CAI 4104/6108 – Machine Learning Engineering:

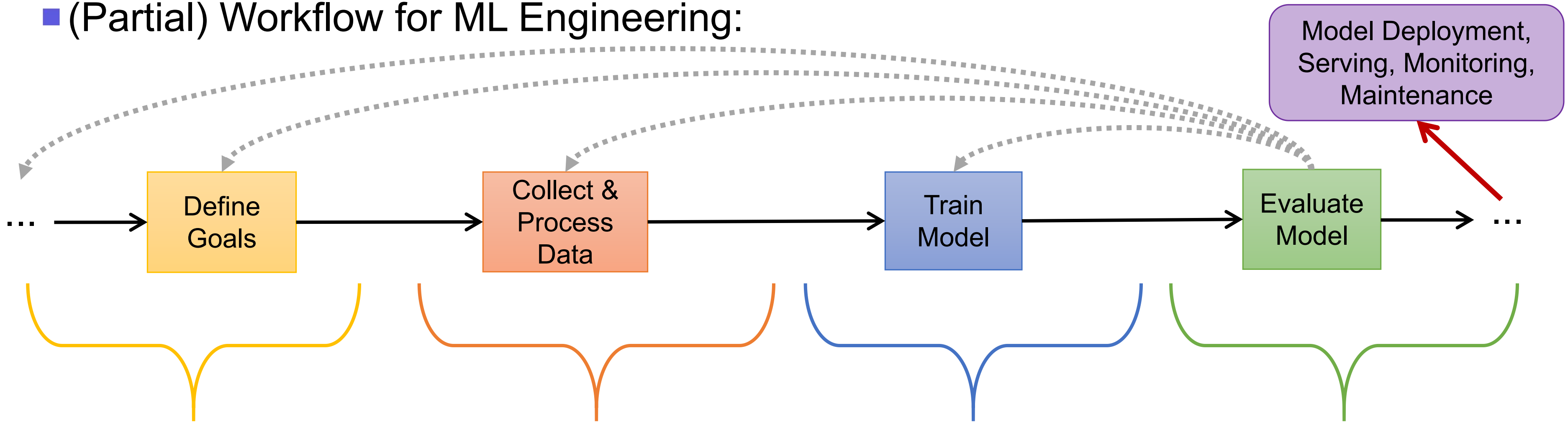
ML Engineering (3)

Prof. Vincent Bindschaedler

Spring 2024

Reminder: Machine Learning Engineering

- (Partial) Workflow for ML Engineering:



Define Goals
- When to use ML?
- What problems can it solve?
- Why do ML projects fail?
- What can ML not do?

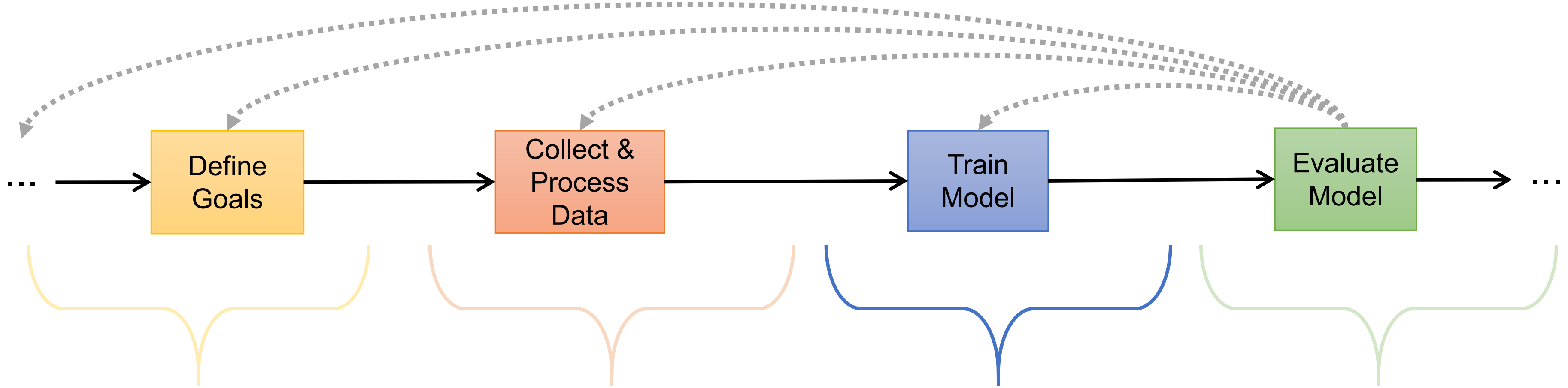
Collect & Process Data
- How to get data?
- Dealing with noisy data?
- How much data is needed?
- How to extract features?

