# Design of ML experiments and evaluation of ML models

# Announcement - Mini Project (Kaggle competition)

- Mini Project for CS 3110 will be online soon - Kaggle Competition
- Will be using the same dataset as CS2500 Data Science Challenge
- Please download the test dataset again since it has been updated

# Lecture outline

1. Design of ML experiments
    a. ML pipelines
    b. Collection & preparation of data
    c. Selecting the type of model to build
        i. Design considerations
2. Evaluation ML models
    a. Model training
    b. Model evaluation
3. Overfitting and underfitting

# Part 1.
# Design of ML experiments

# Working on your ML projects

Say you have trained your classifier and you have about 70% accuracy. Now what?

- Collect more data
- Increase the diversity of the training data
- Train for longer
- Try a different optimizer
- Try a more complex model
- Try a simpler model
- Add/tune regularization
- If you are using a deep neural networks:
- Try different activation functions
- Change the number of hidden units
- …

# Machine learning pipeline - a systematic approach

A machine learning pipeline is a workflow of subtasks that when taken together represents a whole machine learning task.

1. Data extraction, possibly from multiple sources
2. Data cleaning and preparation to be fed into ML models
3. Deciding on which models to use
4. Training models
5. Evaluating model performance
6. Model selection
7. Deployment of selected models

- **Pro tip: Set up the whole ML pipeline as fast as possible. Start simple, and keep improving the whole setup iteratively.**

# Machine learning experiments: what to track

**Machine learning experiment**: A systematic procedure to test the ability of machine learning models to solve a given task

- Data
- Code
- Model architectures
- Trained models and parameters
- Hyperparameters
- Training and validation accuracy (and loss)
- Metadata: experiment name, trail/job name, job parameters

# Setting things up to avoid rookie mistakes

- Programming environment
  - Virtual environments (e.g. conda)
  - Docker containers
  - Jupyter notebooks / Google Colab
- Version control (e.g. git)
- Back up the data
- Documentation/records of experiments and their metadata, hyperparameter values, and corresponding results
  - Log and save all the performance metrics. Use them to plot the results and compare
- Don’t hard code parameter/hyperparameter values.
  - Pass them as arguments
  - Read them from a config file instead (e.g. yaml file)
- For large projects, it’s better to use an established ML experiment management system
  - E.g. Weights & Biases (wandb.ai). See here for a longer list of tools

# Collection and preparation of data

Data is collected from a single or multiple sources. The following issues need to be addressed when preparing the data to be used for machine learning

1. Data cleaning
   a. Handle missing values
   b. Reduce noise
2. Relevance analysis (feature selection)
   a. Remove irrelevant and redundant attributes
   b. Remove correlated features
3. Data transformation
   a. Normalize
   b. Convert (formats)

# Train-Dev-Test Split (The Hold Out Method)

| Training dataset | Dev (validation) dataset | Test dataset |
|---|---|---|
| Used to learn the weights/parameters of the ML models | Used for hyperparameter tuning | Used to measure the generalization accuracy (model evaluation) |

- Should be collected from the same distribution
- Must be taken randomly from the whole dataset
  - **Never pick the data points sequentially!**
- Should be reproducible (using a fixed seed for randomization).
- Test set should be the same across all experiments
- The dev and test sets should reflect the data you expect to get in the future
- Most public datasets have train and test sets. Then you have to extract your own dev set from the training data

# Train-Dev-Test Split: what proportion to use

- There is no rule
- Depends on:
  - Size of the available dataset
  - How much data is required for training for the selected algorithm, using the given dataset
    - How complex is the algorithm (tendency to overfit)
    - How much variability is there in the dataset
- Typical values for train:dev:test partitioning
  - 70:20:10
  - 60:20:20
- If you have a million data points, you might not need to go for the above typical partitioning values
  - 1% of 1 million could be enough for each dev and test sets

