Introductory Applied Machine Learning

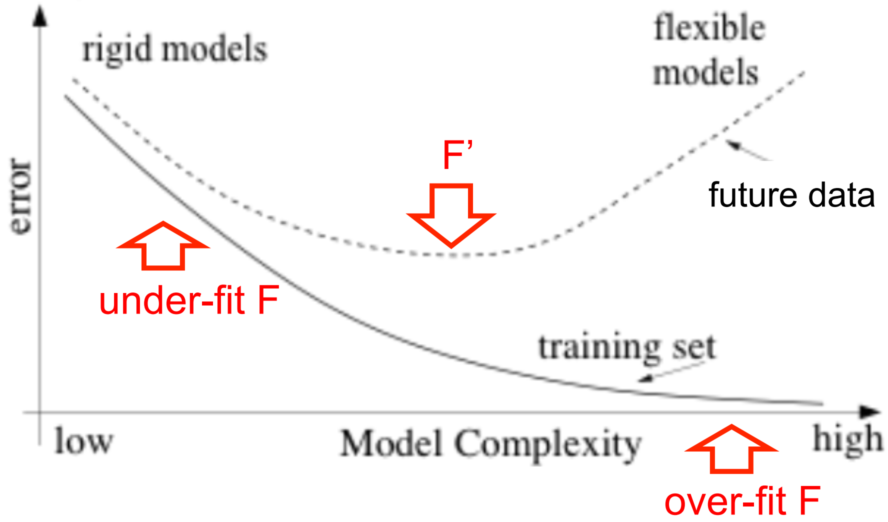
Generalization, Overfitting, Evaluation

Victor Lavrenko and Nigel Goddard
School of Informatics

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: $\{x_i, ?\}$
  - 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

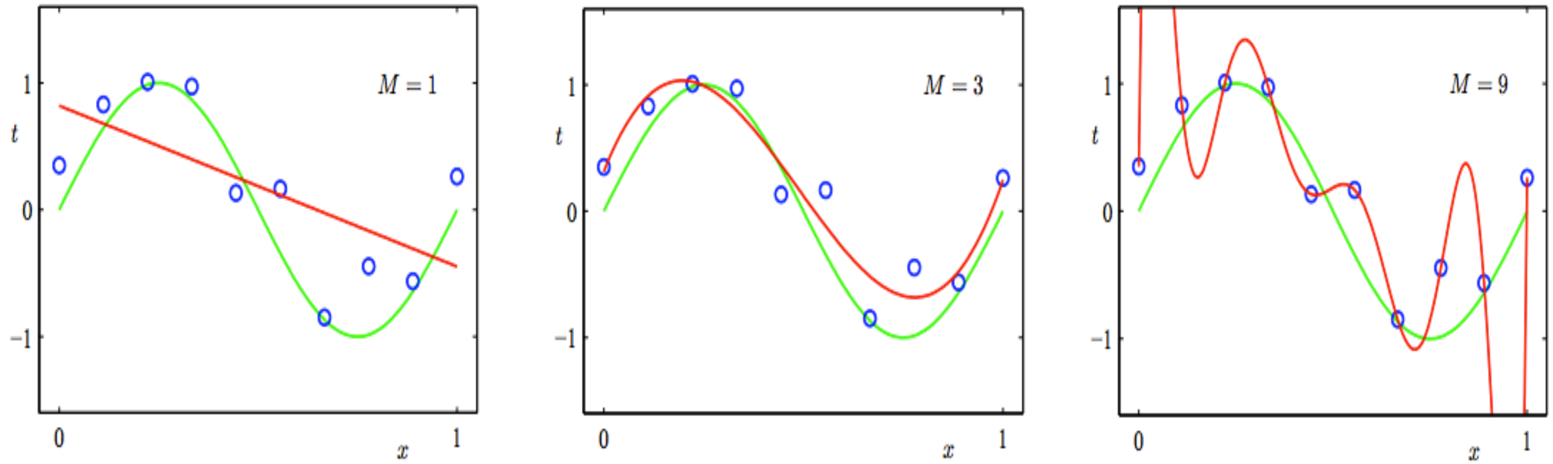
Under- and Over-fitting



- Over-fitting:
  - predictor too complex (flexible)
    - fits "noise" 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: $E_{train}(F') > E_{train}(F)$
    - but fewer mistakes on unseen future data : $E_{gen}(F') < E_{gen}(F)$
- 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_{gen}

