

Evaluation

Evaluating Information Retrieval and Machine Learning systems

Evaluation

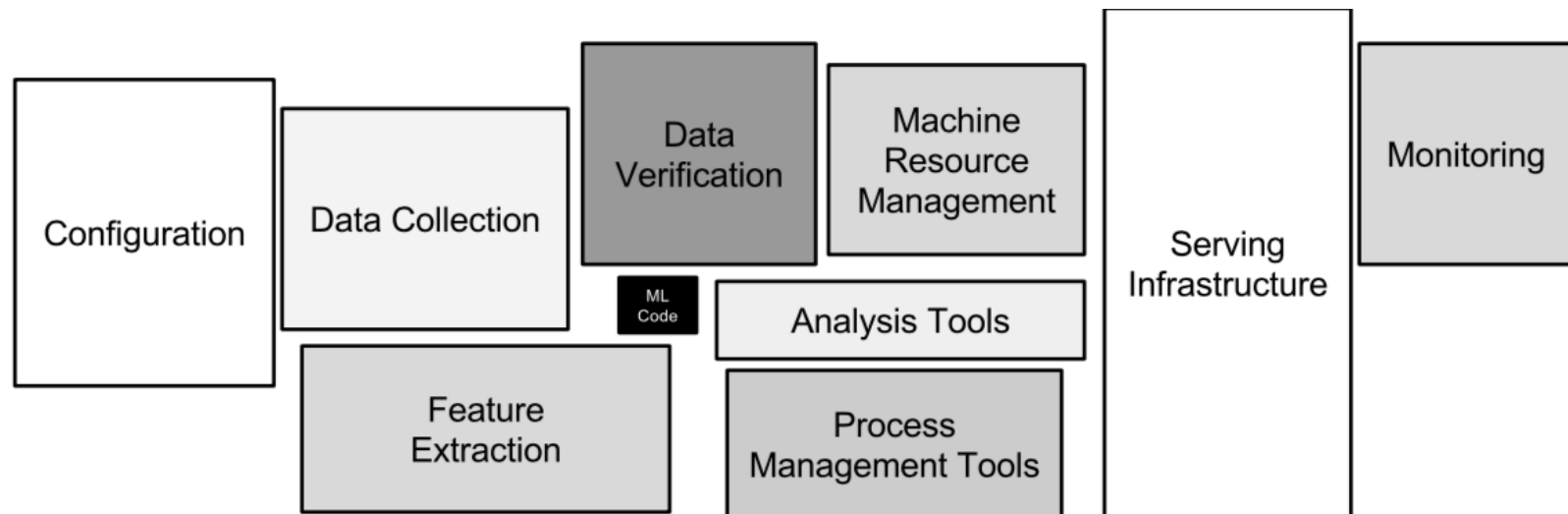
- To know whether we can *trust* our method or system, we need to evaluate it.
- If you cannot measure it, you cannot improve it.
- Choose between different possible techniques in a data-driven way.
- Convince others that your work is meaningful
 - Peers, leadership, clients, yourself(!)
- Keep evaluating relentlessly, adapt to changes

Designing IR/DM systems

- Just running your favourite algorithm is usually not a great way to start
- Consider the problem at large
 - Do you want to understand phenomena or do black box modelling?
 - How to define and measure success? Are there costs involved?
 - Do you have the right data? How can you make it better?
- Build prototypes early-on to evaluate the above.

Designing IR/DM systems

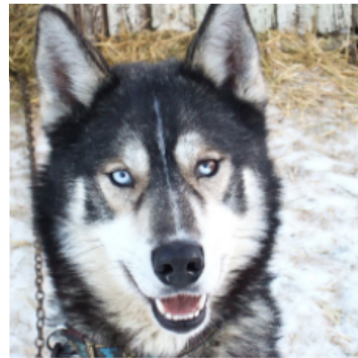
- Analyse your model's mistakes
 - Should you collect more, or additional data?
 - Should the task be reformulated?
 - Often a higher payoff than endless finetuning
- Technical debt: creation-maintenance trade-off
 - Very complex machine learning systems are hard/impossible to put into practice
 - See 'Machine Learning: The High Interest Credit Card of Technical Debt'



Only a small fraction of real-world ML systems is composed of the ML code

Real world evaluations

- Evaluate predictions, but also how outcomes improve *because of them*
- Feedback loops: predictions are fed into the inputs, e.g. as new data, invalidating models
- The signal your model found may just be an artifact of your biased data
 - See 'Why Should I Trust You?' by Marco Ribeiro et al.



(a) Husky classified as wolf



(b) Explanation

- Adversarial situations (e.g. spam filtering) can subvert your predictions
- Do A/B testing (or bandit testing) to evaluate algorithms in the wild

Levels of evaluation

- System-level: Response time, throughput, user retention,...
 - Time for indexing (nr. documents / hour)
 - Search time (latency as function of index size)
- Task-level: User *happiness*, satisfaction, understanding,...
 - Often subjective: novelty, required effort, presentation,...
 - Users find what they want quickly, return to search engine
 - Users find and buy products (e.g. time, percentage of users)
- Method-level: Correctness/effectiveness or predictions
 - Accuracy, precision, recall, ...
 - Cost functions

Difficulties

- How to perform evaluation?
 - Benchmark documents, queries
 - Labelled data vs. user actions
- Are predictions *useful*?
 - Do users act on the results (e.g. click)
 - Do users need to modify the query?
 - Satisfactory / unsatisfactory termination
- Good proxy: are predictions *relevant*?
 - Do they address the 'information need'?
 - Need to consider user preferences, time, place,...

Binary evaluation measures

- First, assume that we have labeled data (ground truth)
- We have positive (relevant) and a negative (irrelevant) predictions
- 2 different kind of errors:
 - False Positive (type I error): model predicts positive while the true label is negative
 - False Negative (type II error): model predicts negative while the true label is positive
- They are usually not equally important
 - Which side do you want to err on for a medical test? For a search engine?

Confusion matrices

- We can represent all predictions (correct and incorrect) in a confusion matrix
 - n by n array (n is the number of classes)
 - Rows correspond to true classes, columns to predicted classes
 - Each entry counts how often a sample that belongs to the class corresponding to the row was classified as the class corresponding to the column.
 - For binary classification, we label these true negative (TN), true positive (TP), false negative (FN), false positive (FP)

negative class	TN	FP
positive class	FN	TP
	predicted negative	predicted positive

Predictive accuracy

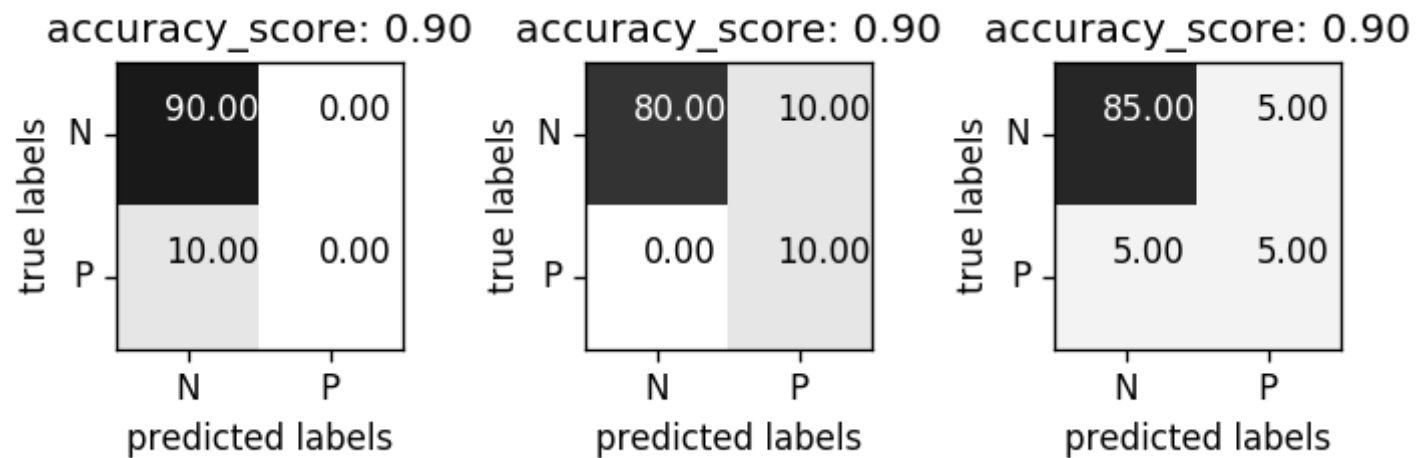
- Accuracy is one of the measures we can compute based on the confusion matrix:

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

- Almost never a good measure for information retrieval (or machine learning)

The problem with accuracy

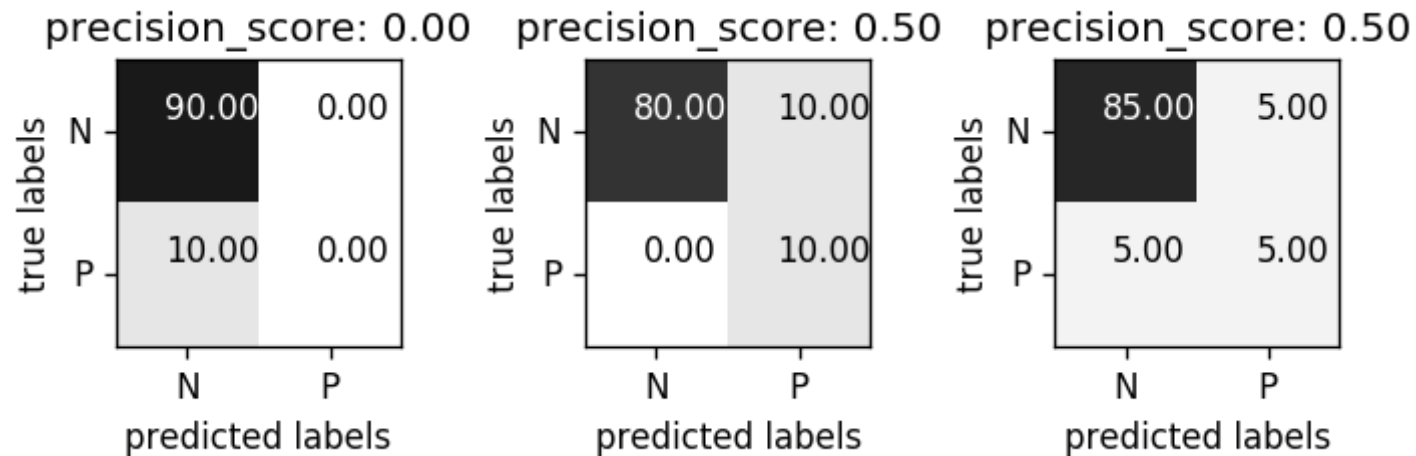
- A search engine that returns very few results can have high accuracy
 - People want to find something, have high tolerance for junk
- Imbalanced data: Many more negatives than positives (or other way around)
 - e.g. credit card fraud, medical test for rare disease, web searches,...
 - Is a 99.99% accuracy good enough?
- Often does not distinguish between very different models
 - Are these three models really equally good?



