Feature and model selection

Subhransu Maji

CMPSCI 689: Machine Learning

10 February 2015

12 February 2015

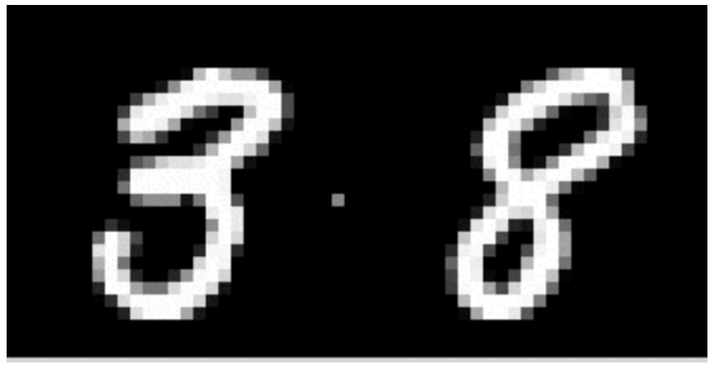
Administrivia

- Homework stuff
  - Homework 3 is out
  - Homework 2 has been graded
  - Ask your TA any questions related to grading
- TA office hours (currently Thursday 2:30-3:30)
  1. Wednesday 3:30 - 4:30?
- Later in the week
  - p1: decision trees and perceptrons
  - due on March 03
- Start thinking about projects
  - Form teams (2+)
  - A proposal describing your project will be due mid March (TBD)

The importance of good features

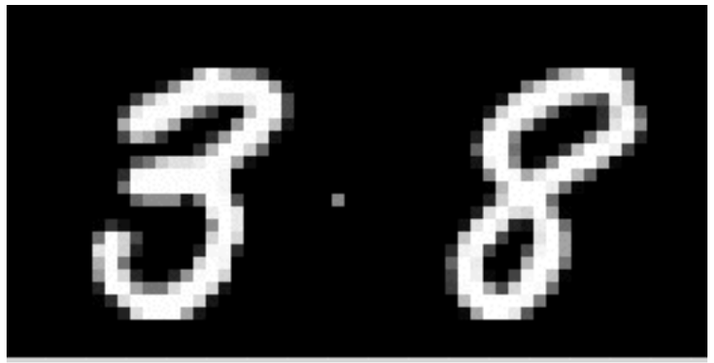
The importance of good features

- Most learning methods are invariant to feature permutation
  - E.g., patch vs. pixel representation of images

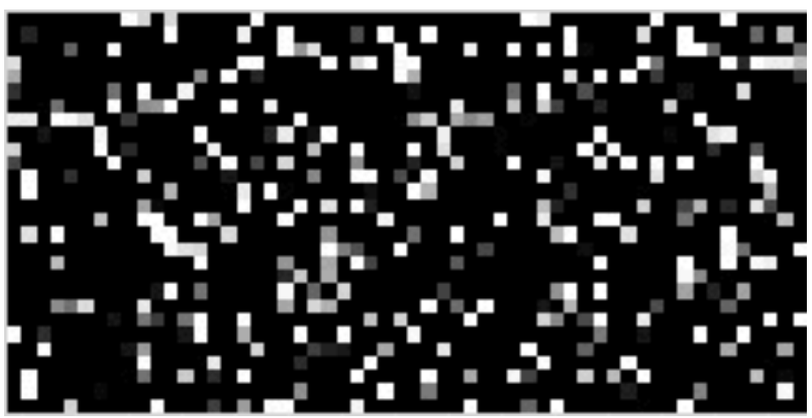


The importance of good features

- Most learning methods are invariant to feature permutation
  - E.g., patch vs. pixel representation of images



permute pixels

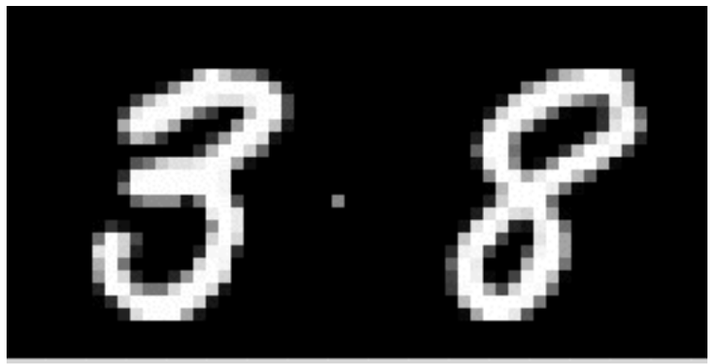


bag of pixels

can you recognize the digits?

The importance of good features

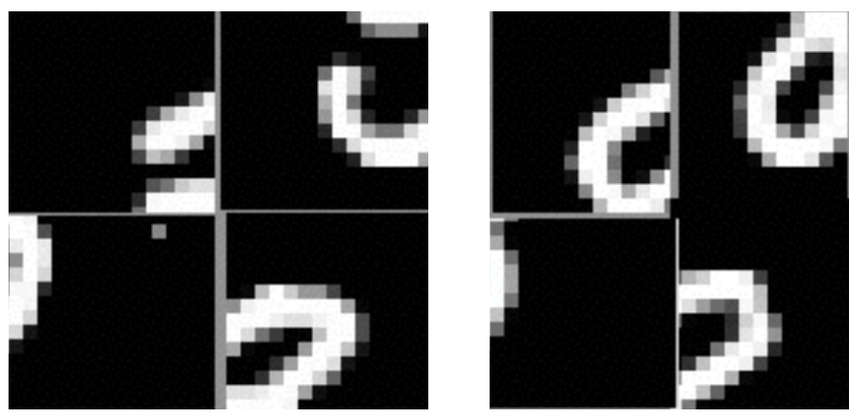
- Most learning methods are invariant to feature permutation
  - E.g., patch vs. pixel representation of images



permute pixels

permute patches

bag of pixels



bag of patches

can you recognize the digits?

Irrelevant and redundant features

- Irrelevant features
  - E.g., a binary feature with $\mathbb{E}[f; C] = \mathbb{E}[f]$
- Redundant features
  - For example, pixels next to each other are highly correlated
- Irrelevant features are not that unusual
  - Consider bag-of-words model for text which typically have on the order of 100,000 features, but only a handful of them are useful for spam classification

- Different learning algorithms are affected differently by irrelevant and redundant features

Irrelevant and redundant features

How do irrelevant features affect decision tree classifiers?