Under- and Over-fitting examples

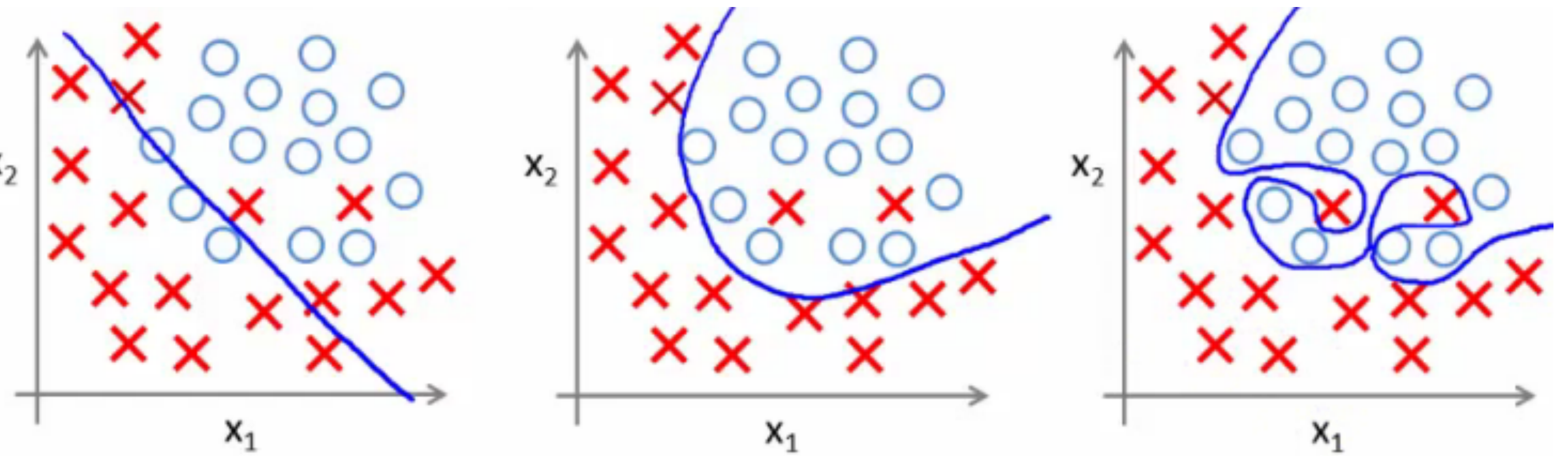
Regression:



predictor too inflexible: cannot capture pattern

predictor too flexible: fits noise in the data

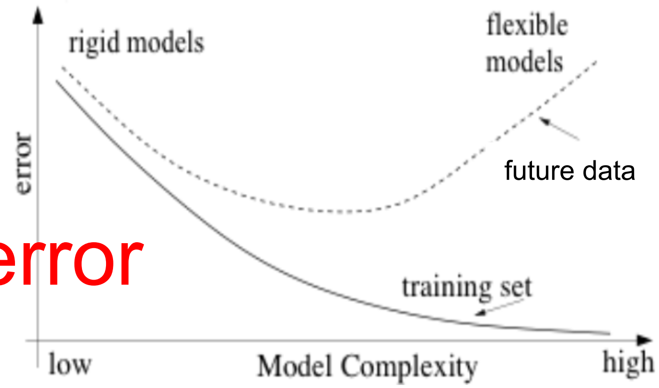
Classification:



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 σ^2, ε
  - 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:

$$E_{train} = \frac{1}{n} \underbrace{\sum_{i=1}^{n}}_{\text{training examples}} \overbrace{error}^{\text{same? different by how much?}}(\underbrace{f_D(\mathbf{x}_i)}_{\text{value we predicted}}, \underbrace{y_i}_{\text{true value}})$$

- Generalization error:
  - how well we will do on future data
  - don't know what future data x_i will be
  - don't know what labels y_i it will have
  - but know the "range" of all possible $\{x,y\}$
    - x: all possible 20x20 black/white bitmaps
    - y: {0,1,…,9} (digits)

Usually $E_{train} \leq E_{gen}$

$$E_{gen} = \underbrace{\int}_{\text{over all possible x,y}} \underbrace{error(f_D(\mathbf{x}), y)}_{\text{error as before}} \underbrace{p(y, \mathbf{x})}_{\text{how often we expect to see such x and y}} d\mathbf{x}$$

Can never compute generalisation error

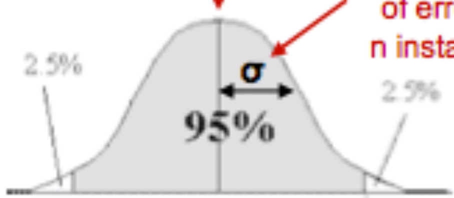
Estimating Generalization Error

- Testing error: $$E_{test} = \frac{1}{n}\sum_{i=1}^{n} error(f_D(\mathbf{x}_i), y_i)$$ over testing set
  - set aside part of training data (testing set)
  - learn a predictor without using any of this data
  - predict values for testing set, compute error
  - gives an estimate of true generalization error
    - if testing set is unbiased sample from $p(x,y)$: $\lim_{n \to \infty} E_{test} = E_{gen}$
    - how close? depends on n
- Ex: binary classification, 100 instances
  - assume: 75 classified correctly, 25 incorrectly
  - $E_{test} = 0.25$, E_{gen} around 0.25, but how close?