Precision is used when the goal is to limit FPs

- Clinical trials: you only want to test drugs that really work
- Search engines: you want to avoid bad search results

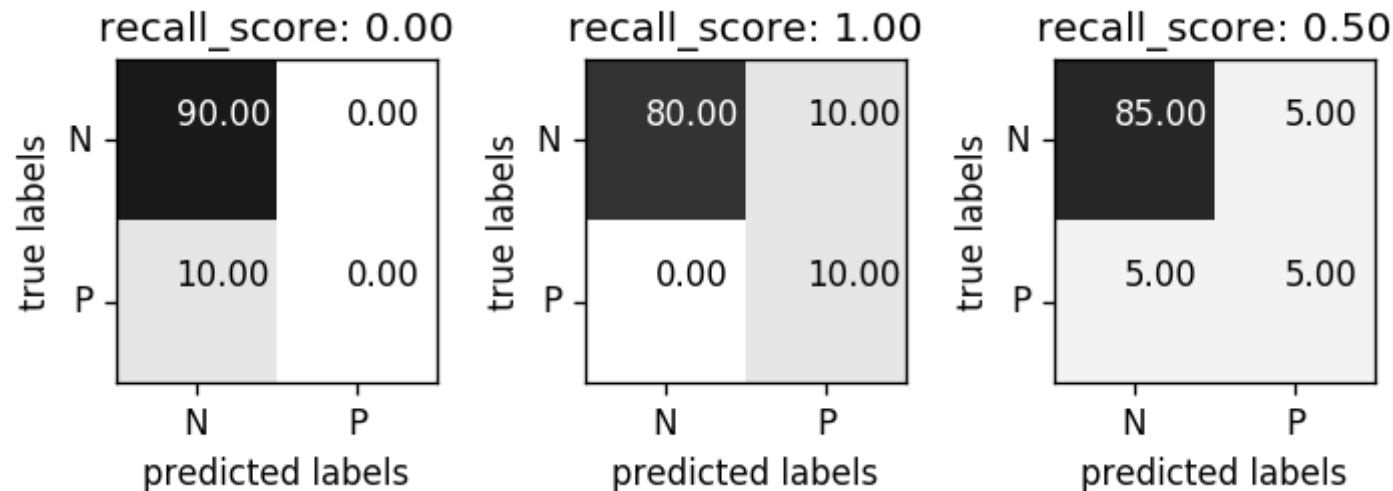
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$



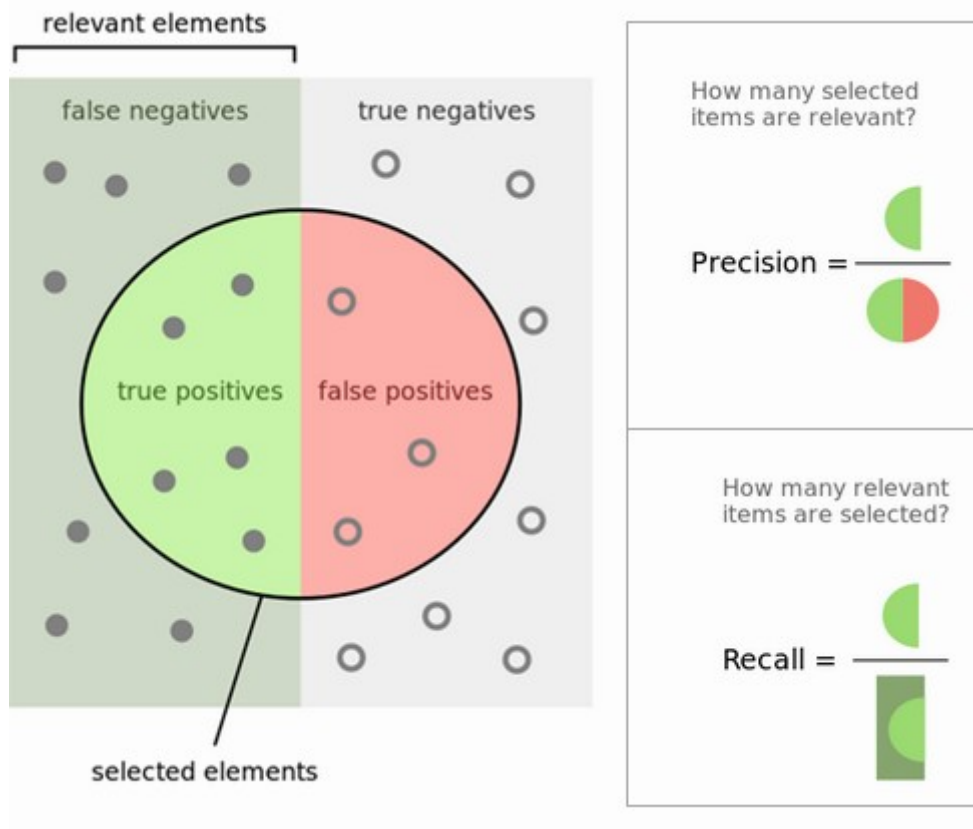
Recall is used when the goal is to limit FNs

- Cancer diagnosis: you don't want to miss a serious disease
- Search engines: You don't want to omit important hits
- Also know as sensitivity, hit rate, true positive rate (TPR)

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$



Precision vs Recall



- Arbitrarily high recall (but low precision) by returning almost all documents
- Arbitrarily high precision (but low recall) by returning almost no documents

Precision-recall difficulties in IR

- Do not take number of irrelevant documents (TN) into account
- Assumes that the corpus (index) is large and complete
 - Recall: undefined when there are no relevant documents
 - Precision: undefined when nothing is retrieved
- Total number of relevant items (TP) can be unknown
 - Take a representative sample from the database
 - Take union of TPs from multiple retrieval techniques
- Needs human labelling (relevant/ not relevant)
- Binary assessment (not gradual)

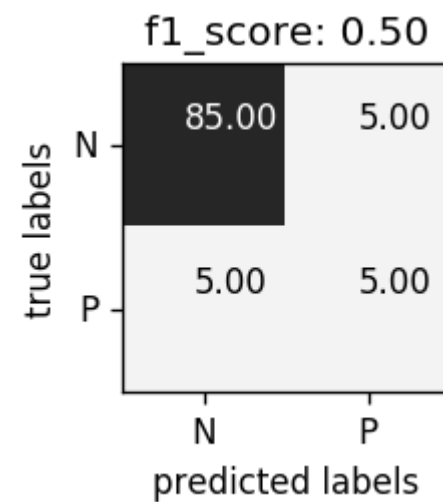
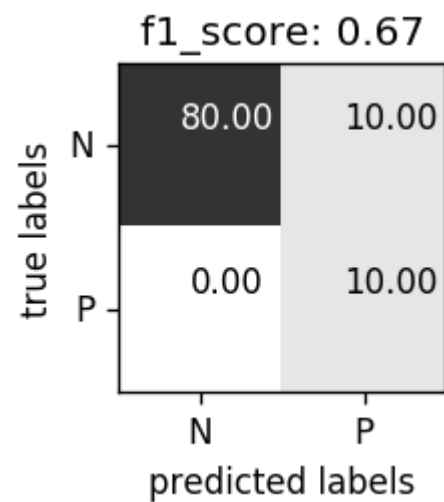
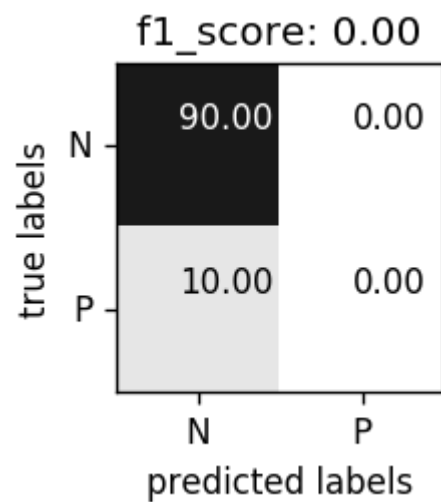
Fallout of FP rate: take number of TN into account

- More systems-oriented: maximal recall, minimal fallout

$$\text{Fallout} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

F1-score or *F1-measure*: trades off precision and recall

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$



E-measure (parameterized F-measure) put emphasis on either precision and recall

- $\beta > 1$: Emphasis on precision
- $\beta < 1$: Emphasis on recall

$$E = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

Classification measure Zoo

		True condition			
Total population		Condition positive	Condition negative	$\text{Prevalence} = \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	$\text{Accuracy (ACC)} = \frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive , Power	False positive , Type I error	$\text{Positive predictive value (PPV), Precision} = \frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	$\text{False discovery rate (FDR)} = \frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative , Type II error	True negative	$\text{False omission rate (FOR)} = \frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	$\text{Negative predictive value (NPV)} = \frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		$\text{True positive rate (TPR), Recall, Sensitivity, probability of detection} = \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	$\text{False positive rate (FPR), Fall-out, probability of false alarm} = \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	$\text{Positive likelihood ratio (LR+)} = \frac{\text{TPR}}{\text{FPR}}$	$\text{Diagnostic odds ratio (DOR)} = \frac{\text{LR+}}{\text{LR-}}$
		$\text{False negative rate (FNR), Miss rate} = \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	$\text{Specificity (SPC), Selectivity, True negative rate (TNR)} = \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	$\text{Negative likelihood ratio (LR-)} = \frac{\text{FNR}}{\text{TNR}}$	
				$F_1 \text{ score} = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$	

[https://en.wikipedia.org/wiki/Precision and recall](https://en.wikipedia.org/wiki/Precision_and_recall)
 ([https://en.wikipedia.org/wiki/Precision and recall](https://en.wikipedia.org/wiki/Precision_and_recall))"

Taking uncertainty into account

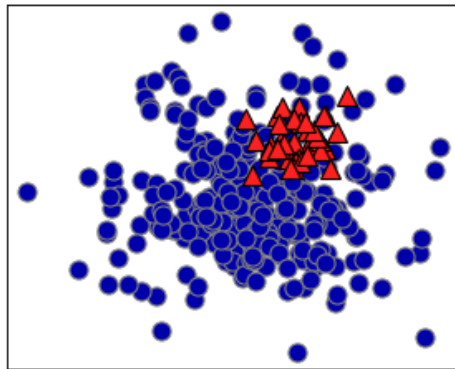
- Many classifiers and retrieval techniques can return a probability per class
 - Based on a `decision_function` (threshold 0)
 - e.g. distance to decision boundary (SVM, linear models)
 - Based on probability (threshold 0.5)
 - Probabilistic (Bayesian) techniques
 - Others (e.g. voting in kNN or ensembling techniques)
- Threshold calibration
 - In some tasks, FP may be much worse than a FN
 - Choose threshold to get fewer FPs, more FNs