- Consider adding 1 binary noisy feature for a binary classification task
  - For simplicity assume that in our dataset there are N/2 instances label=+1 and N/2 instances with label=-1
  - Probability that a noisy feature is perfectly correlated with the labels in the dataset is $2x0.5^N$
  - Very small if N is large (1e-6 for N=21)
    - But things are considerably worse where there are many irrelevant features, or if we allow partial correlation
- For large datasets, the decision tree learner can learn to ignore noisy features that are not correlated with the labels.

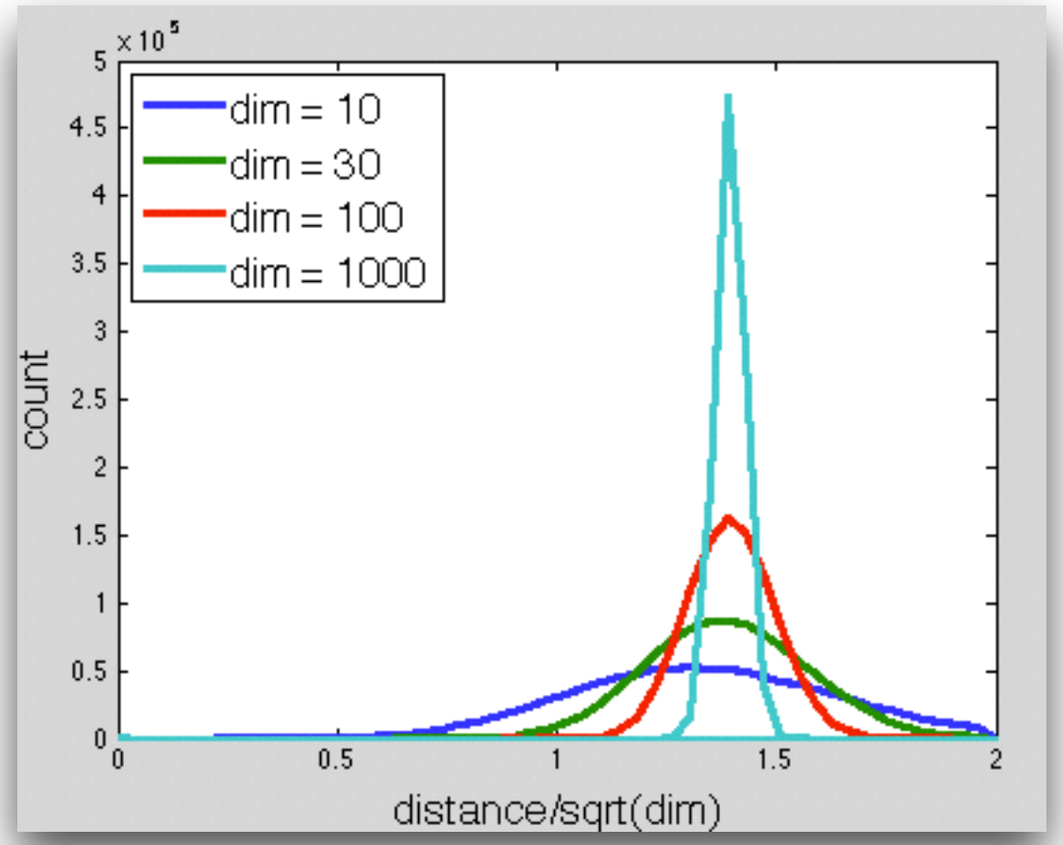
Irrelevant and redundant features

How do irrelevant features affect kNN classifiers?

- kNN classifiers (with Euclidean distance) treat all the features equally
- Noisy dimensions can dominate distance computation
- Randomly distributed points in high dimensions are all (roughly) equally apart!

$$a_i \leftarrow N(0,1) \qquad b_i \leftarrow N(0,1)$$

$$\mathbb{E}\left[||\mathbf{a}-\mathbf{b}||\right] \rightarrow \sqrt{2D}$$



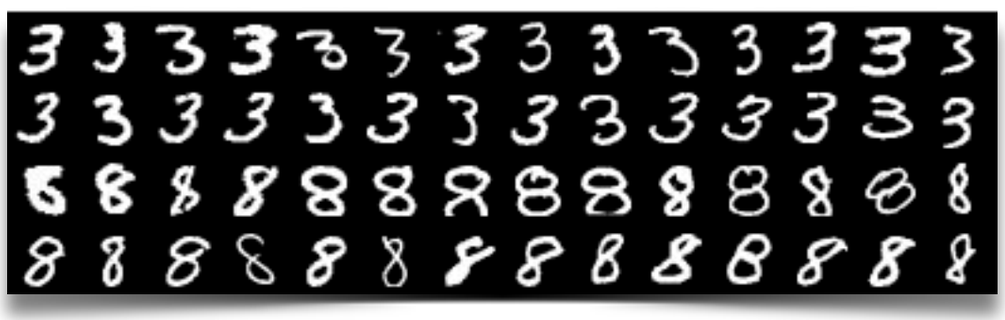
- kNN classifiers can be bad with noisy features even for large N

Irrelevant and redundant features

How do irrelevant features affect perceptron classifiers?

