

Lecture 3: Model Selection

Can I trust you?

Joaquin Vanschoren, Eindhoven University of Technology

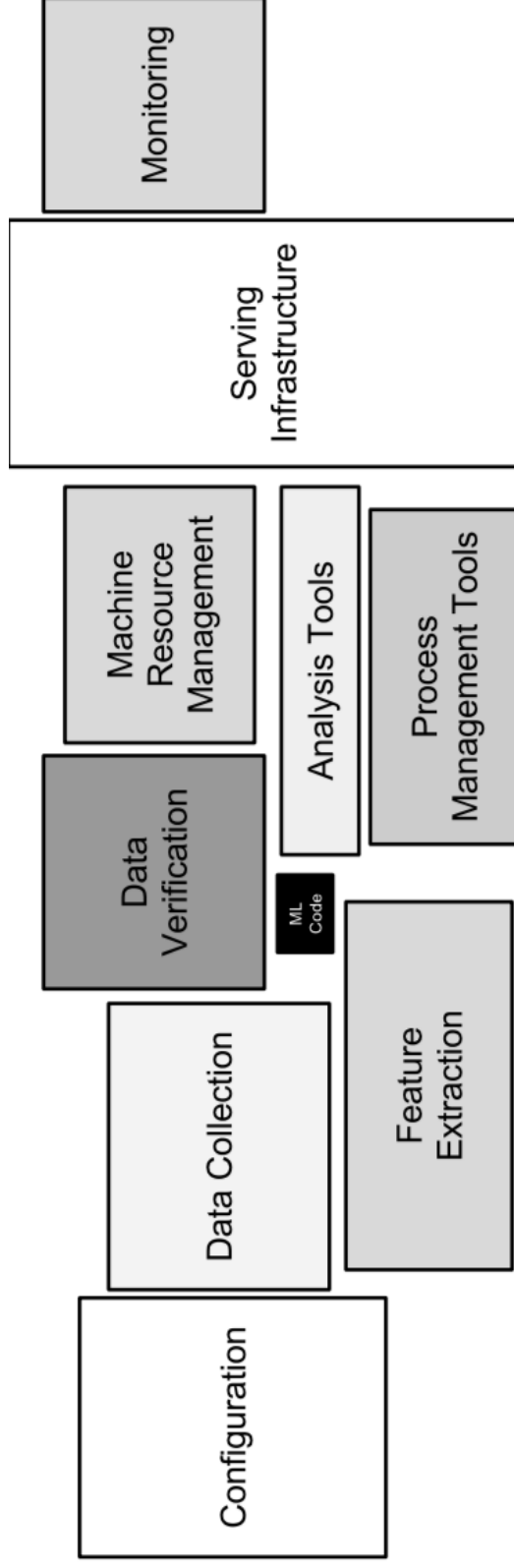
Evaluation

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

Designing Machine Learning 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.

- Analyze 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
 - Example: a medical recommendation model may change future measurements
- The signal your model found may just be an artifact of your biased data
 - When possible, try to *interpret* what your model has learned
 - See 'Why Should I Trust You?' by Marco Ribeiro et al.



