Business 41204

Machine Learning

Notes 1 – Supervised Learning Basics

Christian Hansen

University of Chicago
Booth School of Business

Damian Kozbur

University of Zurich
Economics

Outline: (??)

1. Introduction
2. Supervised Learning: Setup
3. Simple Example to Fix Ideas
4. Validation and Bias/Variance Tradeoff
5. House Price Prediction: Review Linear Models
6. House Price Prediction: Trees
7. House Price Prediction: Categorical Variables
8. Summary

1. Introduction

What is Machine Learning?

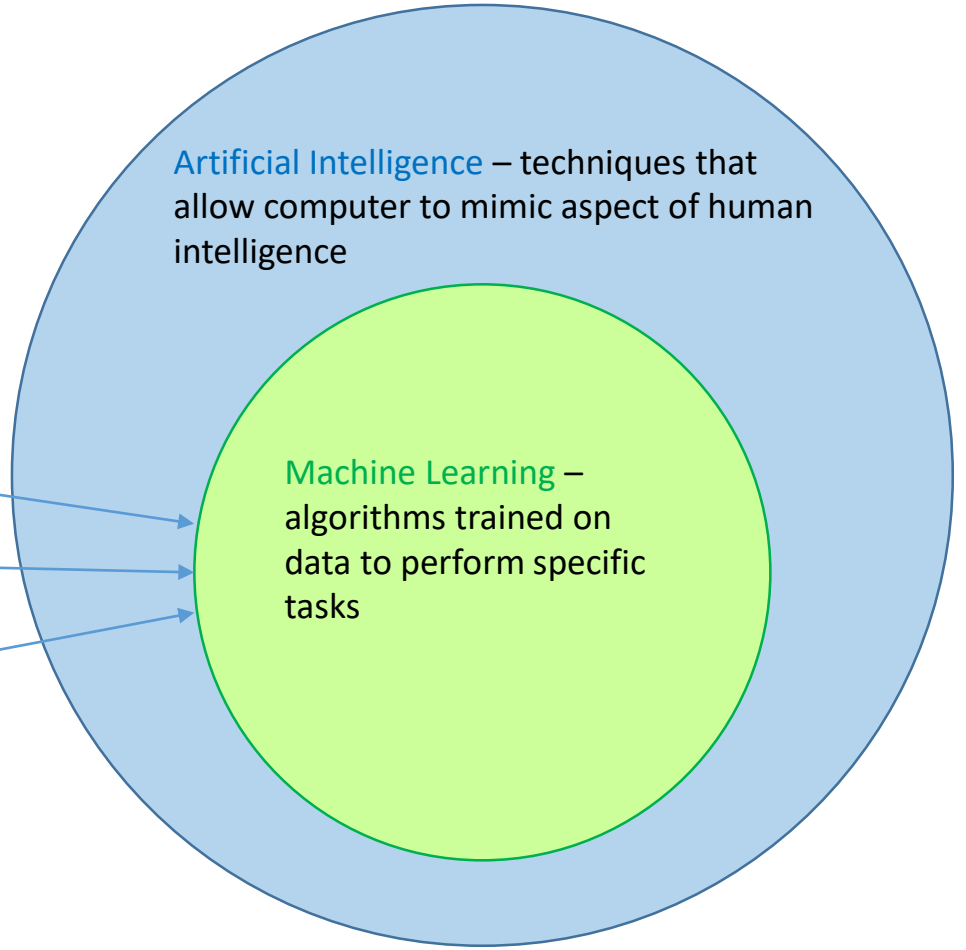
Subfield of AI that uses algorithms trained on data to perform desired tasks

Prominent Types of Machine Learning

Supervised Learning – Learn with *labeled data*

Unsupervised Learning – Extract patterns from *unlabeled data*

Reinforcement Learning – Learn from reward maximization by exploration (creating *new data*)



What is Machine Learning?

Subfield of AI that uses algorithms trained on data to perform desired tasks

ML tends to be outcome oriented:

Does the algorithm do what we want?

- Focus on evaluation in task we want done
- May care less about understanding all the details about why
  - i.e. many ML algorithms are "*black box*"

Source: https://xkcd.com/1838/

Examples of ML Tasks

Supervised Learning

(aka Predictive Analytics)

- Revenue forecasting
- House price forecasting
- Demand prediction
- Customer churn/retention
- Image classification
- Asset return forecasting
- Volatility forecasting
- …

Unsupervised Learning

- Market segmentation
- Recommender systems
- Image generation
- Anomaly detection
- …

Reinforcement Learning

- Gaming (e.g. alphaGo)
- Self-driving cars
- Autonomous robots
- Industrial automation
- …

Course Goals

1. Understand key ML concepts and paradigms
   - Surface understanding of key modeling approaches/algorithms

2. Be able to think critically about ML projects
   - What is achievable?
   - Ethical/regulatory considerations

3. Course is NOT meant to make you an ML engineer
   - Hopefully does give some skills in this direction
   - Helps you meaningfully interact with ML teams

Course structure

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning
4. Generative models/LLMs
5. Topics

2. Supervised Learning: Setup

2. Supervised Learning

Basic goal:

Produce rule to use input X to predict target Y

- X represents observed features such as
  - Customer demographics – age, income, education, …
  - Product characteristics – price, text description, product image, …
- Y represents observed outcome/target (supervisor) such as
  - Whether customer purchases product
  - Quantity of product sold in last week
  - Type of animal in an image

Conceptualization

Useful to think about relation between target (Y) and input (X) as

$$\underbrace{Y}_{\text{target}} = \underbrace{f(X)}_{\text{signal}} + \underbrace{\varepsilon}_{\text{noise}}$$

- $f(X)$ is rule that takes input X and produces a prediction for Y - $\hat{Y} = f(X)$
- ε captures the "noise" or irreducible error in predicting Y
  - E.g. imagine X represents the size of a house – not all houses of the same size have the same price – with only size, even the best prediction rule will have error

GOAL: Use data on Y, X to learn a forecast rule $f(X)$ that produces small forecast error

Loss Function

Need a way to judge what we mean by “small forecast error”.

Use a loss function: $L\big(Y, f(X)\big)$

Common loss functions:

- Squared error loss: $L\big(Y, f(X)\big) = \big(Y - f(X)\big)^2$
  - Most common used loss for regression problems
- Absolute error loss: $L\big(Y, f(X)\big) = |Y - f(X)|$

In principle, can use more tailored loss functions.

Expected Loss Minimization

With our measure of gauging forecast quality defined, can carefully define our target:

Produce a prediction rule $f(X)$ that provides small *expected* loss when used to forecast an *unseen* outcome Y for a new observation with characteristics X. I.e. find

$$f(X) = \text{argmin}_g \, E[L(Y, g)]$$

- Using notion of a population
- Don't know population expectation, so use a *sample*

Empirical Risk Minimization

Have a sample of $\mathrm{i} = 1, \ldots, n$ observations on

- y_i - outcome variable
- x_i - p predictor/input/feature variables (may be produced by *transformations* of underlying input variables)

ML algorithms typically obtain prediction rules by solving the sample version of expected loss minimization:

$$\hat{f}(X) = \operatorname{argmin} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))$$

- Computers are pretty good at solving these problems
- Different ML procedures correspond to different choices for allowable $\hat{f}(X)$

Model Fit

Let $\hat{f}$ be an estimated prediction rule.

Suppose we have $j = 1, \dots, m$ observations with observed x_j and y_j

- Could be the n observations in the sample used to learn the prediction rule
- Should be new observations (more in a couple slides)

Can estimate loss/model fit with sample average loss: $\frac{1}{m}\sum_{j=1}^{m} L\left(y_j, \hat{f}\left(x_j\right)\right)$

E.g.

- Mean squared error (MSE) = $\frac{1}{m}\sum_{j=1}^{m}\left(y_j - \hat{f}\left(x_j\right)\right)^2$
- RMSE = $\sqrt{MSE}$

R^2

Let $\hat{f}$ be an estimated prediction rule and $\hat{f}_0$ be a baseline prediction rule.

Can define $R^2 = 1 - \frac{MSE(\hat{f})}{MSE(\hat{f}_0)}$

- $R^2 \leq 1$
- $R^2 = 1$: predicted values perfectly match observed values on observations used to compute MSE
- $R^2 = 0$: candidate model performance same as baseline model performance
- $R^2 < 0$: candidate model performance worse than baseline model performance
- Typically, baseline model is just sample mean $(\hat{f}_0(x) = \bar{Y})$ which predicts the same value no matter the input

Can use other average loss measures to define performance measures relative to a baseline analogous to R^2.