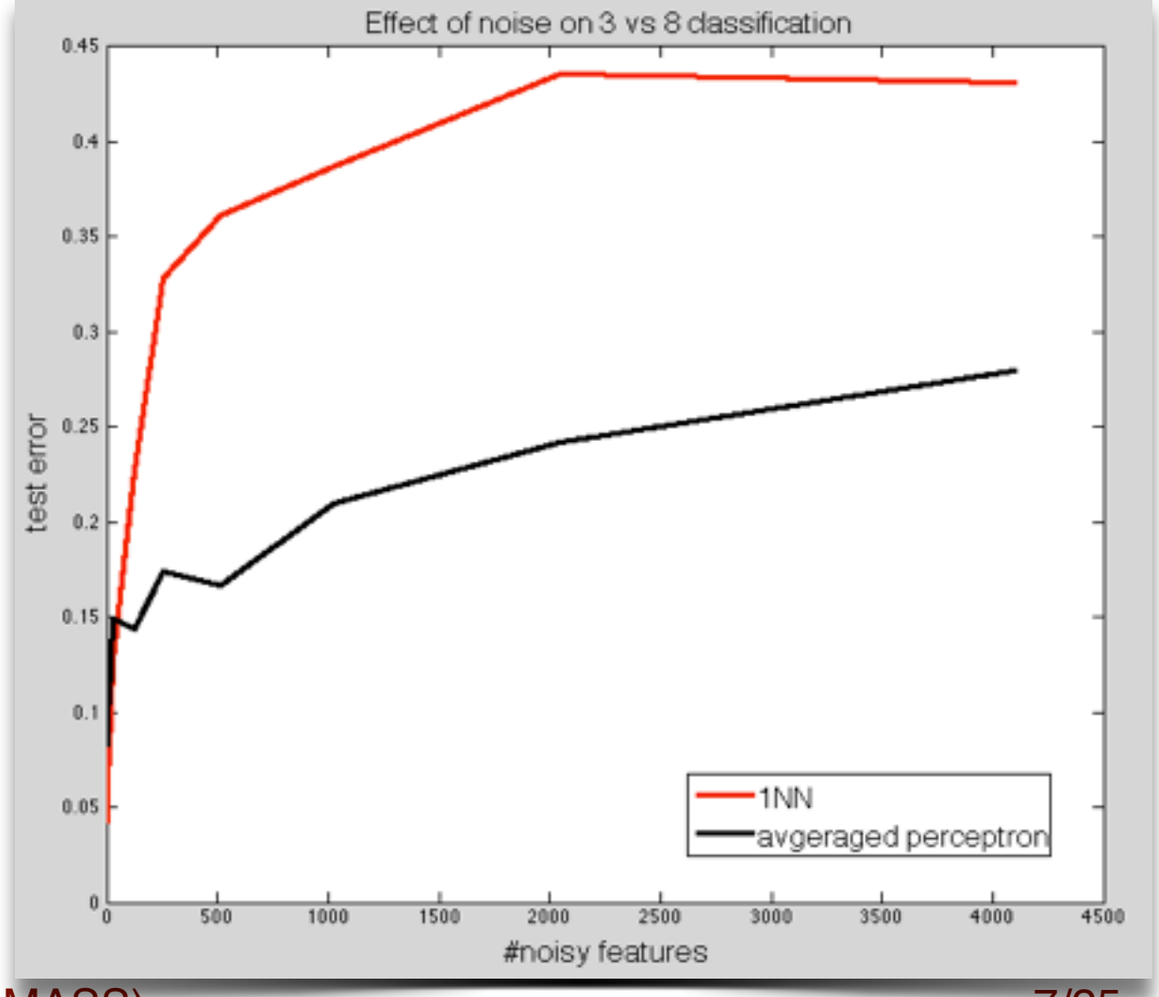
- Perceptrons can learn low weight on irrelevant features
- Irrelevant features can affect the convergence rate
  - updates are wasted on learning low weights on irrelevant features
- But like decision trees, if the dataset is large enough, the perceptron will eventually learn to ignore the weights
- Effect of noise on classifiers:

"3" vs "8" classification using pixel features
(28x28 images = 784 features)



$\mathbf{x} \leftarrow [\mathbf{x}\ \mathbf{z}] \quad z_i = N(0,1), \quad i = 2^0, \ldots, 2^{12}$

vary the number of noisy dimensions



Feature selection

- Selecting a small subset of useful features
- Reasons:
  - Reduce measurement cost
  - Reduces data set and resulting model size
  - Some algorithms scale poorly with increased dimension
  - Irrelevant features can confuse some algorithms
  - Redundant features adversely affect generalization for some learning methods
  - Removal of features can make learning easier and improve generalization (for example by increasing the margin)

Feature selection methods

Feature selection methods

- Methods agnostic to the learning algorithm

Feature selection methods

- Methods agnostic to the learning algorithm
  - Surface heuristics: remove a feature if it rarely changes

Feature selection methods

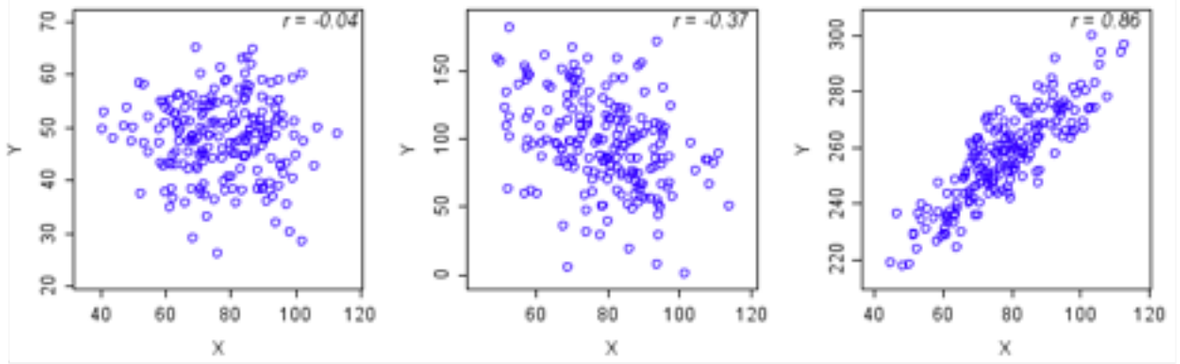
- Methods agnostic to the learning algorithm
  - Surface heuristics: remove a feature if it rarely changes
  - Ranking based: rank features according to some criteria

Feature selection methods

- Methods agnostic to the learning algorithm
  - Surface heuristics: remove a feature if it rarely changes
  - Ranking based: rank features according to some criteria
    - Correlation:

$$\rho_{X,Y} = \operatorname{corr}(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