(a) Husky classified as wolf

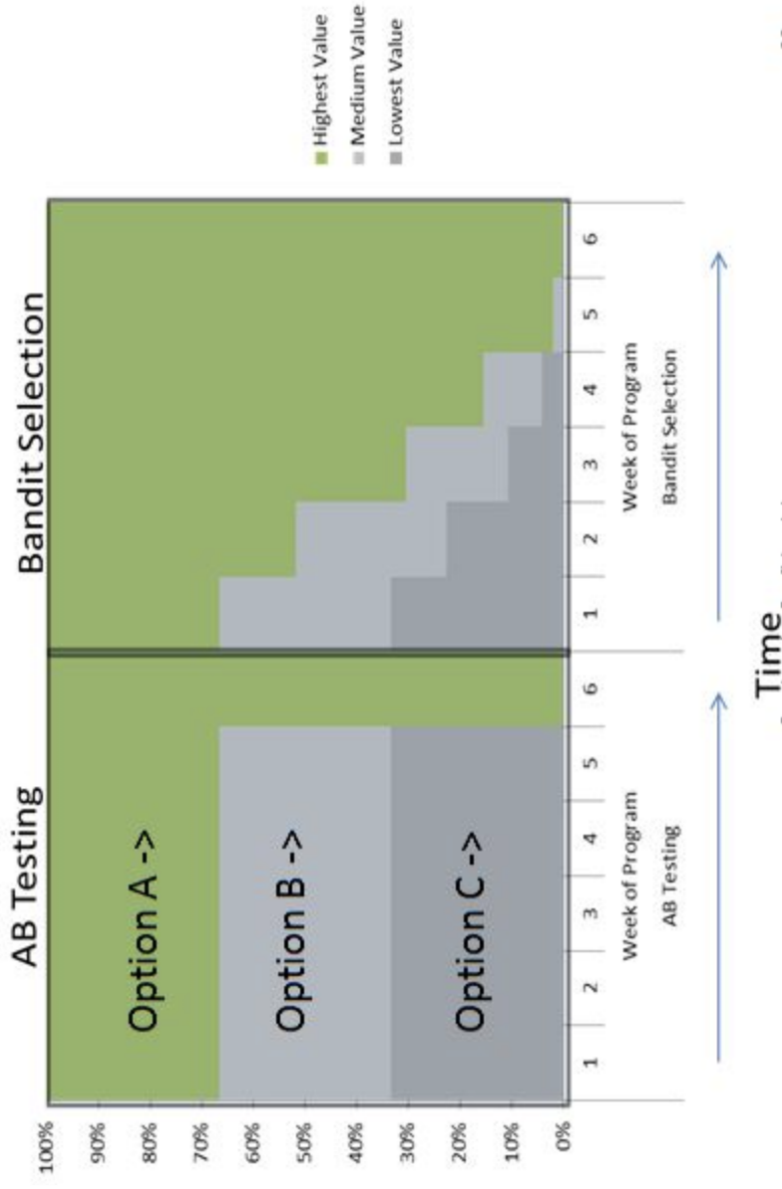


(b) Explanation

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

A/B and bandit testing

- Test a single innovation (or choose between two models)
- 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