Example

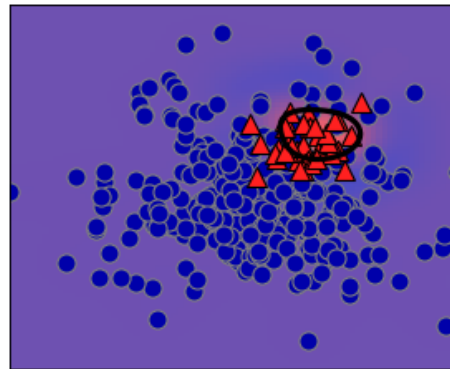
- Avoid misclassifying a positive (red) point
- Points within decision boundary (black line) are classified positive
- Lowering the decision threshold (bottom figure): fewer FN, more FP

decision_threshold

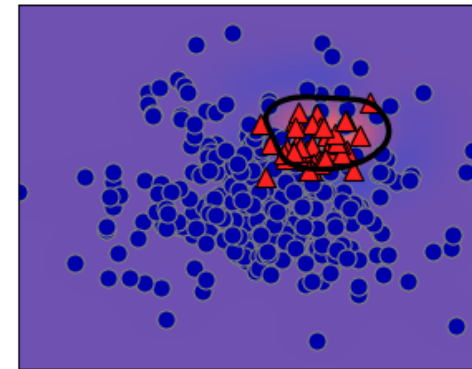
training data



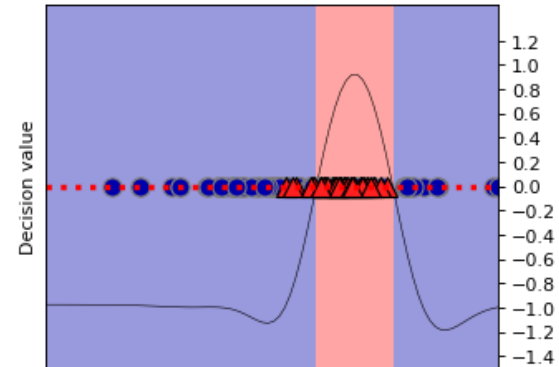
decision with threshold 0



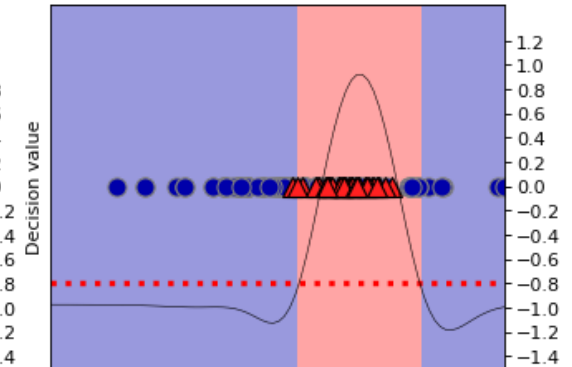
decision with threshold -0.8



Cross-section with threshold 0



Cross-section with threshold -0.8



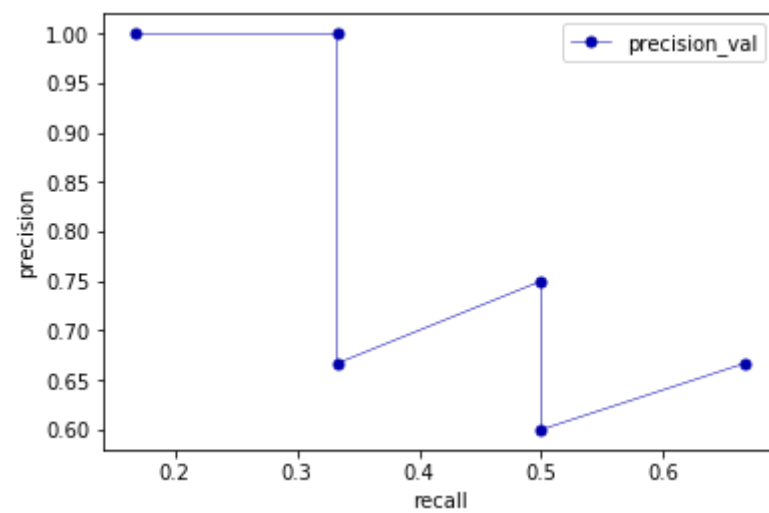
Precision-Recall curves

- The best threshold depends on your application, should be driven by real-world goals.
- You can have arbitrary high recall, but you often want reasonable precision, too.
- It is not clear beforehand where the optimale trade-off (or *operating point*) will be, so it is useful to look at all possible thresholds
- Plotting precision against recall for all thresholds yields a **precision-recall curve**
 - Order predictions by decision function / probability / relevance score
 - Increase threshold gradually, plot all precision and recall values

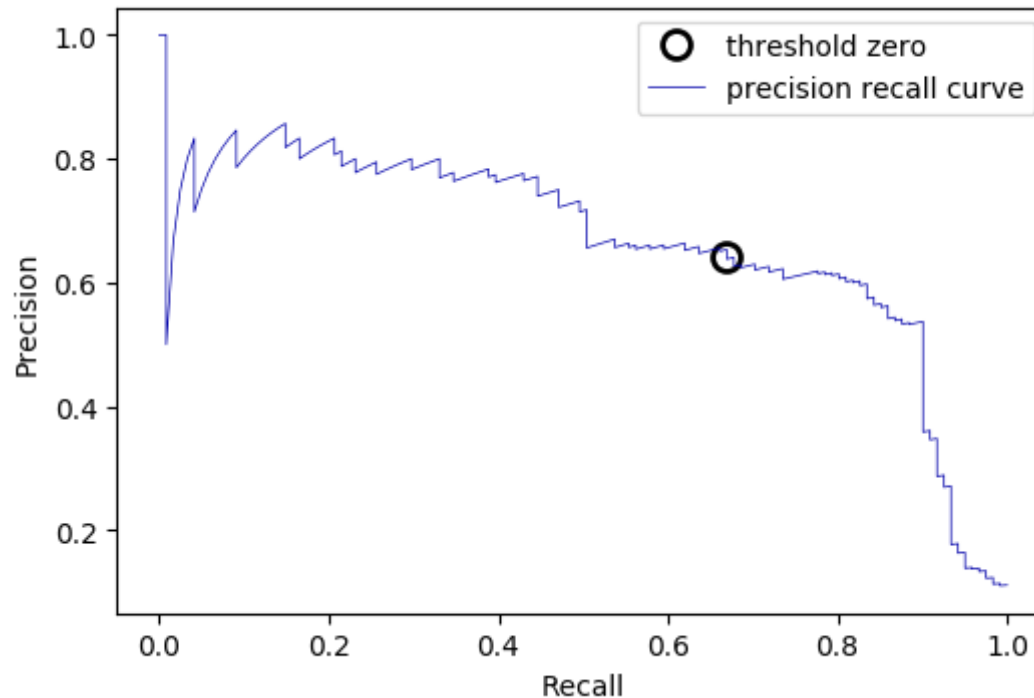
Example

Out[40]:

	n	id	prediction	truth	recall	precision
0	1	588	0.90	1	$1/6=0.167$	$1/1=1$
1	1	589	0.88	1	$2/6=0.333$	$2/2=1$
2	1	576	0.85	0	$2/6=0.333$	$2/3=0.667$
3	1	590	0.84	1	$3/6=0.5$	$3/4=0.75$
4	1	986	0.82	0	$3/6=0.5$	$3/5=0.6$
5	1	592	0.79	1	$4/6=0.667$	$4/6=0.667$

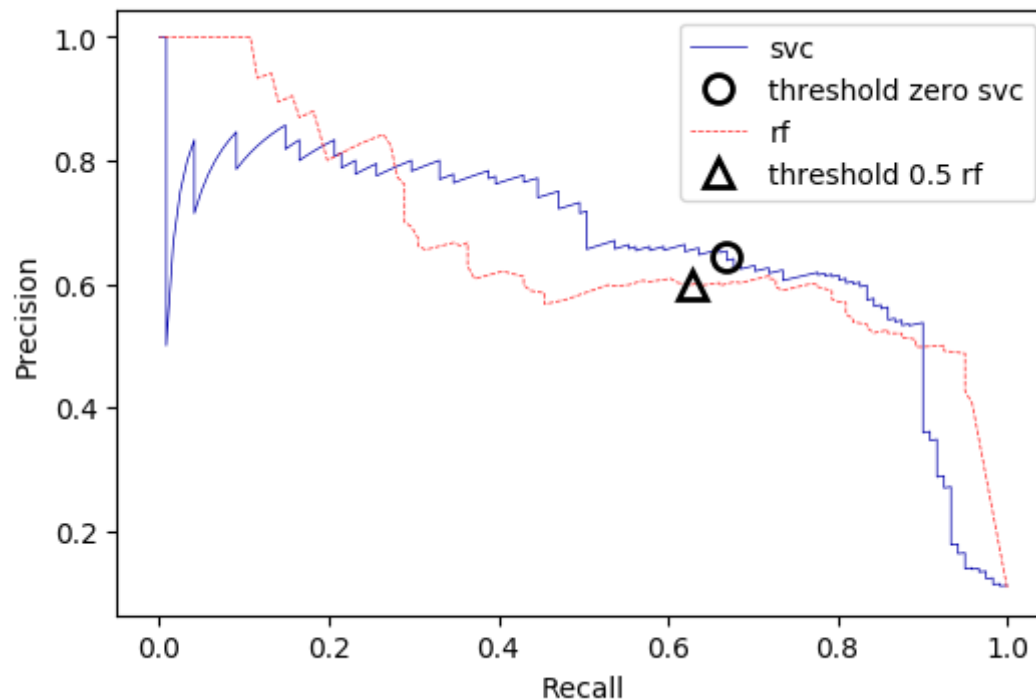


- The default tradeoff (chosen by the `predict` method) is shown as *threshold zero*.
 - Higher threshold, more precision (move left)
 - Lower threshold, more recall (move right)
- The closer the curve stays to the upper-right corner, the better
- Calibration: it is possible to still get a precision of 0.5 with high recall



Comparing models / systems

- Different techniques work best in different parts of the curve (at different operating points)
- RandomForest (in red) performs better at the extremes, SVM better in center
- The area under the precision-recall curve (AUPRC) is sometimes used as a general evaluation measure



Precision-recall across queries

- Precision and recall are computed *per query*
- Compute average over a set of queries
- Take average precision-recall curve to calibrate thresholds
- Average precision at fixed recall level r
 - N_q queries
 - $P_i(r)$: precision at recall level r for query i

$$\bar{P}(r) = \sum_{i=1}^{N_q} \frac{P_i(r)}{N_q}$$

Macro- vs Micro averaging

- Macro-averaging: compute average accross queries:
 - Use when you care about each query equally much

$$\frac{\sum_{i=0}^n score_i}{n}$$

- Micro-averaging: compute total number of FP, FN, TP over all queries, then compute scores using these counts:
 - Use when you want to do equally well on hard queries

$$recall = \frac{\sum_{i=0}^n TP_i}{\sum_{i=0}^n TP_i + \sum_{i=0}^n FN_i}$$

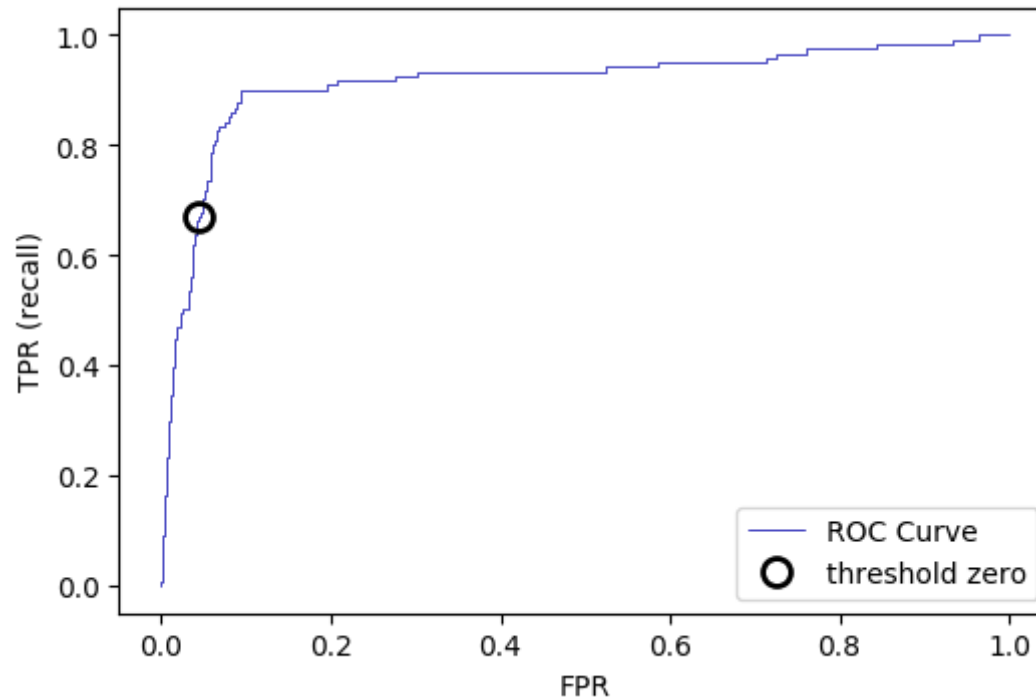
Receiver Operating Characteristics (ROC) and AUC