3. Simple Example to Fix Ideas

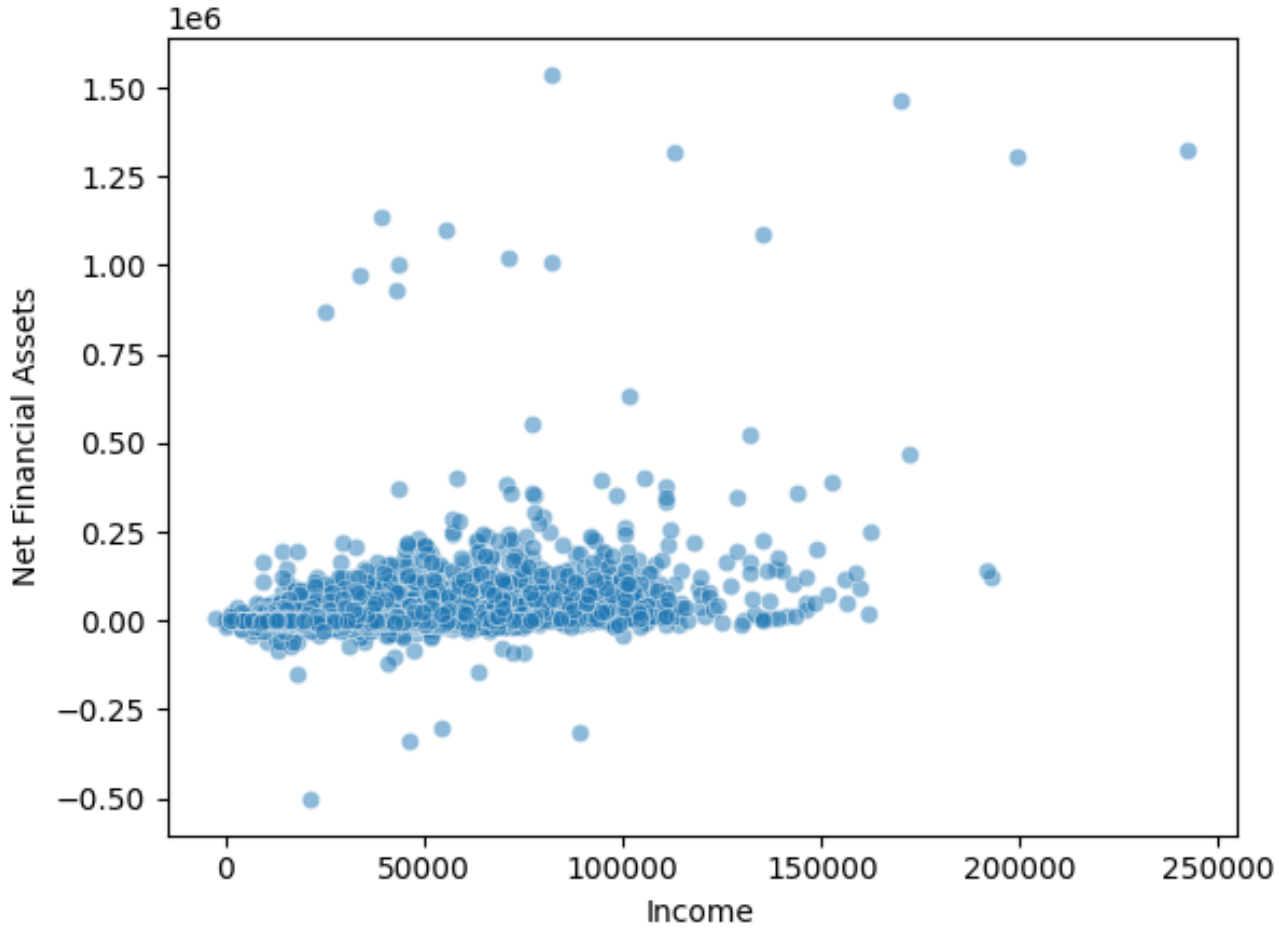
Code for Example 1

Example: Financial Asset Prediction

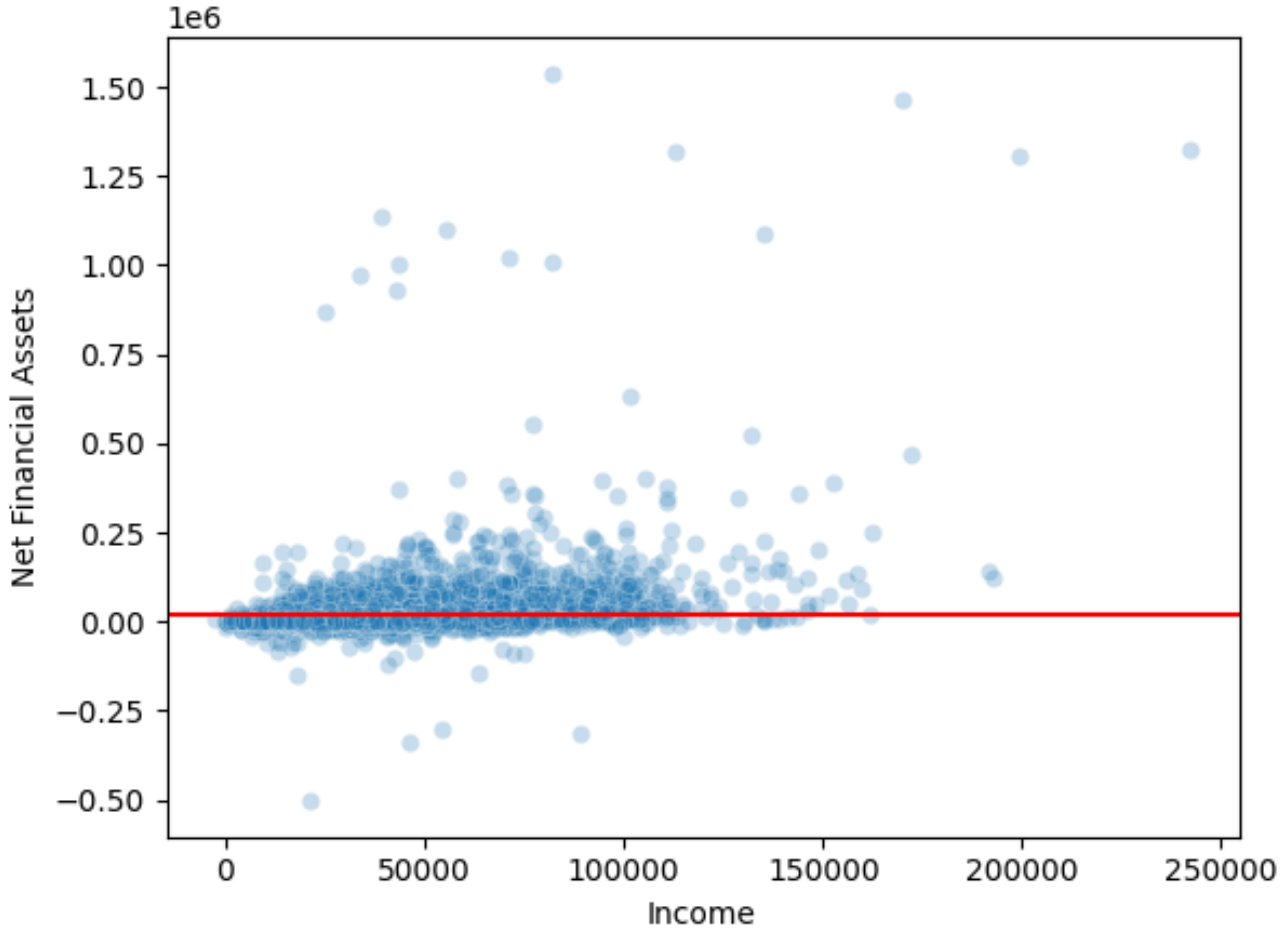
Let's fit some models to predict a household's net financial assets using household income as an input.

Using examples that are simple enough to visualize and run quickly.

[Code for Example 1]



Baseline Model: Constant/sample mean



Sample mean of outcome: 18752.76

MSE of prediction: 4950940890.22

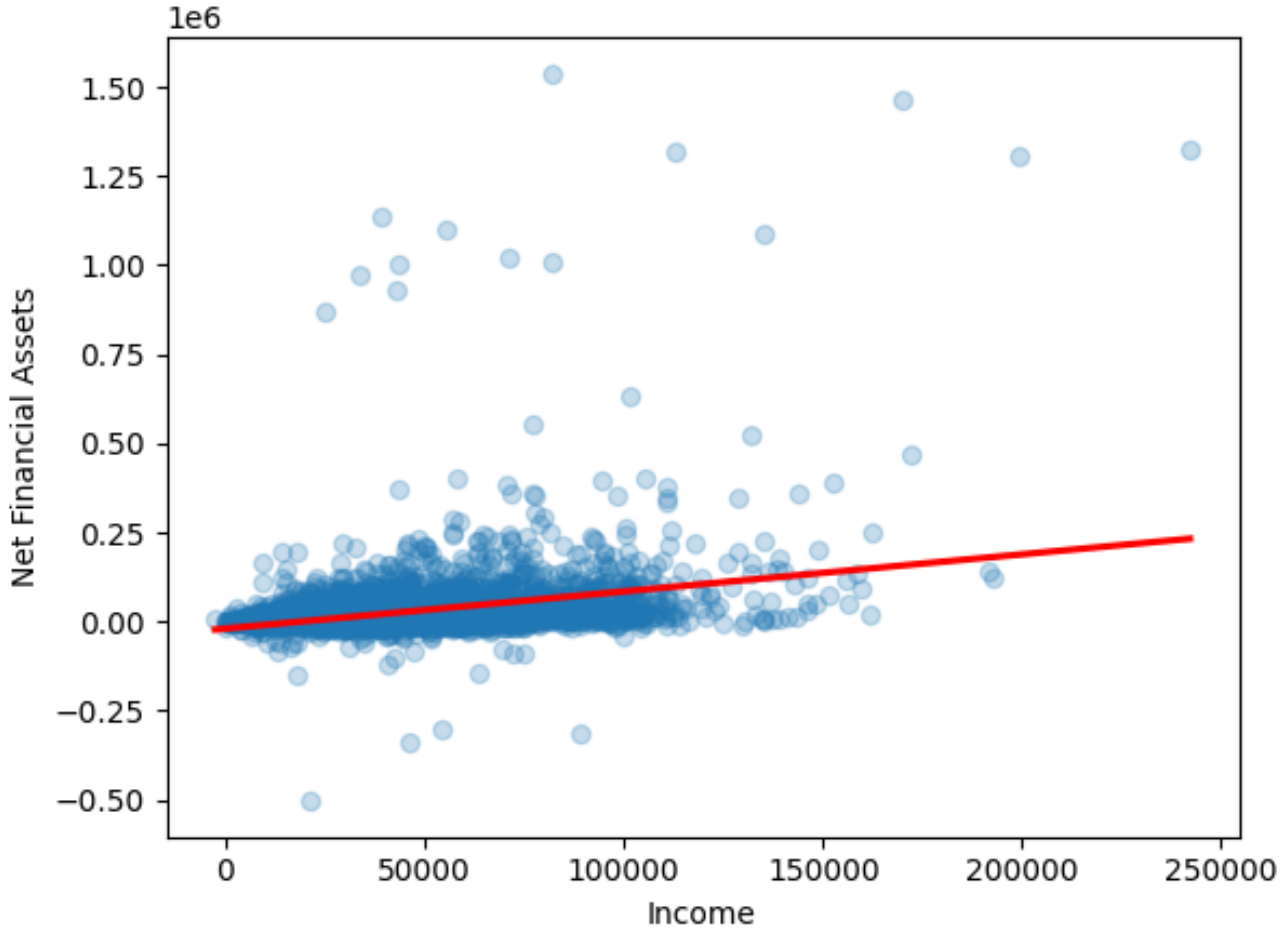
(pretty hard to interpret in isolation)

RMSE: 70362.92
MAE: 27997.19

Prediction rule ignores income.