scatter plot

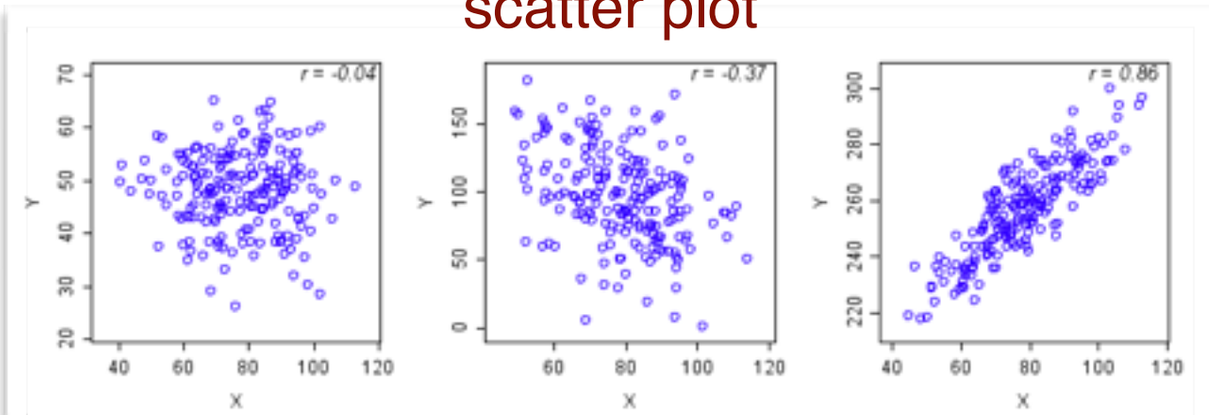


Feature selection methods

- Methods agnostic to the learning algorithm
  - Surface heuristics: remove a feature if it rarely changes
  - Ranking based: rank features according to some criteria
    - Correlation:

$$\rho_{X,Y} = \operatorname{corr}(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

scatter plot



    - Mutual information:

$$I(X;Y) = \int_Y \int_X p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right) dx\, dy,$$

$$H(X) = -\sum_x p(x) \log p(x)$$

entropy

$$I(X;Y) = H(X) - H(X|Y)$$

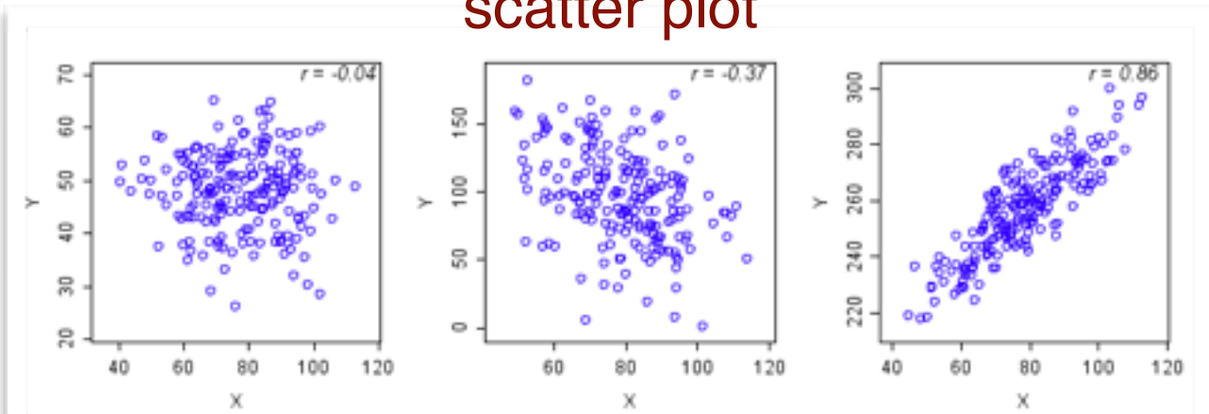
decision trees?

Feature selection methods

- Methods agnostic to the learning algorithm
  - Surface heuristics: remove a feature if it rarely changes
  - Ranking based: rank features according to some criteria
    - Correlation:

$$\rho_{X,Y} = \operatorname{corr}(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

scatter plot



    - Mutual information:

$$I(X;Y) = \int_Y \int_X p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right) dx\, dy,$$

$$H(X) = -\sum_x p(x) \log p(x)$$

entropy

$$I(X;Y) = H(X) - H(X|Y)$$

decision trees?

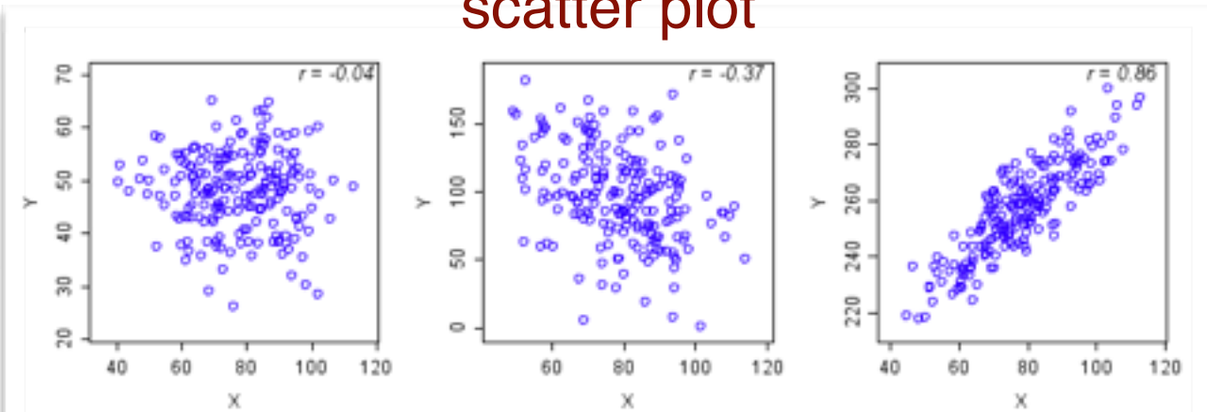
  - Usually cheap

Feature selection methods

- Methods agnostic to the learning algorithm
  - Surface heuristics: remove a feature if it rarely changes
  - Ranking based: rank features according to some criteria
    - Correlation:

$$\rho_{X,Y} = \operatorname{corr}(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y},$$

scatter plot



    - Mutual information:

$$I(X;Y) = \int_Y \int_X p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right) dx\, dy,$$

$$H(X) = -\sum_x p(x) \log p(x)$$

entropy

$$I(X;Y) = H(X) - H(X|Y)$$

decision trees?

