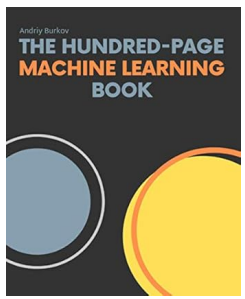
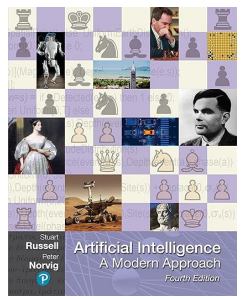
Lesson 02.5: ML Techniques - Model Evaluation

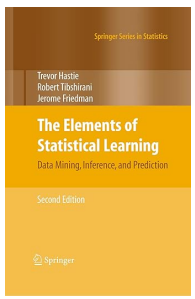
Instructor: Dr. GP Saggese - gsaggese@umd.edu

References:

- Burkov: *"The Hundred-Page Machine Learning Book"* (2019)
- Russell et al.: *"Artificial Intelligence: A Modern Approach"* (4th ed, 2020)
- Hastie et al.: *"The Elements of Statistical Learning"* (2nd ed, 2009)







- ***Performance Metrics***
- Model Selection
- Aggregation

How to Make Progress in ML Research

- In ML there are **many possible directions for research**, e.g.,
  - Different data preprocessing methods
  - Different features
  - Different models
  - Different training algorithms
  - Different evaluation techniques
- **What to do?**
- **Right 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 in real-world

How to Measure Classifier's Performance?

- **Success/hit/win rate (or error/miss rate)**
  - Measures correct predictions proportion
  - Important for overall accuracy
  - E.g., 80 correct out of 100 = 80% success rate
- **Log probability/cross-entropy error**
  - Evaluates classification with probabilities 0 to 1
  - Lower cross-entropy loss = better performance
- **Precision/recall/F-score**
  - Useful for imbalanced data
  - Precision: correctly predicted positives / total predicted positives
    - E.g., 0.75 precision = 75% true positives
  - Recall: correctly predicted positives / actual positives
    - E.g., 0.60 recall = 60% actual positives identified
  - F-score: weighted harmonic mean of precision and recall
- **Utility function**
  - Customizes evaluation to prioritize errors and success
    - E.g., true/false positives/negatives
  - E.g., in medical diagnosis, prioritize minimizing false negatives

Training vs Test Set

- Performance on train set E_{in} is an **optimistic estimate** of E_{out}
  - A model can have:
    - 0% error rate on training data (e.g., memorizing responses for training set)
    - 50% error rate on test set (e.g., answering randomly)
- To **properly evaluate model performance**:
  - Use test set not involved in training
  - Training and test sets should be representative samples
  - E.g., credit risk problem
    - Cannot use data from a Florida bank branch to assess a model built with New York data
    - Populations have different characteristics

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
    - $\rightarrow$ Fit all degrees of freedom of a complex model
  - Predict on lots of data
    - $\rightarrow$ Assess precise out-of-sample performance
- **Scarce data scenario**
  - Often data is scarce
    - E.g., facial recognition datasets with limited annotated data
  - Cannot use all data as a training set
  - Hold out data to estimate performance metrics and bounds
    - Split data 70-30 or 80-20 in train and test sets
    - Use cross-validation to maximize data usage
  - Other approaches:
    - Augment data artificially, like in image processing
    - Use 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 represented according to original data
  - Sized based on data and problem needs
- Ensure sets have the **same distribution**:
  - Stratified sampling
    - E.g., each class label proportionally represented in each set
  - Shuffle and sample
    - Achieves randomization, maintains distribution
  - Sample and check statistics of variables
    - E.g., mean, std dev, PDF
    - Compare statistics to ensure each set mirrors 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 even suitable for this problem?

Can You Use Test Set for Training?

- Once the model is selected and validated, reuse all data (including the test set) for deployment
  - Ensures the model benefits from all information
- More data is generally better, though returns diminish after a certain volume
  - Initially, increasing data size significantly improves performance
  - Eventually, adding more data results in smaller accuracy gains and may not justify increased computational cost
  - Use learning curves

In-Sample vs Out-Of-Sample Error Expressions

- **Goal of ML**: 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")
- Usually the **error is 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** 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** 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 squared difference of error**:

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

  - Measures estimator quality, quantifying difference between estimated and actual values
  - E.g., in house price prediction, determines how close predicted prices are to actual prices
- **Pros:**
  - Related to Gaussian error
  - Optimization friendly
- **Cons:**
  - Doesn't share unit of measure with output
    - Distorts error interpretation
  - Sensitive to outliers
    - Single large error can disproportionately affect 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 Mean Squared Error (MSE)**:

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

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

Median-Based Metrics

- Use metrics based on median (i.e., the 0.5 quantile of absolute error):
- **Median absolute deviation**:

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

- **Median squared error**:

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

- **Pros:**
  - Robust to outliers
- **Cons:**
  - Difficult to evaluate

How to Choose an Error Measure?