Can we do better?

Linear Model



Sample mean of outcome: 18752.76

MSE of prediction: 4298860504.96

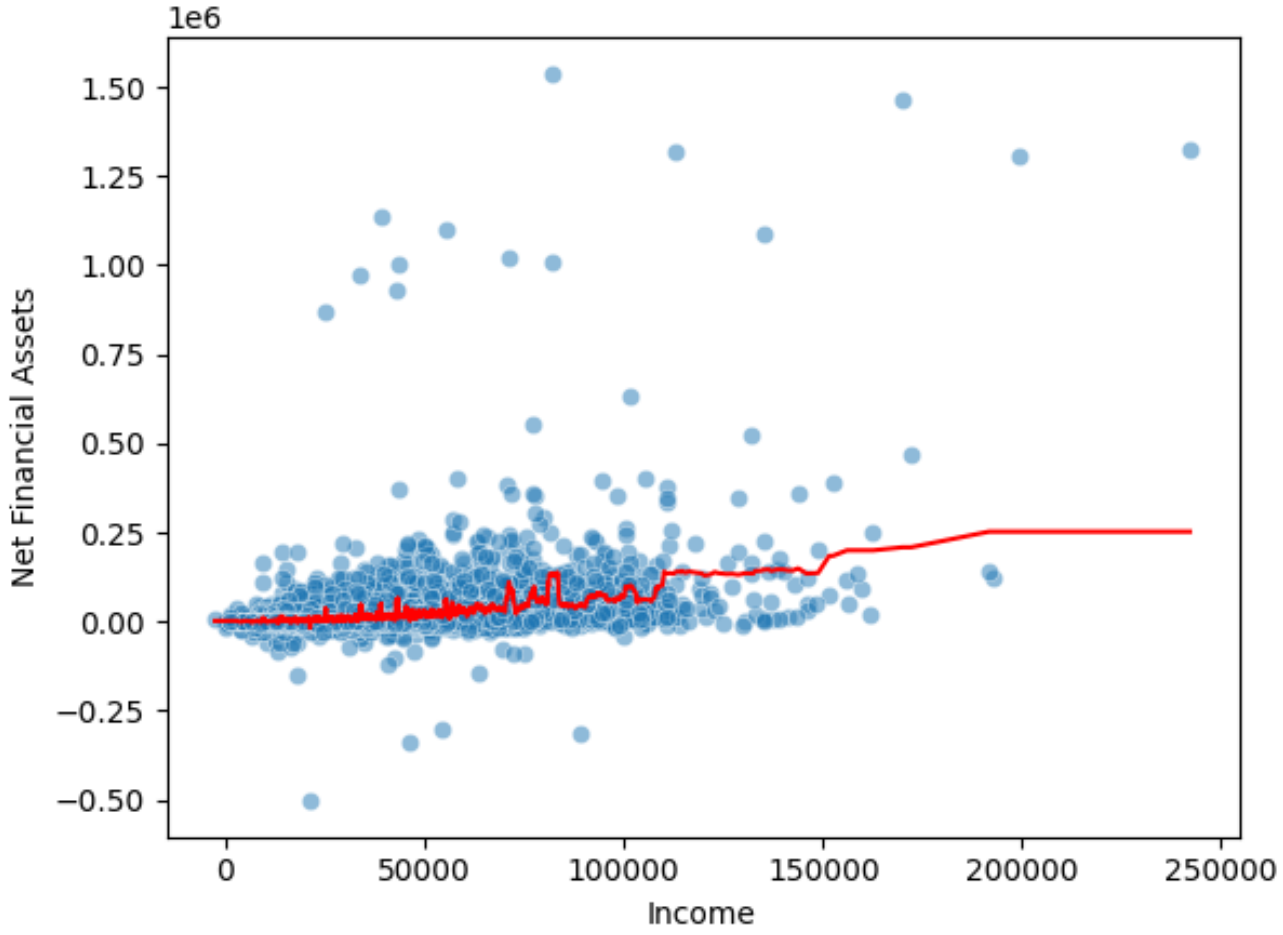
RMSE: 65565.70
MAE: 25193.63

$R^2 = 0.132$

Recall that using just the sample mean we had

MSE: 4950940890.22
RMSE: 70362.92
MAE: 27997.19

KNN with 30 Neighbors



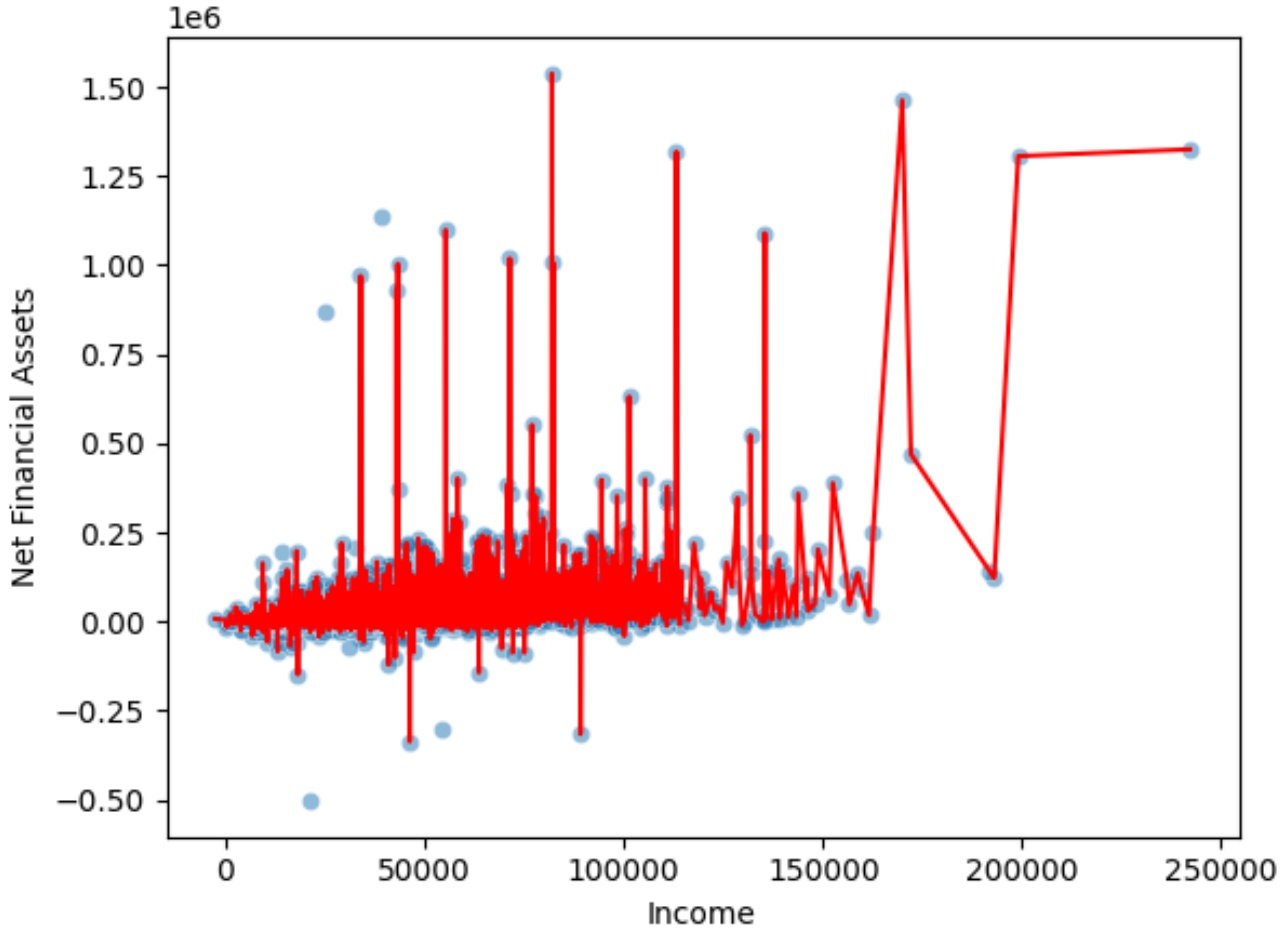
Sample mean of outcome: 18752.76

MSE of prediction: 4076203154.85

RMSE: 63845.15
MAE: 23073.23

$R^2 = 0.177$

KNN with 1 Neighbor



Sample mean of outcome: 18752.76

MSE of prediction: 610846385.67

RMSE: 24715.31
MAE: 3817.69

$R^2 = 0.877$

We have almost perfectly fit/memorized the data!

- We could find an ML procedure that does memorize the data – computers are good at this.

Overfitting/Underfitting

Fundamental problem:

- Find model which is not so simple it misses patterns in data
  - avoid underfitting
- Find model which is not so flexible it captures non-generalizable patterns in data
  - avoid overfitting

How do we decide?

With one input and moderate numbers of observations, we can use our eyes and intuition. We want something more general.

4. Validation and Bias/Variance Tradeoff

Code for Example 1

Prediction Loss

Remember: Our goal is to build a rule to predict *new* observations ("out-of-sample fit")

- When we build a model, we learn how well that model fits the observations used to build it ("in-sample fit")
- In-sample fit approximates out-of-sample fit when n is large and model is "not complex"
- **Most ML methods are "complex" though**

There are ways to formally define complexity. Rather than worry about this, we're just going to note that most ML methods are complex and that validation exercises work for simple AND complex methods.

Punchline: We don't judge model quality on in-sample fit but use validation exercises.

A validation exercise aims to replicate the forecasting environment by splitting data into

- training data – used to learn prediction rules
- testing/validation/hold-out data – used to test estimated rules on new observations
  - Can use same loss metrics as for training

Validation Exercise on Simple Example

How do our simple financial asset prediction rules do in a validation exercise?

