CS6375: Machine Learning

Gautam Kunapuli

Machine Learning Practice

THE UNIVERSITY OF TEXAS AT DALLAS

Erik Jonsson School of Engineering and Computer Science

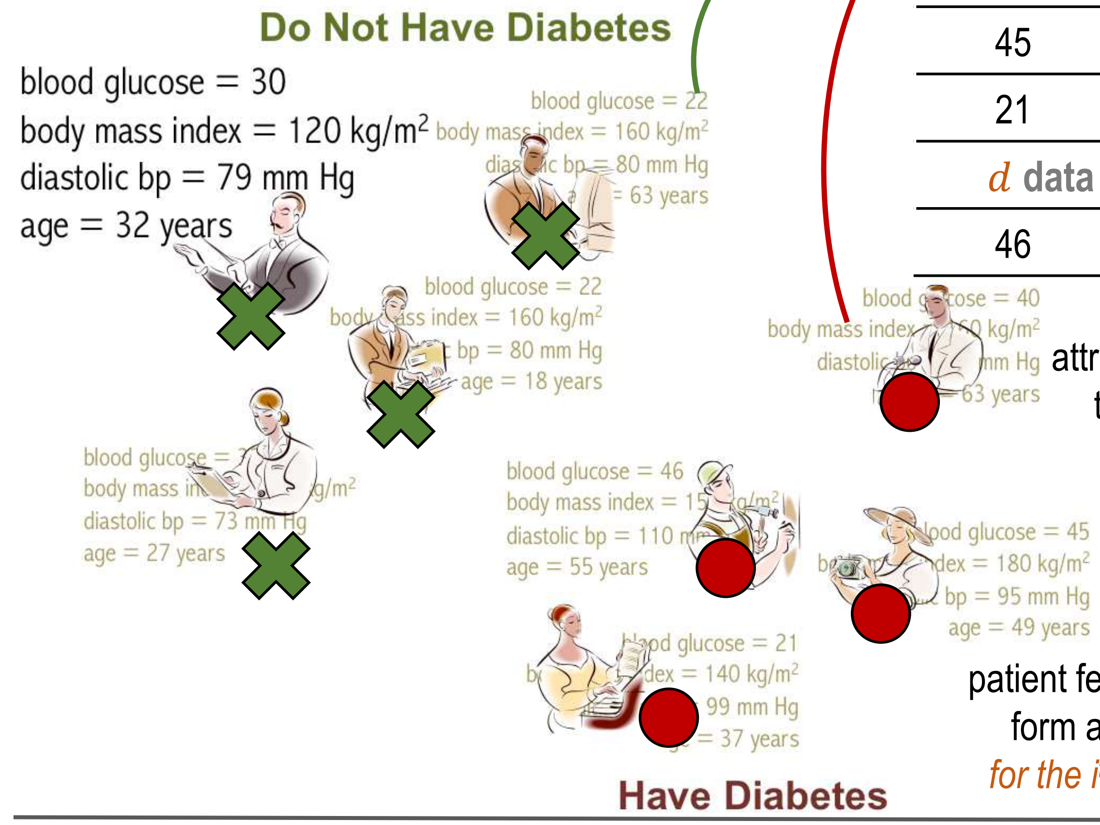
Machine Learning Terminology

exampleample: **Diabetes Diagnosis from Medical Records**. You are developing a clinical decision-support system to **diagnose diabetes** from patient health records.

blood glucose	body mass idx	diastolic blood pr.	age	Diabetes?
30	120	79	32	NO
22	160	80	63	NO
40	160	93	63	YES
22	160	80	18	NO
45	180	95	49	YES
21	140	99	37	YES
				NO
46	153	110	55	YES

d **data features** (aka attributes, variables)

n data examples



attributes and descriptors for each patient are the **features** or **independent variables**
for the ith patient, the kth feature is denoted x_i^k

the diagnosis or the **prediction** is the target or the (training) **label**
for the ith patient, denoted y_i

patient features are collected to form a (training) **example**
for the ith patient, denoted $\boldsymbol{x}_i$