- There is another trade-off between recall (true positive rate, TPR) and fallout (false positive rate, FPR).
- The 2D space created by TPR and FPR is called the Receiver Operating Characteristics (ROC) space
- A model will be at one point in this ROC space

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

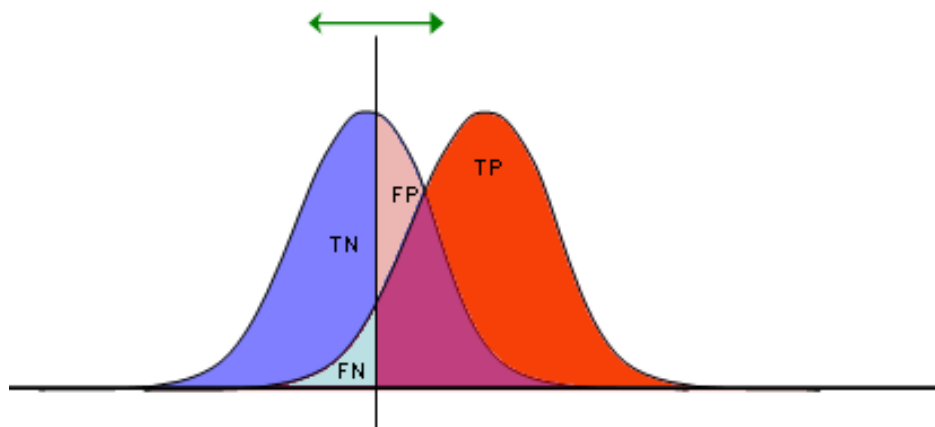
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

- Varying the decision threshold yields the ROC curve
 - Lower threshold, more recall/TPR, move right
 - High threshold, fewer FPs, move left
- Ideal is close to the top left: high recall, low FPR
- Inspect the curve to find the preferred calibration
 - Here, we can get much higher recall with slightly worse FPR

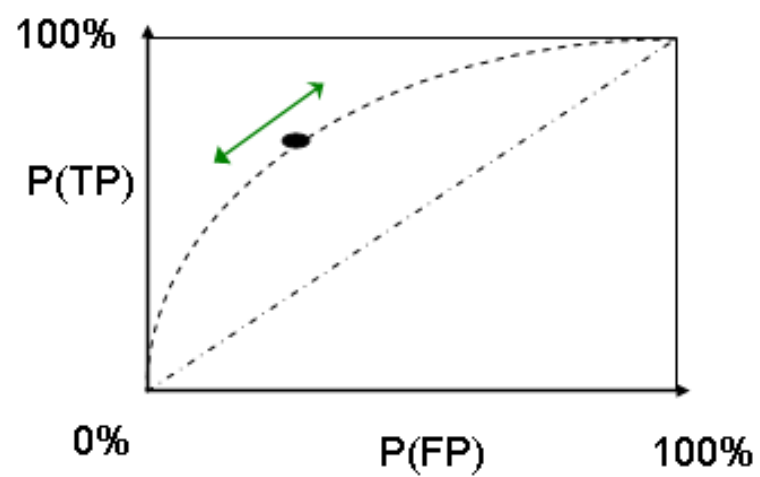


Visualization

- Consider distributions of all negative / positive points
- Hopefully, positive points have higher predicted relevance/probabilities
 - In random predictions the distributions would overlap.
- All points with a predicted probability higher than the threshold are predicted positive, others negative
- As we increase the threshold, we'll get fewer FPs, more FNs. We move from right to left along the ROC curve.

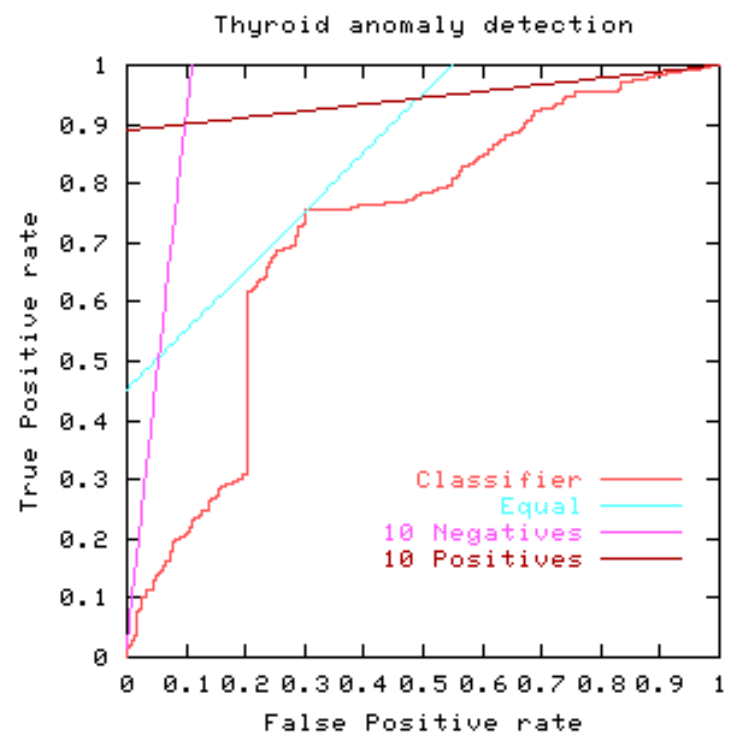


TP	FP
FN	TN
1	1



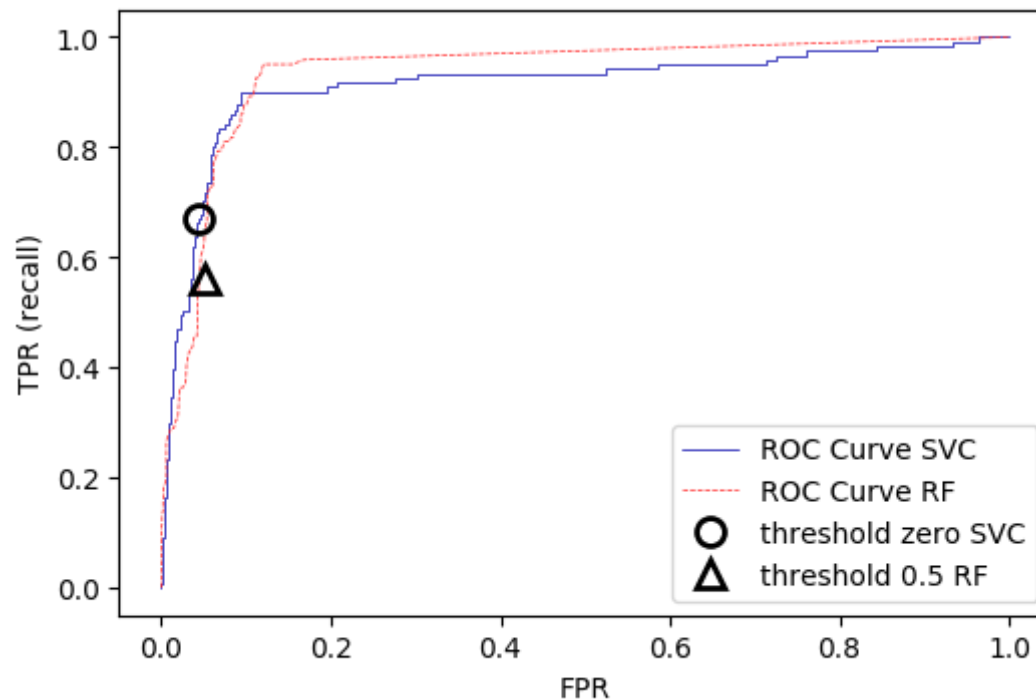
ROC Isometrics

- Different *costs* can be involved for FP and FN
- This yields different *isometrics* (lines of equal cost) in ROC space
- The optimal threshold is the point on the ROC curve where the cost is minimal
 - If a FP and FN are weighed equally, cost lines follow the diagonal (blue line)
 - If a FP is 10 times worse than a FN: pink line
 - If a FN is 10 times worse than a FP: red line



Model selection

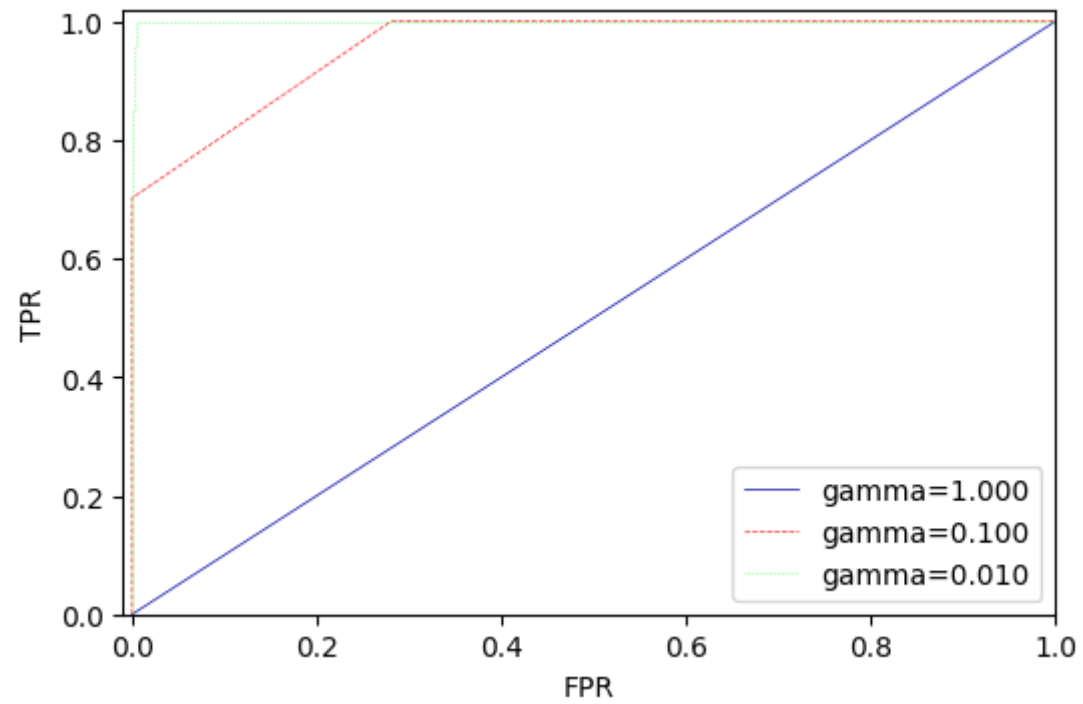
- Again, we can compare multiple models by looking at the ROC curves
- We can calibrate the threshold depending on whether we need high recall or low FPR
- We can select between algorithms (or hyperparameters) depending on the involved costs.
- We can use the area under the ROC curve as an aggregated score



Imbalanced classes

- AUC is popular because it is insensitive to class imbalance
 - Random guessing always yields $TPR=FPR$
 - All points are on the diagonal line, hence an AUC of 0.5
- Example:
 - 3 models (SVM with different gamma values), ACC is the same, AUC not
 - If we optimize for ACC, our model could be just random guessing

gamma = 1.000	accuracy = 0.90	AUC = 0.5000
gamma = 0.100	accuracy = 0.90	AUC = 0.9582
gamma = 0.010	accuracy = 0.90	AUC = 0.9995




Rank-based measures











For ranked results (e.g. search engine)