- Look at prediction performance in a held out sample of 3915 observations

	In-sample			Out-of-sample		
	RMSE	MAE	R^2	RMSE	MAE	R^2
Mean	70362.92	27997.19		51300.56	27068.98	
Linear	65565.70	25193.63	0.132	45792.54	23361.46	0.203
KNN30	63845.15	23073.23	0.177	46512.42	22207.10	0.178
KNN1	24715.31	3817.69	0.877	77966.24	30176.01	-1.310

MSE vs MAE and "outliers"

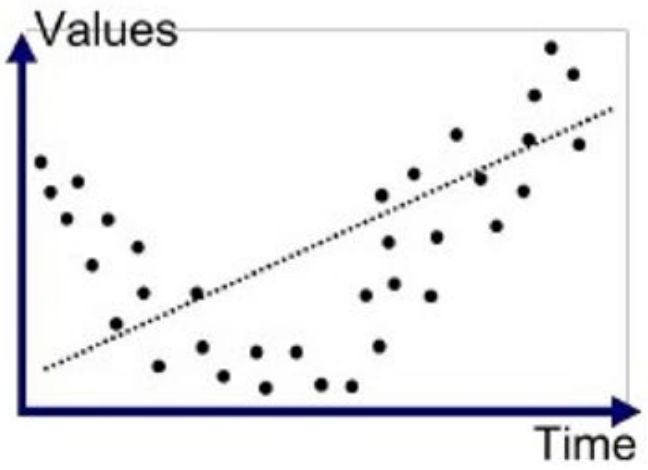
Which rule should we use?

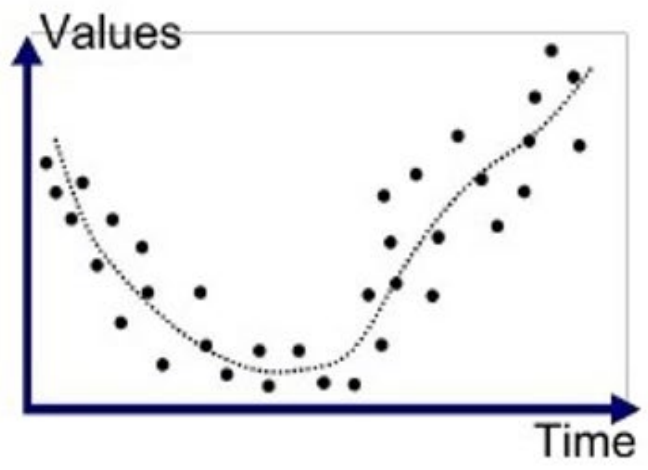
What happened to KNN1?

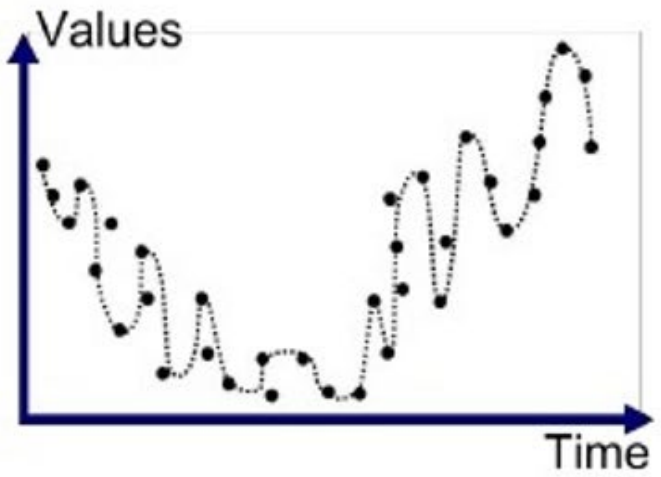
Code for Example 1

Bias/Variance Tradeoff

Can think of "Bias/Variance tradeoff" as shorthand jargon for choosing a rule with the right complexity for the task at hand. Not going to worry about formalizing.







Underfitted

When we build models that are too simple – underfit – we end up with a "biased" prediction rule.

Good Fit/Robust

Good predictive models trade off "bias" and "variance" to land in a sweet spot.

Overfitted

When we build models that are too complex – overfit – we end up with high "variance" prediction rule.

You cannot tell where you are by only looking at the data used to fit the model

⇒ Validation exercise

Train/Validate/Test

Ideally, validation exercises are actually based on splitting the data into three pieces:

- Training data: Data used to fit the candidate prediction rules
  - Want most of the data here
- Validation data: Data used to test out/fine tune prediction rules
- Test data: Final data set used to check performance of model before going into production/giving it to client
  - Training and test data sets typically of similar size

Cross-Validation

Potential issues with basic validation exercise

- Doesn't use all data for training and testing
- Depends on the specific random split

Cross-validation (CV) aims to alleviate these concerns – essentially by repeating the validation exercise

CV also has issues

- Computationally more burdensome that simple validation exercise
- Hard to do with time-series (or with data that is not independent)
- Have to decide what model to use when you're done
- Using the same data repeatedly

K-Fold CV Algorithm:

1. Divide sample into K (approximately) equal size groups
2. For $k = 1, \ldots, K$
    1. Set aside subsample k as validation data
    2. Use remaining subsamples as training data. Call these training data T_k
    3. Use training data T_k to estimate candidate prediction rules, $\hat{f}_k(x)$
    4. Form predictions for observations i in subsample k, $\hat{y}_i = \hat{f}_k(x_i)$
3. Estimate forecast loss as $\frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$
    - E.g. CV-MSE = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Cross-validation in Simple Example

Let's see what 5 Fold cross validation looks like in our financial asset prediction example

- 5 Fold CV here is based on the 6000 training observations (i.e. identical to the **In-sample** data)

	In-sample		**5-Fold CV**		**Out-of-sample**	
	RMSE	R^2	RMSE	R^2	RMSE	R^2
Mean	70362.92		70365.85		51300.56	
Linear	65565.70	0.132	65669.88	0.129	45792.54	0.203
KNN30	63845.15	0.177	65853.75	0.124	46512.42	0.178
KNN1	24715.31	0.877	89571.92	-0.620	77966.24	-1.310

Cross-validation and evaluation in the hold out sample tell roughly the same story.

5. House Price Prediction: Review Linear Models

Code for House Price Example

Interpretable Supervised Learners

Many ML methods are black-box ≈ hard to directly look at model and interpret

- talk about these in the next set of notes

There are "interpretable" supervised learners

- meaning they produce models that are "straightforward" to interpret

Leading examples of interpretable learners:

- linear models (including penalized linear models)
- trees (classification and regression trees specifically)
- logistic regression for classification (later…)

Jargon:

Regression: Predict "continuous" outcome

Classification: Predict "discrete" outcome

We'll talk more about classification later.

House Price Prediction

Let's look at linear models and trees in the context of house price prediction.

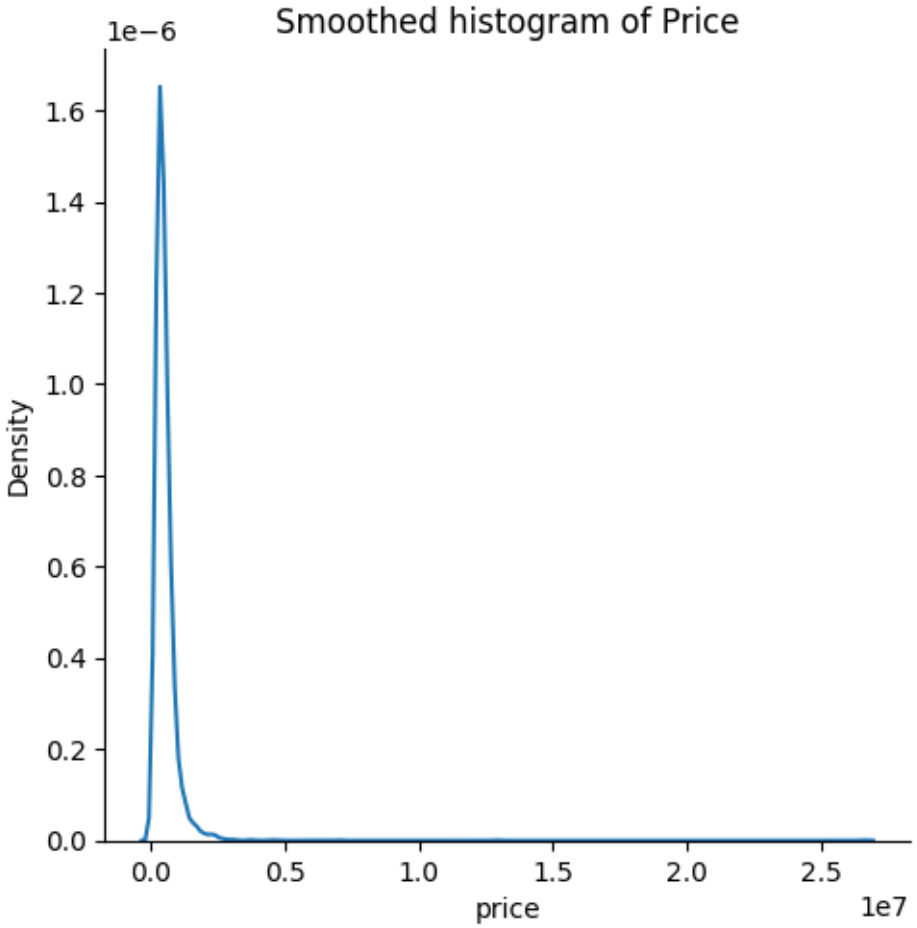
- Why might we be interested in predicting sales prices of houses?
- How might this interest influence our loss function for evaluating prediction models?

