UNIVERSITY OF MARYLAND

MSML610: Advanced Machine Learning

Machine Learning Techniques

Instructor: GP Saggese, PhD - gsaggese@umd.edu

References:

- AIMA: ?
- Hastie: ?

Paradigms

- **Paradigms**
- Techniques

Machine Learning Paradigms with Examples (1/3)

- **Supervised Learning**
  - Learn from labeled data to predict labels for new inputs
  - E.g., image classification using ResNet on ImageNet
- **Unsupervised Learning**
  - Discover hidden patterns or structure in unlabeled data
  - E.g., K-means clustering for customer segmentation
- **Reinforcement Learning**
  - Learn through interaction with an environment, receiving rewards/punishments
  - E.g., deep Q-Learning for playing Atari games
- **Self-Supervised Learning**
  - Generate pseudo-labels from unlabeled data to pre-train models
  - E.g., BERT (Masked Language Modeling)
- **Semi-Supervised Learning**
  - Combine small labeled data with large unlabeled data to improve performance
  - E.g., named entity recognition with few annotated sentences with entity tags, combined with many raw text documents

Machine Learning Paradigms with Examples (2/3)

- **Online Learning**
  - Learn incrementally from a stream of data in real time
  - E.g., online logistic regression for click-through rate prediction
- **Multi-Task Learning**
  - Train a model to perform multiple related tasks simultaneously
  - E.g., learn sentiment analysis and question answering
- **Meta-Learning**
  - "Learning to learn": Adapt quickly to new tasks using prior experience
  - E.g., learn a similarity function between examples
  - E.g., train a model's initialization such that it can be fine-tuned quickly on a new task using just a few gradient steps.
- **Few-Shot / Zero-Shot Learning**
  - Generalize to new tasks with few or no labeled examples
  - E.g., GPT-4 solving tasks with zero-shot prompting
- **Active Learning**
  - The model selects the most informative samples to be labeled by an oracle (e.g., a human)
  - E.g., pick samples where the model is least confident to get more examples

Machine Learning Paradigms with Examples (3/3)

- **Federated Learning**
  - Train models across decentralized devices without sharing raw data
  - E.g., fraud detection or credit scoring across banks
- **Evolutionary Learning**
  - Optimize model structures or parameters using evolutionary algorithms inspired by natural selection and genetics
  - Gradient free, global search, discrete structures, variable length inputs
  - E.g., genetic algorithms
- **Curriculum Learning**
  - Train models on easier tasks first, gradually increasing difficulty
  - E.g., curriculum-based training in robotic control simulations
- **Multi-Agent Learning**
  - Multiple agents learn and interact in shared environments, often in game-theoretic settings
  - E.g., AlphaStar to play StarCraft II

Supervised Learning

- Learn a function $f : X \to Y$ that maps inputs to correct outputs from training examples $(\underline{\boldsymbol{x}}, y)$ where input-output pairs are known
  - Requires labeled data for training
  - Performance measured by error on a separate test set
- **Classification**: output is a discrete label, e.g.,
  - Predict `Spam` vs `Not Spam`
  - Digit recognition `0`, `1`, ...
  - Sentiment analysis `Pos`, `Neg`, `Neutral`
- **Regression**: output is a continuous value, e.g.,
  - Predicting house prices given features like size and location
  - Predict demand
  - Forecast stock prices
- Common algorithms:
  - Linear Regression
  - Decision Trees
  - k-NN
  - Neural Networks
  - ...

Unsupervised Learning

- Learn from data without labeled outputs
  - Goal: discover patterns, groupings, or structure in the data
  - No explicit feedback signal
  - Evaluation is often qualitative
- **Main techniques:**
  - **Clustering**: Group similar examples
    - E.g., customer segmentation
    - E.g., grouping news articles by topic without knowing the topics
  - **Dimensionality Reduction**: Reduce number of features
    - Reduce number of variables with PCA while preserving structure
    - E.g., visualizing high-dimensional data in 2D
  - **Density Estimation**: Estimate probability distribution of data
    - E.g., anomaly detection in server logs
  - **Association Rule Learning**
    - Discover interesting relations between variables
    - E.g., market basket analysis (e.g., "people who buy X also buy Y")
- **Common algorithms:**
  - k-means
  - PCA
  - Autoencoders

Reinforcement Learning

- Learn by interacting with an environment to maximize cumulative reward
  - Learn a policy $\pi(s) \rightarrow a$ that maximizes expected reward
  - Trade-off between exploration (trying new actions) and exploitation (using known good actions)
  - Environments provide clear rules and feedback (win/loss/reward)
  - Often involves physical simulation or real-world interaction
- **Core elements:**
  - Agent: Learner and decision maker
  - Environment: Everything the agent interacts with
  - State s, Action a, Reward r
- **Algorithms:**
  - Q-learning
  - Policy Gradient methods
- **Examples**:
  - In game playing learn strategies through trial and error
    - E.g., AlphaGo mastering the game of Go
  - In robotics learn control policies for movement and manipulation
  - In autonomous driving learn safe and efficient driving behaviors
  - In resource management optimize allocation of limited resources over time
    - E.g., data center cooling or CPU job scheduling
  - In personalized recommendations adapt suggestions based on user interaction

Techniques

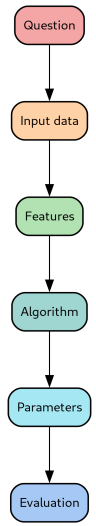
- Paradigms
- **Techniques**
  - Machine learning in practice
  - How to do research
  - Pipeline organization
  - Input processing
  - Learning algorithms
  - Performance metrics
  - Model selection
  - Aggregation

Machine learning in practice

- Paradigms
- Techniques
  - **Machine learning in practice**
  - How to do research
  - Pipeline organization
  - Input processing
  - Learning algorithms
  - Performance metrics
  - Model selection
  - Aggregation

Machine learning flow

- Question
  - E.g., "How can we predict house prices?"
- Input data
  - E.g., historical data of house sales
- Features
  - E.g., number of bedrooms, location, square footage
- Algorithm
  - E.g., linear regression, decision trees
- Parameters
  - E.g., learning rate, number of trees in a random forest
- Evaluation
  - E.g., accuracy, precision, recall
- *"If I were given one hour to save the planet, I would spend 59 minutes defining the problem and one minute resolving it"* (Albert Einstein)
- **Not all phases are equally important!**
  - Question > Data > Features > Algorithm
  - Clarity of the question impacts project success
  - Quality and relevance of data are crucial for performance
  - Proper feature selection simplifies the model and improves accuracy



Question

- The question needs to be as concrete and precise as possible
  - Clearly define the problem you are trying to solve
  - Specify the inputs and expected outputs
  - Align the question with business or research objectives
  - E.g.,:
    - **Bad**: *"How can we improve sales?"*
    - **Good**: *"What factors most significantly impact the sales of product X in region Y during season Z?"*
- This is the most important part of the whole machine learning problem
  - It's like not understanding the problem in an exam: you might solve the problem, but it's the wrong one
  - Misalignment can lead to gathering the wrong data, choosing inappropriate algorithms, ... $\rightarrow$ failing of the ML project

Input data

- Ensure data is specific to the prediction goal
  - E.g., *"Use known movie ratings to predict unseen movie ratings from the same population of people and movie"*
  - Training set $\approx$ test set
- A relationship between data and prediction is not always direct
  - E.g., interested in prices but predict supply and demand instead
- Poor-quality data leads to inaccurate predictions
  - *"Garbage in - garbage out"*
- Recognize when data is insufficient and prevents deriving a valid answer
  - *"Combination of data and desire for an answer does not ensure that a reasonable answer can be extracted from the data"* (John Tukey)
- More data vs better models
  - Meta-studies have shown that the difference between a generic model and the best model for a problem improves like 5%
  - *"It's not who has the best algorithm that wins. It's who has the most data"* (Google researcher)
  - *"Every time I fire a linguist, the performance of the speech recognizer goes up"* (IBM researcher in speech recognition)