# Selecting what ML algorithm to use

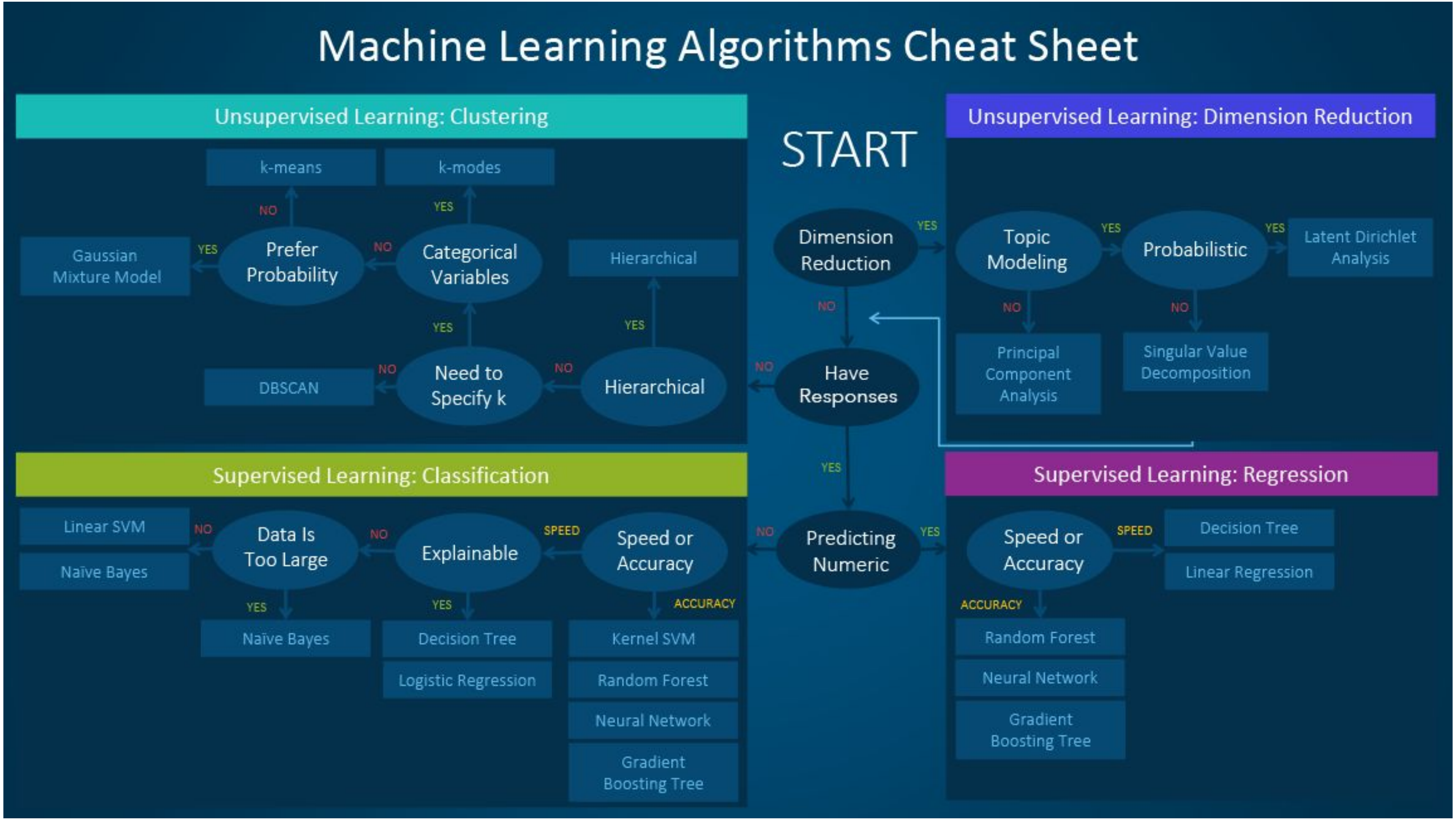
# Types of machine learning tasks

Machine learning paradigms

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

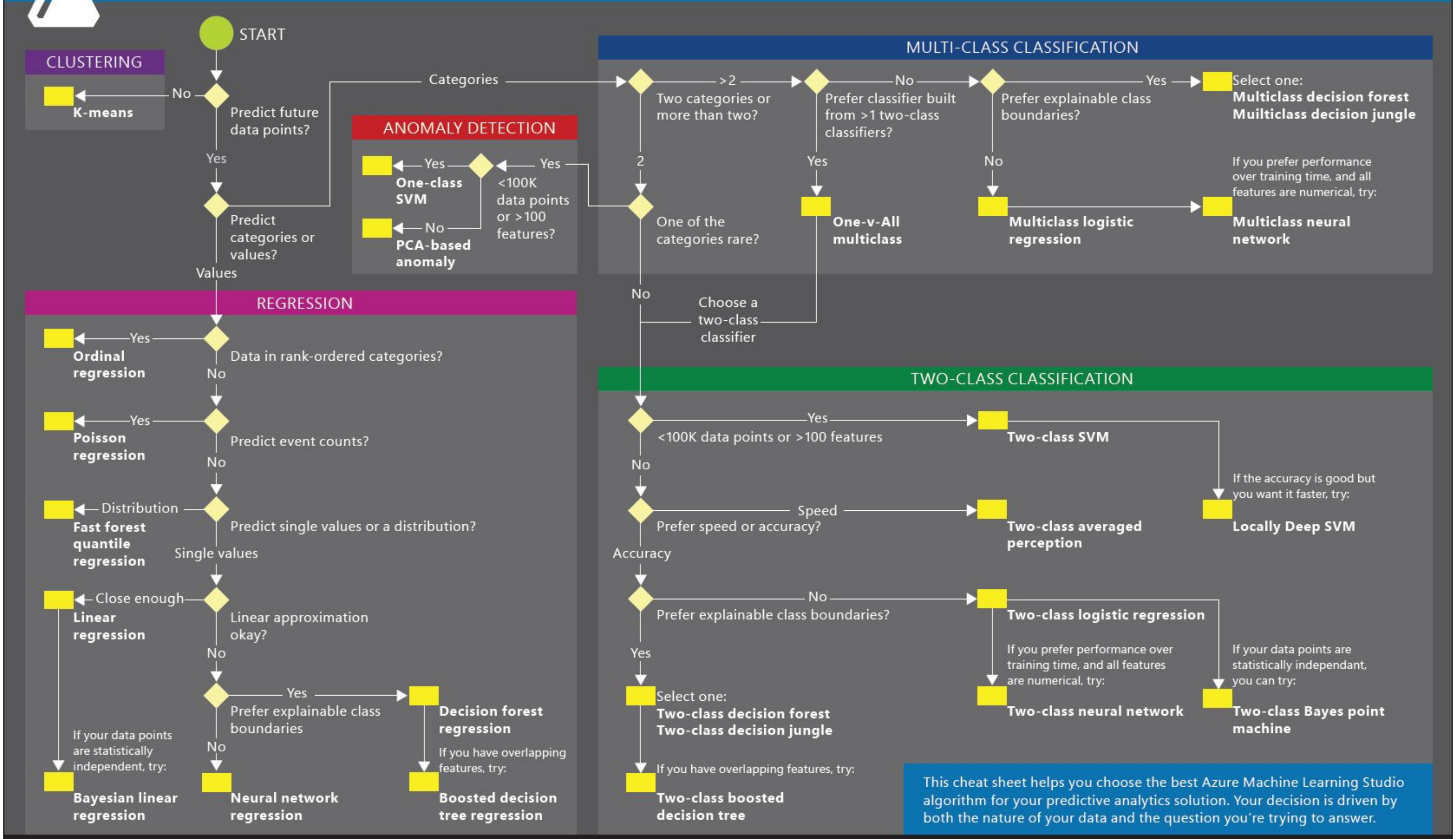
# Selecting the type of ML algorithm to use (supervised)

