Learning Reflection 1

2022. 3. 28 (Mon.)

ML → supervised learning — labeled. with the labels, train to improve
→ unsupervised learning — un-labeled. train to estimate correct answers.
→ reinforcement learning — AI. CSE415 | 473

1. Regression - Predict house prices
2. Classification - Pos. / Neg. Reviews.
3. Document Retrieval + Clustering - Find similar news articles.
4. Recommender Systems - Given past purchase, give recommend.
5. Deep learning - Image Recognition.

Models	Algo.	Concepts
Linear Regression.	Gradient Descent	Point Estimation
regularized approaches (ridge, LASSO)	Boosting	MLE.
Linear Classifiers : Logistic regression	K-means.	Loss Function,
Non-Linear models : decision trees		bias-variance trade off,
Nearest neighbors, clustering		cross-validation.
Recommender systems.		Sparsity
DL.		Overfitting / underfitting
		model selection.
		Decision boundaries.

☆ Case study 1 : Regression : Housing Prices

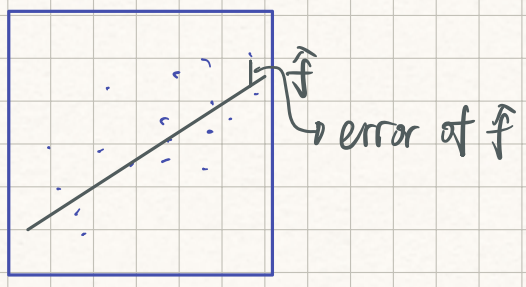
Goal: predict how much my house is worth.

Assumption : There is a relationship b/w $y \in \mathbb{R}$ & $x \in \mathbb{R}^d$, $y \approx f(x)$

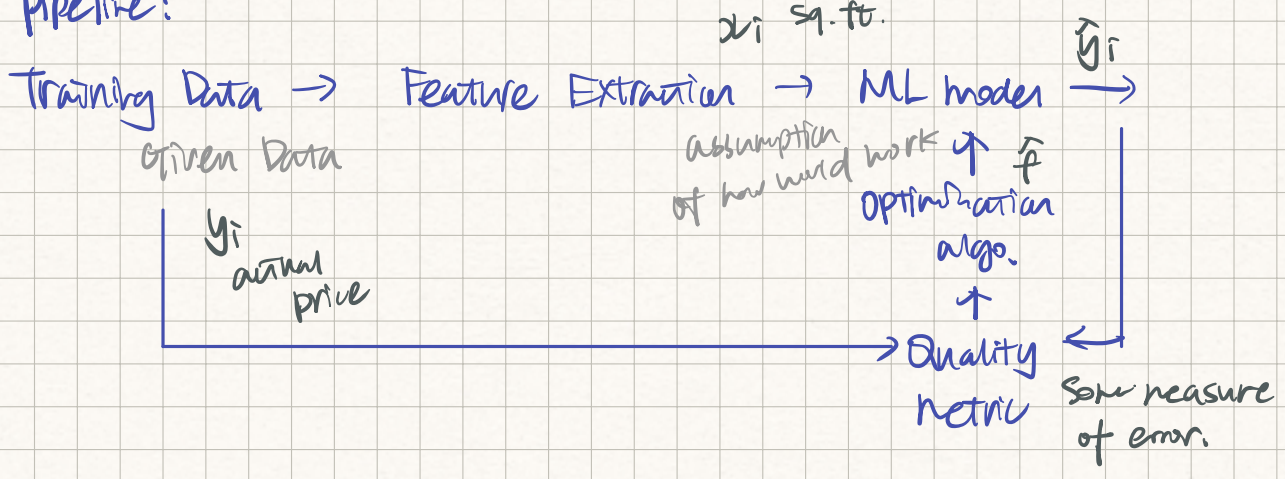
Model: $y \approx f(x)$ allows for some error of not an exact fit to true function.

Predictor: f is unknown. predictor: $\hat{f}$, we predict: $\hat{y} = \hat{f}(x)$

Small error → good fit.



✠ ML pipeline:



✠ Regression.