Features

- Features provide high-level information about inputs
  - Simplify inputs by extracting features based on problem understanding
  - E.g., use intensity and symmetry for scanned numbers instead of raw bit maps
- Characteristics of good features:
  1. Enable data compression
  2. Retain relevant information
  3. Often created with expert knowledge
- Common mistakes in feature building:
  1. Automating feature selection may lead to overfitting
     - Black box predictions can be accurate but stop working at any time
     - E.g., Google Flu's unclear feature-model link
  2. Ignoring data-specific quirks
     - E.g., mislabeling outliers
  3. Unnecessarily discarding information

Models

- **Characteristics of best models**
  - Interpretable
    - Allows users to understand and trust the model's decisions
    - E.g., decision trees are appropriate in medical studies since they produce a "reasoning"
  - Simple
    - Easier to implement and maintain
    - Reduces the risk of overfitting
  - Accurate
    - Often accuracy is traded off for remaining characteristics
    - E.g., accuracy vs interpretability, accuracy vs speed
  - Fast
    - To train and test
    - Essential for real-time applications
    - Reduces computational costs
  - Scalable
    - Can handle large datasets efficiently
    - Important for growing data and user bases
    - E.g., in the Netflix prize, Netflix didn't end up implementing the best algorithm since it wasn't scalable enough

How to do research

- Paradigms
- Techniques
  - Machine learning in practice
  - **How to do research**
  - Pipeline organization
  - Input processing
  - Learning algorithms
  - Performance metrics
  - Model selection
  - Aggregation

Occam's Razor

- *The **simplest** model that fits the data is also the **most plausible*** (Occam)
  - Trim the model to the bare minimum necessary to explain the data
  - *"An explanation of the data should be as simple as possible, but not simpler"* (Einstein?)
  - **Simple** means:
    - Less likely to fit a given data by coincidence
    - An unlikely event is more significant if it happens (formalized in terms entropy)
  - **Better** means better out of sample performance
- An object is **simple** when it is one of few possible objects
  - Polynomial of order 2 is simpler than a polynomial of order 17
    - There are many more polynomials of order 17 compared to order 2, although both are infinite sets
  - SVM (Support Vector Machine) characteristics:
    - The separating hyperplane appears wiggly, but it is defined by a few support vectors
  - Complexity of a hypothesis h
    - E.g., polynomial order, MDL (describe the hypothesis in terms of bits), Kolmogorov complexity
  - Complexity of a hypothesis set $\mathcal{H}$
    - E.g., VC dimension of the model
  - Complexity of h and $\mathcal{H}$ are related by counting: if we need l bits to specify h, then h is one of 2^l elements of a set $\mathcal{H}$

Model soundness

- We cannot blindly accept the result of modeling
  - A model should tell a story
  - Always ask yourself: *"what criticisms would you give to the model if it was presented to us for the first time?"*
- Benchmark models: what are the performance if the model outputs:
  - Outputs always 0 or 1
    - E.g., long-only model for stock predictions
  - Random results
    - I.e., bootstrap of null hypothesis "there is no prediction power"
- A perfect fit can mean nothing, e.g.,
  - Get 2 data points on a plane
    - Fit data with a linear relationship
    - It is a perfect fit
  - This means nothing since:
    - There is always a line between 2 points
    - The data cannot falsify the hypothesis
  - The model (line) is too complex for the data set (only 2 points)

Sampling bias

- A model, when learning, sees the world in terms of the training data
  - If data is sampled in a biased way, learning will produce a biased outcome
- Formally: one of the few hypothesis of Hoeffding in learning theory is that training and testing distributions are the same
- Addressing sampling bias
  - Weight or re-sample data to match testing distribution
  - If data points have zero probability ($\Pr = 0$) in the data set, no remedies are possible

Data snooping

- **Data snooping** is the improper use of data that biases ML model results
  - Common trap for practitioners
- **Sources** of data snooping
  1. Contamination of training and test sets
  2. Multiple testing issue
  3. If data affects any learning step (e.g., feature engineering, model selection, hyperparameter tuning), its assessment becomes optimistic
- **Effects** of data snooping
  - Models show inflated performance metrics which do not translate out of sample
  - Snooping leads to seemingly better performance:
    - It is a "happy minefield"

"Burning the test set"

- Repeatedly using the same data eventually leads to "success"
  - The model starts fitting to specific data quirks
  - The test set should not be used for training; this leads to over-optimism
  - *"If you torture the data long enough, it will confess whatever you want"*
- Solutions:
  - Use the test set *exactly once*
  - The VC dimension applies to the overall learning model, including all attempted models
  - MDL accounts for the number of fitting attempts in overfitting measurement
  - Adjust p-values for multiple experiments

How to achieve out-of-sample fit

- Goal: choose an hypothesis g approximates the unknown target hypothesis f

$$g \approx f \iff E_{out}(g) \approx 0$$

- Solution:
  - Achieve
    1. Good in-sample performance $E_{in}(g) \approx 0$
    2. Good generalization $E_{out}(g) \approx E_{in}(g)$
  - Then 1. + 2. $\implies$ good out-of-sample performance $E_{out}(g) \approx 0$

What to do if out-of-sample fit is poor?

- The model performs well in sample ($E_{in} \approx 0$) but poorly out of sample ($E_{out} \gg E_{in}$)
- What does it mean?
  - The in-sample performance are optimistic
  - The model is overfitted and fails to generalize
- What do we do?
  - Run diagnostics before embarking in long term projects
  - Gain insight on what works / does not work to understand how to improve performance
    - E.g., bias-variance curves and learning curves
- How to fix?
  - It depends on the diagnostics!

1. Training data
   - Get more training data (it can take long time) $\iff$ fixes high variance
2. Features
   - Remove features $\iff$ fixes high variance
   - Add more features (it can take long time) $\iff$ fixes high bias
   - Add derived features (e.g., polynomial features) $\iff$ fixes high bias

Why using a lot of data?

- Several studies show that:
  - Different algorithms/models have remarkably similar performance
  - Increasing training set improves performance
- Thus it holds that:

  $$\text{High capacity model} + \text{massive training set} = \text{good performance}$$

- Using a high capacity model with many parameters (e.g., neural network)

  $$E_{in} \approx 0$$

  due to low bias (and high variance)
- A massive data set helps avoid overfitting

  $$E_{out} \approx E_{in}$$

- These two conditions together

  $$E_{out} \approx E_{in} \approx 0 \implies E_{out} \approx 0$$

What to do when one has lots of data?

- You have $m = 100$M examples in data set, what do you do?
- Training on a lot of data might yield scalability issue:
  - Slow
  - Lots of compute
  - Require work on infrastructure
  - ...
- Plot the learning curves as function of increasing $m = 1k, 10k, 100k, 1M, ...$
  - If the algorithm has large bias, it converges (training and validation performance are similar) at $m = 1000$
    - Add more features and complicate the model rather than training on 100M instances
  - If the variance is large, use all instances to train the model

Why we do things?