Data:

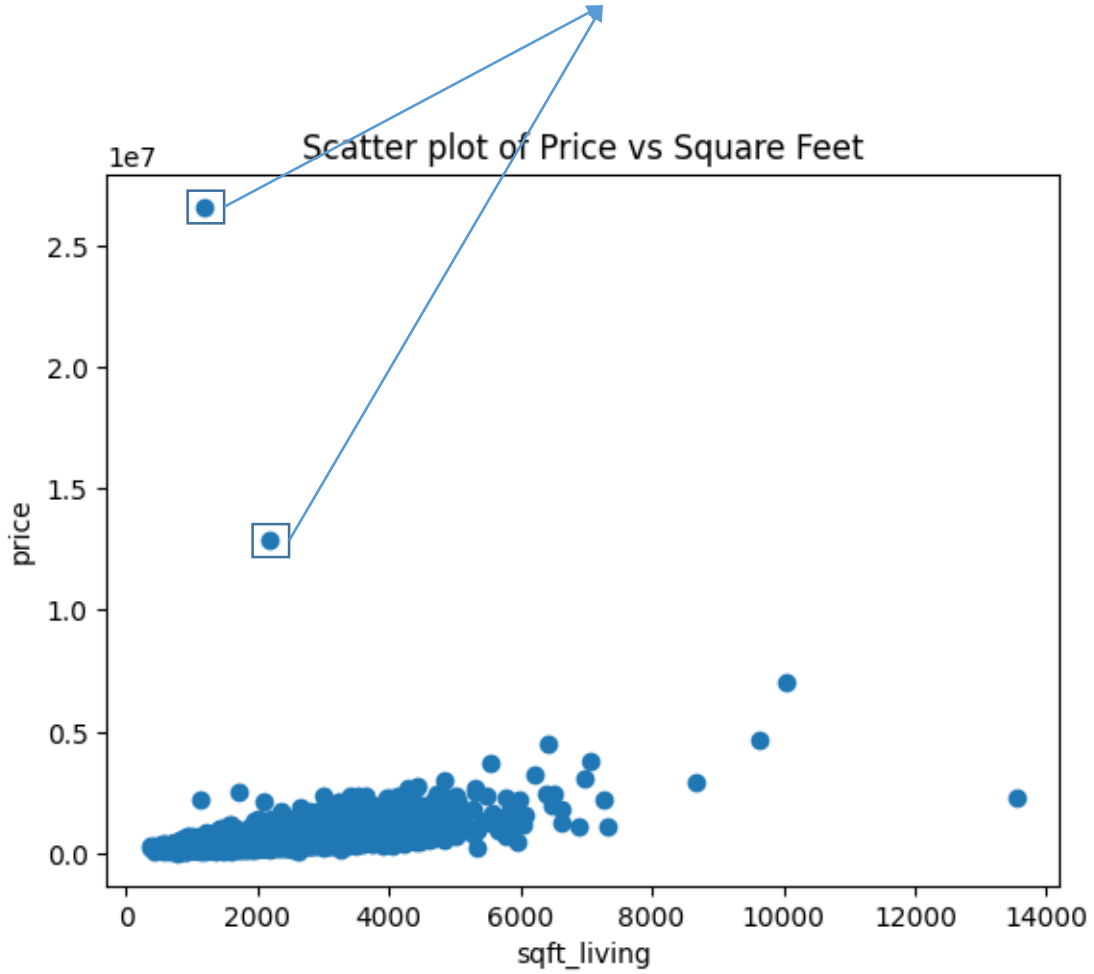
- $n = 4551$ houses that sold in the Seattle, WA area May-July 2014.
  - What should we think about if we wanted to predict house prices today?
- $p = 18$ raw features

Code for House Price Example

House Prices



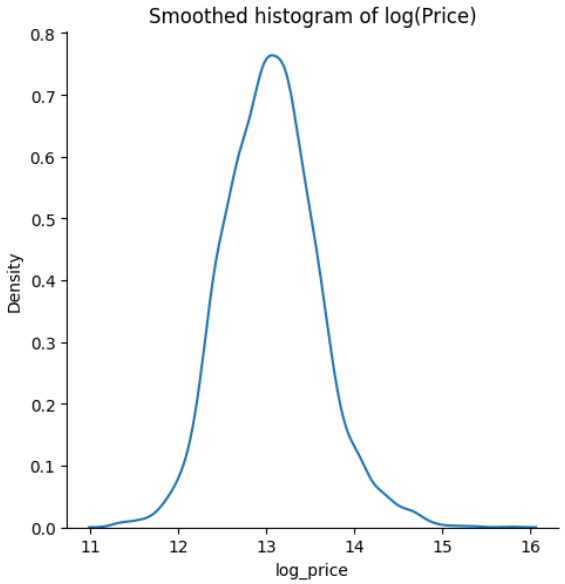
There are a couple of troubling observations. What should we do?

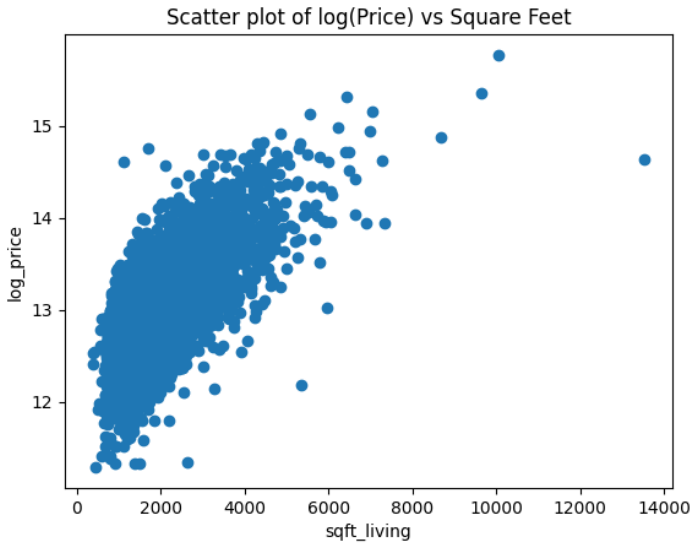


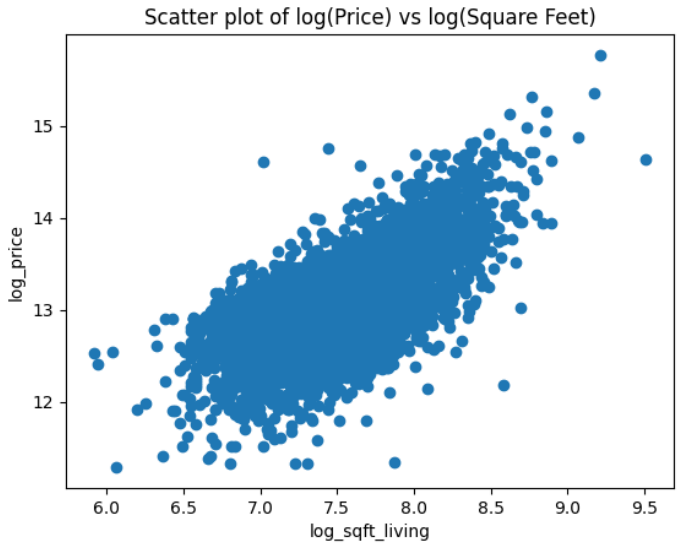
Code for House Price Example

log(House Prices)

Note that I have removed some observations. See the notebook for further discussion.







When Y is highly right skewed and positive, it is common to see model's built for $\log(Y)$ instead.

- Sometimes a good idea (not always)
- Remember that your goal is to predict Y – Need to evaluate model performance in terms of predicting target!
  - A model that predicts $\log(Y)$ well may not do as well for Y
- Using log(square feet) also looks like a good idea

Code for House Price Example

Other Variables

Outside of price, the data includes features

- `date` – date of sale.
- `sqft_living, sqft_lot, sqft_above, sqft_basement` – square footage variables
  - Note `sqft_living = sqft_above+sqft_basement`
- `bathrooms, bedrooms, floors` – numeric variables with small number of values
- `waterfront, view, condition` – ordered categorical
- `yr_built, yr_renovated` – year built and year of most recent renovation
- `street, city, statezip, country` – categorical/qualitative variables

Which should we use? How should we use them?

Code for House Price Example

Linear Models: Review

A linear model is just a prediction rule of the form

$$\hat{Y} = b_0 + b_1 x_1 + b_2 x_2 + \ \ldots + b_p x_p$$

- $x_1, \ldots x_p$ are values of the features we want to make a prediction at
- b_0 is the "estimated intercept" and $b_1, \ldots, b_p$ are the "estimated slopes"
  - b_0 is often referred to as the "offset" in ML circles
- $\hat{Y}$ is the predicted value

Linear models are interpretable:

- We can literally write down the prediction rule as above
- The estimated coefficients tell us exactly how our predictions change as we change values of the features

Linear Models: Review

To form our linear prediction rule, we need to learn parameters $b_0, b_1, \ldots, b_p$ from the training data.

OLS (ordinary least squares) chooses $b_0, b_1, \ldots, b_p$ to minimize MSE:

$$(b_0, b_1, \ldots, b_p) = \arg\min \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_{1,i} - \cdots - \beta_p x_{p,i}\right)^2$$

- Training observations are $(y_i, x_{1,i}, \ldots, x_{p,i})$ for $i = 1, \ldots, n$

Typically what people refer to as linear regression or the linear regression model.

Linear Models: Aside

Linear prediction rules can be learned by targeting loss functions other than MSE.

- Will generally lead to different prediction model than OLS linear regression

E.g. quantile regression chooses $b_0, b_1, \ldots, b_p$ to minimize

$$\frac{1}{n}\sum_{i=1}^{n} \rho_\tau\left(y_i - \beta_0 - \beta_1 x_{1,i} - \cdots - \beta_p x_{p,i}\right)$$

$$\rho_\tau(u) = \begin{cases} \tau|u| & u > 0 \\ (1-\tau)|u| & u \leq 0 \end{cases}$$

Puts different weight on positive and prediction errors.

Special case with $\tau = 1/2$ corresponds to minimizing mean absolute error (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

(also referred to as least absolute values (LAV) or median regression)

Linear Models: Review

Linear models are very general and can capture pretty general patterns.

E.g. the prediction rule

$$\begin{aligned}\widehat{Price} &= b_0 + b_1 square\ feet + b_2 (square\ feet)^2 \\ &= b_0 + b_1 X_1 + b_2 X_2\end{aligned}$$

is linear ($X_1 = square\ feet$ and $X_2 = (square\ feet)^2$)

E.g. the prediction rule

$$\begin{aligned}\widehat{Price} &= b_0 + b_1 square\ feet + b_2 bedrooms + b_3 (square\ feet * bedrooms) \\ &= b_0 + b_1 X_1 + b_2 X_2 + b_2 X_3\end{aligned}$$

is linear ($X_1 = square\ feet$, $X_2 = bedrooms$, and $X_3 = square\ feet * bedrooms$)

Important: Coefficients in these models cannot be interpreted in isolation!

Prediction from log model

Suppose we have build a prediction rule for log(price)

$$\widehat{\log(price)} = b_0 + b_1 x_1 + \cdots b_p x_p$$

Recall that our goal is to predict price.