Data Transformation: Normalization

Normalization is a "scaling down" transformation of **features**

- especially important when there is a **large difference in feature ranges** among **the continuous features**
- especially important for **neural network** and **nearest neighbor** algorithms

blood gluc.	bmi	diast bp	age	Diab?
30	120	79	32	-1
22	160	80	63	-1
40	160	93	63	+1
22	160	80	18	-1
45	180	95	49	+1
21	140	99	37	+1
26	110	73	27	-1
46	153	110	55	+1

n **data examples**, indexed by i

d **data features**, indexed by j

feature normalization is performed on each column

- **min-max normalization**

$$\ell_j = \min_{i=1,\dots,n} x_i^j \quad \textbf{(feature minimum)}$$

$$u_j = \max_{i=1,\dots,n} x_i^j \quad \textbf{(feature maximum)}$$

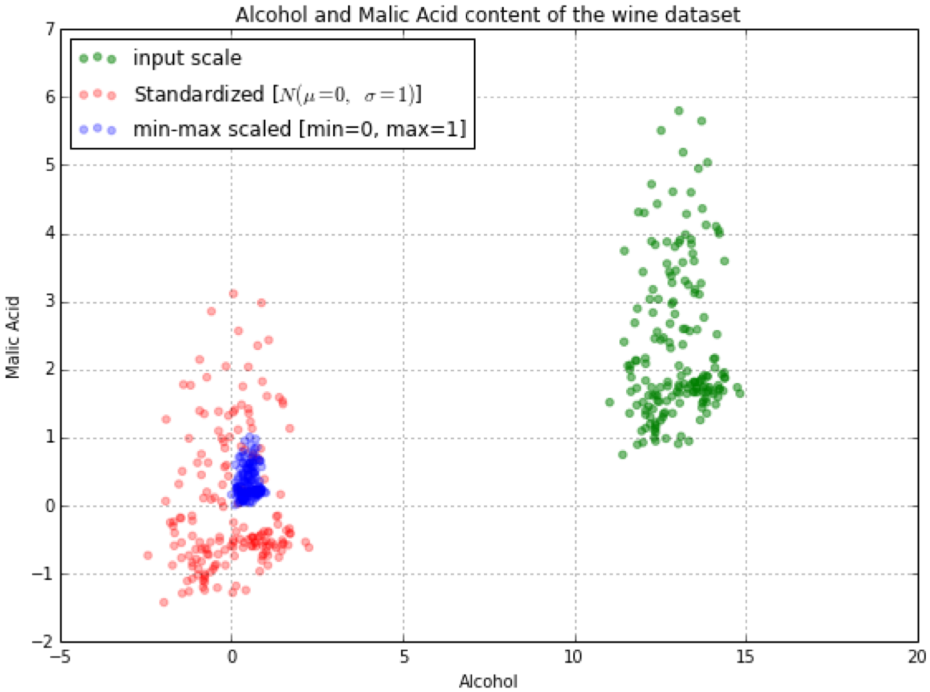
$$\bar{x}_i^j = \frac{x_i^j - \ell_j}{u_j - \ell_j} \quad \textbf{(normalized feature)}$$

- **z-score standardization** (normalize using mean and standard deviation)

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_i^j \quad \textbf{(feature mean)}$$

$$\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^j - \mu_j\right)^2 \quad \textbf{(feature variance)}$$

$$\bar{x}_i^j = \frac{x_i^j - \mu_j}{\sigma_j} \quad \textbf{(normalized feature)}$$



Data Transformation: Discretization

Discretization divides the **range of a continuous attribute** into **intervals** because some learning algorithms **only accept categorical attributes** or considering only categorical attributes can make **learning more efficient**

- can introduce interpretability (for example, replacing raw age attributes with higher-level semantic concepts (such as young, middle-aged, or senior)

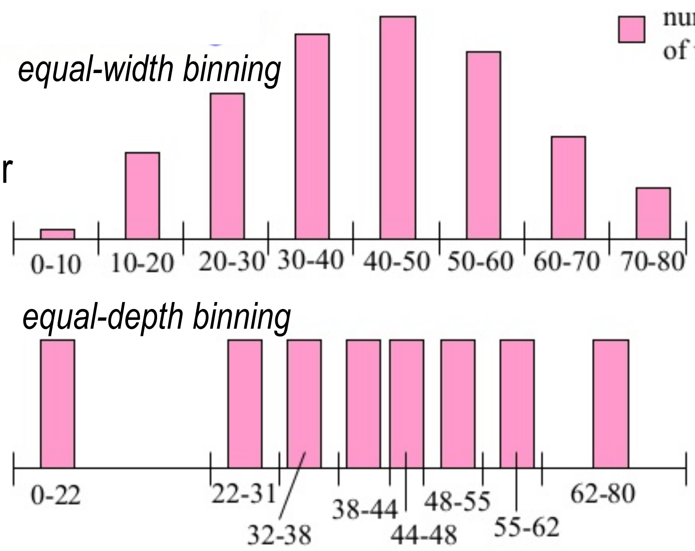
n **data examples**, indexed by i

blood gluc.	bmi	diast bp	age	Diab?
30	120	79	32	-1
22	160	80	63	-1
40	160	93	63	+1
22	160	80	18	-1
45	180	95	49	+1
21	140	99	37	+1
26	110	73	27	-1
46	153	110	55	+1