- Precision@K: precision after K docs are seen (e.g. P@5 or P@10)
 - Users prefer relevant documents at the top
- R-precision: precision at position of last relevant document
- MAP (Mean Average Precision)
 - Mean of precisions at each relevant document
 - Use precision 0 if document not retrieved
- MRR (Mean Reciprocal Rank)
 - Per query, find location of first relevant document
 - When interested in 'first good answer'

MAP: Example











 = relevant documents for query 1

Ranking #1

										
Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5

 = relevant documents for query 2

Ranking #2

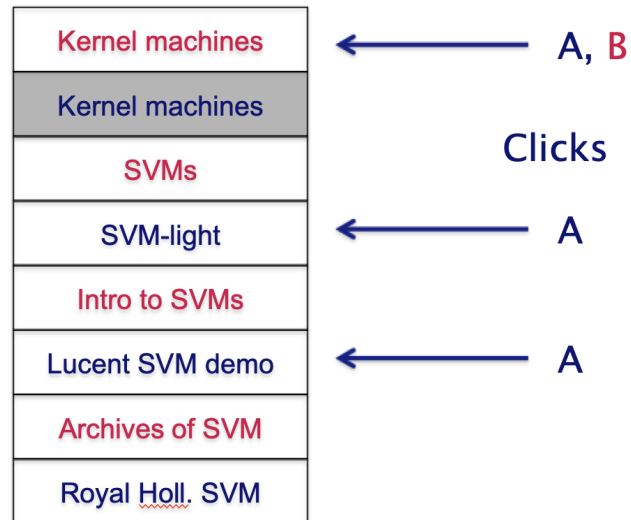
										
Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3

- Average precision query 1: $(1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$
- Average precision query 2: $(0.5 + 0.4 + 0.43)/3 = 0.44$
- MAP: $(0.62 + 0.44)/2 = 0.53$

Counting clicks: Interleaved ranking

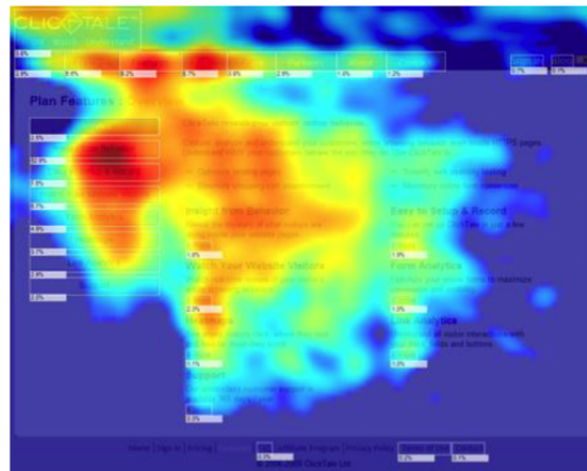
- Compute rankings with different methods (red and blue)
- Interleave the two rankings (drop duplicates), count clicks
- Example: query for 'SVM'

Ranking A: 3
Ranking B: 1



Interpreting clicks

- Clicking an item means that higher, non-clicked items are not relevant
 - Often user-specific (preference learning)
- Click prediction: build a click model that predicts whether a user will click the item
 - Use as additional signal to build final ranking (learning to rank)
- Visibility models:
 - Higher items receive more user attention
 - Can be measured via surveys, mouse/eye tracking



Discounted Cumulative gain

- Assume that highly relevant docs are more useful than marginally relevant ones
- Assume that top-ranked relevant documents are more useful to the user
- *Gain* of examining document is discounted (e.g. by $1/\log(\text{rank})$)
- *Discounted Cumulative gain*: sum of document relevances weighted by discounted gain

$$DCG = r_1 + \frac{r_2}{\log_2(2)} + \frac{r_3}{\log_2(3)} + \dots + \frac{r_n}{\log_2(n)}$$

- Normalized DCG: normalized across queries

Designing IR/DM systems

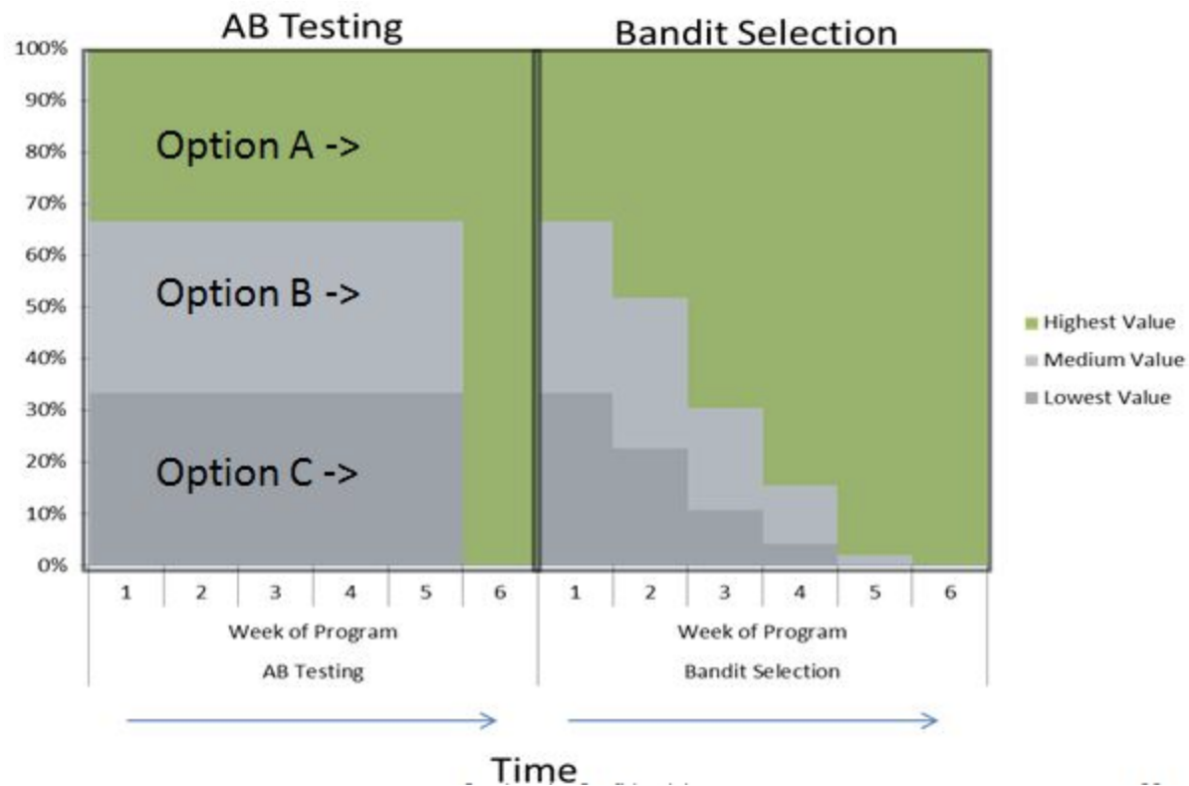
- Use evaluations constantly to select/finetune techniques
 - Does removing or adding features help?
 - Does additional preprocessing help?
 - What are the best models and hyperparameters?
- Measure variance: difference in performance across queries
 - Often the variance across queries is much larger than variance across methods!

Benchmarks

- To evaluate IR/DM systems, we often create a benchmark:
 - Set of standard documents and queries/topics
 - List of relevant documents for each queries
 - E.g. TREC collection
- Expert labels
- Naturally occurring labels:
 - Hashtags, emoticons,...
 - Clicks, bookmarks, downloads
 - ...

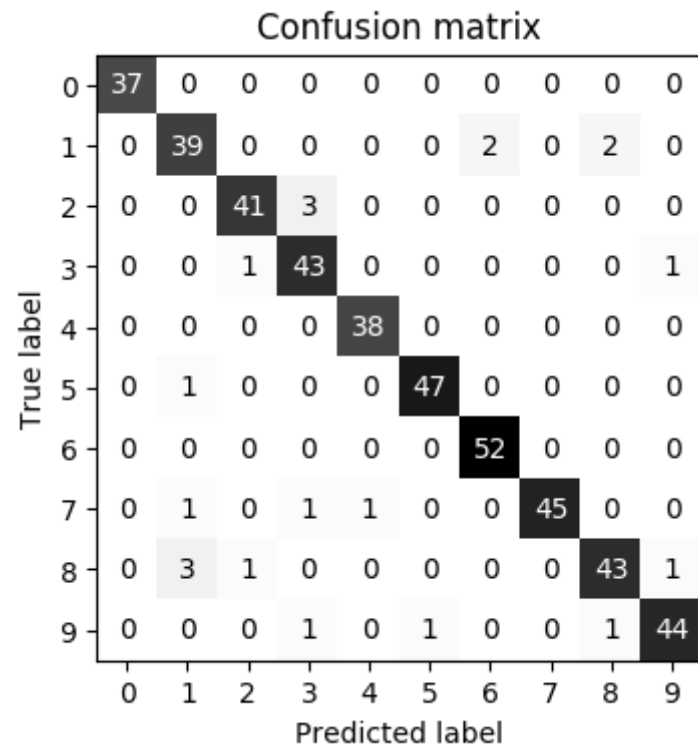
A/B and bandit testing

- Test a single innovations
- Have most users use the old system, divert small group to new system
- Evaluate and compare performance
- Bandit testing: smaller time intervals, direct more users to currently winning system



Multi-class classification

- Multiclass metrics are derived from binary metrics, averaged over all classes
- Example: handwritten digit recognition (10 classes)



Different ways to compute average

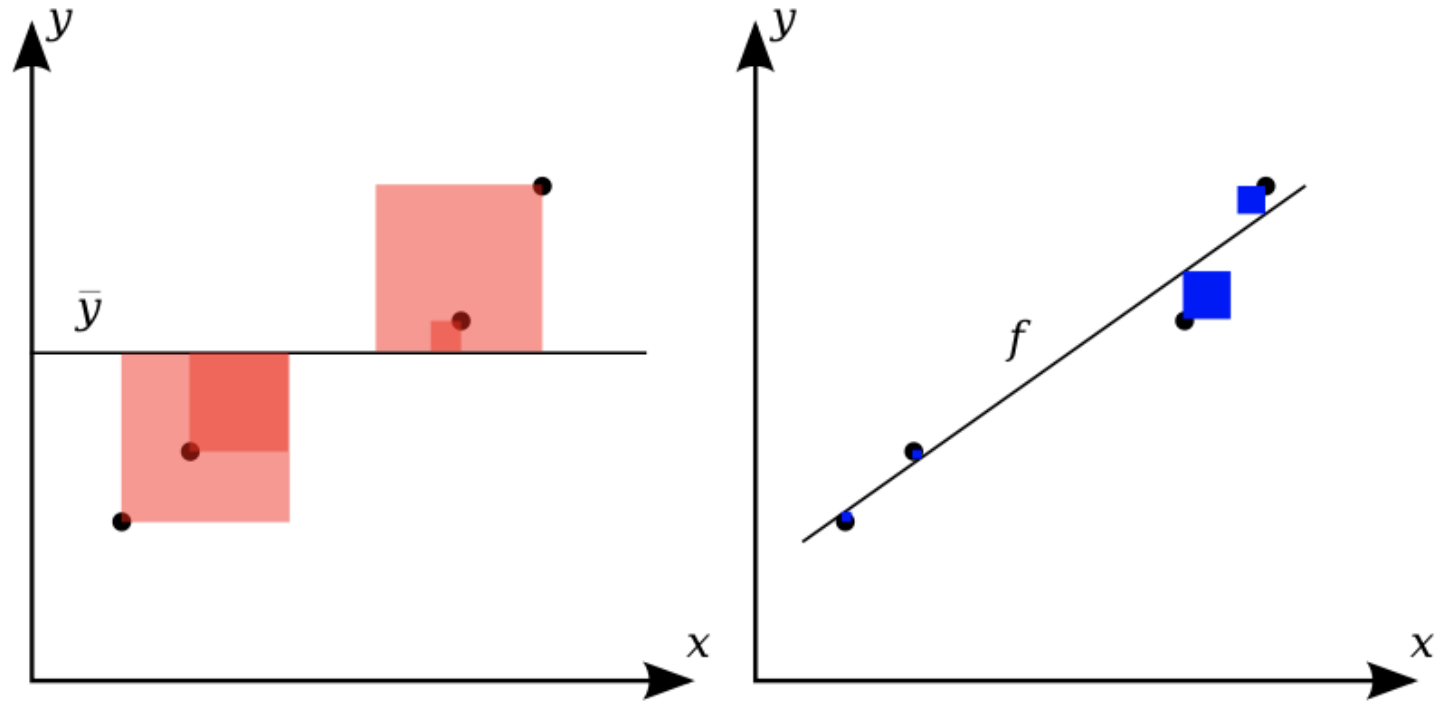
- macro-averaging: computes unweighted per-class scores: $\frac{\sum_{i=0}^n score_i}{n}$
 - Use when you care about each class equally much
- weighted averaging: scores are weighted by the relative size of the classes (support): $\frac{\sum_{i=0}^n score_i weight_i}{n}$
 - Use when data is imbalanced
- micro-averaging: computes total number of FP, FN, TP over all classes, then computes scores using these counts: $recall = \frac{\sum_{i=0}^n TP_i}{\sum_{i=0}^n TP_i + \sum_{i=0}^n FN_i}$
 - Use when you care about each sample equally much