A very simple way to proceed is by taking

$$\widehat{price} = \exp\left(\widehat{\log(price)}\right)$$

- Recall that $\exp(\log(u)) = u$.
  - There are some subtleties here if the prediction model for log price was estimated using OLS and the model is taken very seriously
    - Usually works decently (e.g. Bårdsen and Lütkepohl (2011) "Forecasting levels of log variables in vector autoregressions" *International Journal of Forecasting* 27(4), 1108-1115.)
    - Can be formally motivated under symmetry and arguing that median is being used as prediction rule
    - Less of an issue if LAV is used

Linear Regression in House Price Example

Let's look at some simple linear regression models.

Baseline:

- Start by considering only "generalizable" features: `sqft_living, sqft_lot, sqft_above, sqft_basement, bathrooms, bedrooms, floors, waterfront, view, condition, yr_built, yr_renovated`
  - Note: All square feet variables are actually as "logs": e.g. we're actually using log(`sqft_living`)

Polynomial:

- Include squares of all "continuous" variables
- Interact `sqft_living` with `bathrooms` and `bedrooms`

Will use each of `price` and `log_price` as outcome

Will evaluate models using 5-fold CV based on MSE and MAE

- Baseline prediction rule for MSE will be sample mean of price
- Baseline prediction rule for MAE will be sample median of price

Baseline Model Performance

Model	RMSE	R^2	MAE	Pseudo R^2	p
Mean	368147		236705		
Median	377706		226012		
Baseline	262124	0.493	167493	0.259	13
Baseline – Log	244934	0.557	152559	0.325	13
Polynomial	240846	0.572	153972	0.319	24
Polynomial – Log	240605	0.573	149432	0.339	24
Polynomial (LAV)	244155	0.560	150840	0.333	24
Polynomial (LAV) - Log	237723	0.583	149311	0.339	24

The LAV model trained with log(`price`) does best (by a little).

Have we included all relevant features? Should we allow more interactions, higher order polynomials? Other transformations? Do we need all the features we have?

Code for House Price Example

A Flexible Model?

Maybe we should be more flexible to try to capture any neglected nonlinearity:

1. 4th degree polynomial in `bathrooms`, `bedrooms`, `sqft_lot`, `floors`, `yr_built`, `yr_renovated`
2. 6th degree polynomial in `sqft_living`
3. Dummy variables for all categories of `waterfront`, `view`, `condition`, `renovated_flag`, `basement_flag`
4. All first order interactions of 1-3

Still considering only “generalizable features”

We’ll call this the “flexible model”

Flexible Model Performance

Model	RMSE	R^2	MAE	Pseudo R^2	p
Flexible	≫1M	≪-10	≫1M	≪-10	799
Flexible – Log	≫1M	≪-10	≫1M	≪-10	799
Polynomial (LAV) - Log	237723	0.583	149311	0.339	24

The flexible model does ridiculously bad. It is wildly overfit.

Does this mean nothing in the flexible model could help us?

If we have the universe of features we want to consider for predicting the outcome, we may want to try to find those that are most useful rather than just use all of them.

Code for House Price Example

Lasso

Lasso is a popular approach to estimating linear models that does variable selection while fitting a prediction rule

- i.e. Lasso drops variables that do not seem to strongly improve predictions

Technically Lasso solves the *penalized* least squares problem

$$\min \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \beta_1 x_{1,i} - \cdots - \beta_p x_{p,i}\right)^2 + \lambda \sum_{j=1}^{p}\left|\beta_j\right|$$

Jargon: *Regularization.* Adapting the model fitting process to alleviate overfitting by encouraging "simpler" models

The key is the *penalty term* $\lambda \sum_{j=1}^{p}\left|\beta_j\right|$.

- Imposes cost (λ) of moving increasing magnitude of coefficient.
- Means that a variable that does not improve predictive performance more than the "cost" will be excluded from the model. Lasso chooses the "relevant" features for us!
- Choice of λ determines cost/benefit tradeoff – We typically choose λ by trying many and choosing the best according to (cross-)validation performance

Lasso in the Price Example

used (# available)

Model	RMSE	R^2	MAE	Pseudo R^2	p
Polynomial (LAV) - Log	237723	0.583	149311	0.339	24
Polynomial Lasso	242259	0.567	153547	0.321	19 (24)
Polynomial Lasso – Log	245162	0.557	151620	0.329	13 (24)
Flexible Lasso	241218	0.571	152848	0.324	44 (799)
Flexible Lasso – Log	235301	0.591	149896	0.337	40 (799)

Using Lasso with the flexible model, we use 40 of the available 799 predictors and produce a competitive model.

- Should we conclude these are the "right" 40 variables?
- Should we go farther? What if there are things we haven't tried yet that would really help?

Code for House Price Example

Linear Models: Challenges

Basic issue holds more generally for any approach that does not do representation learning.

The chief difficulty of traditional linear models is choosing how to use the available features.

- What variables should we use?
- What transformations of those variables should we take?
- Should we interact variables? How many and which?
- With a given set of features, Lasso provides a convenient way to choose which to use
  - Many variants of Lasso and related ideas

With large datasets and many variables

- the burden of choice becomes overwhelming
- computation becomes challenging

Despite challenges, linear models provide a useful baseline and are widely used.

- particularly in settings with relatively limited data

More model ML methods bypass the need to specify exactly what transformations of the features to use by doing representation learning:

Algorithm simultaneously discovers how "best" to use features along with learning how to the assigned task

6. House Price Prediction: Trees

Code for House Price Example

Modern Regression

Modern regression aims to obtain a good prediction rule $f(X)$ ***without pre-specifying*** what $f(X)$ looks like

- Directly tries to learn how to represent $f(X)$ from the data – representation learning

Random forests, boosting, deep neural networks fall into this category.

- "black box" – hard to directly interpret.
- We'll tackle these in Notes 2.

Regression trees are simple representation learners that are (relatively) interpretable

- Tend to not predict particularly well in many examples

But

- the building block of more elaborate methods
- simple and fast

Trees

Basic idea of tree-based methods:

- Adaptively partition the feature space into regions
  - Usually rectangles
- Fit a simple prediction rule within each region
  - Usually a constant (e.g. the sample mean within the region)
- Partition and prediction rule jointly chosen to get good in-sample fit
  - Combination of choosing partition and prediction rule means trees are doing representation learning
  - Can choose to never partition on a feature – means this feature is “dropped”
  - In practice, just need to specify the (raw) features and target

Let’s just see what trees look like in the house price data

House Price Regression Trees

We're going to fit a regression tree to `price` using the "generalizable" features:

- `sqft_living, sqft_lot, sqft_above, sqft_basement, bathrooms, bedrooms, floors, waterfront, view, condition, yr_built, yr_renovated`
  - Recall that square feet variables are actually included as "logs"
  - This doesn't actually matter AT ALL for the predictions from a tree, but does matter when we look at the models

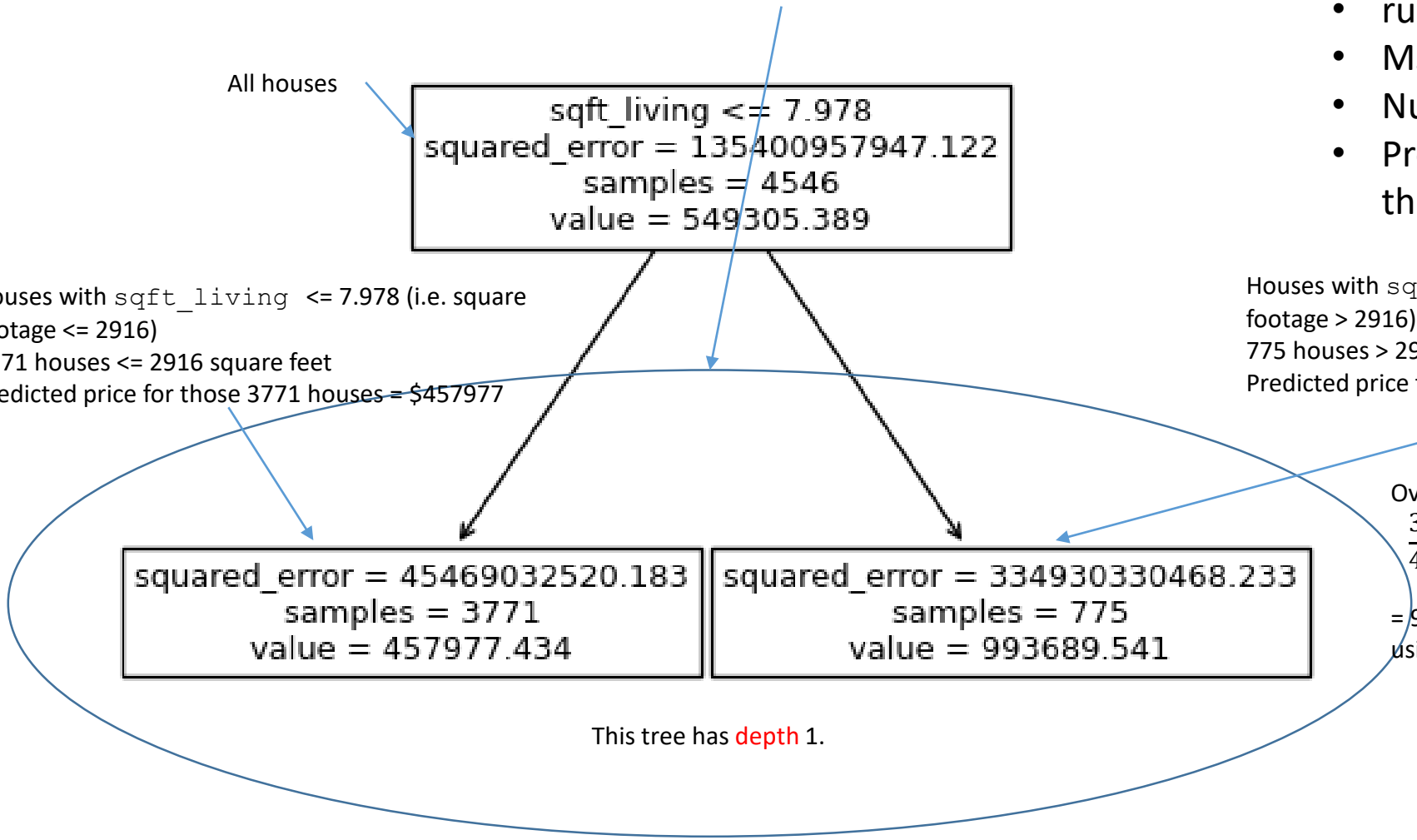
To illustrate, we're going to look at tree with 2, 3, 4, and 8 "leaves"/"terminal nodes"/"decision nodes"

Code for House Price Example

House Price Regression Trees

There are a variety of options for displaying trees. We're just using the simple default, which has plenty of information and is clear enough.