  - Usually cheap
- Wrapper methods
  - Aware of the learning algorithm (forward and backward selection)
  - Can be computationally expensive

Forward and backward selection

Forward and backward selection

- **Given:** a learner L, a dictionary of features D to select from
  - E.g., L = kNN classifier, D = polynomial functions of features
- **Forward selection**
  - Start with an empty set of features F = Φ
  - Repeat till |F| < n
    - For every f in D
      - Evaluate the performance of the learner on F ∪ f
    - Pick the best feature f*
    - F = F ∪ f*, D = D \ f*

Forward and backward selection

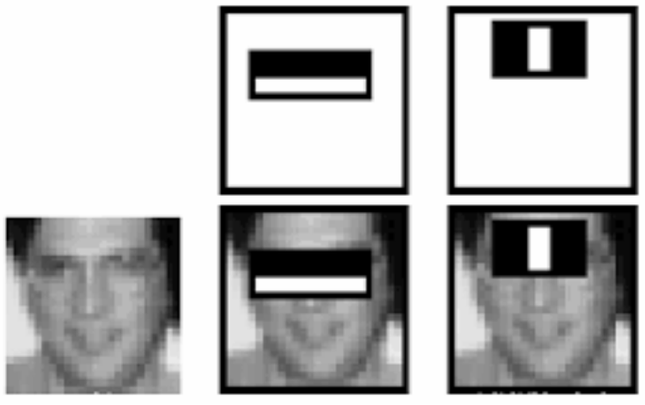
- **Given:** a learner L, a dictionary of features D to select from
  - E.g., L = kNN classifier, D = polynomial functions of features
- **Forward selection**
  - Start with an empty set of features F = Φ
  - Repeat till |F| < n
    - ➡ For every f in D
      - Evaluate the performance of the learner on F ∪ f
    - ➡ Pick the best feature f*
    - ➡ F = F ∪ f*, D = D \ f*
- **Backward selection is similar**
  - Initialize F = D, and iteratively remove the feature that is least useful
  - Much slower than forward selection

Forward and backward selection

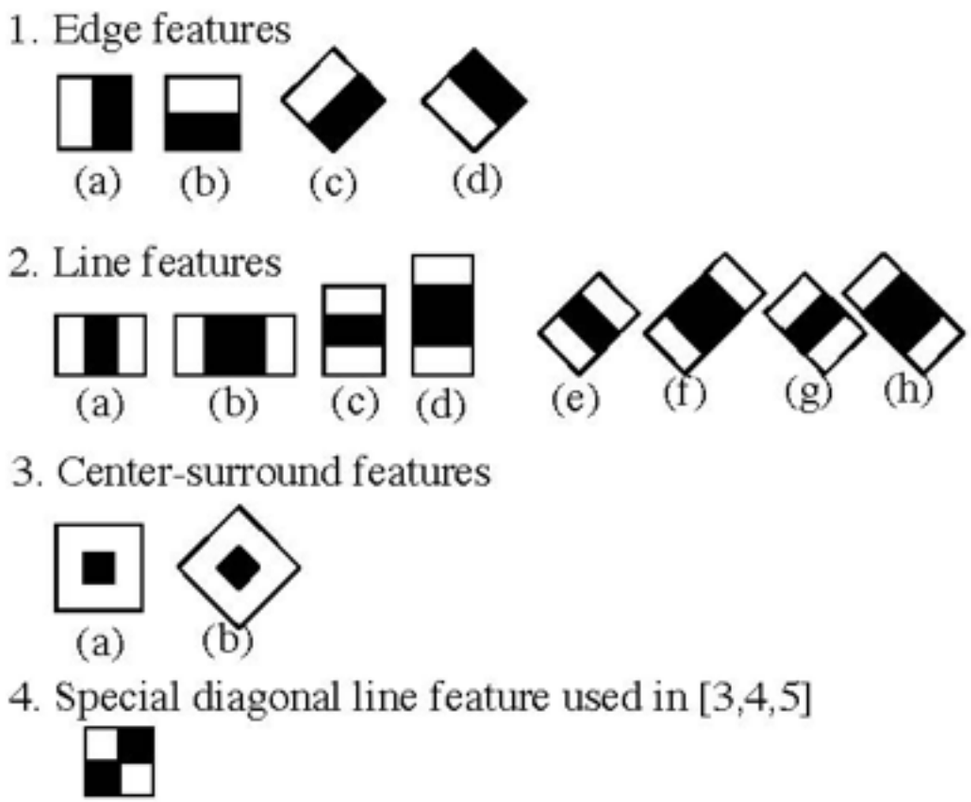
- **Given:** a learner L, a dictionary of features D to select from
  - E.g., L = kNN classifier, D = polynomial functions of features
- **Forward selection**
  - Start with an empty set of features F = Φ
  - Repeat till |F| < n
    - For every f in D
      - Evaluate the performance of the learner on F ∪ f
    - Pick the best feature f*
    - F = F ∪ f*, D = D \ f*
- **Backward selection is similar**
  - Initialize F = D, and iteratively remove the feature that is least useful
  - Much slower than forward selection
- **Greedy, but can be near optimal under certain conditions**

Approximate feature selection

- What if the number of potential features are very large?
  - If may be hard to find the optimal feature



[Viola and Jones, IJCV 01]



- Approximation by sampling: pick the best among a random subset
- If done during decision tree learning, this will give you a random tree
  - We will see later (in the lecture on ensemble learning) that it is good to train *many* random trees and average them (random forest).

Feature normalization

Feature normalization

- Even if a feature is useful some normalization may be good

Feature normalization

- Even if a feature is useful some normalization may be good
- Per-feature normalization
  - Centering $x_{n,d} \leftarrow x_{n,d} - \mu_d$
  - Variance scaling $x_{n,d} \leftarrow x_{n,d}/\sigma_d$
  - Absolute scaling $x_{n,d} \leftarrow x_{n,d}/r_d$

$$\mu_d = \frac{1}{N}\sum_n x_{n,d}$$

$$\sigma_d = \sqrt{\frac{1}{N}\sum_n (x_{n,d} - \mu_d)^2}$$

$$r_d = \max_n |x_{n,d}|$$

Feature normalization

- Even if a feature is useful some normalization may be good
- Per-feature normalization
  - Centering $x_{n,d} \leftarrow x_{n,d} - \mu_d$
  - Variance scaling $x_{n,d} \leftarrow x_{n,d}/\sigma_d$
  - Absolute scaling $x_{n,d} \leftarrow x_{n,d}/r_d$
  - Non-linear transformation
    - ➡ square-root

$$x_{n,d} \leftarrow \sqrt{x_{n,d}}$$

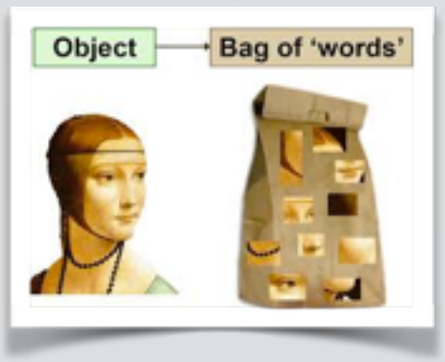
(corrects for burstiness)