d **data features,** indexed by j

feature discretization is performed on each column

- **equal-width (distance) binning**
  - divides feature range into K intervals of **equal size**
  - if ℓ and u are lowest and highest attribute values, bin width will be: $b_w = (u - \ell)/ K$.
  - straightforward to use, but outliers may dominate
  - skewed data is not handled well
- **equal-depth (frequency) binning**
  - divides the range K intervals each containing same number of data points
  - good data scaling
- **bottom-up binning**
  - criteria such as entropy or $\chi^2$to characterize purity of bins
- **clustering**



Data Transformation: Nominal and Ordinal

Nominal and ordinal attributes must be converted to “continuous” attributes in order to ensure that features are handled properly by some machine-learning algorithms (e.g. SVMs)

- Nominal — values from an unordered set, e.g., color, profession
- Ordinal — values from an ordered set, e.g., military or academic rank

blood gluc.	bmi	diast bp	age	Diab?
30	120	79	0	-1
22	160	80	2	-1
40	160	93	1	+1
22	160	80	0	-1
45	180	95	2	+1
21	140	99	1	+1
26	110	73	1	-1
46	153	110	3	+1

n **data examples,** indexed by i

d **data features,** indexed by j

Introduce multiple numeric features for one nominal feature. Consider that the example attribute **age** takes 4 values: child=0, teen=1, adult=2, senior=3

- **binarization:**
  - ○ **0 = 00, 1 = 01, 2 = 10, 3 = 11**
  - ○ requires $\lceil \log_2(\#\mathrm{uniqueValues}(x_j)) \rceil$ columns
- **one-hot vectorization:**
  - ○ **0 = 1000, 1 = 0100, 2 = 0010, 3 = 0001**
  - ○ requires $\#\mathrm{uniqueValues}(x_j)$ columns
- interpretability is lost
- must often ensure that **features are selected together**

blood gluc.	bmi	diast bp	age$_1$	age$_2$	age$_3$	age$_4$	Diab?
30	120	79	1	0	0	0	-1
22	160	80	0	0	1	0	-1
40	160	93	0	1	0	0	+1
22	160	80	1	0	0	0	-1
45	180	95	0	0	1	0	+1
21	140	99	0	1	0	0	+1
26	110	73	0	1	0	0	-1
46	153	110	0	0	0	1	+1

Data Transformation: Missing Data

Data is not always available all the time; missing data may be due to – equipment malfunction, data entry mistakes, inconsistent with other recorded data and thus deleted. In many cases, missing data has to be **imputed**.

- **Manually**: tedious + infeasible; crowdsourcing?
- **Ignoring Instances with Unknown Feature Values**: just ignore instances that have at least one unknown feature value
- **Most Common** Feature Value: most common feature value is selected to be the missing value
  - ○ **Concept Most Common** Feature Value: the value of the feature that is most common within the same class is selected to be the missing value
- **Mean substitution**: substitute feature mean computed from available data as missing value
- **Regression or classification methods**: Develop a regression or classification model based on complete case data for a given feature
- **Nearest-neighbor imputation**: Identify the most similar cases (nearest neighbors) to the case with a missing value to impute missing values as mean/median/most common etc.
- **Treating Missing Feature Values as Special Values**: treating “unknown” itself as a new value for the features that contain missing values
- **Latent Variable Methods**: such as Bayesian models can handle this directly, as they can estimate latent/missing values during learning

blood gluc.	bmi	diast bp	age	Diab?
??	120	79	32	-1
22	160	80	63	-1
40	160	93	??	+1
22	??	80	18	-1
45	180	95	??	+1
21	140	99	37	+1
26	110	73	27	-1
46	??	110	55	+1

n **data examples**

d **data features** (attributes)

Data Reduction: Sampling

Instance selection can be used to handle noise and remove extreme outliers that can harm the classification performance without adding anything meaningful to the models

Instance selection can also be used for for coping with the infeasibility of learning from very large data sets and maintain model quality while reducing sample size

- **random sampling** selects a subset of instances randomly
- **stratified sampling** for imbalanced data sets to ensure that instances of the minority classes are selected with a greater frequency in order to even out the distribution

Other approaches to imbalanced data sets:

- duplicating training examples of the under represented class; re-sampling examples, that is, over-sampling; commonly used in deep learning
- removing training examples of the over represented class; down-sampling

The Importance of Evaluation

When a learning system is **deployed** in the real world, we need to be able to **quantify the performance** of the classifier

- How accurate will the classifier be
- When it is wrong, why is it wrong?