Train Model
- What is the task?
- What kind of model?
- How to tune hyperparameters?

Evaluate Model
- Best practices to evaluate the model?
- What metric(s) to use?
- What is the baseline?
- Underfitting/overfitting?

Machine Learning Engineering

- (Partial) Workflow for ML Engineering:



- When to use ML?
- What problems can it solve?
- Why do ML projects fail?
- What can ML not do?

- How to get data?
- Dealing with noisy data?
- How much data is needed?
- How to extract features?

- What is the task?
- What kind of model?
- How to tune hyperparameters?

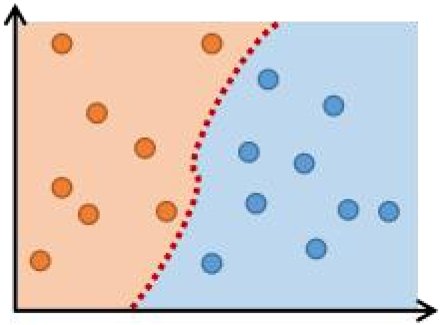
- Best practices to evaluate the model?
- What metric(s) to use?
- What is the baseline?
- Underfitting/overfitting?

Reminder: Types of Learning

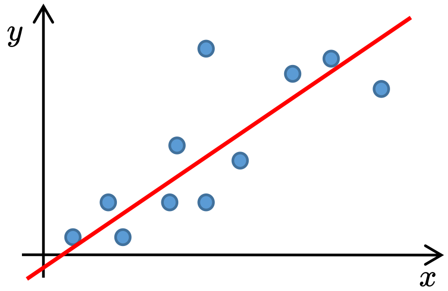
- Supervised Learning
  - Learning from labeled data (i.e., each example or instance in the dataset has a corresponding label)
  - Tasks: classification vs. regression
- Unsupervised Learning
  - Learning from unlabeled data (we must discover patterns in the data)
  - Tasks: clustering (e.g., K-means), dimensionality reduction (e.g., t-SNE, PCA), etc.
- Semi-supervised Learning
  - Learning from partially labeled data
- Reinforcement Learning
  - There is an agent that can interact with its environment and perform actions and get rewards
  - Learning a policy (i.e., a strategy for which actions to take) to get the most rewards over time
- Transfer Learning
  - Learning to repurpose an existing model for a new task

Reminder: Supervised Learning

- Classification
  - Task: predict the corresponding label
  - Different types:
    - Binary classification: there are only two classes (0,1 ; +,-, etc.)
    - Multiclass: more than two classes
    - Multi-label: each instance can belong to more than one class (e.g., label all objects in a photo)
    - One-class: there is only one class, we want to distinguish it from everything else



- Regression
  - Task: predict the corresponding value (typically a real number) or target
    - E.g.: you want to predict a person's future income based on their high school GPA



- Others:
  - Sequence-to-sequence, similarity learning/metric learning, learning to rank, etc.

Reminder: Multiclass Classification

- Multiclass classification (aka *multinomial classification*)
  - There are c >2 distinct classes: $y_i \in \{1,2,...,c\}$ is the label of the i^{th} example
- Wait. How do we do this?
  - Recall the SVM formulation:
    - We want to find a hyperplane $\boldsymbol{w}\,\boldsymbol{x} - b = 0$, also we relabel the classes as +1 and -1. **Problem?**
- Some learning algorithms / models naturally support multiclass classification
  - E.g.: kNN, decision trees, neural networks
- For others, we can transform multiclass classification into a binary classification
  - One-vs-rest (OvR): Train c binary classifiers. f_i to classify class i versus not i
    - Predict: $y = \mathrm{argmax}_i\ f_i(x)$
  - One-vs-one (OvO): Train $c(c\text{-}1)/2$ binary classifiers. $f_{i,j}$ to classify class i versus class j
    - Predict: predict with all $c(c\text{-}1)/2$ and return the class that has the highest number of “votes”

What Is a Learning Algorithm?