Regression metrics

Most commonly used are

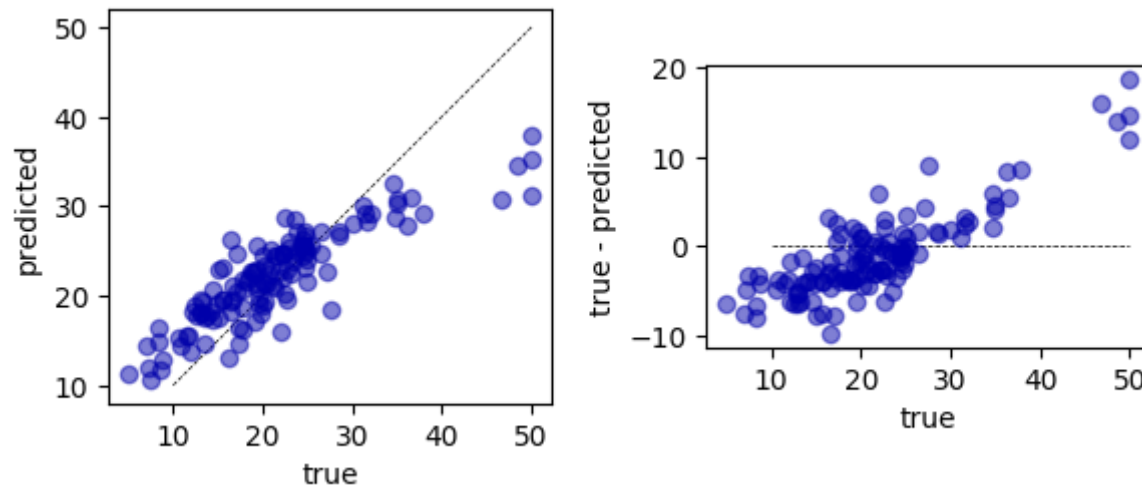
- (root) mean squared error: $\frac{\sum_i (y_{pred_i} - y_{actual_i})^2}{n}$
- mean absolute error: $\frac{\sum_i |y_{pred_i} - y_{actual_i}|}{n}$
 - Less sensitive to outliers and large errors
- R squared (r2): $1 - \frac{\sum_i (y_{pred_i} - y_{actual_i})^2}{\sum_i (y_{mean} - y_{actual_i})^2}$
 - Ratio of variation explained by the model / total variation
 - Between 0 and 1, but *negative* if the model is worse than just predicting the mean
 - Easier to interpret (higher is better).

- R squared: 1 - ratio of $\sum_i (y_{pred_i} - y_{actual_i})^2$ (blue) and $\sum_i (y_{mean} - y_{actual_i})^2$ (red)



Visualizing errors

- Prediction plot (left): predicted vs actual target values
- Residual plot (right): residuals vs actual target values
 - Over- and underpredictions can be given different costs



Final thoughts

- Cost-sensitive classification
 - *Cost matrix*: a confusion matrix with a costs associated to every possible type of error
 - Some algorithms allow optimizing on these costs instead of their usual loss function
 - Meta-cost: builds ensemble of models by relabeling training sets to match a given cost matrix
 - Black-box: can make any algorithm cost sensitive (but slower and less accurate)

More metrics

- There are many more metrics to choose from
 - Cohen's Kappa: accuracy, taking into account the possibility of predicting the right class by chance
 - 1: perfect prediction, 0: random prediction, negative: worse than random
 - With p_0 = accuracy, and p_e = accuracy of random classifier:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

- Balanced accuracy: accuracy where each sample is weighted according to the inverse prevalence of its true class
 - Identical to macro-averaged recall
- Matthews correlation coefficient: another measure that can be used on imbalanced data
 - 1: perfect prediction, 0: random prediction, -1: inverse prediction

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}}$$

Evaluating machine learning techniques

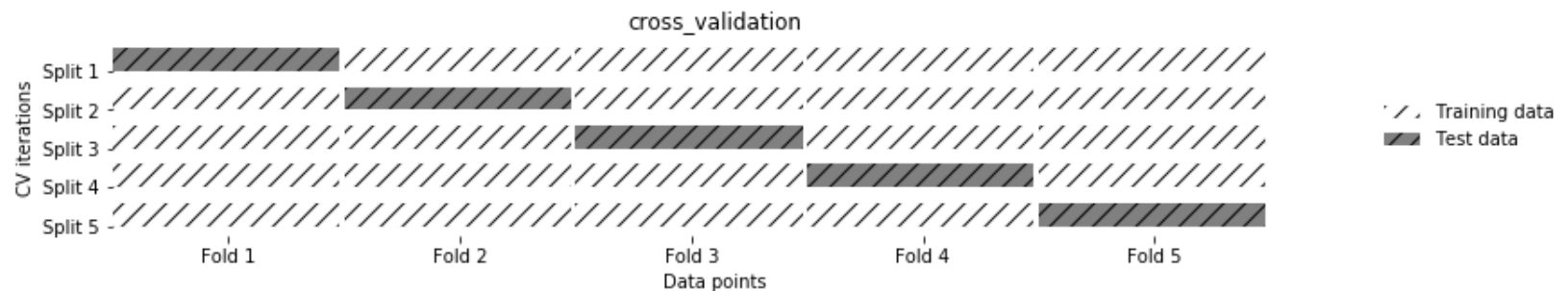
The holdout (simple train-test split)

- Split data into training and test set (75%-25%)
 - In sklearn we split in training and test predictors (X_{train} , X_{test}) and labels (y_{train} , y_{test})
- Train (fit) a model on the training data
- Score a model on the test data (comparing predicted and true labels)
 - We are interested in how well the model *generalizes* to new (test) data

- Why 75%? Are there better ways to split?
- What if one random split yields different models (and scores) than another?
- What if all examples of one class all end up in the training/test set?

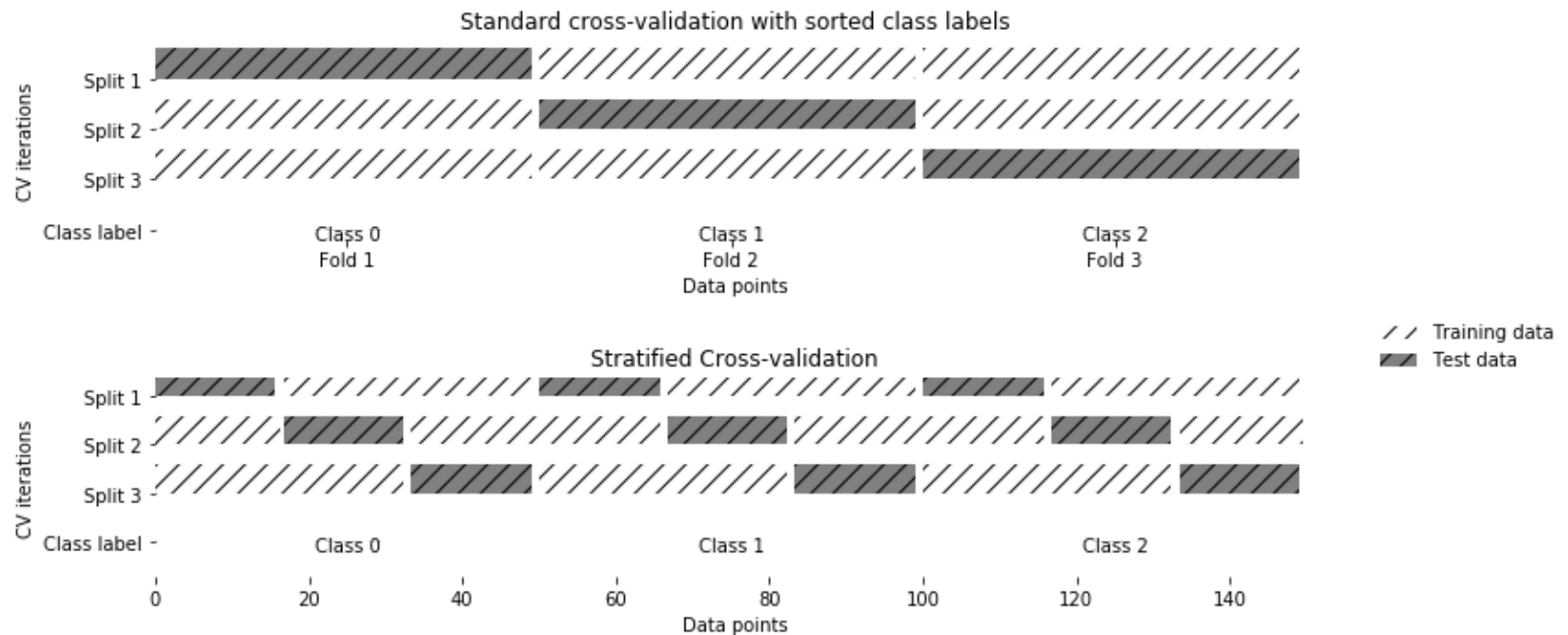
Cross-validation

- More stable, thorough way to estimate generalization performance
- *k-fold cross-validation* (CV): split (randomized) data into k equal-sized parts, called *folds*
 - First, fold 1 is the test set, and folds 2-5 comprise the training set
 - Then, fold 2 is the test set, folds 1,3,4,5 comprise the training set
 - Compute k evaluation scores, aggregate afterwards (e.g. take the mean)



Stratified K-Fold cross-validation

- If the data is *unbalanced*, some classes have many fewer samples
- Likely that some classes are not present in the test set
- Stratification: *proportions* between classes are conserved in each fold
 - Order examples per class
 - Separate the samples of each class in k sets (strata)
 - Combine corresponding strata into folds



Leave-One-Out cross-validation

- k fold cross-validation with k equal to the number of samples
- Completely unbiased (in terms of data splits), but computationally expensive
- But: generalizes *less* well towards unseen data
 - The training sets are correlated (overlap heavily)
 - Overfits on the data used for (the entire) evaluation
 - A different sample of the data can yield different results
- Recommended only for small datasets