- Error measure depends on **application** and **"customer"** should define acceptable error level for their use case, e.g.,
  - Medical applications have low error tolerance
  - Recommendation systems have higher tolerance
- Otherwise, pick error measure that is
  - **Plausible**:
    - E.g., squared error for Gaussian noise
  - **Friendly**:
    - E.g., measures allowing closed-form solutions simplify calculations
    - Convex optimization-friendly measures ensure algorithms find global minimum easily

Error Measures: Fingerprint Verification Example

- In **fingerprint verification**:
  - Recognizing a valid fingerprint $\rightarrow$ no error
  - Otherwise $\rightarrow$ false positive or false negative
- **Error weight depends on the application**

Application	False Positive	False Negative
Supermarket	✅ Minor issue One extra discount	⚠️ Costly Annoyed customer, delays
CIA Building	💀 Critical Security breach	✅ Acceptable Triggers further checks

- For the same problem in two set-ups, the error measure is the opposite!

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)
    - 1% error rate 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

Confusion Matrix

- Assume
  - Actual/predicted class $\in \{0, 1\}$
  - Typically $y = 1$ encodes the rare class to predict
- You have 4 possible cases:
  - $act = 1$, $pred = 1 \rightarrow$ true positive (TP)
  - $act = 0$, $pred = 0 \rightarrow$ true negative (TN)
  - $act = 1$, $pred = 0 \rightarrow$ false negative (FN)
    - Model outputs $pred = 0$, but it is wrong
  - $act = 0$, $pred = 1 \rightarrow$ false positive (FP)
    - Model outputs $pred = 1$, but it is wrong

	pred = 1	pred = 0
act = 1	**TP**	**FN**
act = 0	**FP**	**TN**

- You can aggregate confusion matrix in **precision** and **recall**

Precision vs Recall: Definition

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

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

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

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

- 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 $\rightarrow$ precision $= \frac{20}{30} = \frac{2}{3}$
    - Fails to return other 40 relevant pages $\rightarrow$ recall $= \frac{20}{(40+20)} = \frac{20}{60} = \frac{1}{3}$

Precision / Recall as Quality / Quantity

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

Precision / Recall for Trivial Classifiers

- A **classifier that outputs always most common class** has:

$$\text{precision} \triangleq \Pr(\text{TP}|\text{pred} == 1) = 0 (\text{since TP} = 0)$$
$$\text{recall} \triangleq \Pr(\text{TP}|\text{act} == 1) = 0 \ (\text{since TP} = 0)$$

- A **classifier that outputs always rare class** has:

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

- A trivial classifier has precision or recall close to 0

Trading Off Precision and Recall

- In theory, you want to increase **both precision and recall**
- In practice, you can modify the threshold of a probabilistic classifier to **trade off precision and recall**
- E.g., use logistic regression to predict cancer:
  - With a threshold $\theta = 0.5$, the classifier has:
    - Precision $= \frac{\text{TP}}{|\text{pred} == 1|}$
    - Recall $= \frac{\text{TP}}{|\text{act} == 1|}$
  - Increase threshold θ
    - Output 1 only if more confident
    - I.e., increase precision
  - Decrease threshold θ
    - Output 1 more often
    - Decrease 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:**
  - You have two different numbers, thus it is difficult to compare classifiers to each other
  - Solutions: F-score, AUC

Precision-Recall Curves

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

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 class differentiation
  - AUC $= 0.5$ suggests no discriminative power, like random guessing
  - AUC closer to 1.0 indicates high performance
- **Pros:**
  - Single number summarizing classifier behavior, useful for comparing models
  - Does not require selecting a threshold for performance calculation
  - Handles imbalanced datasets effectively
- E.g., consider a classifier for medical diagnosis
  - AUC helps understand model's ability to distinguish between patients with and without a disease

F-Score

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

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

- **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 $P = 0$
    - $\frac{P+R}{2} = \frac{1}{2}$
    - We prefer a low value (ideally 0)

- Performance Metrics
- ***Model Selection***
- Aggregation

The Problem of Model Selection

- Model selection:
  - Chooses the best model from candidates based on performance
  - Is needed when multiple hypotheses explain data
- Across models certain parameters are fixed
- Others need selection, e.g.,
  - Set of features
    - E.g., select a subset from 100 variables
  - Learning algorithms
    - E.g., decide how to train a neural network
  - Model types
    - E.g., linear regression vs. SVM
  - Model complexity
    - E.g., polynomials of degree $d < 10$
  - Values of regularization parameter
    - E.g., try values like 0.01, 0.1, 1.0
- Evaluate metrics (e.g., model accuracy, precision, recall) with cross-validation

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, you need to **assess the true performance** on D_{test}
  - 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)

- Performance Metrics
- Model Selection
- ***Aggregation***

Ensemble Learning: Intuition

- **Idea**
  - A group of weak learners can form a strong learner
  - Combine multiple models to improve prediction accuracy
- **Combine outputs of models** X_i to build a model X^* better than any X_i
  - Utilize 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 (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 by itself, but together they become reliable

Ensemble Learning: Different Techniques

- **Bagging** (bootstrap + aggregation)
  - Reduces variance by averaging predictions from different models
  - E.g., decision trees + bagging = random forest
    - Creates multiple versions of a decision tree
    - Each tree is trained on a random sample of data
    - Averages 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 logistic regression as a meta-model
    - Combines predictions of other models (e.g., decision trees, support vector machines, 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 the residuals, i.e., difference between predicted and true values
  - Related to "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., you 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
- **Learn a meta-learner** (i.e., 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 highly 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
- **Why boosting works?**
  - 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 is about learning 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