$$\mu_d = \frac{1}{N}\sum_n x_{n,d}$$

$$\sigma_d = \sqrt{\frac{1}{N}\sum_n (x_{n,d} - \mu_d)^2}$$

$$r_d = \max_n |x_{n,d}|$$

Caltech-101 image classification



41.6% linear
63.8% square-root

Feature normalization

- Even if a feature is useful some normalization may be good
- Per-feature normalization
  - Centering $x_{n,d} \leftarrow x_{n,d} - \mu_d$
  - Variance scaling $x_{n,d} \leftarrow x_{n,d}/\sigma_d$
  - Absolute scaling $x_{n,d} \leftarrow x_{n,d}/r_d$
  - Non-linear transformation
    - ➡ square-root

    $$x_{n,d} \leftarrow \sqrt{x_{n,d}}$$

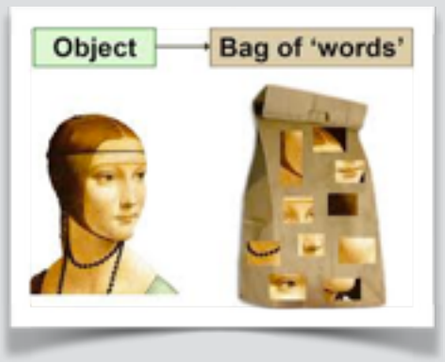
    (corrects for burstiness)

$$\mu_d = \frac{1}{N}\sum_n x_{n,d}$$

$$\sigma_d = \sqrt{\frac{1}{N}\sum_n (x_{n,d} - \mu_d)^2}$$

$$r_d = \max_n |x_{n,d}|$$

Caltech-101 image classification



41.6% linear
63.8% square-root

- Per-example normalization
  - fixed norm for each example $||\mathbf{x}|| = 1$

Feature selection summary

- Choice of features is really important for most learners
- Noisy features:
  - All learners are bad when there are too many noisy features since some of these are likely to correlate well with labels
  - Some learners can learn to ignore noisy features given enough training data (e.g., perceptron and decision trees)
  - kNN suffers in high dimensions with noisy features
- Feature selection
  - May improve generalization and computational efficiency
  - Feature selection methods:
    - Learning agnostic methods:
      - correlation, mutual information, etc
    - Wrapper methods (uses a learner in the loop):
      - forward and backward selection
- Feature normalization:
  - per-feature - centering, variance/absolute scaling, square root
  - per-example - unit norm

Model selection

- Lots of choices when using machine learning techniques
  - learner: kNN classifier, decision trees, perceptrons, et
  - features: what? how many? normalization?
  - hyperparameters
    - ➡ k for kNN classifier
    - ➡ maximum depth of the decision tree
    - ➡ number of iterations for the averaged perceptron training

- How do we measure the performance of models?
  - Ideally we would like models that have low generalization error
  - But we don't have access to the test data or the data distribution

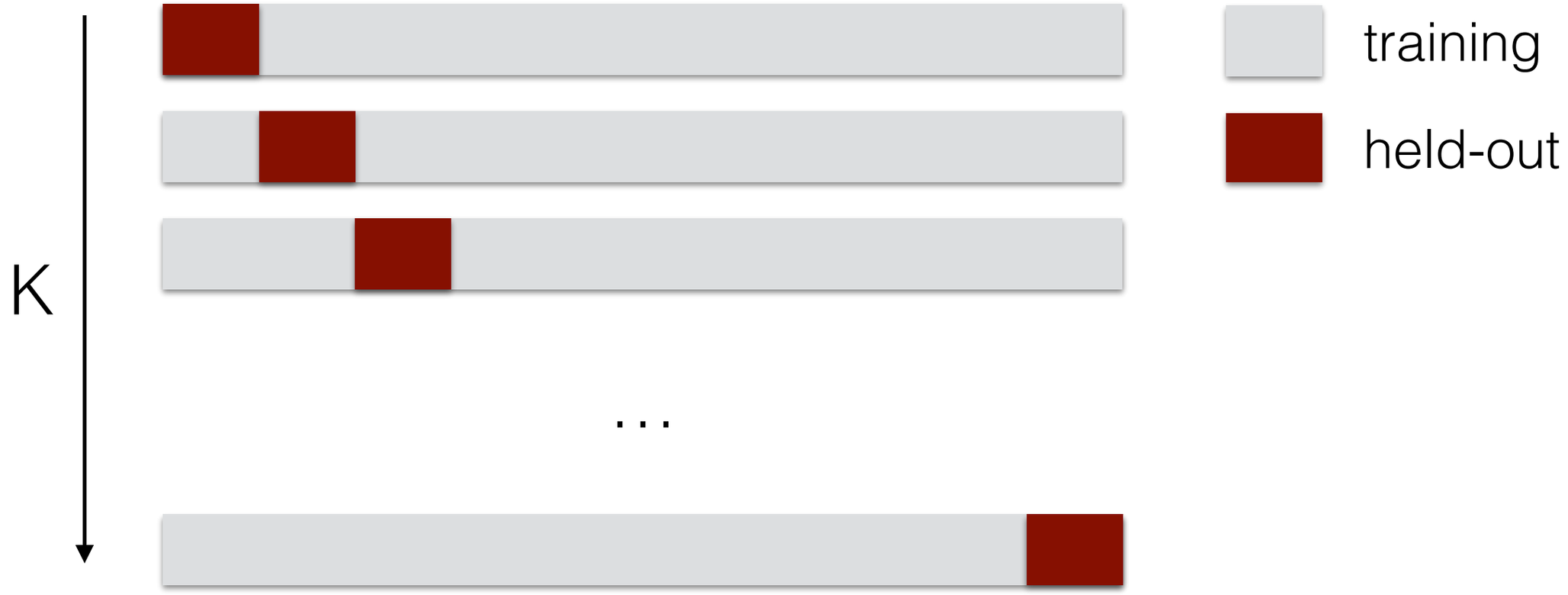
Held-out data

- Set aside a fraction (10%-20%) of the training data
- This becomes our held-out data
  - Other names validation/development data