Confidence Interval for Future Error

- What range of errors can we expect for future test sets?
  - $E_{test} \pm \Delta E$ such that 95% of future test sets fall within that interval
- E_{test} is an unbiased estimate of E = **true error rate**
  - E = probability our system will misclassify a random instance
  - take a random set of n instances, how many misclassified? ← our test set is one such set
    - flip E-biased coin n times, how many heads will we get?
    - Binomial distribution with mean = $n\,E$, variance = $n\,E\,(1-E)$
    - E_{future}= #misclassified / n, ~ Gaussian, mean E, variance = $E\,(1-E)\,/\,n$
      - 2/3 future test sets will have error in E ± $\sqrt{(E(1-E)/n)}$
  - p% confidence interval for future error:
    - for n=100 examples, p=0.95 and E = 0.25
      - $\sigma = \sqrt{(0.25 \cdot 0.75/100)} = .043$
      - CI = 0.25 ± 1.96•σ = **0.25 ± 0.08**
    - n=100, p=0.99 → CI = **0.25 ± 0.11**
    - n=10000, p=0.95 → CI = **0.25 ± 0.008**

$$CI = \underbrace{E}_{\text{expected error}} \pm \underbrace{\sqrt{E(1-E)/n}}_{\sigma\text{: standard deviation of error for n instances}} \cdot \underbrace{\Phi^{-1}\left(\frac{1-p}{2}\right)}_{\text{how many deviations around mean need to get area of p\% } [\sqrt{2}\,\mathrm{erf}^{-1}(-p)]}$$

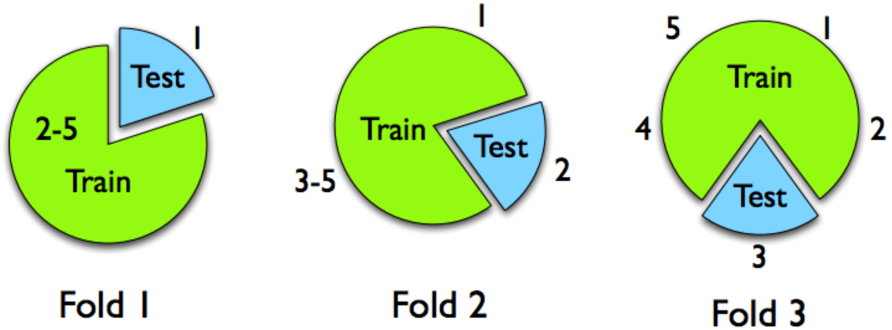


Training, Validation, Testing sets

- 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 n_{test} → tight confidence interval
  - learn classifier as accurately as possible
    - large training set: big n_{train} → better estimates
  - training and testing cannot overlap: $n_{train} + n_{test}$ = const
- Idea: evaluate Train → Test, then Test → 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 (train on 4 others)
    - average the results over 5 folds
  - more common: 10-fold



Leave-one-out

- n-fold cross-validation (n = 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 equi-probable classes → wrong 100% of the time
      - testing balance: {1 of A, 0 of B} vs. training: {n/2 of B, n/2-1 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 → imbalance
- 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 i^{th} part from all classes to make the i^{th} 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

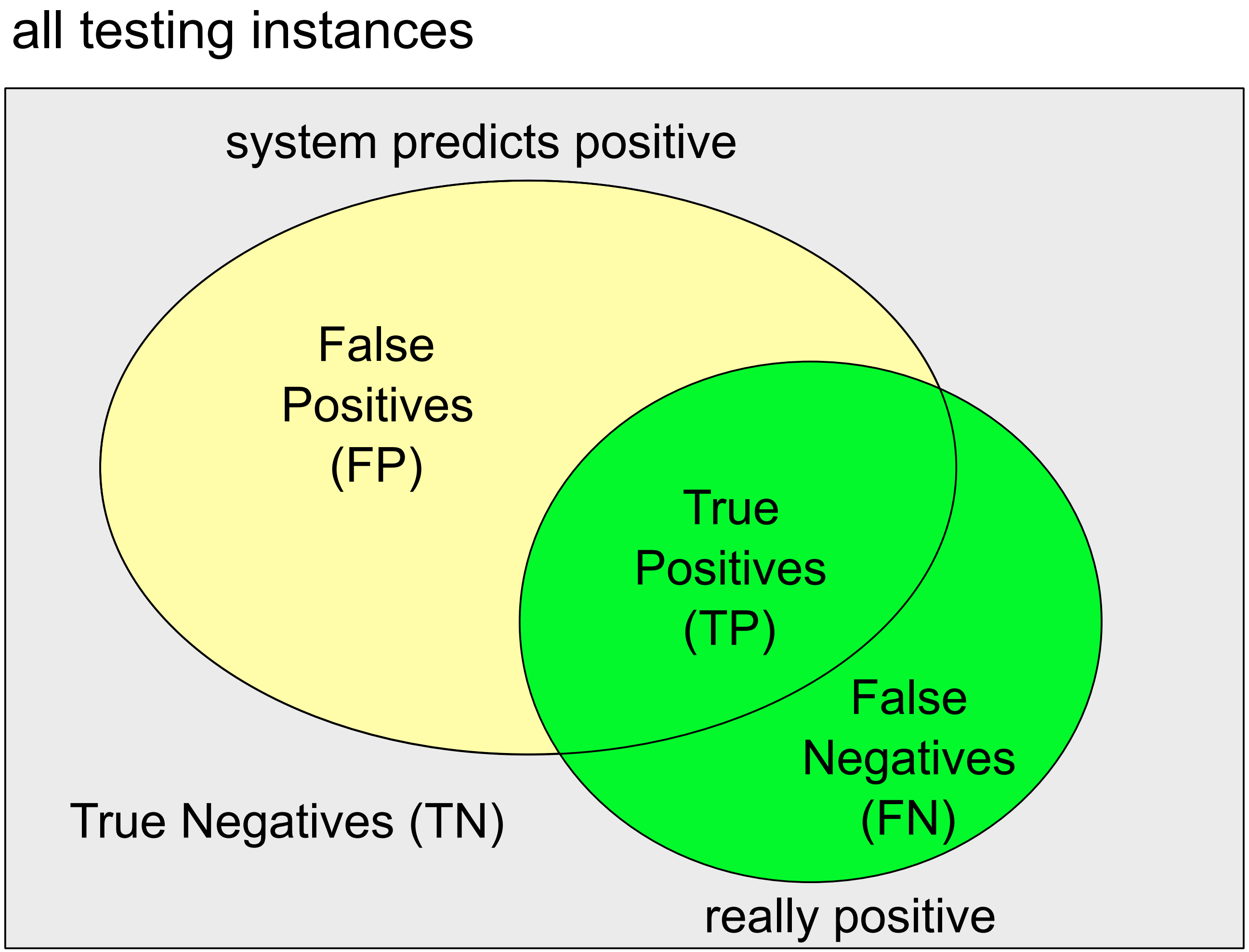
Predict positive?