- Always
  - Ask: *"Why are we doing something?"*
    - To understand the purpose of the task
  - Ask: *"What do we hope to determine by performing the task?"*
    - To clarify goals and outcomes of the task
  - Encourage thinking about actions with the bigger picture in mind
  - Avoid merely going through motions
  - Promote critical thinking and awareness
  - Prioritize tasks by importance and impact
- E.g., when conducting a customer survey, ask:
  - *"Why is feedback being collected?"*
    - To improve product features and customer service
  - *"What is the desired outcome?"*
    - To identify areas for improvement and innovation
- E.g., before starting a marketing campaign, ask:
  - *"Why is this campaign run?"*
    - To increase brand awareness or drive sales
  - *"What are the specific goals?"*
    - Set target number of new leads or click-through rates

Summary of the results, next steps, follow ups

- Always have a summary of the results
  - It's like a high-level map of what we have done and what we have discovered
    - E.g., "smoothing model coefficients helps"
  - Highlight major findings
  - Interpret the results
    - E.g., *"The increase in sales is likely due to the new marketing strategy."*
  - Conclusions
    - Summarize what the data suggests or confirms
    - E.g., *"Our hypothesis that user engagement increases retention is supported"*
- Always have a reference to more detailed results
  - Provide quick insights before diving into details
- Always have next steps / follow-ups
  - What do you expect that will happen?
  - What results do you expect?
    - Like thinking n moves ahead in chess
  - E.g., *"Next, we will conduct a detailed analysis on the demographics contributing most to sales growth"*
  - Outline potential experiments or analyses to validate findings further

Example of spam filter classification

- We use $N = 4$ words in an email to distinguish spam from non-spam emails using logistic regression
  - Words can be: `buy`, `now`, `deal`, `discount`, `<your name>`
- How to improve the performance of this classifier?
  1. Collect more data
     - E.g., honeypot project: set up fake email account and collect spam
  2. Use better features
     - E.g., email routing information: spammers use unusual accounts and mask emails as legitimate
  3. Use better features from message body
  4. Detect intentional misspellings
     - Spammers use misspelled words (e.g., `w4tch` for `watch`) to confuse the classifier
     - Use stemming software

Right and wrong approach to research

- **Bad**
  1. It is not clear how to prioritize the different possible tasks
  2. Use gut feeling and pick one task
  3. Complete the task
  4. Re-evaluate performance
- **Good**
  1. Build a simple algorithm
     - Within 1 day
  2. Set up the performance evaluation framework
     - A single number and bounds to evaluate
     - Aim to improve that number
     - Evaluate with cross-validation
  3. Set up diagnostic tools
     - Compute learning and bias-variance curves
     - Avoid premature optimization by understanding the issue before fixing it
  4. Manually review misclassified emails in the cross-validated set
     - What features might help to improve performance?
     - E.g., what types of emails are misclassified?
- Sometimes an approach must be tried to see if it works
  - E.g., stemming software to consider certain words equivalent

Pipeline organization

- Paradigms
- Techniques
  - Machine learning in practice
  - How to do research
  - **Pipeline organization**
  - Input processing
  - Learning algorithms
  - Performance metrics
  - Model selection
  - Aggregation

How are machine learning systems organized?

- Machine learning systems are typically organized in a pipeline
  1. Break down the problem into sub-problems
  2. Solve problems one at the time
  3. Combine the solutions to the sub-problems into a solution to the initial problem
- The performance p of the entire ML pipeline are given by:

$$p_{system} = \sum_i p_i \cdot \alpha_i$$

where:

- p_i is the performance of each stage p_i
- α_i is the importance of each stage

ML pipeline: Example of Photo OCR system

- Goal: build systems to read text in a picture
  - OCR = "Optical Character Recognition"
- Stages of ML pipeline for OCR:
  - Text detection: find areas of the picture with text
  - Character segmentation: split text into boxes, one per letter
    - E.g., `h e l l o`
  - Character classification: classify characters, one at a time
  - Spelling correction: fix errors in text using context
    - E.g., `hell0` corrected to `hello`
- Issues with text detection:
  - Unknown text location and size
- Solution
  - Use a sliding window classifier
  - Works as evaluating a classifier is often cheap compared to training
  - Sliding window classifiers can be used for text detection and character segmentation
- **Text detection**
  - Train a classifier to recognize letters vs non-letters
  - Scan image in two directions, different sizes looking for text
  - Create a map of text likelihood (e.g., heatmap) using classifier probabilities
  - Enclose text areas in boxes
  - Discard boxes not fitting aspect ratio (valid text width > height)

The ideal recipe for ML

- The ideal recipe for ML is:

  low-bias algorithm + massive amount of data to train

  - Use learning curves to make sure we are taking advantage of more data
- Always ask yourself: *"how much work is to get 10x more data than we currently have?"*
- Often it is not that difficult:
  1. Artificial data
     - E.g., synthesize or amplify data set
  2. Collect and label by hand
     - E.g., crowd sourcing like Amazon Mechanical Turk

OCR pipeline: example of artificial data synthesis

- How can we increase data set size?
  1. Synthesize data set
     - Use font libraries to generate large training sets
     - Paste characters against random backgrounds
     - Apply scaling, distortion, adding noise, etc
  2. Amplify a data set
     - Start from a training set and add examples by warping/distorting existing examples
- Transformations and noise should be specific to the application domain
  - E.g., Gaussian noise is not always appropriate

Ceiling analysis for ML pipeline

- The most valuable resource is time
  - Sometimes one works on an optimization for months
  - The optimization doesn't make much difference
- **Problem**: On which part of the pipeline should time/resource be spent?
- **Solution**: Ceiling analysis
  - Technique to analyze performance of pipelines
  - Have a single number representing the performance of the entire system
    - E.g., accuracy for an OCR system
  - For each component:
    - Mock the component with a box that always gives the correct output (=oracle)
    - Leave the remaining components untouched
    - Compute performance of the entire pipeline
  - Understand which component is critical to performance by estimating an upper bound for overall performance when that component improves 10%
  - Don't trust your gut feeling but measure!

Input processing

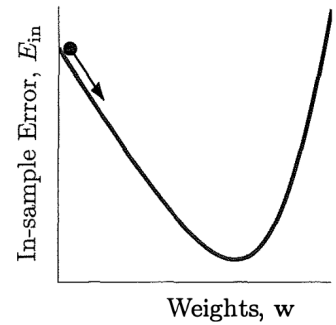
- Paradigms
- Techniques
  - Machine learning in practice
  - How to do research
  - Pipeline organization
  - **Input processing**
  - Learning algorithms
  - Performance metrics
  - Model selection
  - Aggregation
- Data cleaning
- Dimensionality reduction
- Feature engineering

Learning algorithms

- Paradigms
- Techniques
  - Machine learning in practice
  - How to do research
  - Pipeline organization
  - Input processing
  - **Learning algorithms**
  - Performance metrics
  - Model selection
  - Aggregation

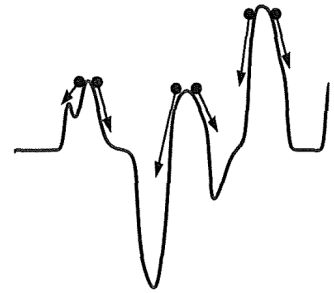
The problem of minimizing a function

- **Goal**: minimize a function $J(\underline{\boldsymbol{w}})$
  - E.g., in-sample error $E_{in}(\underline{\boldsymbol{w}})$
- **Solutions**:
  1. Analytical solution
     - Impose the gradient of $J(\underline{\boldsymbol{w}})$ to equal 0
     - Find a closed-form solution for $\underline{\boldsymbol{w}}^*$
  2. Numerical solution:
     - Use an iterative method to update $\underline{\boldsymbol{w}}$ to reach the minimum value of $J(\underline{\boldsymbol{w}})$
     - E.g., gradient descent
     - It works even if there is an analytical solution