This is very important as it is useful to **decide which classifier to use in which situations**

Our learning methodology should evaluate performance on several fronts

- correctness on **novel** examples (inductive learning)
- learning time
- prediction/testing time
- speedup after learning (explanation-based learning)
- space required to store the models

Basic idea: **repeatedly** use **training**, **tuning** and **testing** sets to estimate future performance

A 2002 paper in *Nature* (a major journal) needed to be corrected due to “training on the testing set”

Original report : 95% accuracy (5% error)

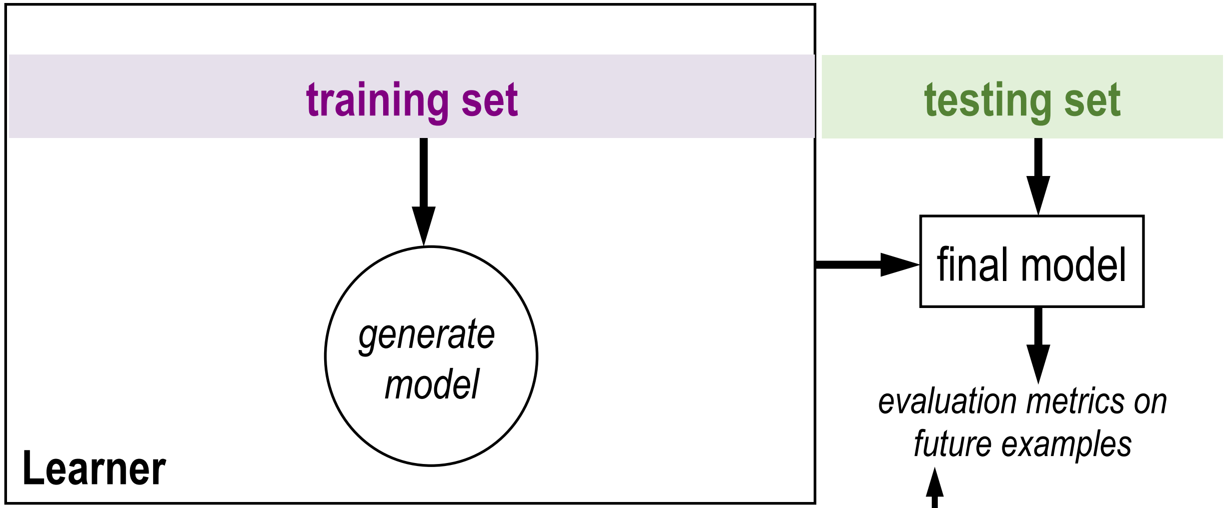
Corrected report (which still is buggy): 73% accuracy (27% error rate)

When following the **correct methodology** of evaluating performance on a hold-out set, **error rate increased by over 400%**

Methodology: The Hold-Out Testing Set

Split the available data into a training set and a hold-out test set

Train the classifier on the training set and **evaluate** on the test set



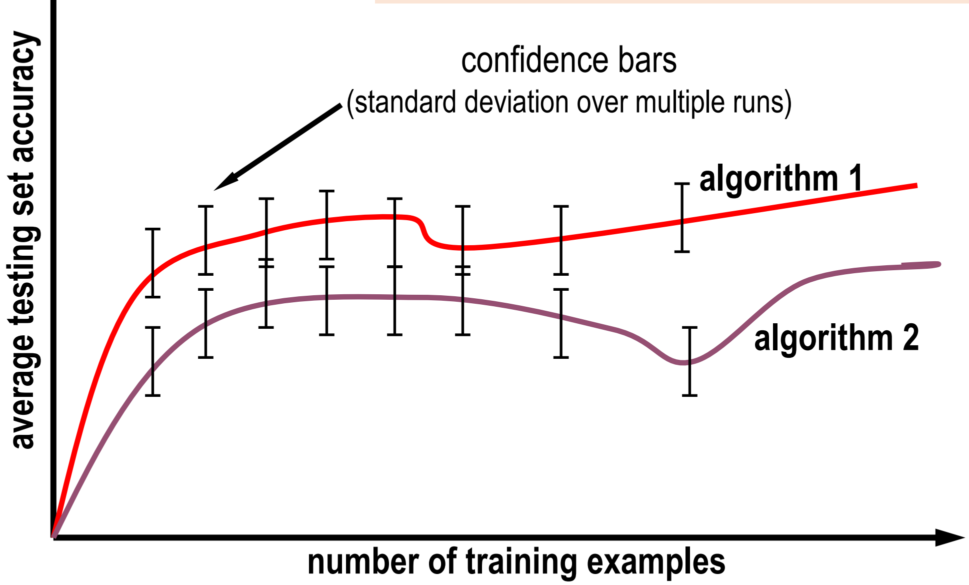
Evaluation Metric: Accuracy

The accuracy of a classifier on a given test set is the percentage of test set examples that are correctly classified by the classifier