Really positive?

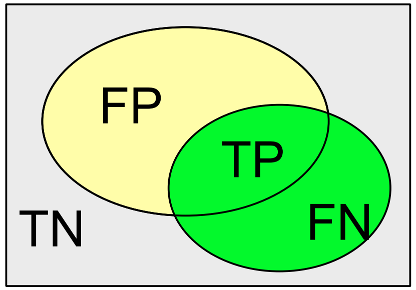
	Yes	No
Yes	TP	FN
No	FP	TN

Confusion matrix for two-class classification

Want: large diagonal, small FP, FN



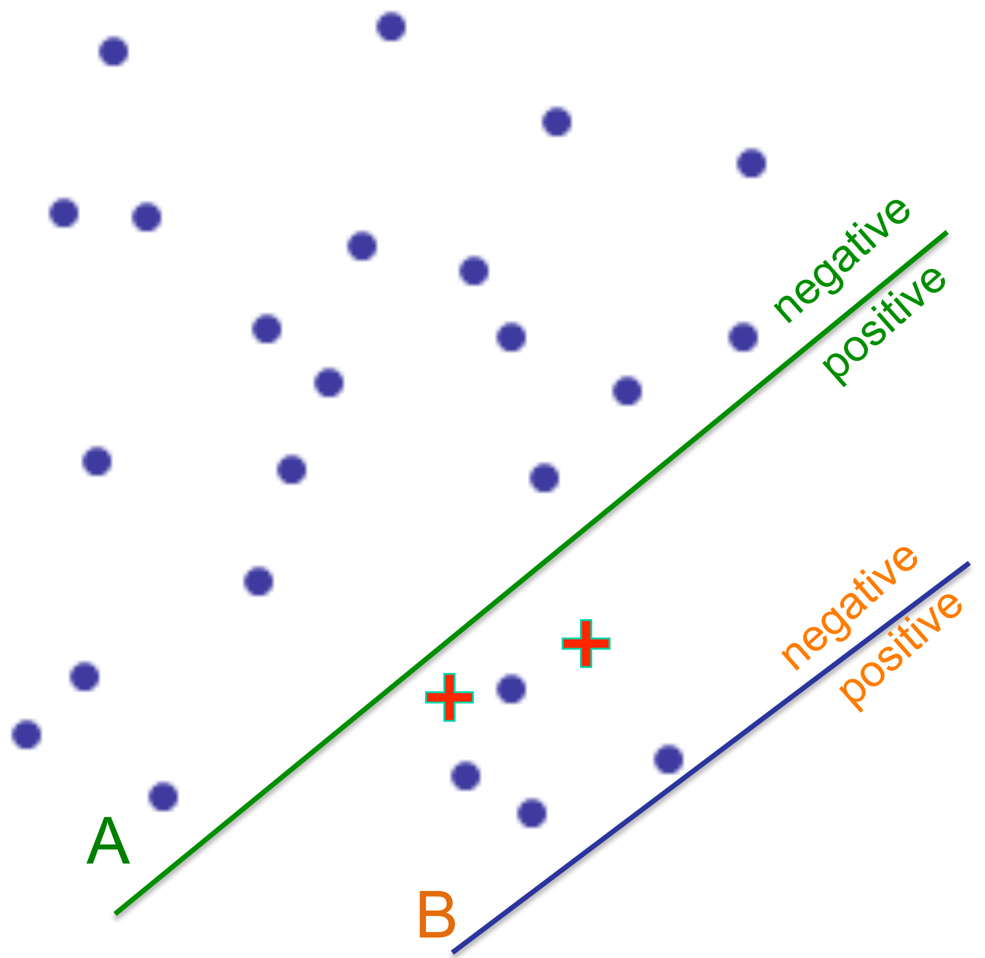
Classification Error



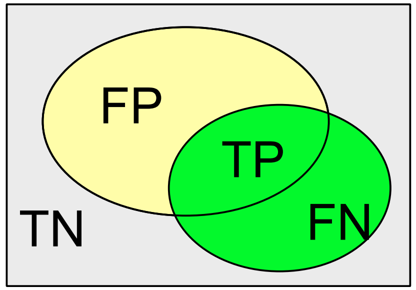
- Classification error = $\frac{errors}{total} = \frac{FP + FN}{TP + TN + FP + FN}$
- Accuracy = (1 – error) = $\frac{correct}{total} = \frac{TP + TN}{TP + TN + FP + FN}$
- Basic measure of "goodness" of a classifier
- Problem: cannot handle unbalanced classes
  - ex1: predict whether an earthquake is about to happen
    - happen very rarely, very good accuracy if always predict "No"
    - solution: make FNs much more "costly" than FPs
  - ex2: web search: decide if a webpage is relevant to user
    - 99.9999% of pages not relevant to any query → 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 = FP / (FP+TN)
  - % of negatives we misclassified as positive
- Miss rate = False Negative rate = FN / (TP+FN)
  - % of positives we misclassified as negative
- Recall = True Positive rate = TP / (TP+FN)
  - % of positives we classified correctly (1 – Miss rate)
- Precision = TP / (TP + FP)
  - % positive out of what we predicted was positive
- Meaningless to report just one of these
  - trivial to get 100% recall or 0% false alarm
  - typical: recall/precision or Miss / FA rate or TP/FP rate

Evaluation (recap)

		Predicted C?	
		Yes	No
Really C?	Yes	TP	FN
	No	FP	TN

- Predicting class C (e.g. spam)
  - classifier can make two types of mistakes:
    - FP: false positives – non-spam emails mistakenly classified as spam
    - FN: false negatives – spam emails mistakenly classified as non-spam