Supervised learning algorithm.

build predictor function learns how to map x^i to y^i

Linear Regression Model.

$$\underline{y}^i = w_1 \underline{X}^i + w_0$$

price Area of a house

w_0 → bias, starting price of a house

w_1 → slope / weight

$$\hat{y} = \hat{w}_1 x + \hat{w}_0$$

Cost / Loss of predictor – minimize the cost

Default loss function for regression: MSE (Mean-Square Errors)

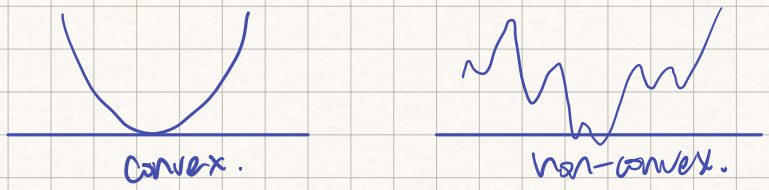
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

($y^i - \hat{y}^i$: residual^2)

y - actual val.

$\hat{y}$ - predicted val.

Gradient Descent.



Start at some (random) weights

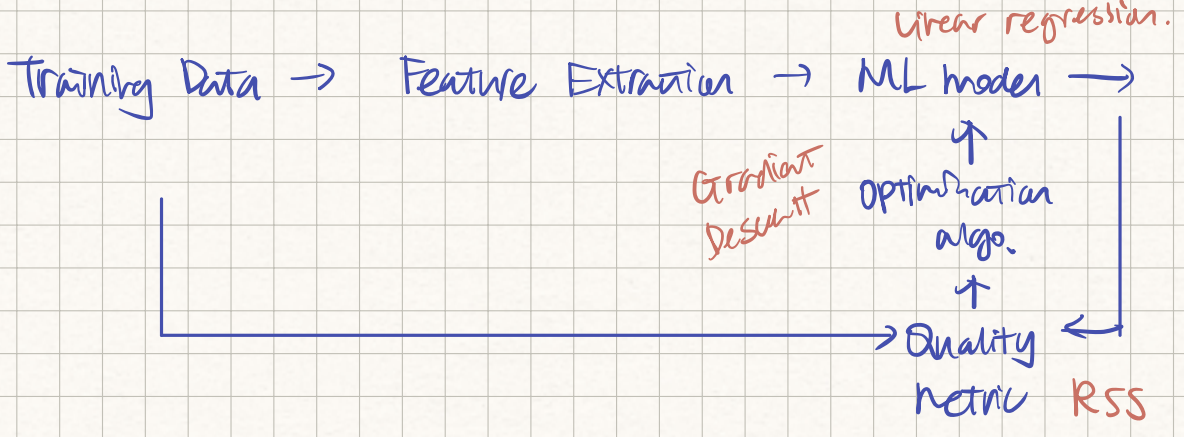
W while we haven't converged:

$$w -= \alpha \nabla L(w)$$

α → learning rate, controls how far you step.

$\nabla L(w)$ → the gradients of loss function L on a set of weights w.

Pre-lectures.



✡ Polynomial Regression

Model: $y_i = w_0 + w_1 x_i + w_2 x_i^2 + \cdots + w_p x_i^p + \varepsilon_i$

use gradient descent to train.

✡ Features

Model: $y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + \cdots + w_p h_p(x_i) + \varepsilon_i$

$= \sum_{j=0}^{p} w_j h_j(x_i) + \varepsilon_i$

e.g.: $y_i = w_0 + w_1(\text{sq. ft.}) + w_2(\#\text{ bathrooms}) + \varepsilon_i$

coefficients tell us rate of change if all other features are constant.

⊗ Distinction: is the difference between a data input & a feature

Data Inputs: columns of the raw data.

Features: values (possibly transformed) for the model (done after feature extraction $h(x)$).

Input: $x_i = (x_i[1], x_i[2], \ldots, x_i[d])$

Output: y_i

x_i is the ith row

$x_i[j]$ is the ith row's j'th data input

$h_j(x_i)$ is the jth feature of the ith row.