Tree for house price prediction with 2 leaves:



This tree has depth 1.

Each box displays

- rule for moving to next box
- MSE for predictions in that box
- Number of observations in the box
- Predicted value for observations in the box

Overall MSE of 2 leaf prediction rule:

$$\frac{3771}{4546} * 45469032520 + \frac{775}{4546} * 334930330468$$

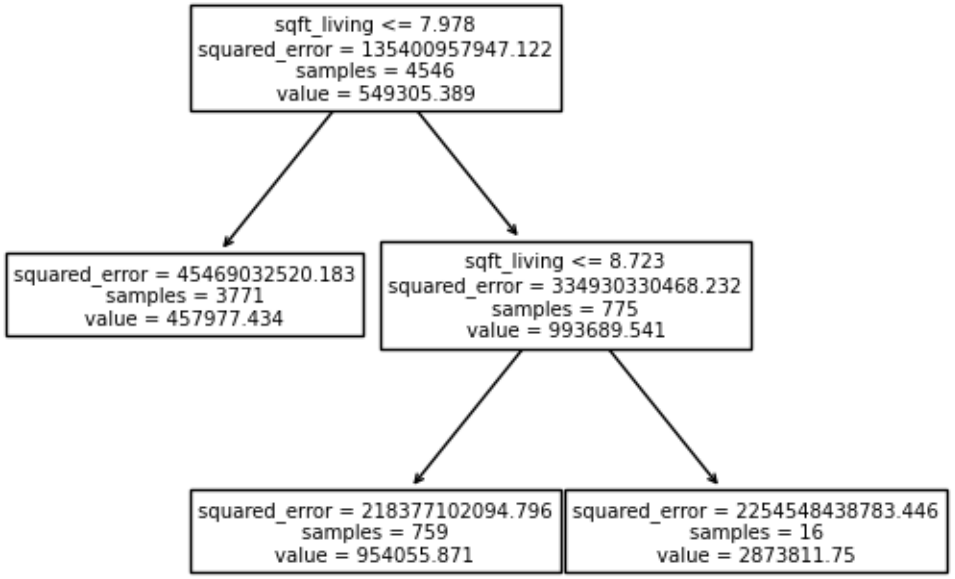
= 94816262153 < 135400957947 = MSE from just using sample mean

Code for House Price Example

Code for House Price Example

House Price Regression Trees

Tree with 3 leaves

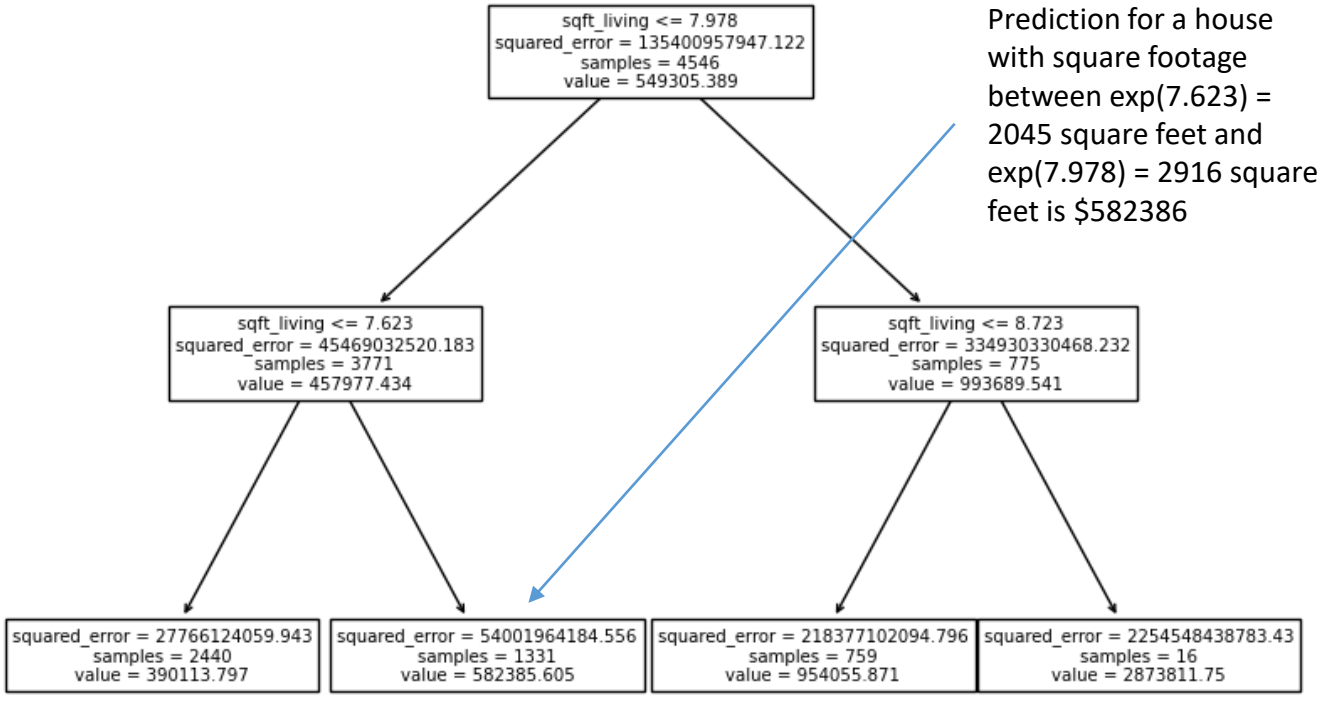


These trees have depth 2.

Tree with 4 leaves

Our 2, 3, and 4 leaf trees only use square footage even though ALL other features were considered.

Prediction for a house with square footage between exp(7.623) = 2045 square feet and exp(7.978) = 2916 square feet is $582386

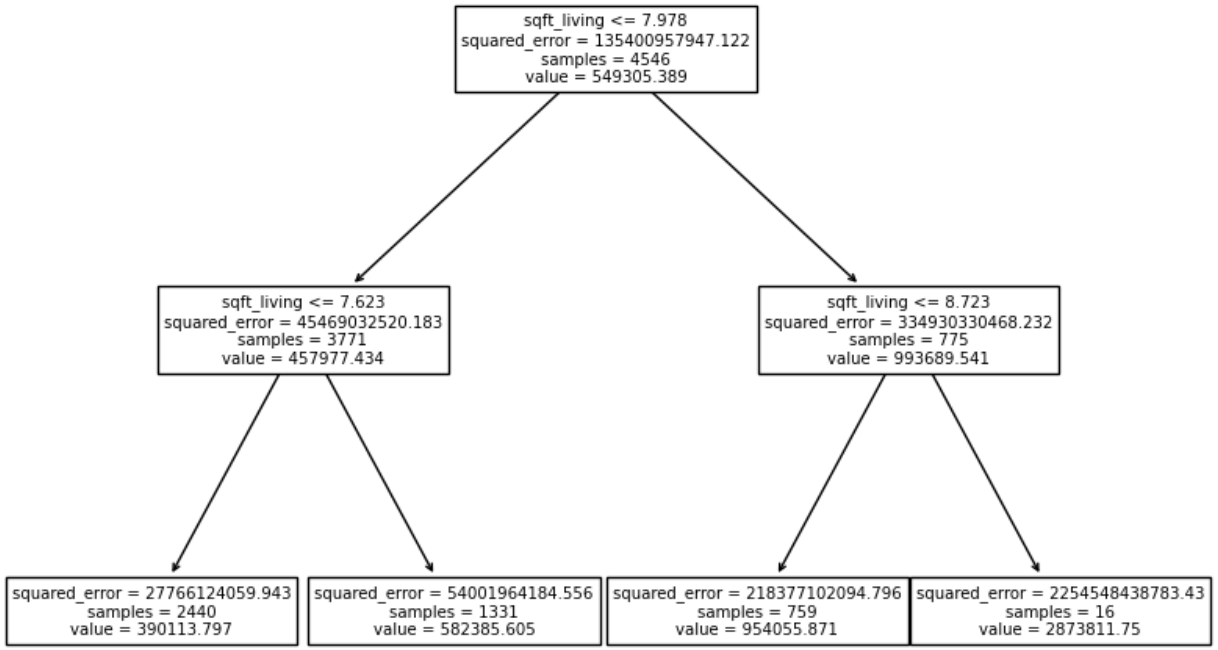


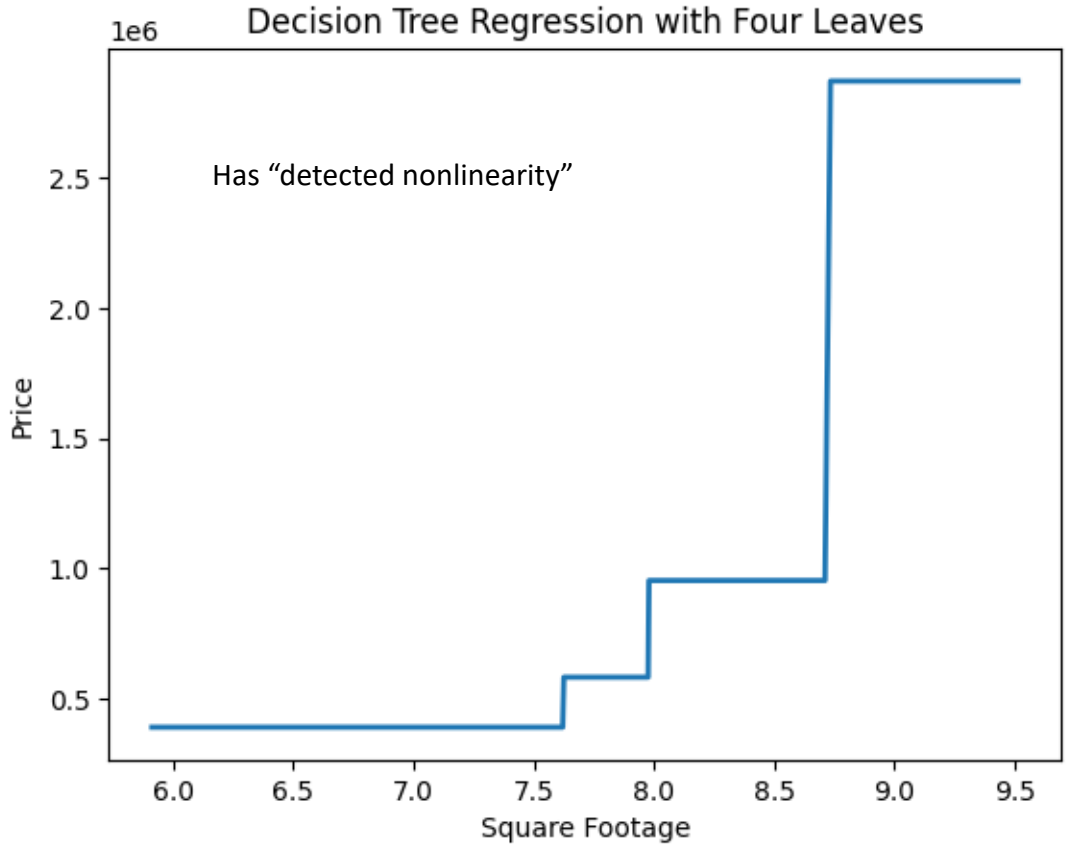
We add a new leaf by looking at the best split to improve the previous tree. E.g. the three leaf tree looked at all ways to split the two leaf tree and chose the one that led to the largest reduction in MSE.

Extremely easy to interpret tree prediction rule – arguably easier than interpreting a linear model.

House Price Regression Trees

Basic trees correspond to step function prediction rules.





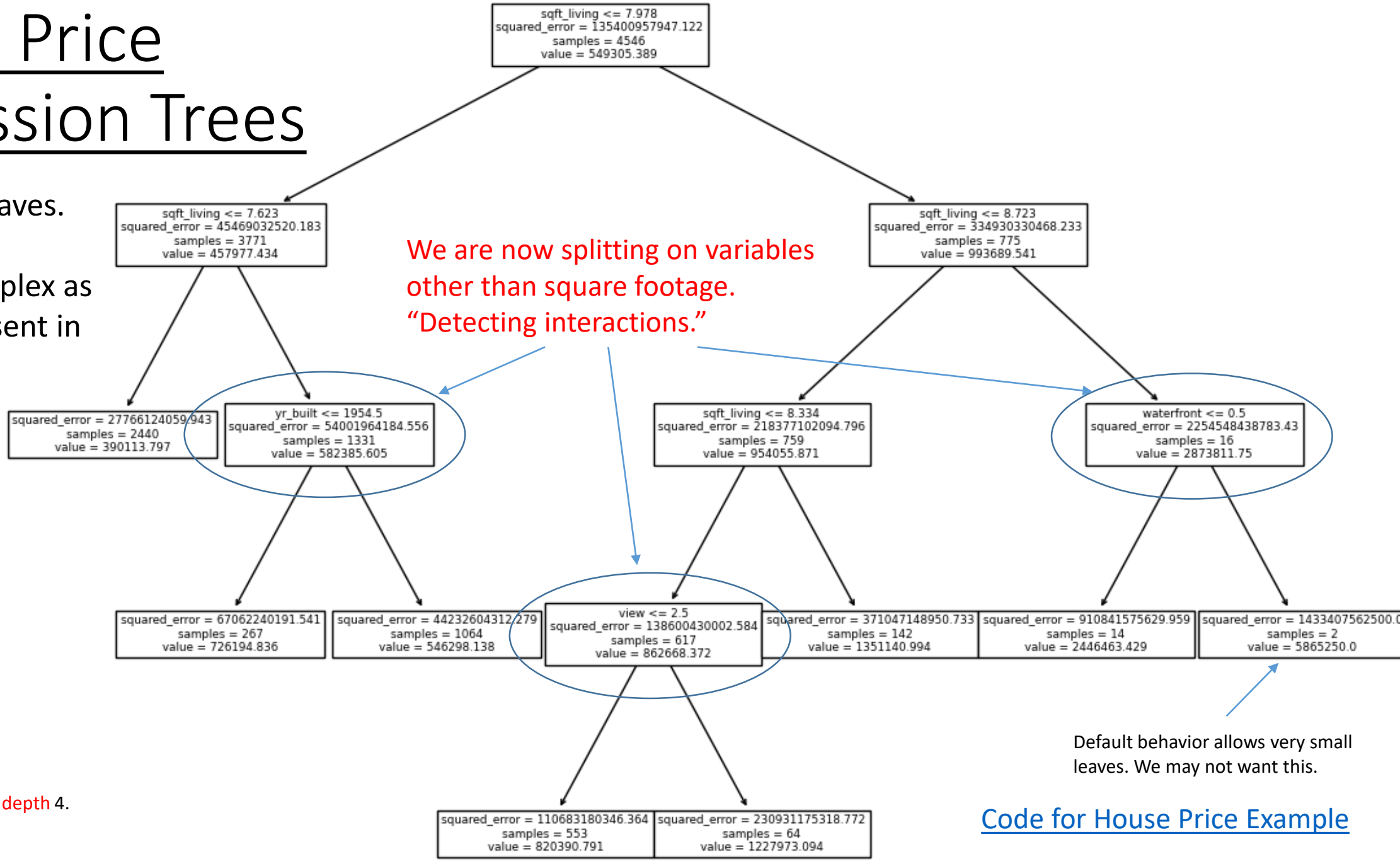
Two ways to represent the prediction rule from the four leaf tree

Code for House Price Example

House Price Regression Trees

Tree with 8 leaves.

About as complex as we can represent in these slides



We are now splitting on variables other than square footage. "Detecting interactions."

Default behavior allows very small leaves. We may not want this.

This tree has depth 4.

[Code for House Price Example]

Regression Tree Fitting

We could keep going in our house price example – growing bigger and bigger trees.

- Eventually, we would perfectly fit the training data.

The key decision is where to stop.

- If interpretability is the key concern, stop as soon as the tree becomes too complex to meaningfully convey.
  - Think of this as a descriptive exercise
- If prediction quality is the key concern, choose the tree to use based on (cross-)validation
  - Typical choices are to tune over depth or number of leaves
  - A related concept called “pruning” which is to grow a big tree and then “prune” back to one that does well according to (cross-)validation

Code for House Price Example

Trees in the House Price Example

Model	RMSE	R^2	MAE	Pseudo R^2	p
Polynomial (LAV) - Log	237723	0.583	149311	0.339	24
Flexible Lasso – Log	235301	0.591	149896	0.337	40 (799)
Tree	260755	0.498	165949	0.266	15
Tree – min 10	254397	0.522	161281	0.286	25
Tree – Log	260144	0.501	160519	0.290	79
Tree – Log – min 10	259071	0.505	159420	0.295	37

Trees based on cross-validation over number of leaves.

- p = number of leaves selected

Trees outperformed by best (and most) linear models in this example

- Fairly typical - step function is crude
  - e.g. has a hard time effectively capturing linearity and other “smooth” patterns

7. House Price Prediction: Categorical Variables

Code for House Price Example

Categorical Variables

Unordered categorical variables capture generic qualitative information. E.g. Zip Code in our example is an unordered categorical variable. In a product pricing example, the product's color would be an unordered categorical variable.

To use unordered categorical variables in predictive models, we first transform them into binary/dummy/one-hot-encoded variables. E.g. for `color` with categories "red" and "blue", we would define dummy variables `red` = 1 if red and 0 if not and `blue` = 1 if blue and 0 if not.

Unordered categorical variables with many categories are a challenge to supervised learners

- Many categories ≈ few observations per category

Basic intuition for challenge is straightforward

- Because there is no order, there is no sense that observations from different categories are "close"
- Learning then relies on having many observations per category
- Many categories means you don't have many observations per category

Punchline: Structured models (e.g. linear models) that rely on pre-specifying how categorical variables enter tend to work relatively well.

City and Zip Code in the House Price Data

In the house price data, we have two unordered categorical variables with many categories: `city` and `statezip`

- `city` has 44 categories
  - Median 33.5 observations
  - Mean 103 observations
  - 15 categories with <20 observations
- `statezip` has 77 categories
  - Median 58 observations
  - Mean 59 observations
  - 11 categories with <20 observations

Why do we think including these variables might add a lot of predictive power?

What if we want to predict outside of the Seattle area?

Code for House Price Example

Code for House Price Example

Categorical Variables in the House Price Example

Model	RMSE	R^2	MAE	Pseudo R^2	p
Flexible Lasso – Log	235301	0.591	149896	0.337	40 (799)
City – Log	196725	0.714	106464	0.529	67
Zip – Log	160547	0.810	85635	0.621	100
Both – Log	162748	0.805	85371	0.622	126
Zip Lasso – Log	165865	0.797	87127	0.615	92 (100)
Both Lasso – Log	167529	0.793	86753	0.616	117 (126)
Flexible + Both Lasso – Log	167206	0.794	87556	0.613	147 (901)
Both Tree – Log	244081	0.560	132772	0.413	84

Only showing results using log(price) for training.

Linear models are “polynomial” model + dummies

Lasso uses “polynomial” and “flexible” model + dummies.

All linear models do much better including location.
Trees, not so much.

8. Summary

Summary

- Supervised learning is fundamentally about building “generalizable” prediction rules
  - Building prediction rules involves a “bias/variance tradeoff”
  - Learn about this tradeoff in our particular scenario with validation exercises
    - Ideally have three splits of data: training, validation, and testing
    - Only use the testing data at the final step to double-check

- Linear models fit in the ML paradigm
  - Often very useful and competitive in settings with limited data

- Tree models are simple representation learners
  - Fast, simple, (relatively) interpretable
  - Tend not to predict as well as other methods