 0.922	0.1185, 0.917
Lasso Regression[13]	0.1049, 0.933	0.1127, 0.927	0.1139, 0.923
Support Vector Regression	0.1184, 0.915	0.1064, 0.935	0.1164, 0.920
Random Forest Regression	0.0506, 0.984	0.1289, 0.904	0.1315, 0.898

Table 5: Results with First 146 Principal Components

	Results: (RMSE, R^2)		
Model	Training	Validation	Test
Linear Regression without Regularization (OLS)	0.1152, 0.919	0.1218, 0.914	0.1209, 0.914
Linear Regression with L2 Regularization	0.1188, 0.914	0.1182, 0.919	0.1220, 0.912
Lasso Regression	0.1205, 0.912	0.1178, 0.920	0.1241, 0.909
Support Vector Regression	0.1224, 0.909	0.1092, 0.931	0.1174, 0.919
Random Forest Regression	0.0688, 0.981	0.1775, 0.818	0.1912, 0.785

For linear regression without regularization, using a selected set of features largely solves the failed generalization problem. However, for all other models, prediction made with fewer features perform strictly worse. Because all models with regularization[14] can learn which features to use, and giving these models less information will

[14]In SVR and Random Forest there is also regularization, although we don't explicitly call something regularization term.

result in strictly worse performance.

Compare table 3 with table 2, table 5 with table 4, we can see that selecting features work strictly better than transforming features with PCA. Because PCA is a lossy data transformation, and it destroys our data. So don't use principal components before making house price prediction. There are many feature selection methods in the supervised learning field, and Lasso is only one of them. If we compare these four tables with table 1, we would agree that a good model doesn't require feature selection beforehand, at least for this house price prediction task.

When we compare results of the different methods within table 2, we see that non-linear models perform strictly better than linear models. Non-linearity models are able to capture more complicated relationships within data, which is an even larger advantage over linear models when the number of features is limited.

8 Conclusion

Here are the things to remember when making house price prediction:

1. The prediction result has to be evaluated using a test dataset that is not included during the training.
2. Always add regularization term (explicitly or implicitly) to the model. When the number of features is limited, it doesn't harm. When the number of features is huge, it saves your life.
3. Principal component analysis is helpful for data visualization, but not for feature selection.
4. Choose support vector regression with no feature selection.

References

Athey, S. (2016). How will machine learning impact economics? `https://www.quora.com/How-will-machine-learning-impact-economics?redirected_qid=6706789`. Accessed: 2017-04-10.

Basak, D., S. Pal, and D. C. Patranabis (2007). Support vector regression. *Neural Information Processing-Letters and Reviews 11*(10), 203–224.

CMU, S. Ridge regression and lasso regression. `http://www.stat.cmu.edu/~ryantibs/datamining/lectures/17-modr2.pdf`.

Cook, J. D. (2010). Don't invert the matrix. `https://www.johndcook.com/blog/2010/01/19/dont-invert-that-matrix/`. Accessed: 2017-04-10.

Costanigro, M., J. J. McCluskey, and R. C. Mittelhammer (2007, 9). Segmenting the wine market based on price: Hedonic regression when different prices mean different products. *Journal of Agricultural Economics 58*(3), 454–466.

De Cock, D. (2011). Ames, iowa: Alternative to the boston housing data as an end of semester regression project. *Journal of Statistics Education 19*(3).

Konstantin. The meaning of gaussian kernel. `http://fouryears.eu/2016/11/11/the-meaning-of-the-gaussian-kernel/`.

Liaw, A. and M. Wiener (2002). Classification and regression by randomforest. *R news 2*(3), 18–22.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Selim, H. (2009). Determinants of house prices in turkey: Hedonic regression versus artificial neural network. *Expert Systems with Applications 36*(2, Part 2), 2843 – 2852.

Shlens, J. (2014). A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*.

Yoo, S., J. Im, and J. E. Wagner (2012). Variable selection for hedonic model using machine learning approaches: A case study in onondaga county, ny. *Landscape and Urban Planning 107*(3), 293–306.

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.