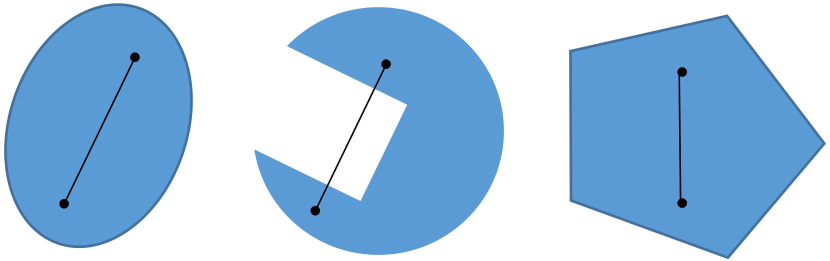
- To train a model we need:
  - Some kind of objection function or criterion often called loss function (or cost function)
    - Some algorithms have a specific criterion (e.g., SVM — hinge loss)
    - Others do not have such a criterion (e.g., kNN)
  - An optimization procedure to find a solution (i.e., parameter values)
    - E.g.: quadratic programming, gradient descent

- We want “nice” loss functions:
  - Ideally: **continuous**, **differentiable**, (strictly) **convex**, **smooth** loss functions
  - Examples:
    - SVM is a convex problem
    - But (in general) the loss function for neural networks is **not** convex

Convex Sets & Convexity

- Convex Sets:

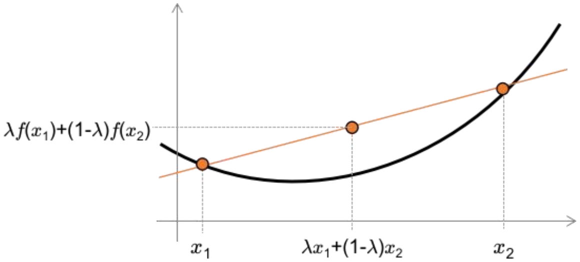
Set $\mathcal{X}$ is convex if for any a, $b \in \mathcal{X}$
the line λa+(1-λ)b $\in \mathcal{X}$ for $\lambda \in [0,1]$

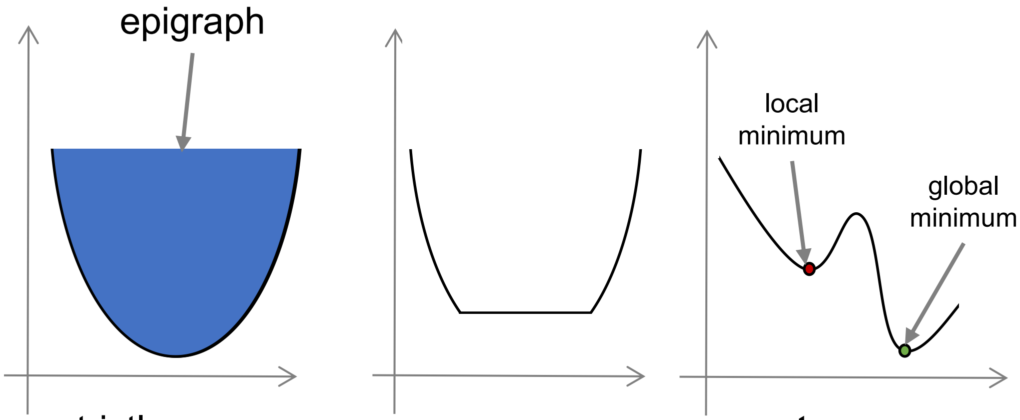


convex **not** convex convex

- Convexity:

Function f on a convex set $\mathcal{X}$ is convex
if for any x, $y \in \mathcal{X}$ and $\lambda \in [0,1]$:
$\lambda f(x)$+(1-λ)$f(y) \geq f(\lambda x$+(1-λ)y)