    - TP/TN: true positives/negatives – correctly classified spam/non-spam
  - common error/accuracy measures:
    - Classification Error: $\frac{errors}{total} = \frac{FP+FN}{TP+TN+FP+FN}$
    - Accuracy = 1-Error: $\frac{correct}{total} = \frac{TP+TN}{TP+TN+FP+FN}$

      meaningless if classes imbalanced
    - False Alarm = False Positive rate = FP / (FP+TN)
    - Miss = False Negative rate = FN / (TP+FN)
    - Recall = True Positive rate = TP / (TP+FN)
    - Precision = TP / (TP+FP)

      always report in pairs, e.g.: Miss / FA or Recall / Prec.

Utility and Cost

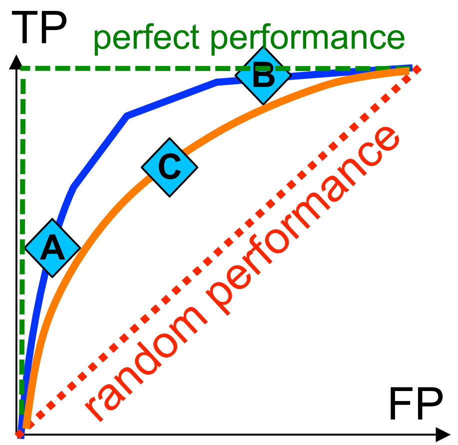
- Sometimes need a single-number evaluation measure
  - optimizing the learner (automatically), competitive evaluation
  - may know costs of different errors, e.g. earthquakes:
    - false positive: cost of preventive measures (evacuation, lost profit)
    - false negative: cost of recovery (reconstruction, liability)
- Detection cost: weighted average of FP, FN rates
  - Cost = C_{FP} * FP + C_{FN} * FN [event detection]
- F-measure: harmonic mean of recall, precision
  - F1 = 2 / (1 / Recall + 1 / Precision) [Information Retrieval]
- Domain-specifc measures:
  - e.g. observed profit/loss from +/- 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: spam if f(x) > t, non-spam if f(x) ≤ t
    - Naïve Bayes: P(spam|x) > 0.5, Linear/Logistic/SVM: $w^T x > 0$, Decision Tree: $p_+/p_- > 1$
  - threshold t determines error rates
    - False Positive rate = P(f(x)>t|ham), True Positive rate = P(f(x)>t|spam)
- Receiver Operating Characteristic (ROC):
  - plot TPR vs. FPR as t varies from ∞ to -∞