  - **Remember:** this is NOT the test data
  - Train each model on the remaining training data
  - Evaluate error on the held-out data
  - Choose model with the smallest held-out error
- **Problems:**
  - Wastes training data
  - May get unlucky with the split leading to a poor estimate of error

Cross-validation

- K-fold cross-validation
  - Create K equal sized partitions of the training data
  - Each partition has N/K examples
  - Train using K – 1 partitions, validate on the remaining partition
  - Repeat the same K times, each with a different validation partition



  - Finally, choose the model with smallest average validation error
  - Usually K is chosen as 10

Leave-one-out (LOO) cross-validation

- K-fold cross-validation with K=N (number of training examples)
  - Each partition contains only one example
  - Train using N–1 examples, validate on the remaining example
  - Repeat the same N times, each with a different validation example

  - Finally, choose the model with smallest average validation error
  - Can be expensive for large N. Typically used when N is small

LOO error example: kNN classifier

- Efficiently picking the k for kNN classifier

Algorithm 9 KNN-TRAIN-LOO(**D**)

$err_k \leftarrow 0, \forall 1 \leq k \leq N-1$ // err_k stores how well you do with kNN
for $n = 1$ **to** N **do**
  $S_m \leftarrow \langle ||\boldsymbol{x}_n - \boldsymbol{x}_m||, m \rangle, \forall m \neq n$ // compute distances to other points
  $S \leftarrow \text{SORT}(S)$ // put lowest-distance objects first
  $\hat{y} \leftarrow 0$ // current label prediction
  for $k = 1$ **to** $N-1$ **do**
    $\langle dist, m \rangle \leftarrow S_k$
    $\hat{y} \leftarrow \hat{y} + y_m$ // let kth closest point vote
    if $\hat{y} \neq y_m$ **then**
      $err_k \leftarrow err_k + 1$ // one more error for kNN
    end if
  end for
end for
return $\text{argmin}_k \, err_k$ // return the K that achieved lowest error

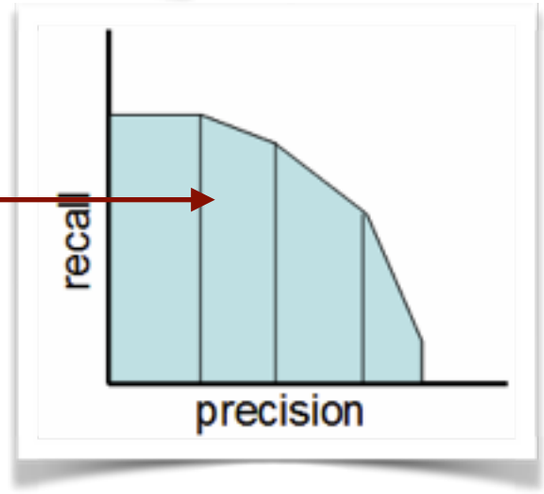
source: CIML book (Hal Daume III)

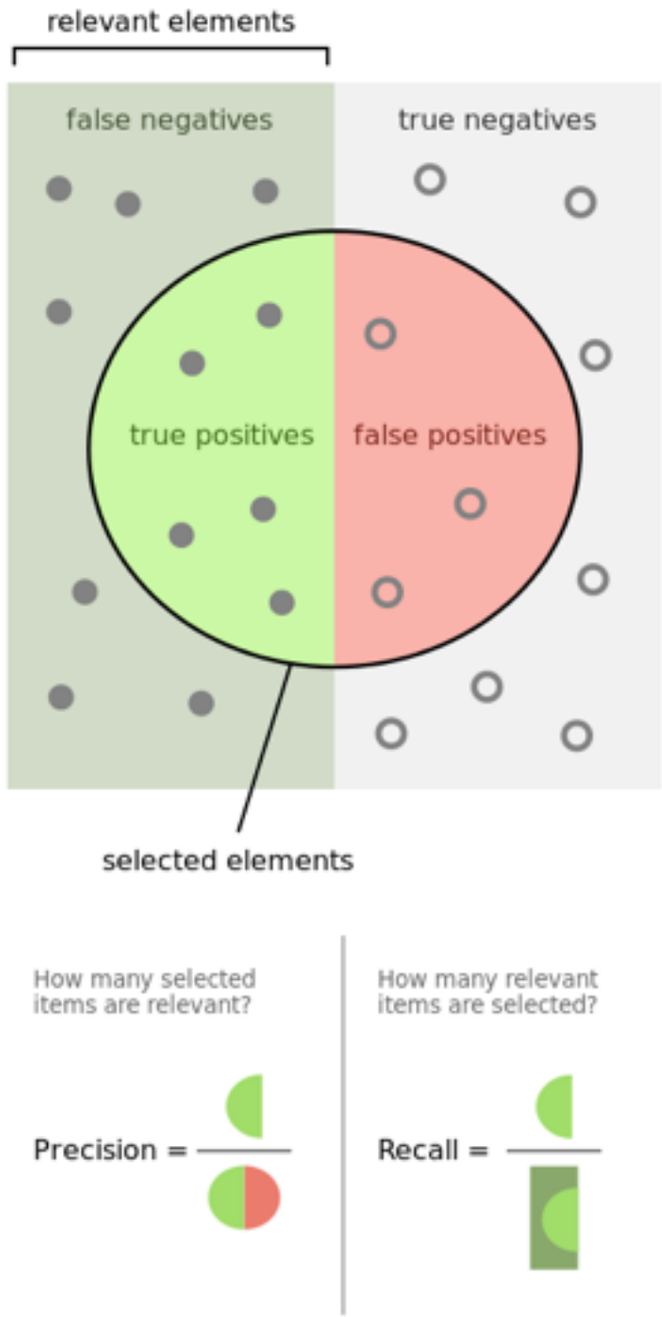
Other performance metrics

- Accuracy is not always a good metric
  - Face detection (1 in a million patches is a face)
  - Accuracy of the classifier that always says no = 99.9999%
- Precision and recall
  - **true positives:** selected elements that are relevant
  - **false positives:** selected elements that are irrelevant
  - **true negatives:** missed elements that are irrelevant
  - **false negatives:** missed elements that are relevant
  - **precision** = true positives/(true positives + false positives)
  - **recall** = true positives/(true positives + false negatives)
  - **f-score =** harmonic mean of precision and recall

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

  - precision vs. recall curve
    - vary the threshold
    - average precision (AP)





source: wikipedia

Statistical significance

- Classifier A achieves 7.0% error
- Classifier B achieves 6.9% error

- How significant is the 0.1% difference in error
  - Depends on how much data did we test it on
    - 1000 examples: not so much (random luck)
    - 1m examples: probably

- Statistical significance tests
  - "There is a 95% chance that classifier A is better than classifier B"
  - We accept the hypothesis if the chance is greater than 95%
    - "Classifier A is better than classifier B" (hypothesis)
    - "Classifier A is is no better than classifier B" (null-hypothesis)
  - 95% is arbitrary (you could also report 90% or 99.99%)
  - A common example is "is treatment A better than placebo"

“Lady tasting tea”

- The experiment provided the Lady with 8 randomly ordered cups of tea – 4 prepared by first adding milk, 4 prepared by first adding the tea. She was to select the 4 cups prepared by one method.
  - The Lady was fully informed of the experimental method.
- The “null hypothesis” was that the Lady had no such ability (i.e., randomly guessing)
- The Lady correctly categorized all the cups!
- There are (8 choose 4) = 70 possible combinations. Thus, the probability that the lady got this by chance = 1/70 (1.4%)

Ronald Fisher

Fisher exact test

http://en.wikipedia.org/wiki/Lady_tasting_tea

“Lady tasting tea”

- The experiment provided the Lady with 8 randomly ordered cups of tea – 4 prepared by first adding milk, 4 prepared by first adding the tea. She was to select the 4 cups prepared by one method.
  - The Lady was fully informed of the experimental method.
- The “null hypothesis” was that the Lady had no such ability (i.e., randomly guessing)
- The Lady correctly categorized all the cups!
- There are (8 choose 4) = 70 possible combinations. Thus, the probability that the lady got this by chance = 1/70 (1.4%)

Ronald Fisher

Fisher exact test

http://en.wikipedia.org/wiki/Lady_tasting_tea

Statistical significance: paired t-test

- Suppose you have two algorithms evaluated on N examples with error

$$\text{A, with } \mathbf{a} = a_1, a_2, \ldots, a_N \qquad \text{B, with } \mathbf{b} = b_1, b_2, \ldots, b_N$$

$$\hat{a} = \mathbf{a} - \mu_a \qquad \hat{b} = \mathbf{b} - \mu_b$$

- The t-statistic is defined as:

$$t = (\mu_a - \mu_b)\sqrt{\frac{N(N-1)}{\sum_n (\hat{a}_n - \hat{b}_n)^2}}$$

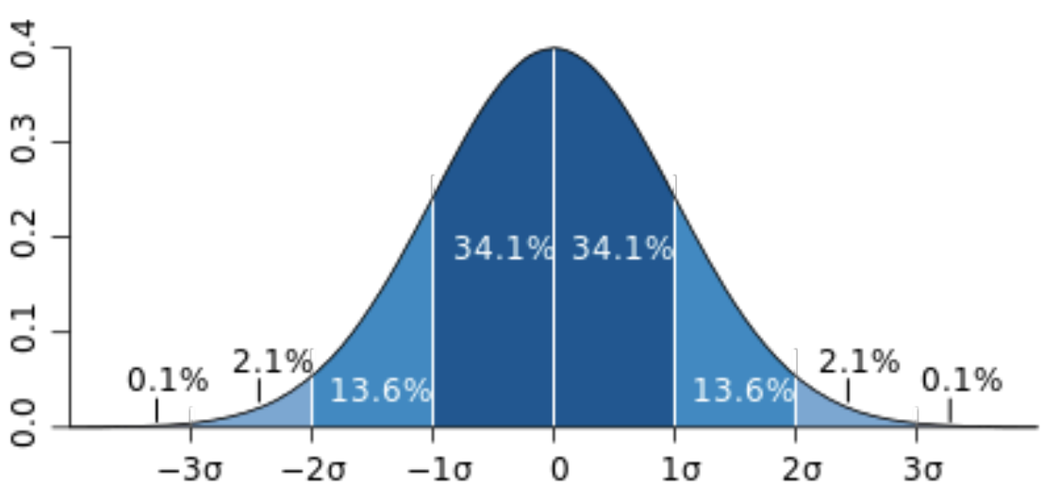
N has to be large (>100)

- Once you have a t value, compare it to a list of values on this table and report the significance level of the difference:

t	significance
≥ 1.28	90.0%
≥ 1.64	95.0%
≥ 1.96	97.5%
≥ 2.58	99.5%

Confidence intervals: cross-validation

- Paired t-test cannot be applied to metrics that measure accuracy on the entire set (e.g. f-score, average precision, etc)
- Fortunately we can use cross-validation
  - For example, you run 5-fold cross validation
  - Method A gets f-scores **92.4**, **93.9**, **96.1**, **92.2** and **94.4**
    - ➡ Average f-score **93.8**, standard deviation **1.595**
  - Assuming the distribution of scores is a Gaussian:
    - ➡ 70% prob. mass lies in $[\mu - \sigma, \mu + \sigma]$
    - ➡ 95% prob. mass lies in $[\mu - 2\sigma, \mu + 2\sigma]$
    - ➡ 99.5% prob. mass lies in $[\mu - 3\sigma, \mu + 3\sigma]$



  - So, if we were comparing this algorithm with another whose average f-score was **90.6%**, we could be **95%** certain that the better performance of A is not due to chance.

Confidence intervals: bootstrapping

- Sometimes we cannot re-train the classifier
  - E.g., a black-box classifier you downloaded from the web
- All we have is a single test dataset of size N
  - How do we generate confidence intervals?
- Bootstrapping: a method to generate new datasets from a single one
  - Generate M copies of the dataset by sampling N points uniformly at random with replacement
    - ➡ without replacement the copies will be identical to the original
  - Measure f-score on each of these M datasets
  - Derive confidence intervals for the estimate of f-score
- Closely related to jackknife resampling
  - Generate N copies of the data of size (N-1) by leaving out each instance one by one

http://en.wikipedia.org/wiki/Bootstrapping_statistics

http://en.wikipedia.org/wiki/Jackknife_resampling

Slides credit

- Slides are adapted from CIML book by Hal Daume, slides by Piyush Rai at Duke University, and Wikipedia
- Digit images are from the MNIST dataset by Yann LeCun