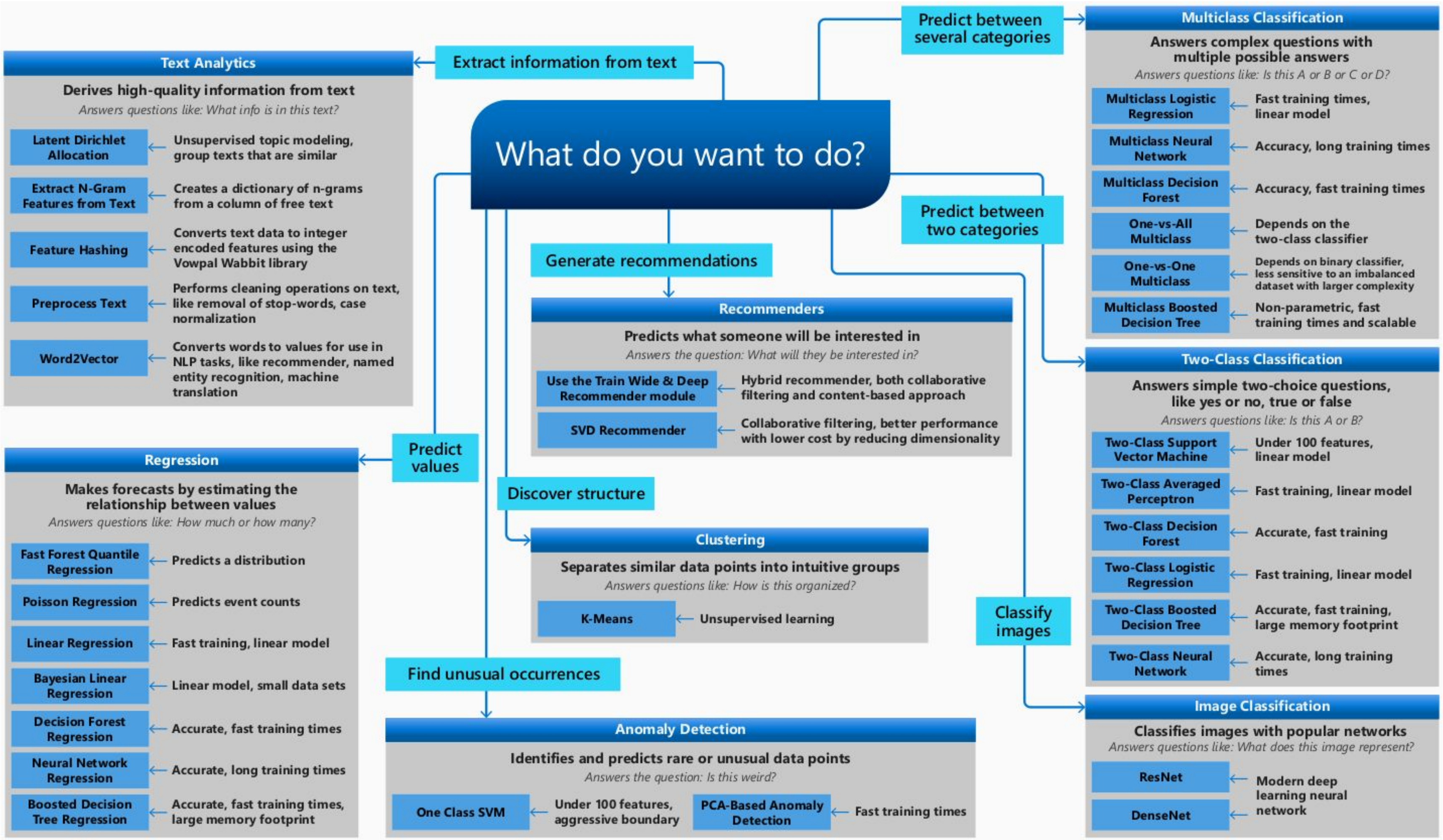
1. What's the nature of your data and what do you want to do with it?
   a. Predicting a value of a continuous variable
   b. Assigning a class label to a set of attributes
   c. Clustering similar data points
2. What are the priorities/requirements/constraints of the solution?
   a. Accuracy
   b. Interpretability
   c. Speed
   d. Number of parameters
   e. Number of features
   f. Resources Memory and CPU/GPU




Source: blogs.sas.com

# Microsoft Azure Machine Learning: Algorithm Cheat Sheet






Source: docs.microsoft.com

| Classification Algorithm | Accuracy | Training time | Linear | Param |
|---|---|---|---|---|
| Two-Class logistic regression | Good | Fast | Yes | 4 |
| Two-class decision forest | Excellent | Moderate | No | 5 |
| Two-class boosted decision tree | Excellent | Moderate | No | 6 |
| Two-class neural network | Good | Moderate | No | 8 |
| Two-class averaged perceptron | Good | Moderate | Yes | 4 |
| Two-class support vector machine | Good | Fast | Yes | 5 |
| Multiclass logistic regression | Good | Fast | Yes | 4 |
| Multiclass decision forest | Excellent | Moderate | No | 5 |
| Multiclass boosted decision tree | Excellent | Moderate | No | 6 |
| Multiclass neural network | Good | Moderate | No | 8 |
| One-vs-all multiclass | - | - | - | - |

| Regression Algorithm | Accuracy | Training time | Linear | Param |
| --- | --- | --- | --- | --- |
| Linear regression | Good | Fast | Yes | 4 |
| Decision forest regression | Excellent | Moderate | No | 5 |
| Boosted decision tree regression | Excellent | Moderate | No | 6 |
| Neural network regression | Good | Moderate | No | 8 |

# Announcement - Mini Project (Kaggle competition)

- Mini Project for CS 3110 will be online soon - Kaggle Competition
- Will be using the same dataset as CS2500 Data Science Challenge
- Please download the test dataset again since it has been updated