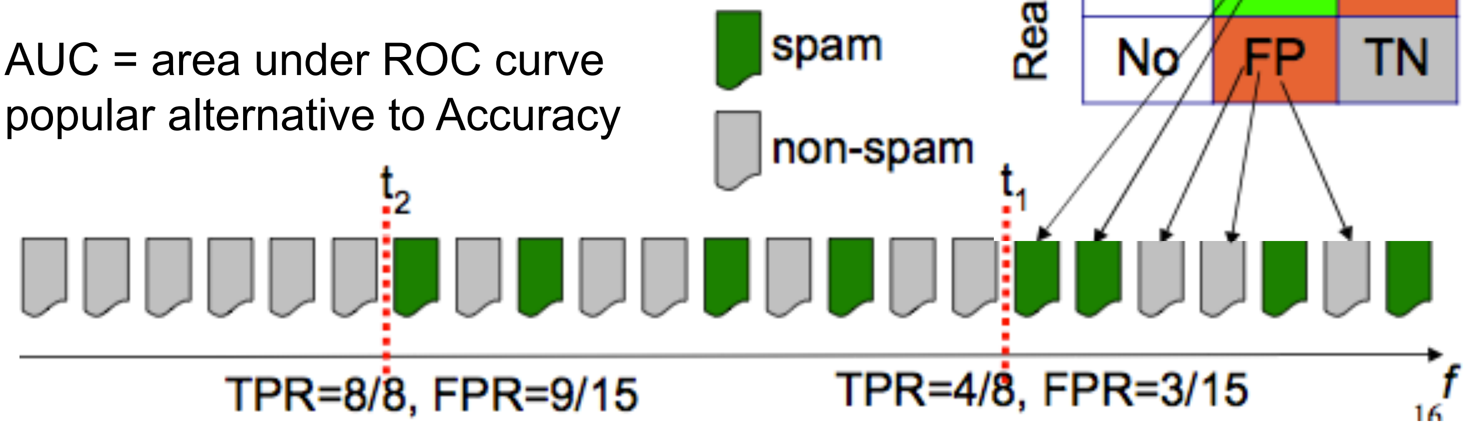


shows performance of system across all possible thresholds

AUC = area under ROC curve
popular alternative to Accuracy

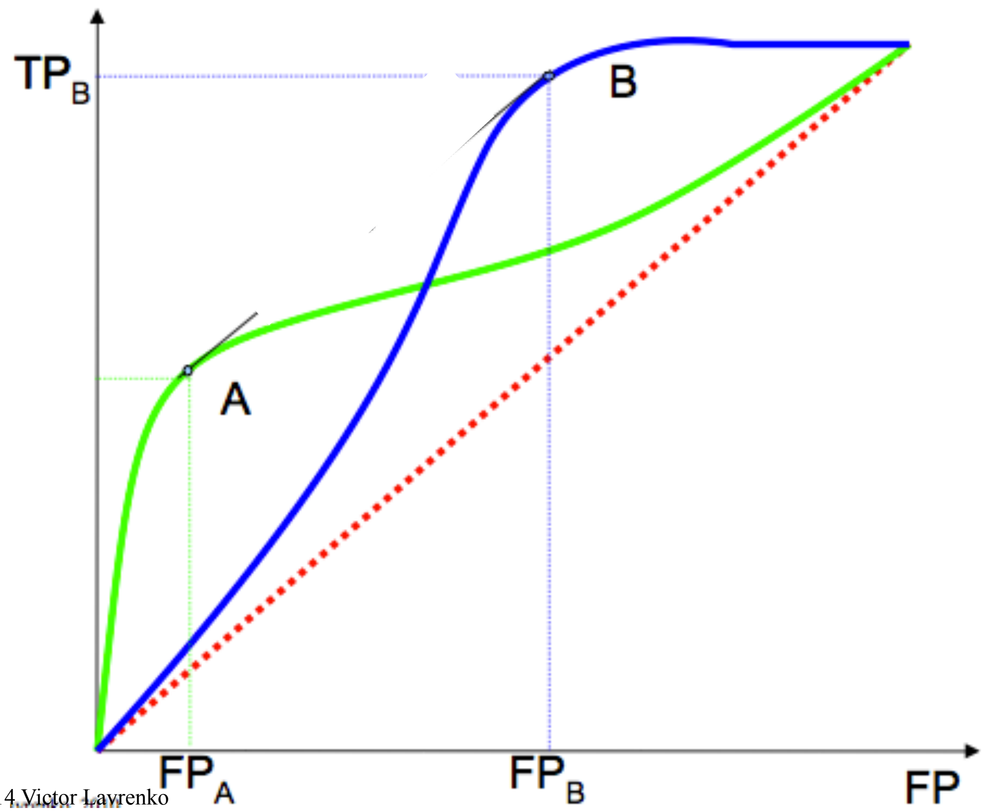
Predicted spam?

Really spam?	Yes	No
Yes	TP	FN
No	FP	TN

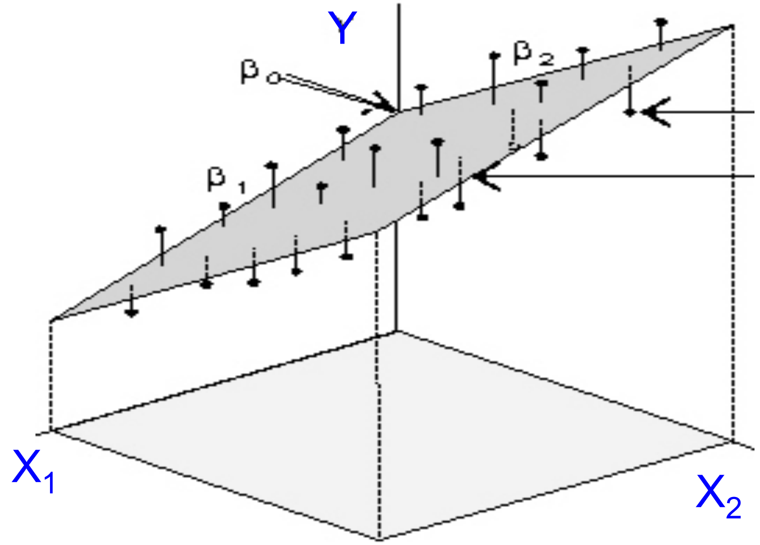


ROC convex hull

- System A: better at high thresholds (high-precision)
- System B: better at low thresholds (high-recall)
- System C: for each x: flip a *p*-coin, heads: A(x), tails: B(x)
  - if x was really positive:
    - P(correct) = p * P(A(x) > t_A | +) + (1-p) * P(B(x) > t_B | +)
    - $TP_C = p\ TP_A + (1-p)\ TP_B$
  - if x was really negative:
    - P(error) = p * P(A(x) > t_A | -) + (1-p) * P(B(x) > t_B | -)
    - $FP_C = p\ FP_A + (1-p)\ FP_B$
  - may be better than either A or B
  - example: Netflix challenge



Evaluating regression



- Classification:
  - count how often we are wrong
- Regression:
  - predict numbers y_i from inputs x_i
  - always wrong, but by how much?
  - distance between predicted & true values
    - (root) mean squared error:
      - popular, well-understood, nicely differentiable
      - sensitive to single large errors (outliers)
    - mean absolute error:
      - less sensitive to outliers
    - correlation coefficient
      - insensitive to mean & scale

$$\sqrt{\underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{testing set}}\Big(\underbrace{f(x_i)}_{\text{predicted}}-\underbrace{y_i}_{\text{true}}\Big)^2}$$

$$\frac{1}{n}\sum_{i=1}^{n}\left|f(x_i)-y_i\right|$$

$$\frac{n\sum_{i=1}^{n}\left(f(x_i)-\mu_f\right)\left(y_i-\mu_y\right)}{\sqrt{\sum_{i=1}^{n}\left(f(x_i)-\mu_f\right)^2\cdot\sum_{i=1}^{n}\left(y_i-\mu_y\right)^2}}$$

Mean Squared Error

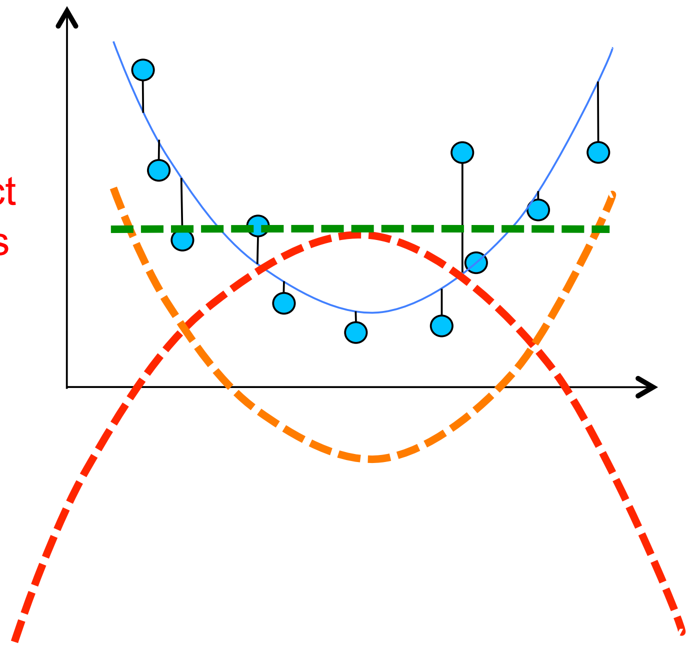
- Average (squared) deviation from truth $\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(f(x_i)-y_i\right)^2}$
- Very sensitive to outliers
  - 99 exact, 1 off by $10
  - all 100 wrong by $1

  same MSE → large effect on models
- Sensitive to mean / scale
  - $\mu_y = {}^1/_n \Sigma_i y_i$ ... good baseline
- Relative squared error (Weka)

$$\sqrt{\frac{\sum_{i=1}^{n}\left(f(x_i)-y_i\right)^2}{\sum_{i=1}^{n}\left(\mu_y-y_i\right)^2}}$$

MSE of predictor

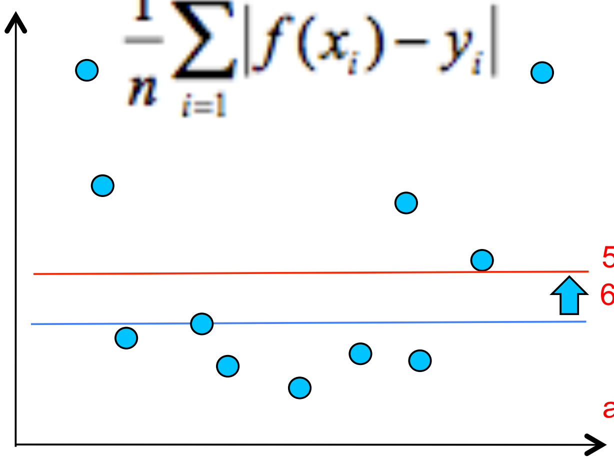
MSE when using the mean as a predictor



Mean Absolute Error

- Mean Absolute Error (MAE):

$$\frac{1}{n}\sum_{i=1}^{n}|f(x_i)-y_i|$$

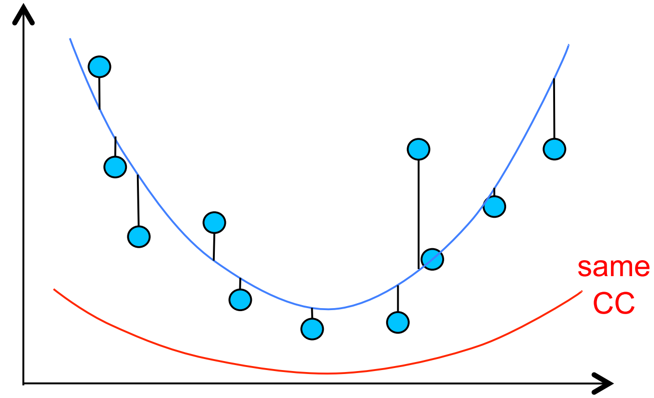


  - less sensitive to outliers
  - many small errors = one large error
  - best 0^{th} order baseline: median$\{y_i\}$
    - not the mean as for MSE
- Median Absolute Deviation (MAD): med$\{|f(x_i)-y_i|\}$
  - robust, completely ignores outliers
  - can define similar squared error: median$\{(f(x_i)-y_i)^2\}$
  - difficult to work with (can't take derivatives)
- Sensitive to mean, scale

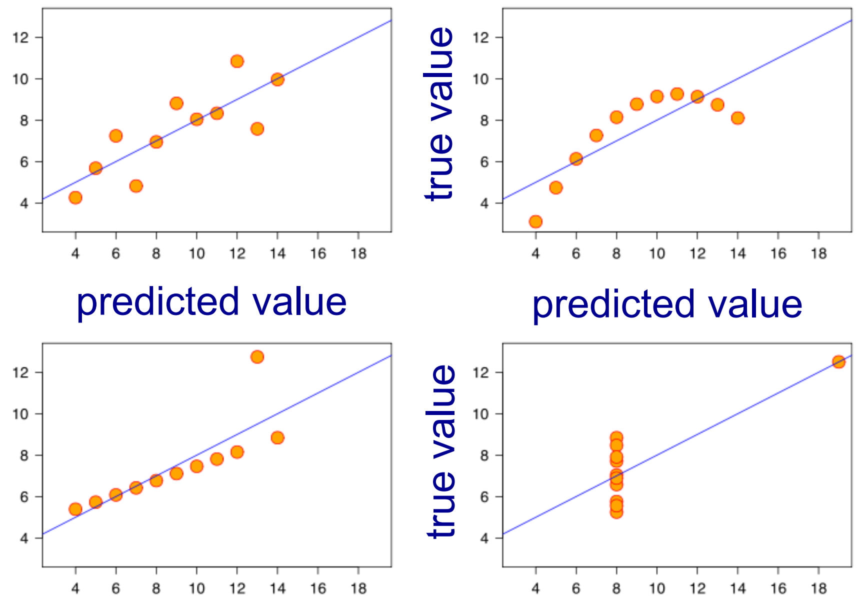
Correlation Coefficient

- Completely insensitive to mean / scale:

$$\frac{n\sum_{i=1}^{n}\left(f(x_i)-\mu_f\right)\left(y_i-\mu_y\right)}{\sqrt{\sum_{i=1}^{n}\left(f(x_i)-\mu_f\right)^2\cdot\sum_{i=1}^{n}\left(y_i-\mu_y\right)^2}} = n\sum_{i=1}^{n}\underbrace{\frac{f(x_i)-\mu_f}{\sigma_f}}_{\text{prediction relative to mean}}\cdot\underbrace{\frac{y_i-\mu_y}{\sigma_y}}_{\text{truth relative to mean}}$$



- Intuition: did you capture the relative ordering?
  - output larger $f(x_i)$ for larger y_i
  - output smaller $f(x_i)$ for smaller y_i
  - useful for ranking tasks:
    - e.g. recommend a movie to a user
- Important to visualize data
  - same CC for 4 predictors ➔



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

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 $e_{A,1} \dots e_{A,K}$, $e_{B,1} \dots e_{B,K}$
  - 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:
    - $e_{A,1} \dots e_{A,K}$, $e_{B,1} \dots e_{B,K}$ are random samples from the same population
    - want to show $P(H_0)$ is very small → reject H_0 as improbable
  - let $d_i = e_{A,i} - e_{B,i}$ $\quad t = \frac{\sum_i d_i}{\sqrt{\sum_i (d_i - \mu)^2}}$ ~ Student's t distribution
  - caution: d_i must be independent (no overlap in data)