$$acc = \frac{\#\ correctly\ classified}{\#\ total\ examples}$$

Error rate is the opposite of accuracy

$$err = 1 - acc$$



Multiple runs: Usually a good idea to repeat the split-train-test procedure several times with different random splits and average the results

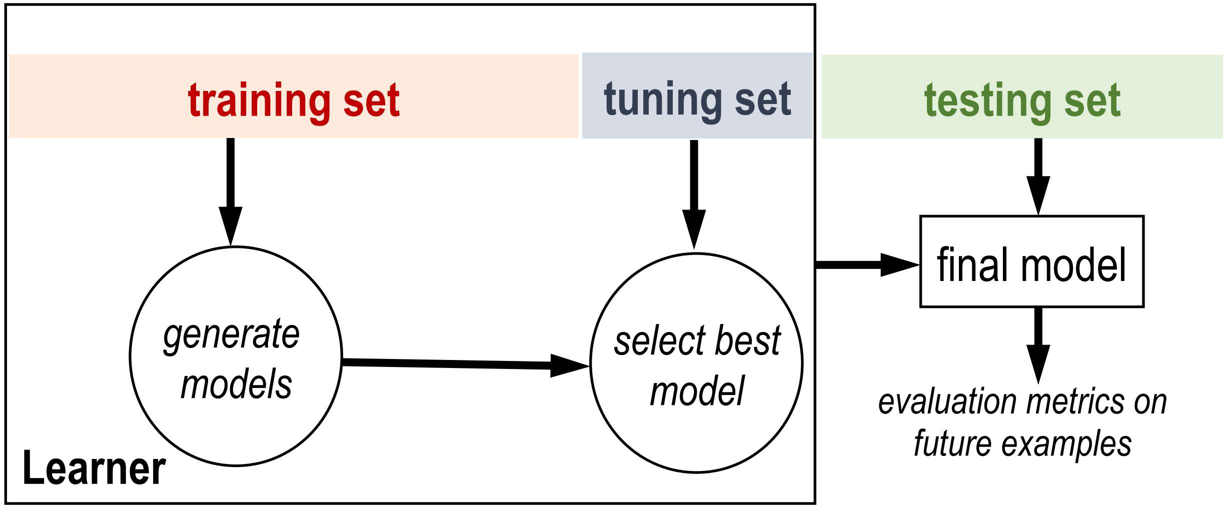
Methodology: The Tuning Set

Often, an ML system has to make several decisions (when to stop learning, select among alternative answers, identify best model parameters) that lead to very different models

- One wants the model that produces the highest accuracy on **future** examples ("overfitting avoidance")
- It is a **"cheat"** to look at the **test** set while still learning

Methodology using a Tuning or Validation Set:

- set aside part of the training set for **tuning** or **validation**
- measure performance on this tuning/validation set to **estimate future performance** for a particular algorithm setting
  - **SVMs**: *regularization parameter, kernel choice and kernel parameters*
  - **Decision Trees**: *tree depth, splitting criterion, pruning options*
  - **Neural Networks**: *learning rates, activation thresholds*
- use **best parameter settings**, train with **all training data** (**except test set**)
- estimate future performance on **testing set**



Methodology: Cross Validation

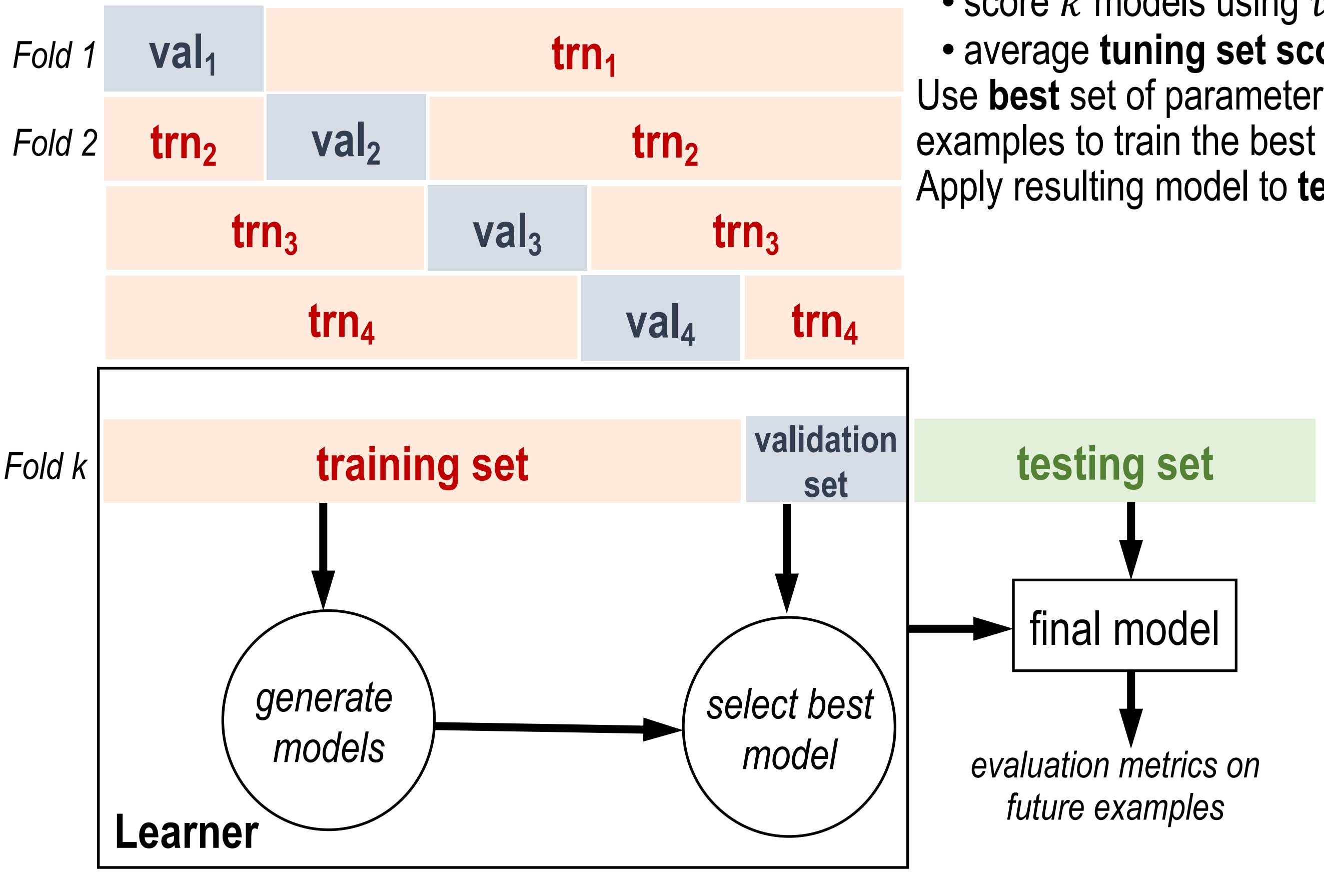
Using a **single** tuning set can be **unreliable** predictor, plus some data "**wasted**"; cross validation can help with **model selection**:

For each possible set of parameters, θ_p

- Divide training data into k **folds**
- **train** k models using $trn_k$with θ_p
- score k models using val_k
- average **tuning set score** over the k models

Use **best** set of parameters $\theta_*$and **all** **(train + tune)** examples to train the best model

Apply resulting model to **test set**



Evaluation: False Positives and False Negatives

Sometimes accuracy is simply not sufficient, especially for **imbalanced data sets**. Furthermore, different applications may have very different requirements.

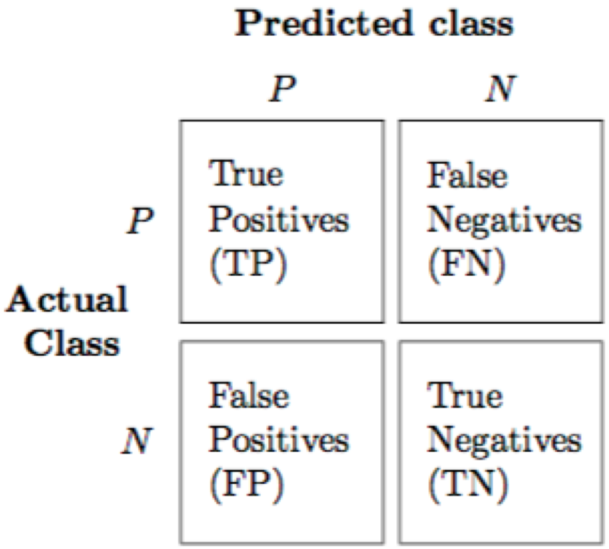
Case Study 1: Consider a **medical diagnosis application**, where the task is to **predict if a patient has cancer**. The data set contains $n_+ = 100$ (positive examples) and $n_- = 10{,}000$ (negative examples). If our classifier classified all training examples as negative examples (i.e., $f(\boldsymbol{x}) = -1$), then accuracy is $\frac{100}{10{,}100} \times 100 = 99.01\%$.

However, we have misdiagnosed 100 patients, who are false negatives. The cost associated with this misdiagnosis is very high, both monetarily and medically.

Case Study 2: Consider a **spam-filtering application**, where the task is to **predict incoming e-mail as spam** to maximize spam-blocking and minimize legitimate emails incorrectly flagged as spam. Of the 101 e-mail flagged as spam, 100 are legitimately spam but 1 is an e-mail informing you of a prestigious job offer. The accuracy is $> 99\%$.

However, we have flagged a very important e-mail as spam, which makes this misclassification a false positive. The cost associated with such a mistake is very high, personally.

Evaluation: The Confusion Matrix



		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

the **confusion matrix** (aka error matrix) is a specific visualizes the performance of a supervised classification algorithm

- illustrates how a model is performing in terms of **false positives** and **false negatives**
- gives more information than a single accuracy figure
- allows us to think about the cost of mistakes
- can be extended to any number of classes

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Misclassification\ Rate = \frac{FP + FN}{TP + FP + TN + FN}$$

$$True\ Positive\ Rate\,(sensitivity) = \frac{TP}{TP + FN}$$

$$True\ Negative\ Rate\,(specificity) = \frac{TN}{TN + FP}$$

Evaluation: Receiver-Operator Characteristic Curves

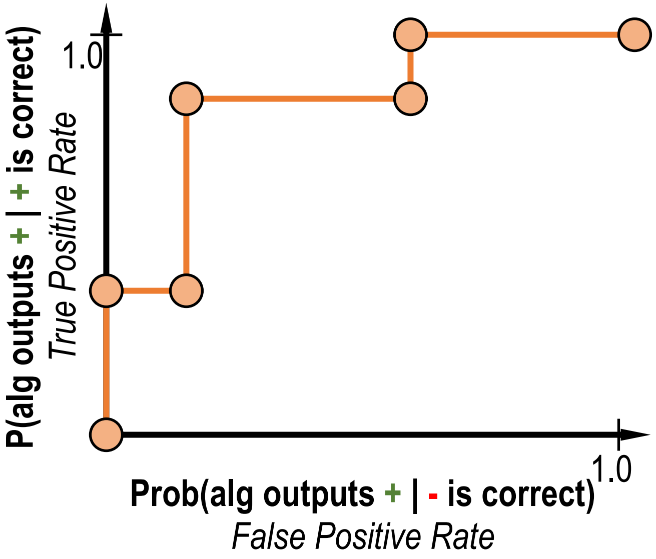
An **ROC curve** (receiver operating characteristic curve) is a graph showing the performance of a classification model at **all classification thresholds**.

- originally developed for radar research during WWII
- judging algorithms on accuracy alone may not be good enough when getting a positive example wrong **costs more** than getting a negative example wrong (**or vice versa**)
- **lowering** the classification **threshold** classifies **more** items as **positive**, thus increasing both False Positives and True Positives

Procedure to construct an ROC curve:

- **sort** predictions on test set
- locate a **threshold** between examples with opposite categories
- **compute** TPR & FPR for each threshold
- **connect** the dots

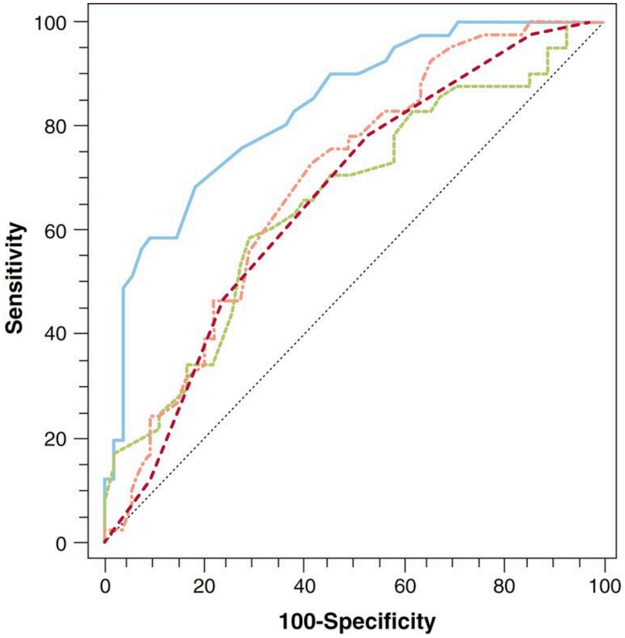
	Pred(y=+)	**True**	
example 9	0.99	+	
example 7	0.98	+	TPR = 2/5, FPR = 0/5
example 1	0.72	-	TPR = 2/5, FPR = 1/5
example 2	0.70	+	
example 6	0.65	+	TPR = 4/5, FPR = 1/5
example 10	0.51	-	
example 3	0.39	-	TPR = 4/5, FPR = 3/5
example 5	0.24	+	TPR = 5/5, FPR = 3/5
example 4	0.11	-	
example 8	0.01	-	TPR = 5/5, FPR = 5/5



Evaluation: Area Under the ROC Curve

Area under the ROC Curve (AUC) provides an aggregate measure of performance across all possible classification thresholds

- One way of interpreting AUC is as the **probability** that the model ranks a random **positive example** more **highly** than a random **negative example**
- can compare performance of different algorithms using AUC
- can use AUC/ROC to select a good threshold for classification in order to weight false positives and false negatives differently



Asymmetric Error Costs

Assume that $cost(FP) \neq cost(FN)$

You would like to **pick a threshold** that minimizes

$$E(cost) = cost(FP) \times Prob(FP) \times \#neg + cost(FN) \times Prob(FN) \times \#pos$$

Evaluation: Precision and Recall

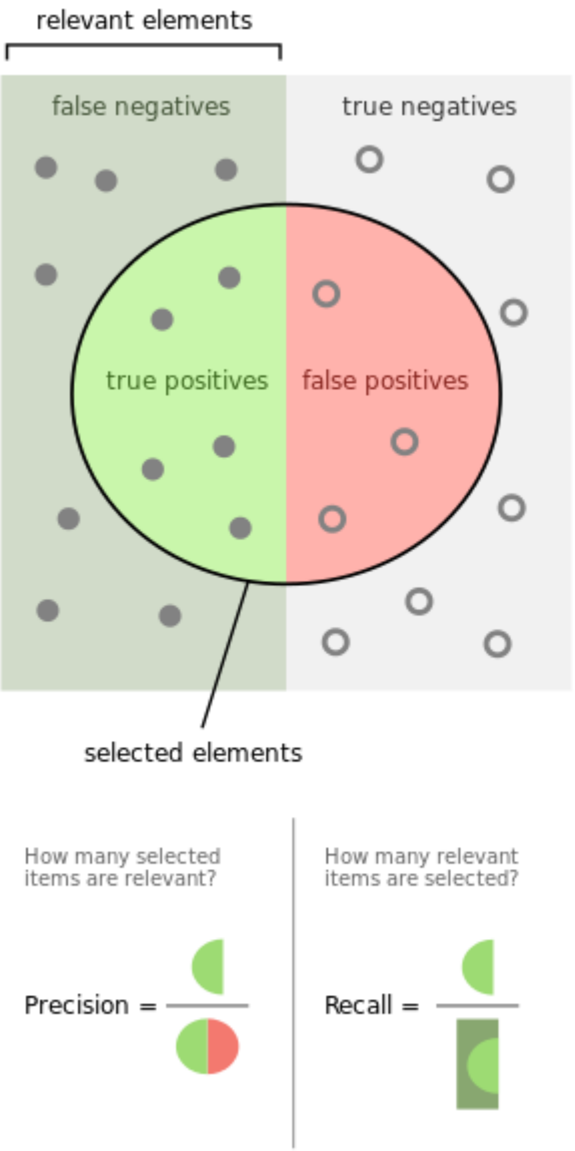
precision $= \dfrac{\#\, of\ relevant\ items\ retrieved}{total\ \#\ of\ items\ retrieved} = \dfrac{TP}{TP+FP}$

interpretation: Prob(is positive | called positive)

recall $= \dfrac{\#\, of\ relevant\ items\ retrieved}{total\ \#\ of\ items\ that\ exist} = \dfrac{TP}{TP+FN}$

interpretation: Prob(called positive | is positive)

Notice that the count of true negatives (TN) is not used in either formula; therefore you get **no credit for filtering out irrelevant items**



Case Study 1: For applications such as **medical diagnosis,** require **high recall to reduce false negatives**

Case Study 2: For applications such as **spam-filtering** and **recommendations systems**, require **high precision to reduce false positives**