Choosing a performance estimation procedure

No strict rules, only guidelines:

- Always use stratification for classification
- Use holdout for very large datasets (e.g. >1.000.000 examples)
 - Or when learners don't always converge (e.g. deep learning)
- Choose k depending on dataset size and resources
 - Use leave-one-out for small datasets (e.g. <500 examples)
 - Use cross-validation otherwise
 - Most popular (and theoretically sound): 10-fold CV
 - Literature suggests 5x2-fold CV is better
- Use grouping or leave-one-subject-out for grouped data

Bias-Variance decomposition

- When we repeat evaluation procedures multiple times, we can distinguish two sources of errors:
 - Bias: systematic error (independent of the training sample).
The classifier always gets certain points wrong
 - Variance: error due to variability of the model with respect to the training sample. The classifier predicts some points accurately on some training sets, but inaccurately on others.
- There is also an intrinsic (noise) error, but there's nothing we can do against that.
- Bias is associated with underfitting, and variance with overfitting
- Bias-variance trade-off: you can often exchange variance for bias through regularization (and vice versa)
 - The challenge is to find the right trade-off (minimizing total error)
- Useful to understand how to tune or adapt learning algorithm

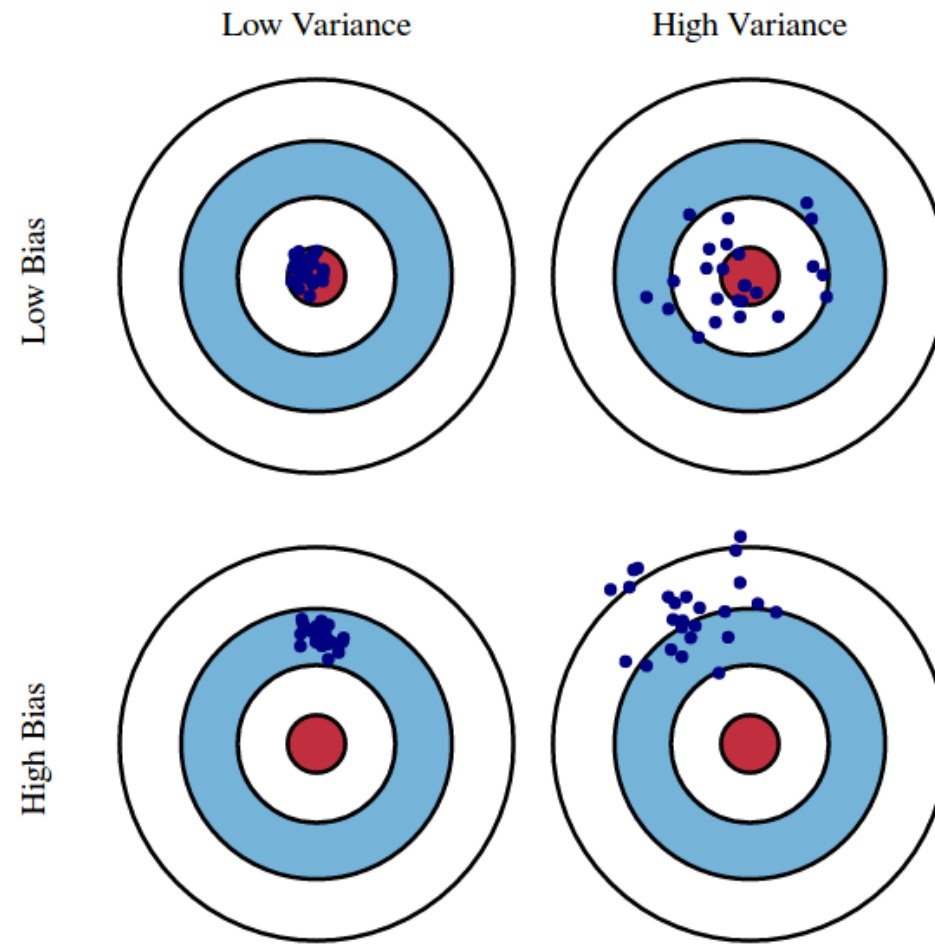
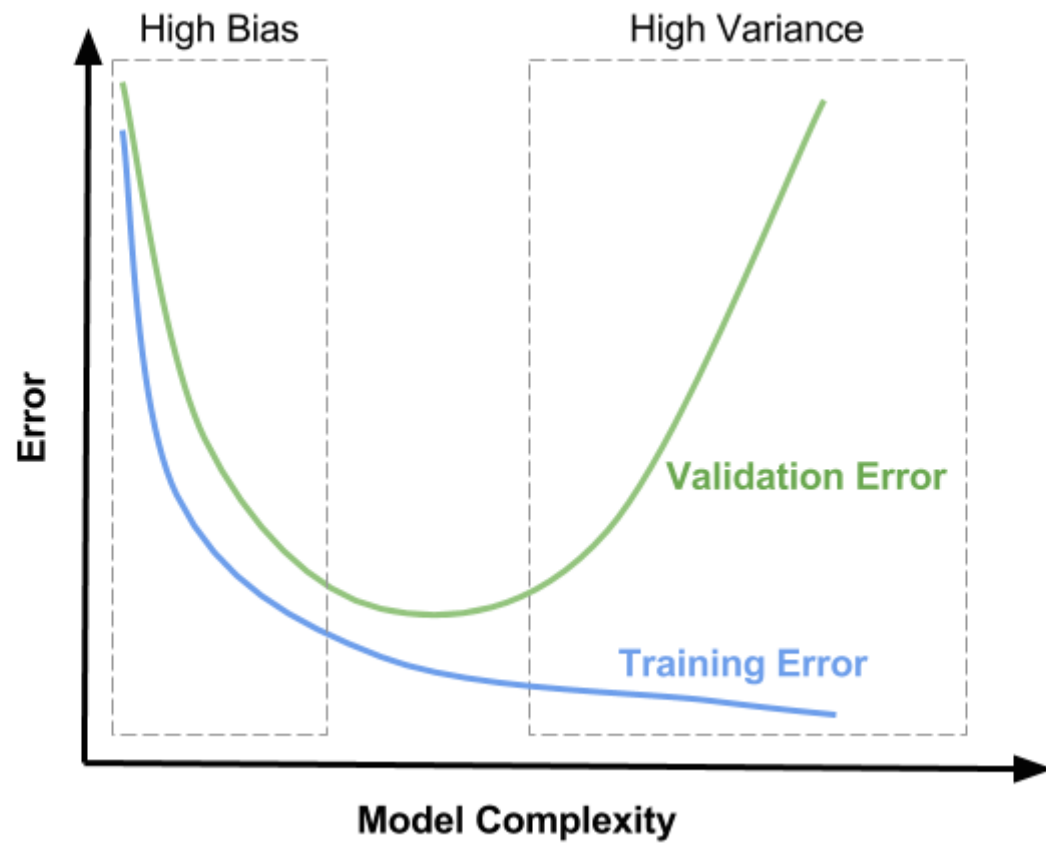


Fig. 1 Graphical illustration of bias and variance.

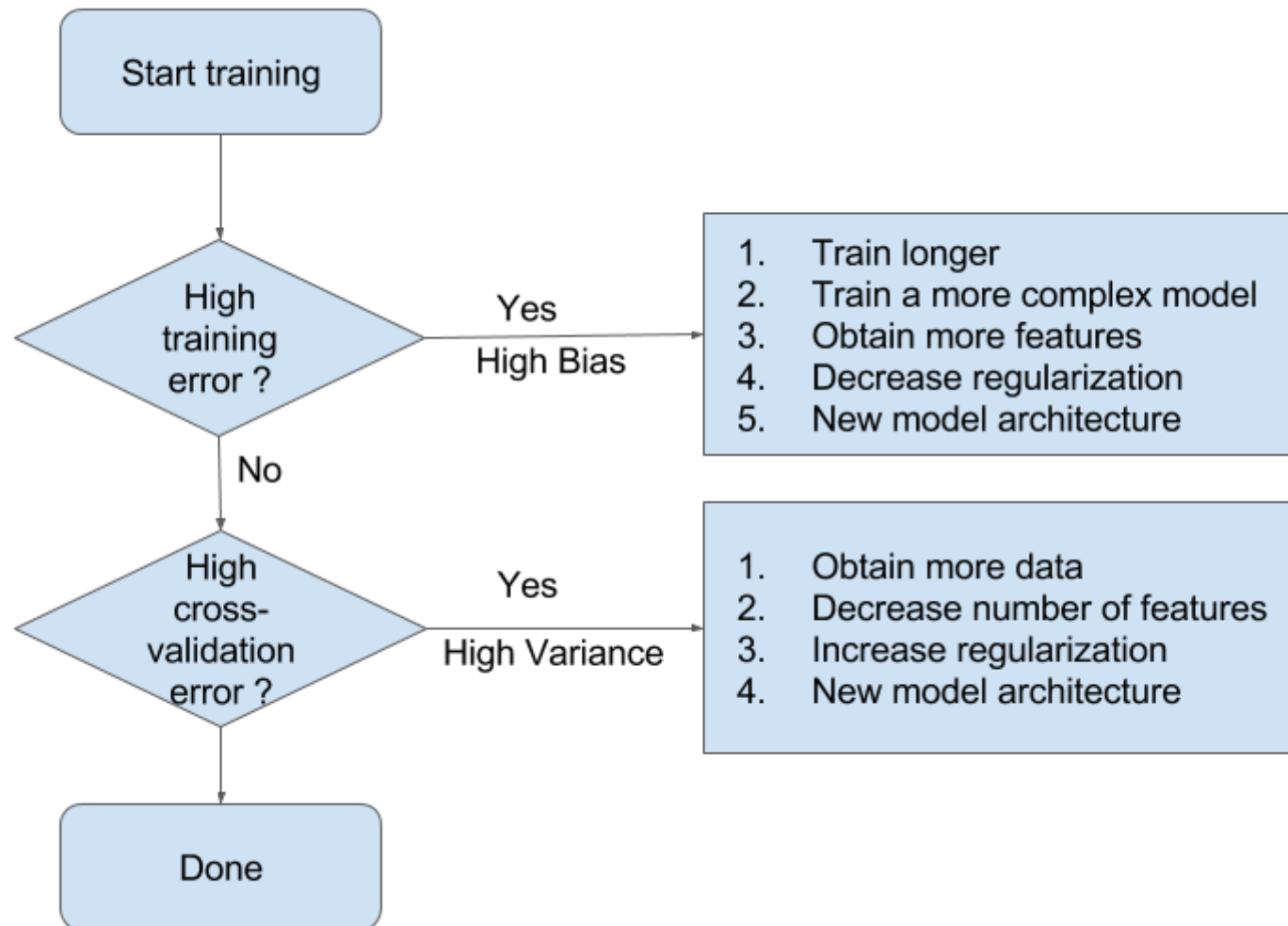
- How to measure bias and variance (for regression):
 - Take 100 or more bootstraps (or shuffle-splits)
 - For each data point x :
 - $bias(x)^2 = (x_{true} - mean(x_{predicted}))^2$
 - $variance(x) = var(x_{predicted})$
 - Total bias: $\sum_x bias(x)^2 * w_x$, with w_x the ratio of x occurring in the test set
 - Total variance: $\sum_x variance(x) * w_x$
- General procedure for (binary) classification:
 - Take 100 or more bootstraps (or shuffle-splits)
 - Bias for any point x = misclassification ratio
 - If misclassified 50% of the time: $bias(x) = 0.5$
 - Variance for any point x is $(1 - (P(class_1)^2 + P(class_2)^2))/2$
 - $P(class_i)$ is ratio of class i predictions
 - When each class predicted half of the time:
 $variance(x) = (1 - (0.5^2 + 0.5^2))/2 = 0.25$
 - Total bias: $\sum_x bias(x)^2 * w_x$, with w_x the ratio of x occurring in the test data
 - Total variance: $\sum_x variance(x) * w_x$