Gradient descent: intuition

- **Problem**:
  - We are on a hilly surface and we want to walk down to the bottom of the hill
- **Solution**:
  - At each point:
    - We look around
    - We move a step in the direction where the surface is steepest
  - We keep doing until we reach the bottom
- Gradient descent
  - Is a general technique for minimizing a twice-differentiable function
  - Converges to
    - A local minimum in general
    - The global minimum if $J(\underline{\boldsymbol{w}})$ is convex (e.g., logistic regression and linear models)



Gradient descent with fixed learning rate (1/3)

- Consider the contour plot of a function
- Start from a point $\underline{\boldsymbol{w}}(0)$ (random, the origin, ...)
- At each step, move a fixed amount η in the weight space (fixed learning rate):

$$\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) + \eta \hat{\underline{\boldsymbol{v}}}$$

  where $\hat{\underline{\boldsymbol{v}}}$ is a unit vector

- Pick $\hat{\underline{\boldsymbol{v}}}$ to move to a value of $E_{in}(\underline{\boldsymbol{w}})$ as negative as possible
  - The change for E_{in} is:

$$\begin{aligned}
\Delta E_{in} &= E_{in}(\underline{\boldsymbol{w}}(t+1)) - E_{in}(\underline{\boldsymbol{w}}(t)) \\
&= E_{in}(\underline{\boldsymbol{w}}(t) + \eta \hat{\underline{\boldsymbol{v}}}) - E_{in}(\underline{\boldsymbol{w}}(t)) && \text{(replacing the expression of } \underline{\boldsymbol{w}}(t+1)) \\
&= \eta \nabla E_{in}(\underline{\boldsymbol{w}}(t))^T \hat{\underline{\boldsymbol{v}}} + O(\eta^2) && \text{(using Taylor expansion)}
\end{aligned}$$

  - Gradient descent keeps only $O(\eta)$ the term and ignores the rest
  - Conjugate gradient considers up to $O(\eta^2)$ and ignores higher order infinitesimals

Gradient descent with fixed learning rate (2/3)

- The minimal value of the scalar product
  - Is $-\eta \|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|$,
  - Happens when $\hat{\underline{\boldsymbol{v}}} = -\frac{\nabla E_{in}(\underline{\boldsymbol{w}}(t))}{\|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|}$
- The change in weights is:

$$\Delta \underline{\boldsymbol{w}} = -\eta \frac{\nabla}{\|\nabla\|}$$

- It is called "gradient descent" since we descend along the gradient of the function to optimize

Gradient descent with fixed learning rate (3/3)

- Each component of the weight $\underline{\boldsymbol{w}}$ is updated with the partial derivative with respect to that coordinate:

$$\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) - \eta \hat{v}$$

$$\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) - \eta \frac{\nabla E_{in}(\underline{\boldsymbol{w}}(t))}{\|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|}$$

$$w_j(t+1) = w_j(t) - \eta \frac{1}{\|\nabla E_{in}(\underline{\boldsymbol{w}}(t))\|} \frac{\partial E_{in}(\underline{\boldsymbol{w}})}{\partial w_j}$$

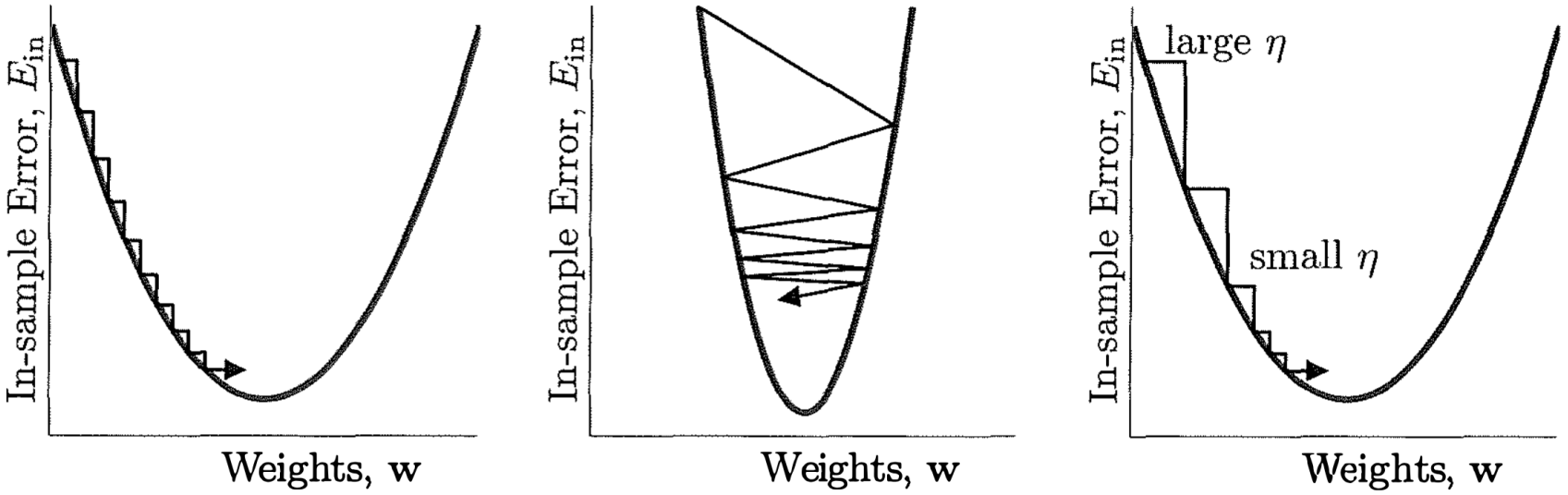
- The update of all components should be simultaneous, i.e., computed at once
- A step of the optimization when we update the solution (weights) is called epoch

Gradient descent: stopping criteria

- In theory, stop when $\Delta E_{in} = \underline{\mathbf{0}}$
  - Numerically this might not occur
- In practice, stop when
  - The variation of E_{in} is smaller than a given threshold $\Delta E_{in} < \theta$
  - We have reached a certain number of iterations
- Monitoring gradient descent
  - In theory, only need to compute the derivatives of the function $J(\underline{\boldsymbol{w}})$ to optimize
  - In practice, need to monitor the algorithm progress by recomputing the cost function $J(\underline{\boldsymbol{w}})$ periodically to make sure it is decreasing

Setting η in gradient descent with fixed learning rate

- Consider a 1D convex function
  - If η is small:
    - The linear approximation of E_{in} is effective
    - Many steps are needed to converge to the minimum
  - If η is large:
    - The linear approximation fails (higher terms affect values)
    - It "bounces around"



η too small η too large variable η just right

- Idea: vary learning rate η during gradient descent
  - Smaller learning rates may find a better minimum

Gradient descent with variable learning rate

- In gradient descent with fixed learning rate (i.e., constant change in weight space), we use:

$$\Delta \underline{\boldsymbol{w}} = -\eta \frac{\nabla J}{\|\nabla J\|}$$

- To converge quickly, we want to:
  - Move fast in weight space (large η) when the surface is steep (large gradient)
  - Move slow in weight space (small η) near the minimum to avoid bouncing around (small gradient)
- Ideally, η should increase with the slope: $\eta \propto \|\nabla J\|$
- This is called gradient descent with variable learning rate:

$$\Delta \underline{\boldsymbol{w}} = -\eta \nabla J$$

Feature scaling in gradient descent

- Gradient descent converges faster if features are scaled to the same range
  - Feature scaling techniques include min-max scaling and standardization
  - E.g., applying standardization to a dataset can transform feature values to have a mean of 0 and a standard deviation of 1
- Otherwise, different gradient components have different errors due to numerical approximation, causing the gradient to bounce around
  - Unscaled features can lead to slow and unstable convergence due to varying magnitudes
  - E.g., if one feature ranges from 1 to 1000 and another ranges from 0.01 to 1, the large disparity can cause inefficient updates

Issues with Batch Gradient Descent

- Consider the case of squared error with n samples

$$E_{in}(\underline{\boldsymbol{w}}) = \frac{1}{n}\sum_i e(h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_i) - y_i) = \frac{1}{n}\sum_i (h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_i) - y_i)^2$$

- The Batch Gradient Descent (BSD) requires to update each component of the weight vector with an expression like:

$$\underline{\boldsymbol{w}}(t+1) = \underline{\boldsymbol{w}}(t) - \eta \frac{\nabla E_{in}}{\|\nabla E_{in}\|}$$

- In terms of coordinates for squared error:

$$w_j(t+1) = w_j(t) - \eta \frac{2}{n}\sum_{i=0}^{n} (h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_i) - y_i)\frac{\partial h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_i)}{\partial w_j}$$

- With a large number of training examples (e.g., $N = 10^6$), gradient descent:
  - Is computationally expensive as it requires evaluating the gradient from all examples for a single update
  - Requires storing all the data in memory

Stochastic Gradient Descent

- **Idea** of Stochastic Gradient Descent (SGD)
  - Update the weights only for one training example picked at random
- **Algorithm**
  - Pick one $(\underline{\boldsymbol{x}}_n, y_n)$ at a time from the available examples
  - Compute $\nabla e(h(\underline{\boldsymbol{x}}_n), y_n)$ to update the weights:

$$\Delta \underline{\boldsymbol{w}} = -\eta \nabla e$$

  - Update the weight considering only one random example:

$$w_j(t+1) = w_j(t) - \eta \frac{2}{n}(h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_t) - y_t)\frac{\partial h_{\underline{\boldsymbol{w}}}(\underline{\boldsymbol{x}}_t)}{\partial w_j}$$

- ∇e is a function of a random var $\underline{\boldsymbol{x}}_n$
  - The average direction of SGD is the same direction as batch version

$$\mathbb{E}[\nabla e] = \frac{1}{N}\sum \nabla e(h(\underline{\boldsymbol{x}}_n), y_n) = \nabla \frac{1}{N}\sum e(h(\underline{\boldsymbol{x}}_n), y_n) = \nabla E_{in}$$

- In Stochastic Gradient Descent (SGD):
  - The path in the weight space is more random
  - It does not even converge but rather oscillates around the local minimum

Mini-batch Gradient Descent

- Bring together characteristics of both Batch and Stochastic Gradient Descent
- Use b examples to make an update to the current weight
  - b represents the batch size, which is a hyperparameter you can choose
  - A common choice for b might be $b = 32$ or $b = 64$
- Mini-batch GD offers a balance between SGD noisiness and full-batch approaches, using small, random data samples for updates

On-line learning and gradient descent

- Continuous stream of training examples requires updating the model
  - In real-time systems, new data points arrive and the model adapts without fully retraining
  - E.g., in stock market prediction models, each transaction can dynamically adjust model weights
  - Handle variation in the dynamics of the underlying process
- Stochastic gradient (SGD) and mini-batch descent are suitable for online learning, updating the model one example at a time
- Discard examples for a "compressed" model representation
  - Useful for large data streams where storing every data point is impractical
  - E.g., in training a language model on live chat data, older conversations might be discarded after updates to maintain relevant patterns in the model

SGD vs BGD vs Mini-batch

- To update the weights:
  - BGD (batch gradient descent) uses all the training examples
  - SGD (stochastic gradient descent) uses a single (random) training example
  - Mini-batch GD uses only a subset of training examples

Aspect	**Batch Gradient Descent**	**Stochastic Gradient Descent**
Computation	Uses all examples	One example at a time
Memory	Requires all examples in memory	Require less memory
Randomization	More likely to terminate in flat regions	Avoid local minima due to randomness
Regularization	No implicit regularization	Oscillations act as regularization
Parallelization	Can be parallelized	Less parallel-friendly
Online Learning	Not suitable	Suitable for online learning

Map-reduce for batch gradient descent

- In map-reduce we use k machines to parallelize the summation (map step) and then we send the k partial sums to a single node to accumulate the result (reduce step)
- Batch GD (and many learning algorithms) can be expressed in this map-reduce form

Coordinate descend

- Minimize $J(x_0, ..., x_n)$ by optimizing along one direction x_i at a time
  - Instead of computing all derivatives
- **Algorithm**
  - Pick a random starting point $\underline{\boldsymbol{w}}(0)$
  - Pick a random order for the coordinates $\{x_i\}$
  - Find the minimum along the current coordinate (1D optimization problem)
  - Move to the next coordinate x_{i+1}
  - The sequence of $\underline{\boldsymbol{w}}(t)$ is decreasing
  - A minimum is found if there is no improvement after one cycle of scanning all coordinates
  - The minimum is local

Gradient descent vs pseudo-inverse for linear models

- For linear models we can use either pseudo-inverse or gradient descent to find optimal $\underline{\boldsymbol{w}}^*$
- **Gradient descent**
  - Choose learning rate η
  - Requires many iterations to converge
  - Monitor stopping criteria, oscillations, etc
  - Effective for many features P
- **Pseudo-inverse**
  - No parameter selection needed
  - Converges in one iteration (with nested loops)
  - Computes $(\underline{\underline{\boldsymbol{X}}}^T \underline{\underline{\boldsymbol{X}}})^{-1}$, a $P \times P$ matrix
    - Inverse complexity $O(P^3)$
    - E.g., for $P \approx 10,000$, gradient descent is preferable

Performance metrics

- Paradigms
- Techniques
  - Machine learning in practice
  - How to do research
  - Pipeline organization
  - Input processing
  - Learning algorithms
  - **Performance metrics**
  - Model selection
  - Aggregation

How to make progress in ML research

- There are many possible directions for research
  - Different features
  - Different data preprocessing methods
  - Different models
  - Different training algorithms
  - Different evaluation techniques
  - Explore optimization strategies
- What to do?
- Approach
  - Evaluate models systematically using a single number
    - Implement metrics (E.g., accuracy, F1 score) for insight
    - Use cross-validation for model validation
  - Statistical tests to ensure differences are not random
    - Utilize hypothesis testing for genuine improvements
    - Conduct A/B testing for real-world validation

How to measure classifier's performance?

- Success / hit / win rate (or error / miss rate)
  - Measures the proportion of correct predictions by the model
  - Important for understanding overall accuracy
  - E.g., in binary classification, 80 correct predictions out of 100 result in an 80% success rate
- Log probability / cross-entropy error
  - Evaluates classification model with probabilities between 0 and 1
  - E.g., lower cross-entropy loss indicates better performance
- **Precision / recall / F-score**
  - Useful for evaluating models in imbalanced data scenarios
  - Precision: ratio of correctly predicted positive observations to total predicted positives
    - E.g., a precision of 0.75 means 75% of identified positives are true positives
  - Recall: ratio of correctly predicted positive observations to actual positives
    - E.g., a recall of 0.60 means 60% of actual positives were correctly identified
  - F-score: weighted harmonic mean of precision and recall
- **Utility function**
  - Customizes the evaluation metric to prioritize types of errors and success
    - E.g., true / false positives / negatives
  - E.g., in medical diagnosis, a utility function might give higher weight to minimizing false negatives to prevent missed diagnoses

Training vs test set

- Performance on train set E_{in} is an optimistic estimate of E_{out}
  - One can have:
    - 0% error rate on training data (e.g., memorizing responses for training set)
    - 50% error rate on test set (e.g., by answering randomly)
- To evaluate model performance, use a test set that played no role in training
- Training and test sets should be representative samples of the problem
  - E.g., credit risk problem
    - One cannot use data from a bank branch in Florida to assess a model built with data from a bank branch in New York
    - Characteristics of the populations are very different

Lots of data scenario vs scarce data scenario

- **Lots of data scenario**
  - Ideal to have lots of data (ideally infinite)
  - Learn on lots of data
    - Fit all degrees of freedom of a complex model
  - Predict on lots of data
    - Assess precise out-of-sample performance
- **Scarce data scenario**
  - Often data (especially data of high quality) is scarce
    - E.g., facial recognition datasets with limited annotated data needing careful management
  - Cannot use all data as a training set
  - Need to hold out data to estimate performance metrics and bounds
    - Split the data 70-30 or 80-20 in train and test sets
    - Consider cross-validation techniques to maximize data usage
  - Other approaches:
    - Augment data artificially, like data augmentation in image processing
    - Utilize transfer learning with pre-trained models on related tasks

Splitting Data into Training, Validation, Test Sets

- Training, validation, and test sets must be:
  - Distinct
  - Representative of the problem
    - E.g., each class in all sets must be represented according to the original data
  - Sized based on available data and problem needs
- To ensure sets have the same distribution:
  - Stratified sampling
    - E.g., each class label is proportionally represented in each set
  - Shuffle and then sample
    - Achieves randomization, maintaining distribution
  - Sample and check statistics of variables (e.g., mean, std dev, PDF)
    - Compare these statistics to ensure each set mirrors the broader dataset

Rule of thumbs for data set splits

- If n is **large** $\rightarrow$ use a 60-20-20 split
  - Training: 60%
  - Validation: 20%
  - Test: 20%
- If n is **medium** $\rightarrow$ use a 60-40 split
  - Training: 60%
  - Test: 40%
  - Not possible to learn hyperparameters, so no validation set
- If n is **small** $\rightarrow$ use cross-validation and report "small data size"
  - Use K-fold cross-validation
  - Be cautious of the increased chance of high accuracy by chance
  - Is machine learning for the given sample size even suitable?

Can we ever use test set as training set?

- Once the model is selected and validated, reuse all available data (including the test set) to generate the model for deployment
  - This ensures the model benefits from all available information
- Generally, more data is better, though returns diminish after exceeding a certain volume
  - Initially, increasing data size can significantly improve model performance
  - Eventually, adding more data results in smaller accuracy gains and may not justify the increased computational cost

In-sample vs out-of-sample error expressions

- We want to find a function h that approximates the unknown function f, $h \approx f$ over the space of inputs $\underline{\boldsymbol{x}} \in \mathcal{X}$ ("script X")
- The error is usually defined point-wise:

$$e(h(\underline{\boldsymbol{x}}_i), f(\underline{\boldsymbol{x}}_i))$$

  - E.g.,
    - Squared error: $e(\underline{\boldsymbol{x}}) = (h(\underline{\boldsymbol{x}}) - f(\underline{\boldsymbol{x}}))^2$
    - 0-1 binary error: $e(\underline{\boldsymbol{x}}) = I[h(\underline{\boldsymbol{x}}) == f(\underline{\boldsymbol{x}})]$
    - Log probability: $e(\underline{\boldsymbol{x}}) = -\log(\Pr(h(\underline{\boldsymbol{x}}) == f(\underline{\boldsymbol{x}})))$
- In-sample error is computed using all points in the training set:

$$E_{in}(h) = \frac{1}{N} \sum_{i=1}^{N} e(h(\underline{\boldsymbol{x}}_i), f(\underline{\boldsymbol{x}}_i))$$

- Out-of-sample error is computed on the entire space of inputs $\mathcal{X}$

$$E_{out}(h) = \mathbb{E}_{\underline{\boldsymbol{x}} \in \mathcal{X}}[e(h(\underline{\boldsymbol{x}}), f(\underline{\boldsymbol{x}}))]$$

Mean squared error (MSE)

- MSE is the average difference of squared error:

$$\text{MSE} \overset{def}{=} \frac{1}{N}\sum_{i=1}^{N}(h(\underline{\boldsymbol{x}}_i) - f(\underline{\boldsymbol{x}}_i))^2$$

  - MSE measures the estimator quality, quantifying the difference between estimated and actual values
  - E.g., in a house price prediction model, MSE determines how close predicted prices are to actual prices
- **Cons:**
  - It doesn't share the unit of measure with the output
    - Distorts error interpretation; predicted and actual values are usually in different units
  - Sensitive to outliers
    - A single large error can disproportionately affect the MSE
    - Use median absolute deviation (MAD), median of squared error for robustness against outliers

Root mean squared error (RMSE)

- RMSE is the standard deviation of the Mean Squared Error (MSE):

$$\text{RMSE} \overset{def}{=} \sqrt{\text{MSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(h(\underline{\boldsymbol{x}}_i) - f(\underline{\boldsymbol{x}}_i))^2}$$

- **Pros:**
  - Same units as the output, allowing intuition of its magnitude compared to the mean
  - Facilitates comparison between different data sets or models since the metric is normalized to the output's scale
- **Cons:**
  - Sensitive to outliers (like MSE) which can excessively affect the metric
  - May not be suitable for ranking models when outliers or skewed distributions are present

Median-based metrics

- We can use metric based on median (i.e., the 0.5 quantile of absolute error):
- Median absolute deviation:

$$\text{MAD} \stackrel{def}{=} \text{median}_i(|h(\underline{\boldsymbol{x}}_i) - f(\underline{\boldsymbol{x}}_i)|)$$

- Median squared error:

$$\stackrel{def}{=} \text{median}_i(|h(\underline{\boldsymbol{x}}_i) - f(\underline{\boldsymbol{x}}_i)|^2)$$

How to choose an error measure?

Error measure depends on the **application** and should be **specified by the "customer"**: - The customer needs to define what constitutes an acceptable level of error for their specific use case - E.g., medical applications might have a low tolerance for errors, while a recommendation system might have a higher tolerance

- Otherwise, we can pick:
  - A **plausible error measure**:
    - E.g., squared error is commonly used when assuming Gaussian noise in the data
  - A **"friendly error" measure**:
    - E.g., measures that allow for closed-form solutions simplify calculations significantly
    - Convex optimization-friendly measures ensure optimization algorithms find the global minimum easily

Error measures: fingerprint verification example

- In fingerprint verification:
  - Recognizing a valid fingerprint has no error
  - Otherwise, it is a false positive or a false negative
- Error weight depends on the application
  - For the same problem in two set-ups, the error measure is the opposite
  - For supermarket applications:
    - False positives are minor (e.g., one more discount)
    - False negatives are costly (e.g., annoyed customer, slow line)
  - For CIA building access:
    - False negatives are acceptable (triggers further security)
    - False positives are disastrous

Error metrics for skewed classes

- When classes are skewed (i.e., one class is very rare), accuracy can be misleading
  - Use metrics like confusion matrix, precision, and recall
- Example:
  - Train a classifier to distinguish tumors as:
    - $y = 1$: malignant
    - $y = 0$: benign
  - Classifier's error rate is 1% (i.e., guess correctly 99% of the time) seems good
  - But only 0.5% of patients have cancer
    - A trivial classifier that always outputs $y = 0$ has a 0.5% error rate!
    - Now a 1% error rate does not look good anymore

Decision matrix

- Aka confusion matrix
- Typically $y = 1$ encodes the rare class to predict
- Assuming actual and predicted class $\in \{0, 1\}$, we have 4 possible cases:
  - $act = 1$, $pred = 1$: true positive (TP)
  - $act = 0$, $pred = 0$: true negative (TN)
  - $act = 1$, $pred = 0$: false negative (FN) (output $pred = 0$, but it is wrong)
  - $act = 0$, $pred = 1$: false positive (FP) (output $pred = 1$, but it is wrong)
- Aggregate decision matrix in precision and recall

	$\text{pred} = 1$	$\text{pred} = 0$
$\text{act} = 1$	**TP**	**FN**
$\text{act} = 0$	**FP**	**TN**

Precision vs recall

- Assume that $y = 1$ encodes the rare event we want to detect
- **Precision** measures how often there is a true positive *given that pred = 1*

$$\text{precision} \stackrel{def}{=} \Pr(\text{TP}|\text{pred} == 1) = \frac{|\text{pred} == 1 \wedge \text{act} == 1|}{|\text{pred} == 1|} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall** measures how often there is a true positive *given that act = 1*

$$\text{recall} \stackrel{def}{=} \Pr(\text{TP}|\text{act} == 1) = \frac{\text{TP}}{|\text{act} == 1|} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- Both are conditional probability measuring the fraction of TP under different circumstances:
  - (Pre)cision: pred = 1
  - Rec(a)ll: act = 1
- Precision/recall are widely used in information retrieval
  - E.g., a search engine:
    - Returns 30 pages; only 20 are relevant $\implies$ precision = 20 / 30 = 2 / 3
    - Fails to return another 40 relevant pages $\implies$ recall = 20 / (40 + 20) = 20 / 60 = 1 / 3

Precision / recall in terms of quality / quantity

- **Precision**
  - Increasing precision means when we predict 1, we are more likely to be right
    - E.g., in a spam email detection system, "precision is 90%" means 90% of the emails marked as spam are actually spam
  - A higher precision indicates fewer false positives
  - Measures "quality" of prediction
- **Recall**
  - Increasing recall means we predict more instances when the outcome is 1
    - E.g., in a spam email detection system, "recall is 80%" indicates 80% of all actual spam emails were correctly identified as spam
  - A higher recall means fewer false negatives
  - Measures "quantity" of prediction (coverage)

Precision / recall for trivial classifiers

- A classifier that outputs always the most common class 0 has:

$$\text{precision} = 0 (\text{since TP} = 0)$$
$$\text{recall} = 0 (\text{since TP} = 0)$$

- A classifier that outputs always the rare class 1 has:

$$\begin{aligned}
\text{recall} &= 1 && (\text{since FN} = 0) \\
\text{precision} &\overset{def}{=} \Pr(\text{TP}|\text{pred} == 1) && (\text{by definition}) \\
&= \frac{\text{TP}}{\text{TP} + \text{FP}} && (TP + FP = n \text{ because} \\
&= \frac{\#(y = 1)}{n} && \text{classifier always emits 1}) \\
&= \Pr(\text{pos}) \approx 0 && (\text{the positive class is very rare})
\end{aligned}$$

- A trivial classifier has precision or recall close to 0

Trading off precision and recall

- In theory, we want to increase both precision and recall
- In practice, modify the threshold of a probabilistic classifier to trade off precision and recall in practice
- E.g., use logistic regression to predict cancer:
  - With a threshold $= 0.5$, the classifier has:
    - Precision $= \frac{\text{TP}}{|\text{pred} == 1|}$
    - Recall $= \frac{\text{TP}}{|\text{act} == 1|}$
  - Increase the threshold $\implies$ output 1 only if more confident, i.e., increase precision
  - Decrease the threshold $\implies$ output 1 more often, decreasing the chances of missing a possible case of cancer, i.e., increase recall

Precision-recall: pros / cons

- Pros:
  - Give insight on the behavior of a classifier (e.g., confusion matrix)
  - Avoid mistaking a trivial classifier for a good classifier
- Cons:
  - We have two different numbers, thus it is difficult to compare classifiers to each other
  - Solutions: F-score, AUC

Precision-recall curves

- **Aka ROC curves**
- Plot the curve on a precision-recall plane: (y = precision, $1 - x$ = recall) to show the precision vs recall trade-off for a classifier
  - E.g., changing the threshold of logistic regression
- A curve higher than another means a better classifier, since for the same recall we can get a higher precision
  - The best classifier (precision = recall = 1) is in the top-right corner
- The precision-recall plot can have different shapes, e.g.,
  - Diagonal (pure luck)
  - Convex up (better than luck)
  - Convex down (worse than luck)

Area under the curve

- **AUC** is the area under the precision-recall curve
  - Provides a robust metric by integrating over all thresholds
  - Higher AUC indicates better performance in differentiating between classes
  - AUC $= 0.5$ suggests no discriminative power, similar to random guessing,
  - AUC closer to 1.0 indicates high performance
- **Pros**:
  - Single number summarizing classifier behavior, useful for comparing different models
  - Does not require selecting a threshold for performance calculation
  - Can handle imbalanced datasets effectively
- E.g., consider a classifier for medical diagnosis
  - The AUC helps understand how well the model distinguishes between patients with and without a disease across all thresholds

F-score

- The F-score is the harmonic mean of precision and recall:

$$\text{F-score} \stackrel{def}{=} \frac{2}{\frac{1}{P} + \frac{1}{R}} = 2\frac{P \cdot R}{P + R}$$

- **Interpretation:**
  - Trivial classifiers: $P = 0$ or $R = 0 \implies$ F-score $= 0$
  - Perfect classifiers: $P = R = 1 \implies$ F-score $= 1$
  - For F-score to be large, both P and R must be high
- Why not just averaging P, R?
  - A classifier that always outputs 1 has $R = 1$ and $P = 0$
  - $\frac{P+R}{2} = \frac{1}{2}$, while we prefer a low value (ideally 0)

Model selection

- Paradigms
- Techniques
  - Machine learning in practice
  - How to do research
  - Pipeline organization
  - Input processing
  - Learning algorithms
  - Performance metrics
  - **Model selection**
  - Aggregation

Model selection problem

- Model selection chooses the best model from a set of candidates based on performance
  - Needed when multiple hypotheses can explain the data
- Certain parameters are fixed, while others need to be picked, e.g.,
  - Set of features
    - E.g., selecting a subset of features from a dataset with 100 variables
  - Learning algorithms
    - E.g., deciding how to train a neural network
  - Model types
    - E.g., linear regression model vs. Support Vector Machine (SVM)
  - Model complexity
    - E.g., models with polynomials of degree $d < 10$
  - Values of the regularization parameter
    - E.g., trying different values like 0.01, 0.1, and 1.0
- Evaluate model accuracy, precision, and recall
- Perform cross-validation to assess model performance
- Consider computational cost
  - E.g., a simple logistic regression is faster than a complex neural network

Model selection process

1. Split data into $D_{train}, D_{val}, D_{test}$
   - Commonly: 60% training, 20% validation, 20% test
   - Like splitting 80% training between two learning phases
2. Given N hypotheses, learn on D_{train} to get $g_1, ..., g_N$
3. Evaluate hypotheses on D_{val} estimating errors $E_{val}^{(1)}, ..., E_{val}^{(N)}$
4. Pick model g_m with minimum $E_{val}^{(m)}$
5. Use test set D_{test} to estimate fair performance of model g_m, i.e., $E_{val} \approx E_{out}$
6. Retrain model with entire $D = D_{train} \cup D_{val} \cup D_{test}$ to get final g_m^*

Model selection as learning

- "Picking the model with smallest E_{val}" is a form of learning:
  - Hypothesis set: $\{g_1, ..., g_N\}$
  - Training set: D_{val}
  - Pick the best model g_m
- After model selection
  - Experimentally $E_{val}(g_m) < E_{out}(g_m)$, i.e., $E_{val}(g_m)$ is a (optimistically) biased estimate of $E_{out}(g_m)$
  - Theoretically:
    - The penalty for model complexity with a finite set of hypotheses is

$$E_{out}(g_m) \leq E_{val}(g_m) + O(\sqrt{\log(N/K)})$$

    - Use VC dimension for an infinite number of hypotheses (e.g., choice of λ for regularization)

Aggregation

- Paradigms
- Techniques
  - Machine learning in practice
  - How to do research
  - Pipeline organization
  - Input processing
  - Learning algorithms
  - Performance metrics
  - Model selection
  - **Aggregation**

Ensemble learning: intuition

- Ensemble learning combines multiple models to improve prediction accuracy
  - **Idea**: a group of weak learners can form a strong learner
- Combine outputs of models X_i to build a model X^* better than any X_i, with the wisdom of all
  - Utilizes diversity in model predictions to improve accuracy
  - Each model contributes its unique perspective, reducing overfitting
  - E.g., like a panel of voting experts
- Example: in computer vision detecting a face is difficult task (at least circa 2010)
  - Look for different features:
    - Are there eyes?
    - Is there a nose?
    - Are eyes and nose in the correct position?
    - ...
  - Each feature is weak per-se, but together they become reliable

Ensemble learning: Different Techniques

- **Bagging** (bootstrap + aggregation)
  - Reduces variance by averaging predictions from different models
  - E.g., decision trees $\rightarrow$ bagging $\rightarrow$ random forest
    - Bagging creates multiple versions of a decision tree (each trained on a random sample of data)
    - Average their predictions to improve accuracy
- **Boosting**
  - Reduces bias by focusing on errors made by previous models
  - Sequentially adds models, each correcting its predecessor
  - E.g., `adaBoost` increases weights of incorrectly classified data points to learn the next model
- **Stacking**
  - Uses a meta-model to combine separate models using weights
  - E.g., a stacking ensemble
    - Uses a logistic regression as a meta-model
    - Combines the predictions of other models (e.g., decision trees, support vector machines, and neural networks)

Ensemble learning: relation with statistics

- **Bagging**
  - Improves performance by adding randomized variants (mimicking multiple training sets)
  - Reduce variance without affecting bias
- **Boosting**
  - Use another model to learn residuals, i.e., difference between predicted and true values
  - Related to the statistical technique of "forward stagewise additive models"
- **Stacking**
  - If we have 3 independent classifiers, each with $\Pr(\text{correct}) = 0.7$

$$\begin{aligned}\Pr(\text{majority correct}) &= \Pr(\text{at least 2 classifiers correct}) \\ &= \binom{3}{2} 0.7^2 0.3 + 0.7^3 \\ &= 3 \times 0.7^2 \times 0.3 + 0.7^3 \\ &\approx 0.78 > 0.7\end{aligned}$$

Ensemble learning: pros and cons

- **Pros**
  - Hypothesis set $\mathcal{H}$ is increased by combining hypotheses from different models
- **Cons**
  - More computationally intensive to train and evaluate
  - Loss of interpretability
  - Risk of overfitting (model complexity is increased)
  - Ensemble learning contradicts Occam's razor, which advocates simplicity

When ensemble learning works

- Combining multiple models with ensemble learning works when models:
  - Are very different from each other
  - Treat a reasonable percentage of the data correctly
    - E.g., one cannot do much if all classifiers have 50% accuracy
  - Complement each other: they are specialists in a part of the domain where the others don't perform well

How to combine outputs in ensemble learning

- **Regression**
  - Weighted average of prediction
  - E.g., by accuracy of each model or by a prior
- **Classification**
  - Weighted vote of predicted classes
  - It needs an odd number of models to break ties
- **Probabilistic classification**
  - Weighted average of class probabilities
- We can also learn a meta-learner (stacking) to combine multiple models

Bagging

- Bagging stands for "Bootstrap AGGregation"
- **Learning procedure**
  - Several training datasets are extracted randomly by sampling with replacement from the original dataset (i.e., bootstrap)
  - Learn multiple models, one for each training set
  - Combine outputs using various methods
  - Result is a better model than a single model
- **Why bagging works?**
  - From the bias-variance decomposition view, combining multiple models:
    - Reduces the variance component
    - Without compromising the bias (bagged models are typically unbiased)
  - Bagging mimics extracting more training sets (though not independent) from the unknown distribution

Bagging and instability in learning algorithms

- Bagging works best with different models, especially non-linear models
- Introduce randomization in the learning algorithm intentionally
- **Decision Trees**
  - Disable pruning
  - Break ties randomly when selecting the best attribute to split
  - E.g., bagging trees results in random forests
- **Multilayer Perceptrons**
  - Use different initial weights in backpropagation to reach different local minima
- **Nearest Neighbor Classifier**
  - Use a random subset of features
  - Resampling the training set has limited impact, as it is equivalent to changing example weights

Boosting

- Boosting builds models that complement each other
  - Typically use homogeneous models, i.e., parametrized models from $\mathcal{H}$
- Strong classifiers can be built from weak classifiers
  - E.g., decision stumps = decision trees with one level
- Statistical meaning of boosting:
  - Boosting implements forward stagewise additive modeling
  - Use another model to learn residuals (difference between predicted and true values)
- Boosting does not work for linear regression:
  - Combination of linear models is still a linear model
  - OLS finds optimal weights in one step
  - Combining linear regressions from different attributes is equivalent to a single multiple linear regression

Adaboost.M1

- Widely used for classification
- Assume examples can be weighted in the cost function used to learn
  - Otherwise use resampling
- **Learning procedure**
  - Start with equal weights for all examples
  - Iterate:
    - Learn a classifier based on current weights for examples
    - Weight the answer of each model by overall score (e.g., accuracy) or probability
    - Evaluate the ensemble
    - Adjust weights for examples classified correctly/incorrectly

Stacking

- Stacking learns how to combine models (not necessarily of the same type)
- The problem is that with voting / averaging we don't know which model to trust
- Instead of voting or weighting we can use a meta-learner (level 1) to learn how to pick / mix models (level 0)
- **Learning procedure**
  - Learn "level 0" models
  - Learn "level 1" model using hold-out data from learning of level 0 models (like in model selection)
    - Build training data with predicted values from level 0 models
    - Then learn level 1
    - Use a simple model for level 1 (e.g., linear models or trees) to avoid overfitting
    - Use probabilities from level 0, so level 1 can assess the confidence of each model

Boosting vs bagging vs stacking

Aspect	Bagging	Boosting	Stacking
Combines	Models of the same type	Models of the same type	Models of different types
Learning	Models trained independently	Iterative training	Models trained independently
Predicting	Uses uniform or data-driven weights	Uses learned weights from training	Uses learned weights or confidence
Main Objective	Reduce variance	Reduce bias	Improve generalization through diversity
Base Learners	Often strong learners	Often weak learners	Any model type (heterogeneous ensemble)
Sensitivity to Noise	Low	High	Medium
Parallelizable	Yes	No (sequential dependency)	Partially (base models parallelized)
Meta-model	Not used	Not used	Required
Examples	Random Forest	AdaBoost, Gradient Boosting	Stacked Generalization, Blending