⊗ Recap. Linear Regression

Dataset: $[(x_i, y_i)]_{i=1}^n$ where $x \in \mathbb{R}^d$, $y \in \mathbb{R}$

Feature Extraction:

$h(x): \mathbb{R}^d \to \mathbb{R}^D$

$h(x) = (h_0(x), h_1(x), \ldots, h_D(x))$

Regression Model:

$y = f(x) + \epsilon$

$= \sum_{j=0}^{D} w_j h_j(x) + \epsilon$

$= w^T h(x) + \epsilon$

Quality Metric

$RSS(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2$

Predictor

$\hat{w} = \min_w RSS(w)$

Prediction

$\hat{y} = \hat{w}^T h(x)$

ML Algo.

optimized using Gradient Descent

⊠ Assessing Performance

Why train ML models ? do well on future data.

Minimizing RSS ≠ the model fits future data better

- General loss function $L(y, \hat{f}(x)) = (y - \hat{f}(x))^2$

- True Error : $E[L(y, \hat{f}(x))] = \sum_{x \in X} \sum_{y \in Y} L(y, \hat{f}(x)) \, p(x, y)$

  → all possible (x, y) pairs.

Need future data to calculate True Error, however we don't have future data in real cases. Try to minimize the true error is our goal.

Split dataset into train set & test set

- train set → train model
- test set → estimate performance in future. (true error)

Test Error :

$$RSS_{test}(w) = \sum_{i \in Test} (y_i - f_{\hat{w}}(x_i))^2$$

If the test set is large enough, this can approx. the true error.

⊠ Train / Test split.

Bigger test ⇒ Better estimate of true error.

But reduce the training set. Won't learn as well.

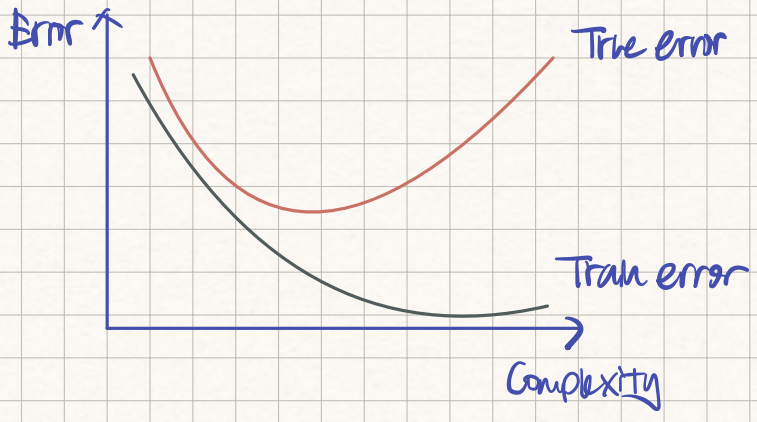
either { 80 : 20 / 90 : 10

Never train model on data in test set.

2022/3/30 (Wed.)

Model Complexity. A model w/ more para. usually is more complex.

Train / True Error:



Underfitting: fix → remove uninformative features; increase complexity

(adding more features or introduce more non-linearities to capture non-linear patterns)

More complex → easy to overfit; less complex → easier to underfit.

Bias - Variance Tradeoff.

Error { Bias
Variance
Irreducible Errors (noise from observations)

* { Model too simple (low complexity), high bias. low variance
Model too complicated (high complexity), high variance. low bias

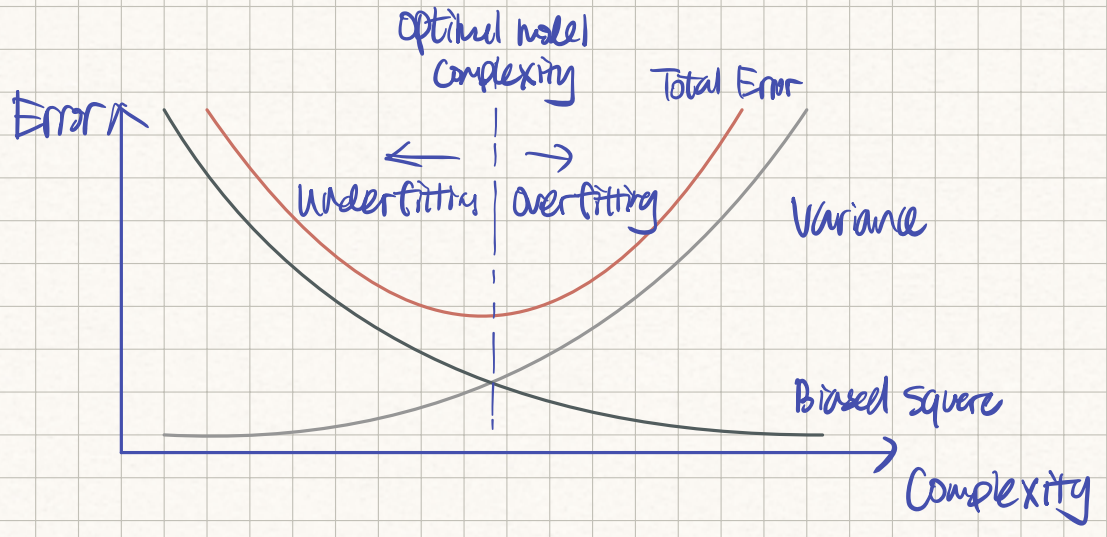
Bias: difference between the average prediction of our model and the expected value which we are trying to predict.

Variance: variability in model prediction, meaning how much the predictions will change if a different training dataset is used.

Source of errors for a particular model $\hat{f}$ using MSE loss function:

$$E[(y - \hat{f}(x))^2] = bias[\hat{f}(x)]^2 + Var(\hat{f}(x)) + \theta_\epsilon^2$$

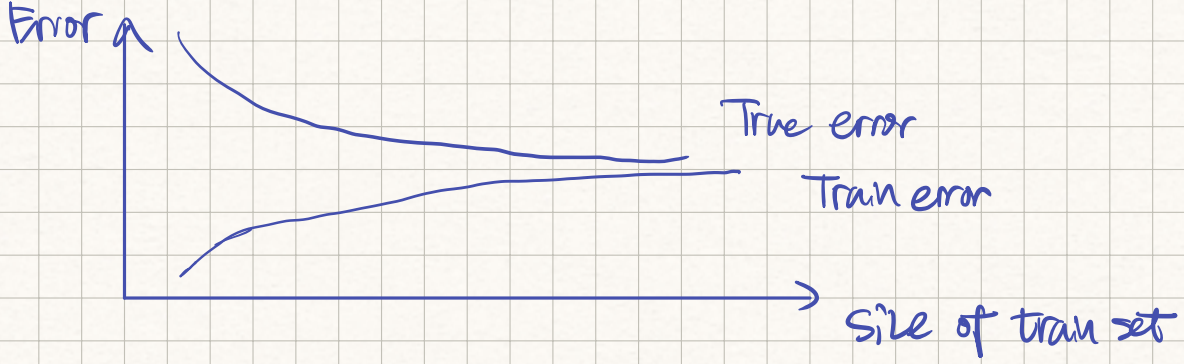
Error = Biased squared + Variance + Irreducible Error



✱ Model complexity doesn't depend on the size of the training set.

The larger the training set,

the lower the variance of the model. thus less overfitting.



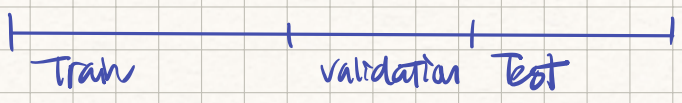
✱ Choosing complexity

two ways to pick the model complexity w/o running our test set.

Using a validation set. ⇒ Doing cross validation.

Validation set.

We can't use Test to choose our model complexity, instead,



↳ still not a fully unbiased estimator of the true error, but less biased than the train set.

Pros: Easy to describe and implement
Pretty fast.

Cons: Have to sacrifice more training data.
Prone to overfitting

Summary

1. Use ML to build models and train them to well predict unseen (future) data.
2. Use True Error to make sure our models predict more accurately.
3. However, we need infinite data to calculate True Error but we don't have infinite data. Then, we need to estimate the True Error. Therefore, we split the dataset into Train Set & Test Set with usually 90:10 or 80:20. Even though, a large Test Set decreases the error of estimating the True Error (can be more confident in the prediction), it decreases the accuracy of the Training Set.
4. Never use test error as the standard to pick models. We choose models only based on the Train Set. Then we use the Test Set for only testing the model.