Bias-variance and overfitting



- High bias means that you are likely underfitting
 - Do less regularization
 - Use a more flexible/complex model (another algorithm)
 - Use a bias-reduction technique (e.g. boosting, see later)
- High variance means that you are likely overfitting
 - Use more regularization
 - Get more data
 - Use a simpler model (another algorithm)
 - Use a variance-reduction techniques (e.g. bagging, see later)

Bias-Variance Flowchart (Andrew Ng, Coursera)



Overfitting the parameters and the validation set

- Simply taking the best performing model yields optimistic results
- We've already used the test data to evaluate each hyperparameter setting!
- Hence, we don't have an independent test set to evaluate these hyperparameter settings
 - Information 'leaks' from test set into the final model
- Solution: Set aside part of the training data to evaluate the hyperparameter settings
 - Select best hyperparameters on validation set
 - Rebuild the model on the training+validation set
 - Evaluate optimal model on the test set



Nested cross-validation

- Note that we are still using a single split to create the outer test set
- We can also use cross-validation here
- Nested cross-validation:
 - Outer loop: split data in training and test sets
 - Inner loop: run grid search, splitting the training data into train and validation sets
- Result is a just a list of scores
 - There will be multiple optimized models and hyperparameter settings (not returned)
- To apply on future data, we need to train `GridSearchCV` on all data again

Hyperparameter tuning

Now that we know how to evaluate models, we can improve them by tuning their hyperparameters

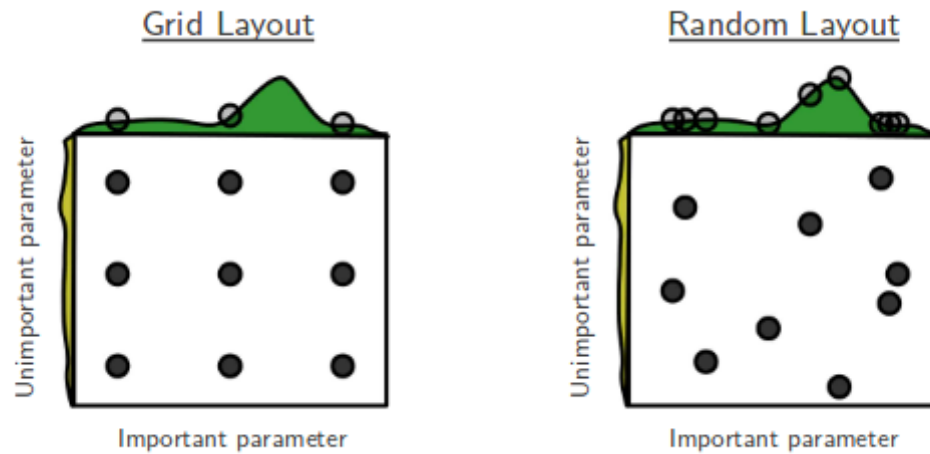
We can basically use any optimization technique to optimize hyperparameters:

- **Grid search**
- **Random search**

More advanced techniques:

- Local search
- Racing algorithms
- Model-based optimization (see later)
- Multi-armed bandits
- Genetic algorithms

Grid vs Random Search



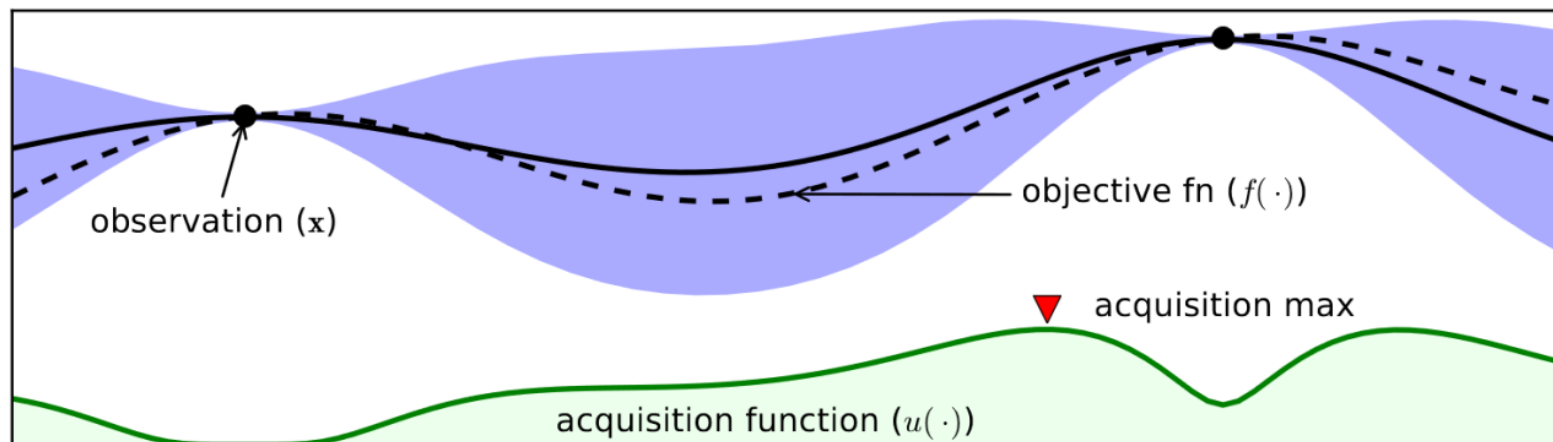
Bayesian optimization

- The incremental updates you can do with Bayesian models allow a more effective way to optimize functions
 - E.g. to optimize the hyperparameter settings of a machine learning algorithm/pipeline
- After a number of random search iterations we know more about the performance of hyperparameter settings on the given dataset
- We can use this data to train a model, and predict which other hyperparameter values might be useful
 - More generally, this is called model-based optimization
 - This model is called a *surrogate model*
- This is often a probabilistic (e.g. Bayesian) model that predicts confidence intervals for all hyperparameter settings
- We use the predictions of this model to choose the next point to evaluate
- With every new evaluation, we update the surrogate model and repeat

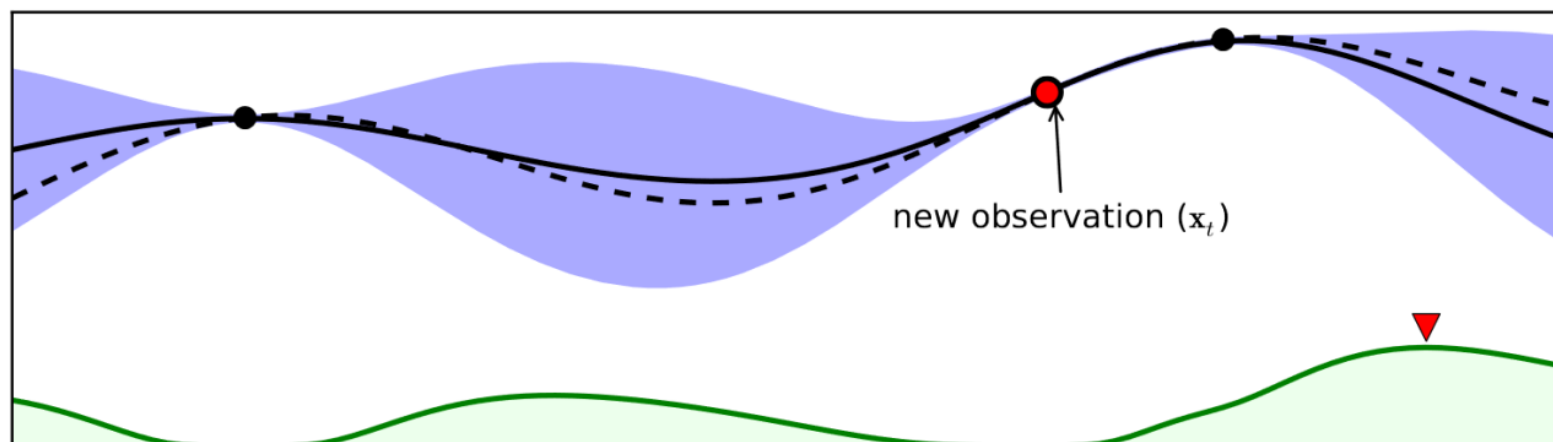
Example (see figure):

- Consider only 1 continuous hyperparameter (X-axis)
 - You can also do this for many more hyperparameters
- Y-axis shows cross-validation performance
- Evaluate a number of random hyperparameter settings (black dots)
 - Sometimes an initialization design is used
- Train a model, and predict the expected performance of other (unseen) hyperparameter values
 - Mean value (black line) and distribution (blue band)
- An *acquisition function* (green line) trades off maximal expected performance and maximal uncertainty
 - Exploitation vs exploration
- Optimal value of the acquisition function is the next hyperparameter setting to be evaluated
- Repeat a fixed number of times, or until time budget runs out

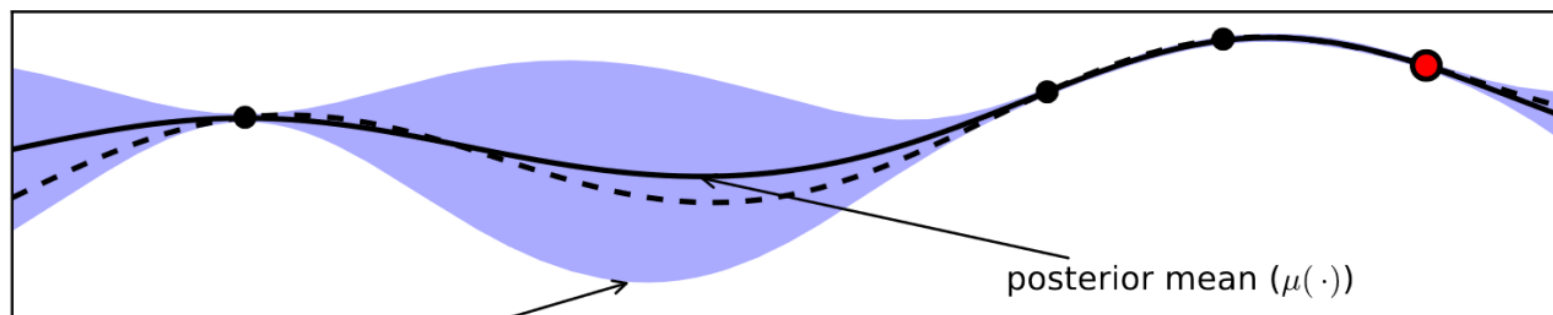
$t = 2$

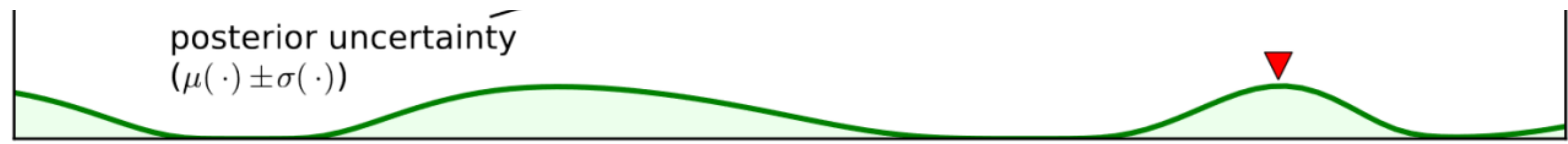


$t = 3$



$t = 4$





In 2 dimensions: