Generalization

- Training data: \( (x_i, y_i) \)
  - examples that we used to train our predictor
  - e.g., all emails that our users labelled ham/spam
- Future data: \( (x, y) \)
  - examples that our classifier has never seen before
  - e.g., emails that will arrive tomorrow
- Want to do well on future data, not training
  - not very useful: we already know \( y_i \)
  - easy to be perfect on training data (DT, kNN, kernels)
  - does not mean you will do well on future data
  - can over-fit to idiosyncrasies of our training data

Under- and Over-fitting

- Over-fitting:
  - predictor too complex (flexible)
    - too “noisy” in the training data
    - patterns that will not re-appear
  - predictor \( F \) over-fits the data if:
    - we can find another predictor \( F’ \)
    - which makes more mistakes on training data: \( E_{test}(F) > E_{test}(F’) \)
    - but fewer mistakes on unseen future data: \( E_{test}(F’) < E_{test}(F) \)
- Under-fitting:
  - predictor too simplistic (too rigid)
  - not powerful enough to capture salient patterns in data
  - can find another predictor \( F’ \) with smaller \( E_{train} \) and \( E_{test} \)

Under- and Over-fitting examples

Regression:
- predictor too inflexible: cannot capture pattern
- predictor too flexible: fits noise in the data

Classification:
- predictor too inflexible: cannot capture pattern
- predictor too flexible: fits noise in the data

Flexible vs. inflexible predictors

- Each dataset needs different level of “flexibility”
  - depends on task complexity + available data
  - want a “knob” to get rigid / flexible predictors
- Most learning algorithms have such knobs:
  - regression: order of the polynomial
  - NB: number of attributes, limits on \( c, \varepsilon \)
  - DT: \#nodes in the tree / pruning confidence
  - kNN: number of nearest neighbors
  - SVM: kernel type, cost parameter
  - Tune to minimize generalization error

Training vs. Generalization Error

- Training error:
  \[
  E_{train} = \frac{1}{n} \sum_{i=1}^{n} \text{error}(\hat{f}(x_i), y_i)
  \]
- Generalization error:
  \[
  E_{test} = \frac{1}{n} \sum_{i=1}^{n} \text{error}(f(x_i), y_i)
  \]
- \( E_{test} \) is an unbiased estimate of \( E_{true} \)
- Usually
  \[
  E_{train} \leq E_{test} \leq E_{true}
  \]

Confidence Interval for Future Error

- What range of errors can we expect for future test sets?
  \[
  E_{true} \pm 0.95 \text{ such that } 95\% 	ext{ of future test sets fall within that interval}
  \]
- \( E_{true} \) has an unbiased estimate of \( E_{true} \)
- \( E \) probability our system will misclassify a random instance
- Take a random set of \( n \) instances, how many misclassified?
  - 80% correct coins in \( n \) times, how many heads will we get?
  - Binomial distribution with \( n \), \( \varepsilon \), variance \( n \cdot E[(1-E)] \)
- \( E_{true} \) distributed as \( n \cdot \text{Gaussian}, \text{mean } E, \text{variance } n \cdot E[(1-E)] \)
- \( E_{true} \) has \( n \) confidence interval for future error:
  \[
  \pm 0.95 \text{ confidence interval for future error:}
  \]
  - \( n = 100 \) examples, \( p = 0.05 \) and \( E = 0.25 \)
  - \( E = 0.25 \pm 0.05 \)
  - \( n = 1000 \), \( p = 0.05 \) to \( E = 0.25 \pm 0.01 \)
  - \( n = 10000 \), \( p = 0.05 \) to \( E = 0.25 \pm 0.005 \)

Estimating Generalization Error

- Testing error:
  \[
  E_{test} = \frac{1}{n} \sum_{i=1}^{n} \text{error}(f(x_i), y_i)
  \]
- Set aside part of training data (testing set)
- Learn a predictor without using any of this data
- Predict values for testing set
- Gives an estimate of true generalization error
- If testing set is unbiased sample from \( p(x, y) \):
  \[
  \lim_{n \to \infty} E_{test} = E_{true}
  \]
- How close? depends on \( n \)
- Ex: binary classification, 100 instances
  - Assume: 75 classified correctly, 25 incorrectly
  - \( E_{test} = 0.25 \), \( E_{true} \) around 0.25, but how close?

Training, Validation, Testing sets

- Training set: construct classifier
  - NB: count frequencies, DT: pick attributes to split on
- Validation set: pick algorithm + knob settings
  - Pick best-performing algorithm (NB vs. DT vs. …)
  - Fine-tune knobs (tree depth, k in kNN, \( C \) in SVM…)
- Testing set: estimate future error rate
  - Never report best of many runs
  - Run only once, or report results of every run
  - Split randomly to avoid bias
Cross-validation

- Conflicting priorities when splitting the dataset
  - estimate future error as accurately as possible
    - large testing set; big $\frac{n_{test}}{n_{train}}$, tight confidence interval
  - learn classifier as accurately as possible
    - large training set; big $\frac{n_{train}}{n_{test}}$ better estimates
  - training and testing cannot overlap; $\frac{n_{train}}{n_{test}} = \frac{n_{test}}{n_{train}}$ in cross

- Idea: evaluate Train $\rightarrow$ Test, then Test $\rightarrow$ Train, average results
  - every point is both training and testing, never at the same time
  - reduces chances of getting an unusual (biased) testing set
  - 5-fold cross-validation
    - randomly split the data into 5 sets
    - test on each in turn (leave out 4 others)
    - average the results over 5 folds
  - more common: 10-fold

Leave-one-out

- $n$-fold cross-validation ($n = \text{total number of instances}$)
  - predict each instance, training on all ($n-1$) other instances
- Pros and cons:
  - best possible classifier learned: $n-1$ training examples
  - high computational cost: re-learn everything $n$ times
- not an issue for instance-based methods like KNN
- there are tricks to make such learning faster
- classes not balanced in training/testing sets
- random data, 2 equal-probable classes $\rightarrow$ wrong 100% of the time
- testing balance (1 of A, 0 of B) vs. training (1/2 of B, 0/1 of A)
- duplicated data: nothing can beat 1NN (0% error)
- wouldn’t happen with 10-fold cross-validation

Stratification

- Problems with leave-one-out:
  - training/testing sets have classes in different proportions
  - not limited to leave-one-out
  - $K$-fold cross-validation: random splits $\rightarrow$ imbalances
- Stratification
  - keep class labels balanced across training/testing sets
  - simple way to guard against unlucky splits
  - recipe:
    - randomly split each class into $K$ parts
    - assemble $P$ part from all classes to make the $P$ fold

Evaluation measures

- Are we doing well? Is system A better than B?
- A measure of how (in)accurate a system is on a task
- in many cases Error (Accuracy / PC) is not the best measure
- using the appropriate measure will help select best algorithm
- Classification
  - how often we classify something right/wrong
- Regression
  - how close are we to what we’re trying to predict
- Unsupervised
  - how well do we describe our data
  - in general: really hard

Classification measures: basics

- $\text{Classification error} = \frac{\text{FP + FN}}{\text{FP + TP + TN + FN}}$
- $\text{Accuracy} = (1 - \text{error})$
- Basic measure of “goodness” of a classifier
- Problem: cannot handle unbalanced classes
  - ex1: predict whether an earthquake is about to happen
    - happen very rarely, very good accuracy if always predict ‘no’
    - solution: make this much more “costly” than FPs
  - ex2: web search: decide if a webpage is relevant to user
    - 90% of pages not relevant to any query $\rightarrow$ retrieve nothing
    - solution: use measures that don’t involve TN (recall / precision)

Accuracy and un-balanced classes

- You’re predicting Nobel prize (+) vs. not (-)
- Human would prefer classifier A.
- Accuracy will prefer classifier B (fewer errors)
- Accuracy poor metric here

Misses and False Alarms

- False Alarm rate = False Positive rate = $\text{FP}/(\text{FP} + \text{TN})$
  - $\%$ of negatives we misclassified as positive
- Miss rate = False Negative rate = $\text{FN}/(\text{FP} + \text{FN})$
  - $\%$ of positives we misclassified as negative
- Recall = True Positive rate = $\text{TP}/(\text{TP} + \text{FN})$
  - $\%$ of positives we classified correctly ($1 - \text{Miss rate}$)
- Precision = $\text{TP}/(\text{TP} + \text{FP})$
  - $\%$ positive out of what we predicted was positive
- Meaningless to report just one of these
  - linked to get 100% recall or 0% false alarm
  - typical recall/precision or Miss / FA rate or TP/FP rate

Evaluation (recap)

- Predicting class C (e.g. spam)
  - classifier can make two types of mistakes:
    - $\text{FP}$: false positives – non-spam emails mistakenly classified as spam
    - $\text{FN}$: false negatives – spam emails mistakenly classified as non-spam
    - $\text{TP}$: true positives/negatives $\rightarrow$ correctly classified spam/spam
- $\text{common error/accuracy measures:}$
  - $\text{Classification Error} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}$
  - $\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}$
  - $\text{False Alarm} = \frac{\text{FP}}{\text{FP} + \text{FN}}$
  - $\text{Miss} = \frac{\text{FN}}{\text{FP} + \text{FN}}$
  - $\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
  - $\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$

always report in pairs, e.g.
- Miss / FA in Recall / Prec.
Utility and Cost

- Sometimes need a single, single evaluation measure
  - optimizing the learner (automatically), competitive evaluation
  - may know costs of different errors, e.g. earthquakes:
    - false positive: cost of preventive measures (evacuation, lost profit)
    - false negative: cost of recovery (inconvenience, liability)
- Detection cost: weighted average of FP, FN rates
  - Cost = Cost(FP) + Cost(FN)
- F-measure: harmonic mean of recall, precision
  - \( F = 2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \)
- Domain-specific measures:
  - e.g. observed profit/loss from +1/-1 market prediction

Thresholds in Classification

- Two systems have the following performance:
  - A: True Positive = 50%, False Positive = 20%
  - B: True Positive = 100%, False Positive = 60%
- Which is better? (assume no-apriori utility)
  - very misleading question
  - A and B could be the same exact system
  - operating at different thresholds

ROC curves

- Many algorithms compute “confidence” \( f(x) \)
  - threshold to get decision: span if \( f(x) > 1 \), non-span if \( f(x) < 1 \)
    - Naive Bayes: \( P(y|\text{spam}) > 0.5 \),\( \text{minimize} \log(1-P) + \text{error rate} \)
    - Decision Trees: \( f(x) > 1 \)
  - threshold determines error rates
    - False Positive rate \( = P(\text{spam} | \text{non-spam}) \)
    - True Positive rate \( = P(\text{spam} | \text{spam}) \)
- Receiver Operating Characteristic (ROC):
  - plot TPR vs. FPR as \( f(x) \) varies from \( 0 \) to \( \infty \)
  - shows performance of system across all possible thresholds
  - AUC = area under ROC curve
  - popular alternative to accuracy

Evaluating regression

- Classification:
  - count how often we are wrong
- Regression:
  - predict \( y \) from inputs \( x \)
    - always wrong, but by how much?
    - distance between predicted \& true values
      - (root) mean squared error:
        - popular, well-understood, analytically derivable
        - less sensitive to outliers
  - mean absolute error:
    - insensitive to mean \& scale
  - correlation coefficient:
    - insensitive to mean \& scale

Mean Squared Error

- Average (squared) deviation from truth
  - \( \frac{1}{n} \sum (f(x) - y)^2 \)
  - very sensitive to outliers
    - 99 exact, 1 off by \$10
    - all 100 wrong by \$1
  - Sensitive to mean / scale
    - \( \mu = \frac{\sum x}{n} \) good baseline
  - Relative squared error (Weka)
    - \( \frac{\sum (f(x) - y)^2}{\sum (y - \mu)^2} \)
    - USE when using a mean as a predictor

Summary

- Training vs. generalization error
  - under-fitting and over-fitting
- Estimate how well your system is doing its job
  - how does it compare to other approaches?
  - what will be the error rate on future data?
- Training and testing
  - cross-validation, leave-one-out, stratification, significance
- Evaluation measures
  - accuracy, miss / false alarm rates, detection cost
  - ROC curves
    - regression: (root) mean squared/absolute error, correlation
  - Mean Absolute Error (MAE):
    - less sensitive to outliers
    - many small errors + one large error
    - best 0th order baseline: median \( |y| \)
      - not the mean as for MSE
  - Median Absolute Deviation (MAD): \( \text{med}(\{|f(x) - y|\}) \)
    - robust, completely ignores outliers
    - can define similar squared error: \( \text{med}((f(x) - y)^2) \)
    - difficult to work with (can’t take derivatives)
  - Sensitive to mean, scale

Correlation Coefficient

- Completely insensitive to mean / scale:
  - \( \frac{\sum (f(x) - \mu)(y - \mu)}{\sqrt{\sum (f(x) - \mu)^2 \sum (y - \mu)^2}} \)
  - Intuition: did you capture the relative ordering?
    - output larger \( f(x) \) for larger \( y \)
    - output smaller \( f(x) \) for smaller \( y \)
  - useful for ranking tasks:
    - e.g. recommend a movie to a user
  - Important to visualize data:
    - same for 4 predictors
Evaluating unsupervised methods

- Generally hard and subjective
  - broad aim: did we capture the structure of the dataset?
  - if possible: does it help us do some (supervised) task

- Dimensionality reduction
  - distance between data in original & reduced space

- Mixture models
  - do we assign high probability to the training data?

- Clustering
  - did we “discover” the latent sub-populations?

Significance tests

- Often need to compare two systems: A, B
  - perform cross-validation: errors $e_{a1} ... e_{aK}, e_{b1} ... e_{bK}$
  - average errors: $e_a < e_b$
  - does this mean that A better than B?
    - look at the variance of errors

- Significance: could the difference be due to chance?
  - analogy: 3 coin flips, always large difference, pure chance
  - null hypothesis $H_0$
    - $e_{a1} ... e_{aK}, e_{b1} ... e_{bK}$ are random samples from the same population
    - want to show $P(H_0)$ is very small → reject $H_0$ as improbable
  - let $d_i = e_{ai} - e_{bi}$ → $\frac{\sum d_i}{\sigma}$ ~ Student’s t distribution
  - caution: $d_i$ must be independent (no overlap in data)