# Part 2.
# Evaluation of ML models

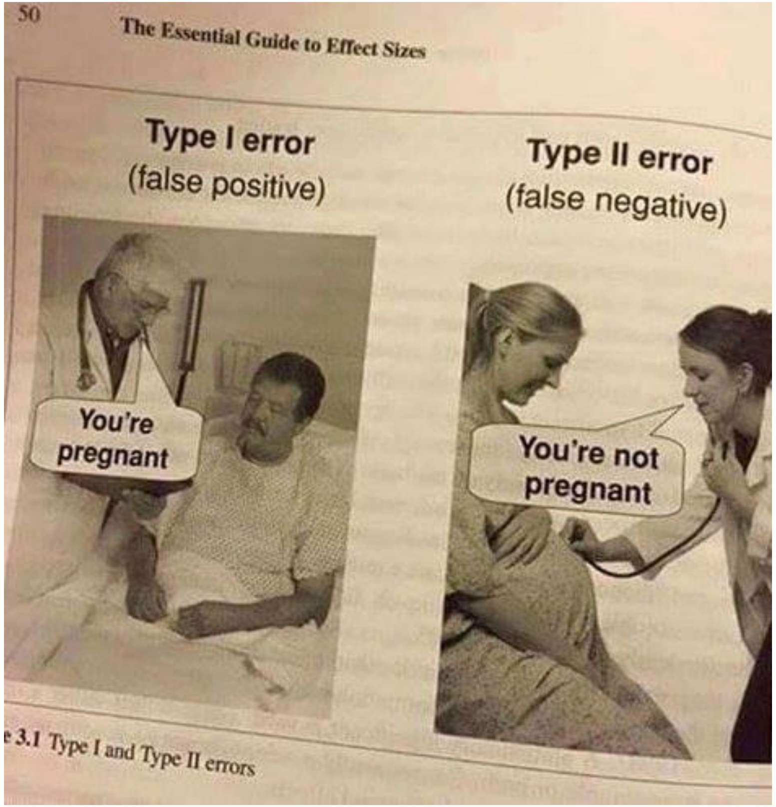
# Why evaluate?

- To know if your model will perform well with new/unseen data
    - Generalization
- To get an idea of what weaknesses your model has
- To know what aspects of the model should be improved
- To compare multiple models (model selection)

# Considerations when evaluating a model

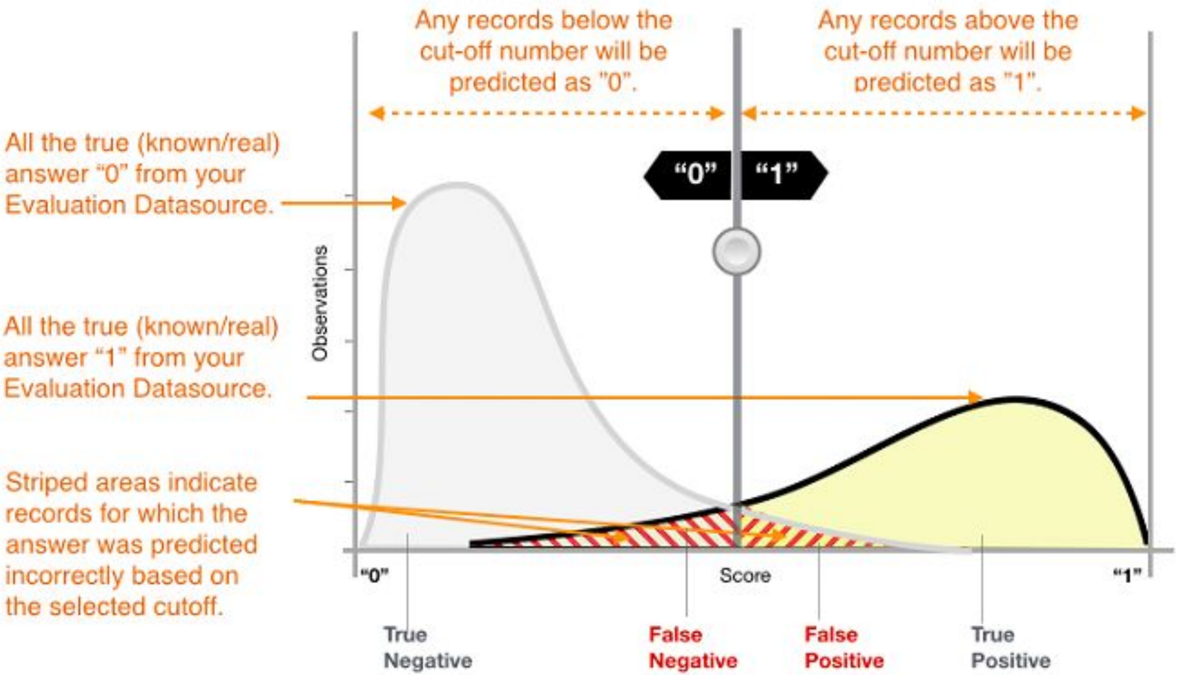
1. Accuracy
2. Speed
3. Robustness
4. Scalability
5. Interpretability (insights)
6. Size/compactness
7. Resource requirements

# Some key definitions




- **TP** (**true positives**): num samples correctly classified as positive (yes)
- **TN** (**true negatives**): num samples correctly classified as negative (no)
- **FP** (**false positives**): num samples incorrectly classified as positive **[Type I error]**
- **FN** (**false negatives**): num samples incorrectly classified as negative **[Type II error]**

# Binary classification: False positives and false negatives



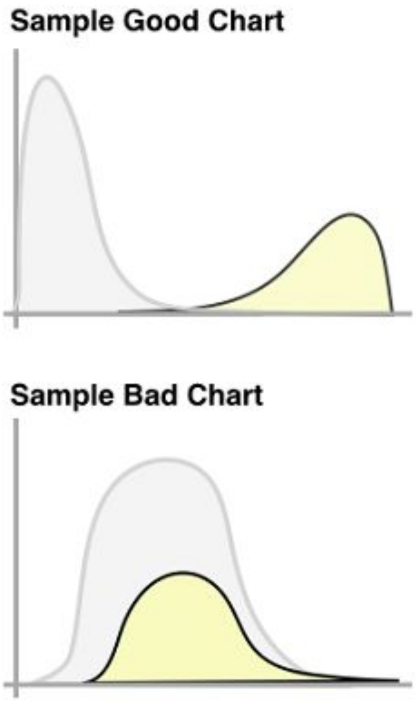




Image source: Amazon ML

# Confusion matrix for binary classification

| | | actual class | |
|---|---|---|---|
| | | positive | negative |
| predicted class | positive | true positives (TP) | false positives (FP) |
| | negative | false negatives (FN) | true negatives (TN) |