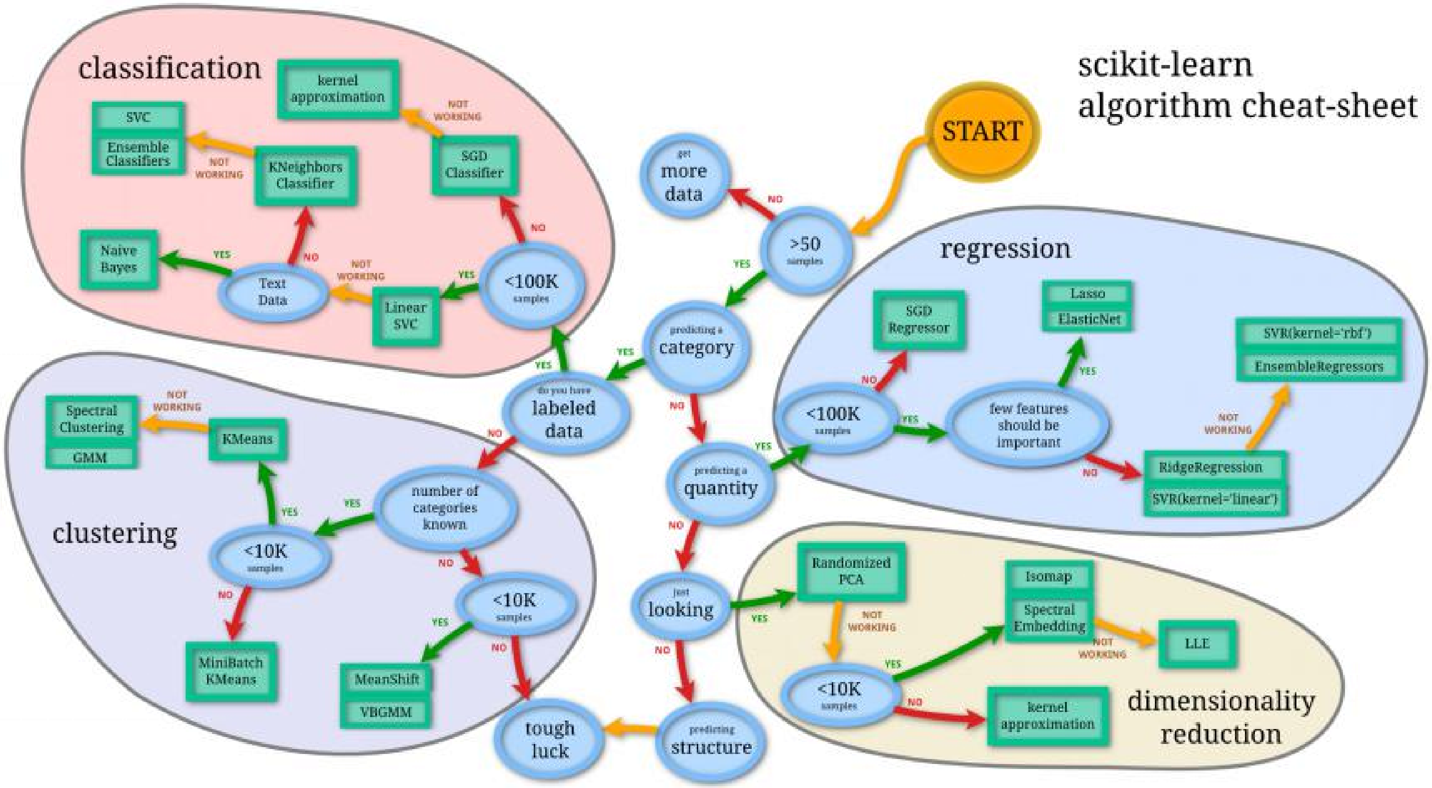


strictly convex convex **not** convex

No Free Lunch Theorem

- Famous ML result
  - No Free Lunch (NFL) Theorem
    - *David H. Wolpert. "The lack of a priori distinctions between learning algorithms." Neural computation 8.7 (1996).*
- What does the theorem say?
  - Informally: Given two learning algorithms A and B, there are just as many problem instances/datasets where A performs better than B as vice-versa (B performs better than A)
  - In other words:
    - There is **no** learning algorithm that is **guaranteed** to work well on our data a priori (i.e., before we try it)
- Why?
  - When we train a model (i.e., use a learning algorithm as opposed to another), we are making assumptions about how features are related to target/label
    - E.g.: we assume two classes can be separated by a linear decision boundary based features (if we use linear SVM)
- Consequences?
  - The only way to know for sure what learning algorithm is best is to evaluate them all
  - Or in practice: start with reasonable assumptions, then evaluate (only) a few algorithms

Model Selection



Source: scikit-learn.org

Parameters & Hyperparameters

- Parameters
  - Sometimes called “weights”
  - Model parameters are determined by the training data (e.g., through some optimization procedure)
- Hyperparameters
  - These are not learned when we trained the model; the machine learning engineer sets those
    - E.g.: k in kNN is a hyperparameter; for SVM is the non-linearly separable case there is C (regularization hyperparameter)
  - However, hyperparameters should be tuned
    - If we want the best model, we also need the best values of hyperparameters!
- Hyperparameter tuning/optimization strategies
  - Grid search: try all combinations of hyperparam values
    - For example: for hyperparam $a \in \mathcal{A}$ and $b \in \mathcal{B}$, try all pairs $(a,b) \in \mathcal{A} \times \mathcal{B}$
  - Random search: given distribution of hyperparam values, we randomly sample from them
  - Many others: e.g., bayesian hyperparameter optimization, evolutionary optimization, etc.

Machine Learning Engineering

- (Partial) Workflow for ML Engineering:

… → Define Goals → Collect & Process Data → Train Model → Evaluate Model → …

Define Goals
- When to use ML?
- What problems can it solve?
- Why do ML projects fail?
- What can ML not do?

Collect & Process Data
- How to get data?
- Dealing with noisy data?
- How much data is needed?
- How to extract features?

Train Model
- What is the task?
- What kind of model?
- How to tune hyperparameters?

Evaluate Model
- Best practices to evaluate the model?
- What metric(s) to use?
- What is the baseline?
- Underfitting/overfitting?

Training, Test, Validation

- Before training a model (or looking at the data ideally)
  - Divide the dataset into **three disjoint parts**
    1. Training dataset
    2. Test dataset
    3. Validation dataset
- Why? Why do we need the validation dataset? Why not just training and test?
  - We need it for hyperparameter optimization!
    - Q: Why can't we use the training set for that?
    - Q: Why can't we use the test set for that?
- What proportion of the dataset to allocate to each?
  - (Rule of thumb:) For small datasets (i.e., < 100k examples): 70% training, 15% validation, 15% test
  - For large datasets (e.g., deep learning): 95% training, 2.5% validation, 2.5% test
  - For very small datasets (e.g., <1000 examples): check the raw numbers (e.g., how many examples is 10%?)
  - What if you don't have enough data to afford leaving some aside for validation/testing?
    - Use k-fold validation: divide the data into k equal parts, then train on k-1, test on the remaining part, repeat k times and average!

(Training) Data Leakage

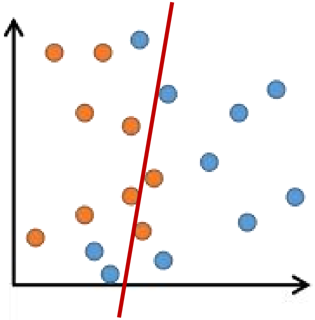
- Subtle failure more for ML: ***data leakage***
  - Occurs if the model is given access to information (at **training time**) that would **not** be available at **inference time**.
  - Row-wise leakage (training example) or column-wise leakage
  - Examples:
    - Duplicate data
    - Preprocessing leakage (e.g., premature feature engineering)
    - Improper hyperparameter tuning
    - Proxy attributes (e.g., want to predict `age` but `year_of_birth` is a feature)
    - Time leakage (e.g., time series data improperly split between train and test)
    - Etc.
  - Major concern: **reproducibility** crisis
    - Ref: Kapoor and Narayanan. "*Leakage and the reproducibility crisis in machine-learning-based science*." Patterns 4, no. 9 (2023).

Bias and Variance

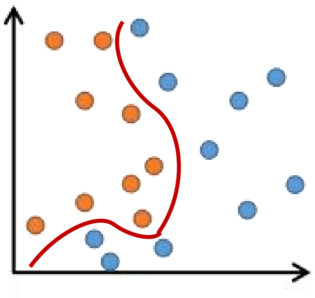
- Bias
  - Error due to incorrect assumptions in the model
  - *Inability to capture the true relationship*
    - If a model is too simple to capture the true relationship between features and label/target, it will have high bias!
  - High bias means underfitting!
  - Terminology:
    - do not confuse this with the bias term in the parameters of a model (i.e., the intercept)
- Variance
  - Sensitivity to small variation in the training data
    - Think of training a model as a repeated randomized process
    - If the model is highly influenced by a few data points, then it has high variance (it models the random noise!)
  - High variance means overfitting!

Bias and Variance

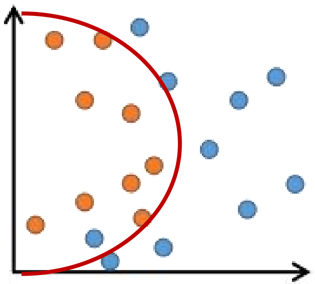
- Bias
  - Error due to incorrect assumptions in the model
  - *Inability to capture the true relationship*
- Variance
  - Sensitivity to small variations in the training data
- Ideally, we want: low bias **and** low variance
  - Strategies to lower bias:
    - Increase model complexity
    - Use more features
  - Strategies to lower variance:
    - Reduce model complexity
    - Use more training data



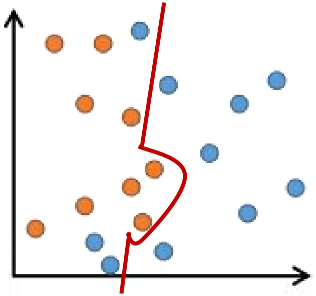
High bias



High variance



Low(er) bias & low(er) variance



High bias & High variance

Bias-Variance Tradeoff

- Generalization error (aka out-of-sample error or risk)
  - Prediction error on *unseen* data
  - Related to overfitting
    - If the model overfits, then the generalization error will be large
- Bias-Variance Tradeoff
  - Generalization error: `bias² + variance + irreducible error`
    - For more details:
      - Geman et al. "*Neural networks and the bias/variance dilemma.*" Neural computation (1992)
      - Kohavi et al. "Bias plus variance decomposition for zero-one loss functions." ICML, 1996.
  - Why is it a tradeoff?
    - Increasing model complexity => lower bias
    - Decreasing model complexity => lower variance
    - Note: there has been some debate of whether this applies to neural networks
      - E.g.: see Neal et al. "A modern take on the bias-variance tradeoff in neural networks." arXiv, 2018.

Regularization

- Most models can be regularized
  - Typically tuned through a regularization constant (hyperparameter)
  - Effect: lower variance at the cost of (slightly?) higher bias

- Regularization reduces model complexity
  - It decreases the degrees of freedom of the model
    - E.g.: for linear SVM, regularization controls the cost of misclassification in the loss function
  - Note: there are several types of regularization and regularization techniques

- If your model is overfitted
  - Regularization is (one of) the first things you should try

Measuring Bias & Variance

- Key quantities:
  - Error on training dataset
  - Error on validation/test dataset
  - To keep in mind: the irreducible error
- Examples (classification):
  - Assumptions:
    - We measure the error using 1-accuracy
    - Irreducible error is 0%
  - Diagnoses:
    1. Training error: 1%; validation error: 20% => low bias; high variance (**overfitted**)
    2. Training error: 20%; validation error: 21% => high bias; low variance (**underfitted; generalizes well**)
    3. Training error: 20%; validation error: 35% => high bias; high variance (**worst case**)
    4. Training error: 1%; validation error: 2% => low bias; low variance (**best case**)

Baseline(s)

- In general, we do not know the irreducible error
  - Suppose the training error of a classifier is 20%
  - Q: Does the classifier have high bias (is it underfitted)?
    - **It depends what the irreducible error is!**

- It is critical to have an appropriate baseline!
  - Given a baseline, we can at least know if the model learned anything at all!
  - Baseline(s) for classification tasks
    - Random guessing:
      - If there c classes, the baseline accuracy is 1/c (baseline error is 1-1/c)
    - Guessing the mode (most frequent class)
      - If q_i is the frequency of class i, then baseline error is $\min(1-q_i) = 1-\max(q_i)$
    - If the problem is well-studied, use benchmarks as a baseline!
      - Note: *if humans can perform the task with almost 0% error, then the irreducible error is probably 0*

Next Time

- Friday (1/26): Exercise 2

- Upcoming:
  - Homework 1 will be out today (due 2/2 by 11:59pm)