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

# Is “accuracy” good enough?

… to measure predictive performance?

accuracy may not be useful measure in cases where

- there is a large class skew
  - Is 98% accuracy good if 97% of the instances are negative?
- there are differential misclassification costs – say, getting a positive wrong costs more than getting a negative wrong
  - Consider a medical domain in which a false positive results in an extraneous test but a false negative results in a failure to treat a disease
- we are most interested in a subset of high-confidence predictions

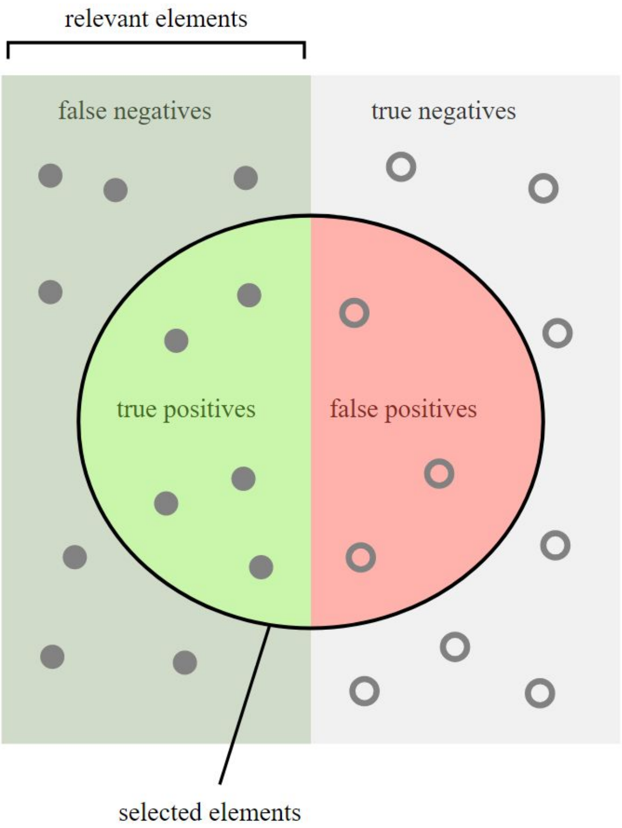
# TPR and FPR

| | | actual class | |
|---|---|---|---|
| | | positive | negative |
| predicted class | positive | true positives (TP) | false positives (FP) |
| | negative | false negatives (FN) | true negatives (TN) |

$$\text{true positive rate (recall)} = \frac{\text{TP}}{\text{actual pos}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{false positive rate} = \frac{\text{FP}}{\text{actual neg}} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

# Precision & Recall




$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$


- **Precision**: measure of ***exactness***. What percentage of positive classifications are actually positive?
- **Recall**: measure of ***completeness***. What percentage of positive tuples are classified as such? Similar to ***sensitivity***
- [Recommended learning material]() (video 13 minutes)

Image source: wikipedia.org

# F1-Score

- F1-score is computed as the weighted average of precision and recall between 0 and 1, where the ideal F1 score value is 1
- It’s good to have a single metric for optimization and comparison purposes, instead of two (precision and recall)
- It only takes into account the combined effect of precision and recall - treats them equally

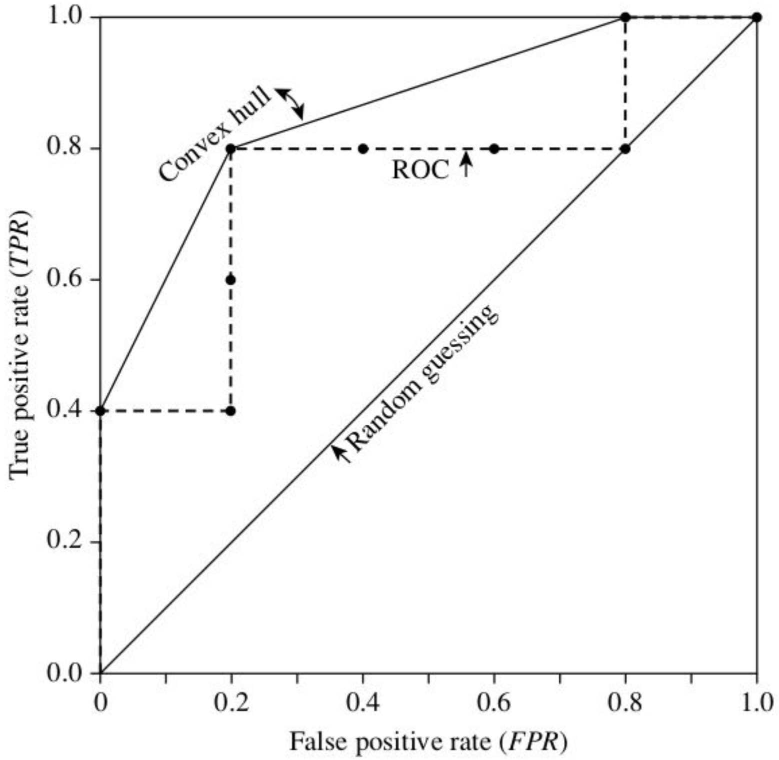
$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}.$$

# Evaluating binary classifiers

| *Measure* | *Formula* |
|---|---|
| accuracy, recognition rate | $\frac{TP+TN}{P+N}$ |
| error rate, misclassification rate | $\frac{FP+FN}{P+N}$ |
| sensitivity, true positive rate, recall | $\frac{TP}{P}$ |
| specificity, true negative rate | $\frac{TN}{N}$ |
| precision | $\frac{TP}{TP+FP}$ |
| $F$, $F_1$, $F$-score, harmonic mean of precision and recall | $\frac{2 \times precision \times recall}{precision + recall}$ |

- **TP** (true positives): num samples correctly classified as positive (yes)
- **TN** (true negatives): num samples correctly classified as negative (no)
- **FP** (false positives): num samples incorrectly classified as positive
- **FN** (false negatives): num samples incorrectly classified as negative

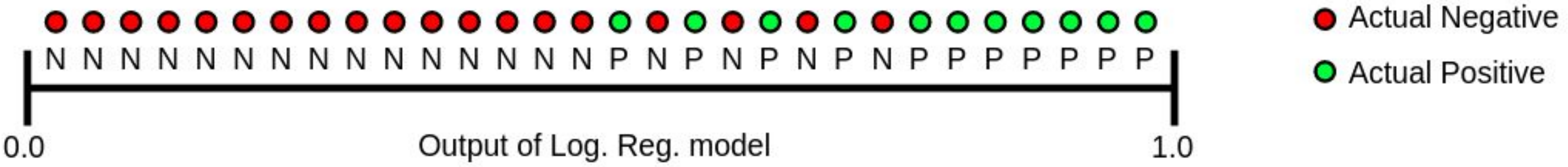
# ROC (Receiver Operator Characteristic) Curve, and the Area Under the Curve (AUC)

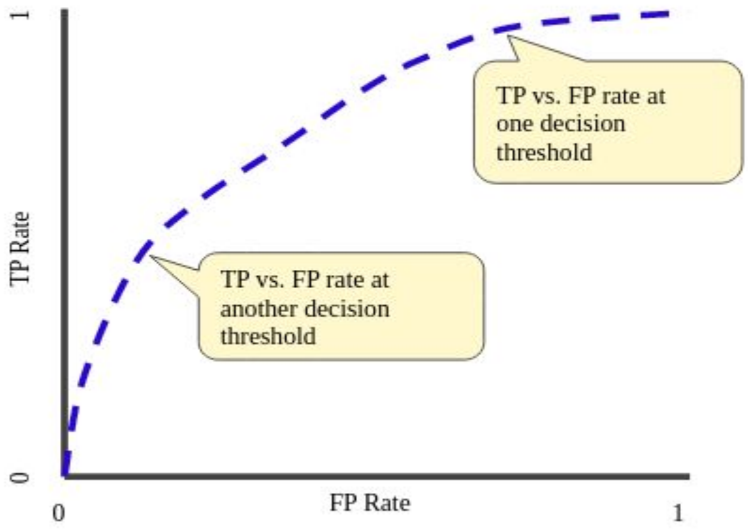


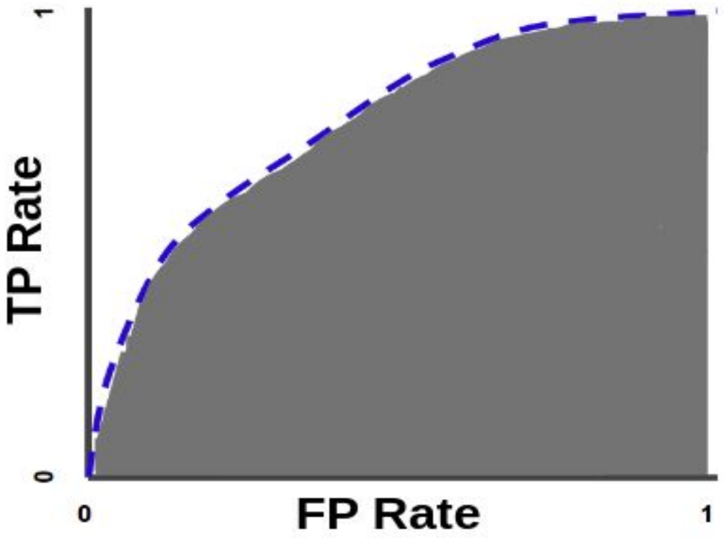

*Sample ROC curve*

- Assume a two-class classification
- For a given classifier, pick a range of values for the threshold ***t*** to determine yes/no for each data point
    - Each value of ***t*** should result in a ***(TPR, FPR)*** tuple
    - Plotting all these (TPR, FPR) values results in the ROC curve
- Shows the tradeoff between true positives and false positives
- Closer the curve is to the diagonal, worse the accuracy
    - Lower area under the curve (AUC)
    - AUC = 1, when accuracy is perfect

# ROC construction



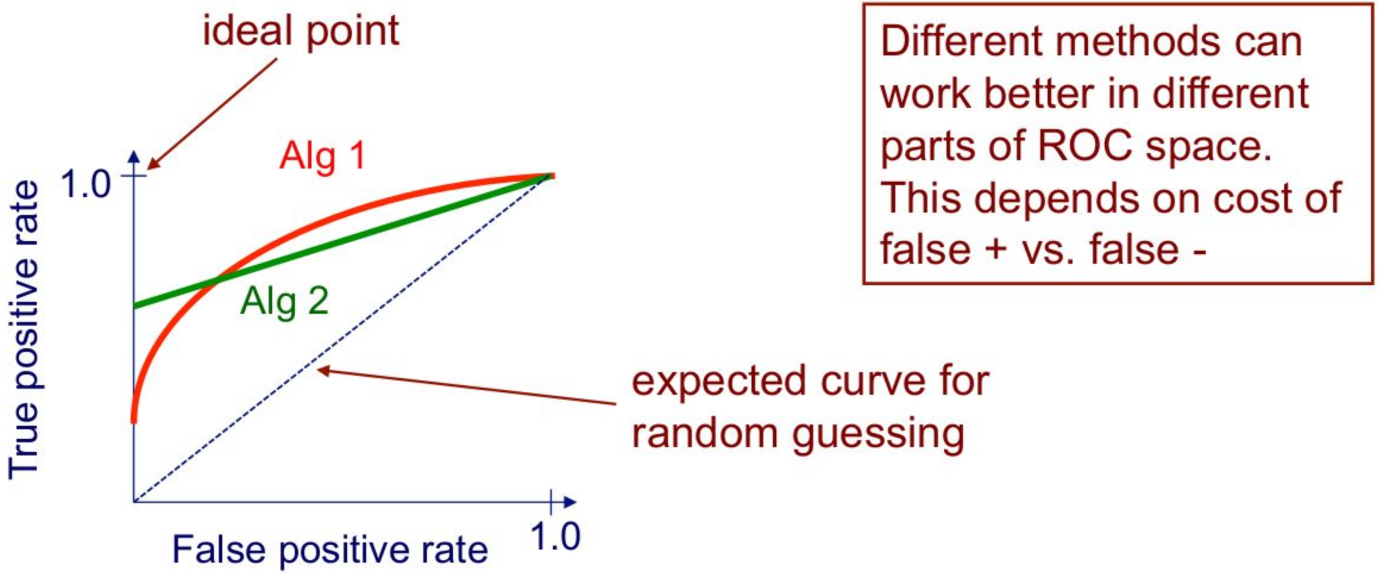




[Reference]

# ROC curves

A ROC curve plots the TP-rate vs. the FP-rate as a threshold on the confidence of an instance being positive is varied



# ROC curve: example

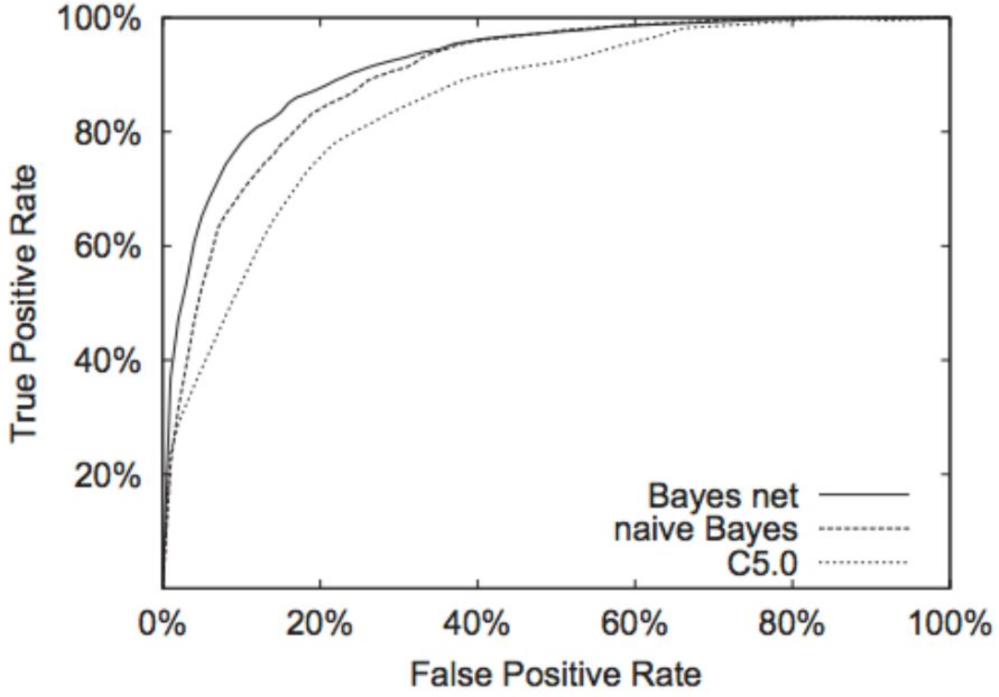



Figure from Bockhorst et al., Bioinformatics 2003

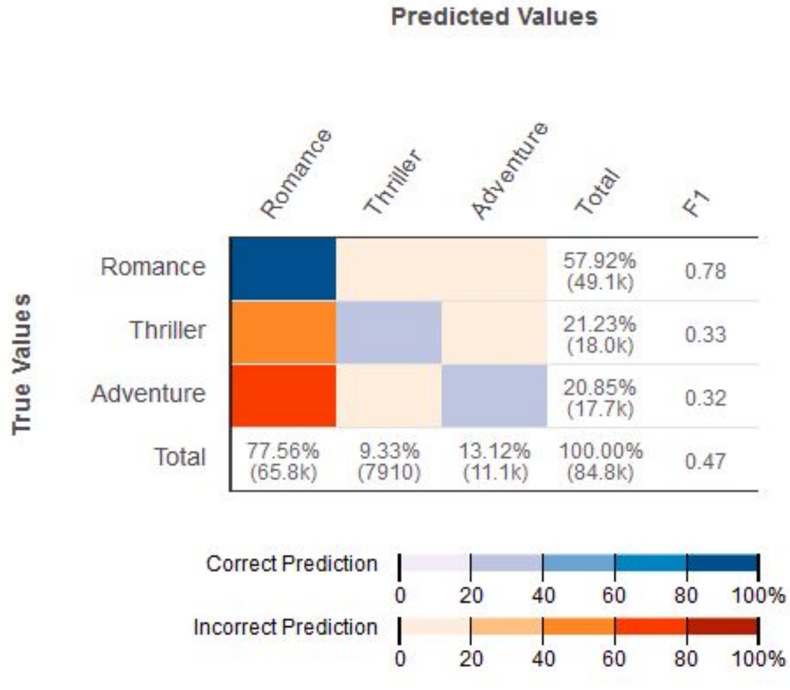
# Evaluating multiclass classifiers

- Macro average F1-score

$$F1\ score = \frac{2 * \text{precision} * \text{recall}}{\text{precision}\ + \text{recall}}$$

$$Macro\ average\ F1\ score = \frac{1}{K}\sum_{k=1}^{K} F1\ score\ for\ class\ k$$

- Confusion matrix (multiclass)



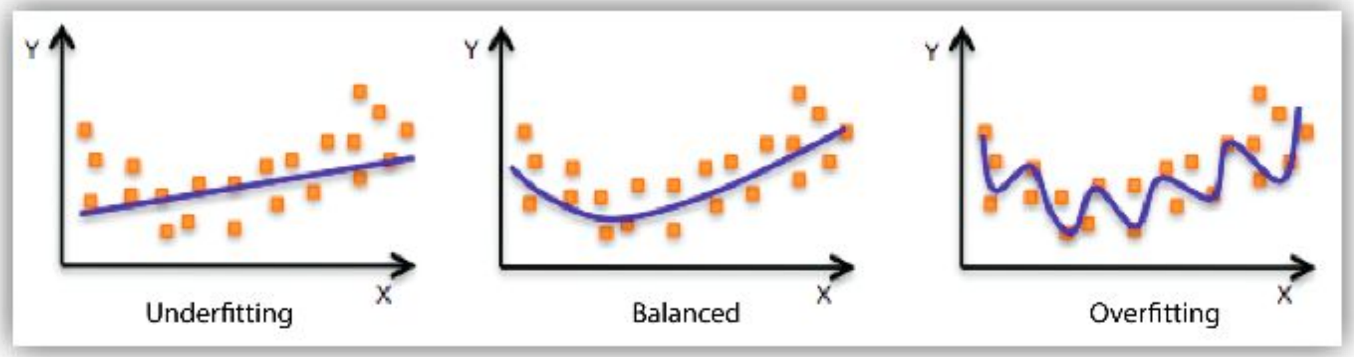
# Choosing evaluation metrics

- When you are building a model for a machine learning task, it's good to try out multiple evaluation methods because different evaluation metrics capture different aspects of the model's performance.
- But to decide if you are making progress or not with training/developing your model, you have to choose a single metric (a goal to optimize)
- Refer to the exact requirements/constraints/priorities (accuracy, precision, recall, speed,...) when choosing which evaluation metrics to consider and what to prioritize

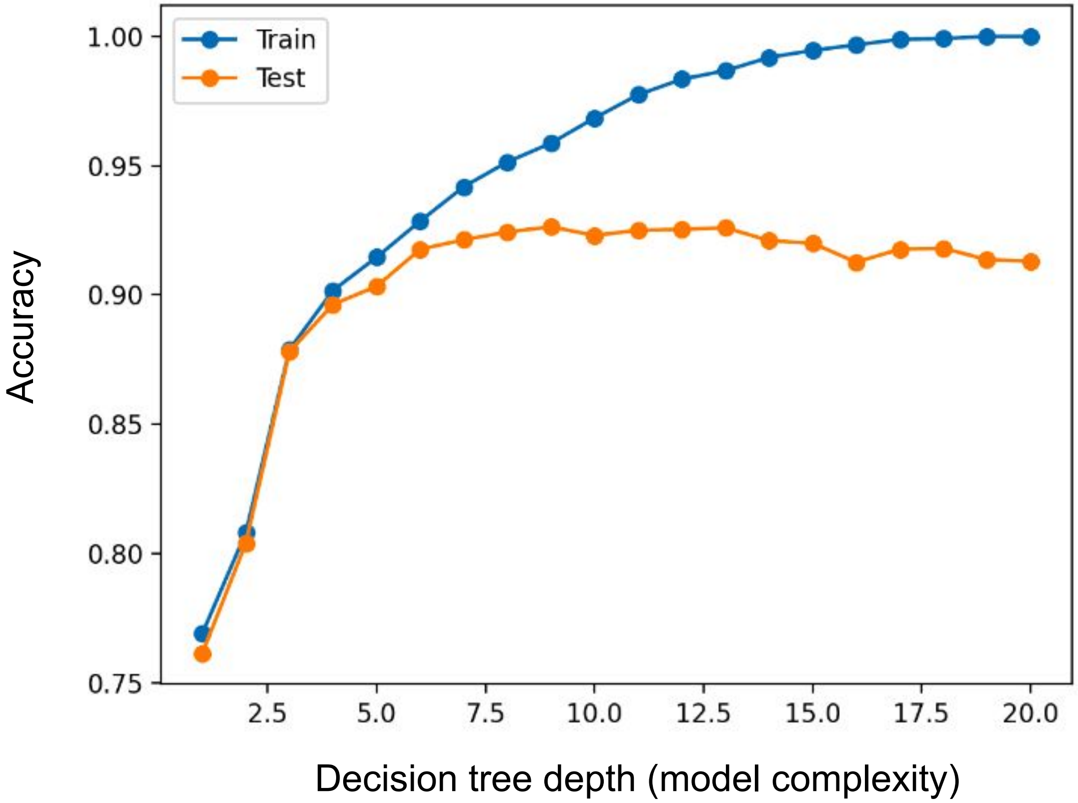
# Overfitting and Underfitting

# Overfitting and underfitting

- Your model is **underfitting** the training data when the model performs poorly on the training data.
- Your model is **overfitting** your training data when you see that the model performs well on the training data but does not perform well on the evaluation data.



# Overfitting example



# What to do when your model is underfitting

- Add new domain-specific features and more feature Cartesian products, and change the types of feature processing used (e.g., increasing n-grams size)
- Decrease the amount of regularization used
- Increase the complexity (e.g. number of parameters) of the model

# What to do when you model is overfitting

- Feature selection: consider using fewer feature combinations, and decrease the number of numeric attribute bins.
- Increase the amount of regularization used.
- Reduce the complexity (e.g. number of parameters) of the model

# Further reading

- Design and analysis of ML experiments (Purdue)
- How to select a machine learning algorithm (Microsoft)
- Machine learning experiment management
- Train / dev / test split

# Announcement - Mini Project (Kaggle competition)

- Mini Project for CS 3110 will be online soon - Kaggle Competition
- Will be using the same dataset as CS2500 Data Science Challenge
- Please download the test dataset again since it has been updated