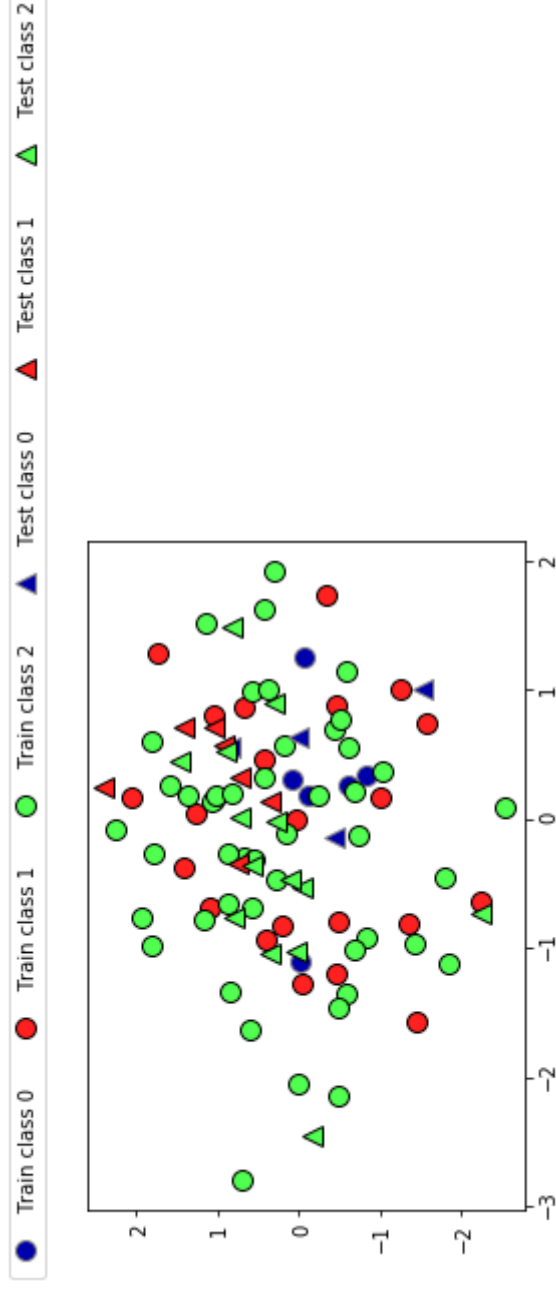


Performance estimation techniques

- We do not have access to future observations
- Always evaluate models *as if they are predicting the future*
- Set aside data for objective evaluation
 - How?

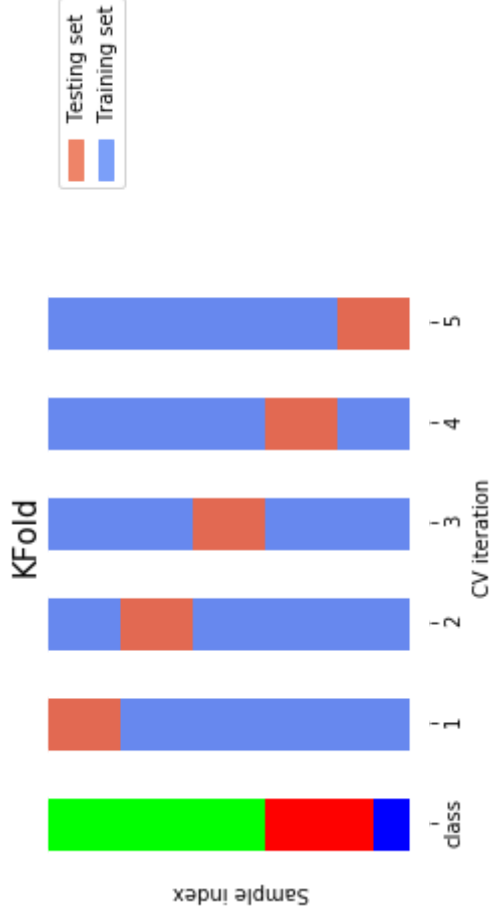
The holdout (simple train-test split)

- *Randomly* split data (and corresponding labels) into training and test set (e.g. 75%-25%)
- Train (fit) a model on the training data, score on the test data



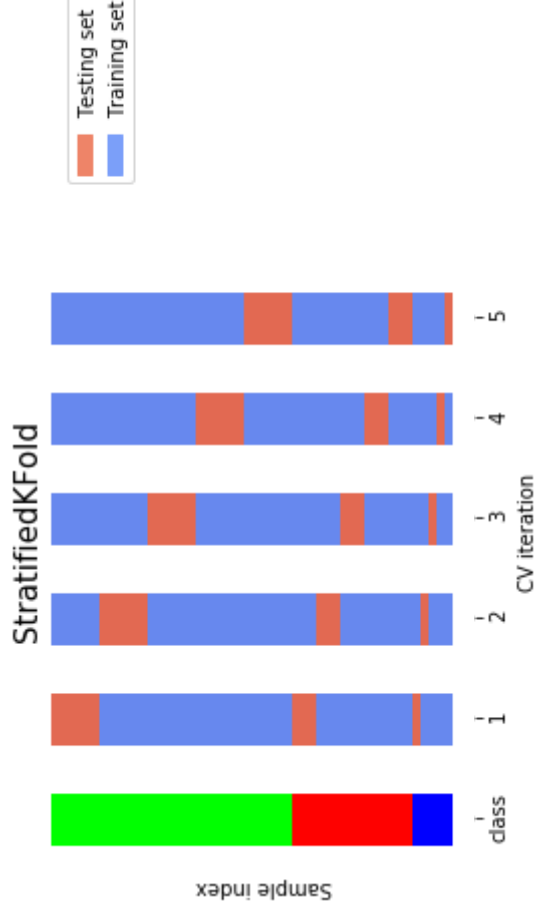
K-fold Cross-validation

- Each random split can yield very different models (and scores)
 - e.g. all easy (of hard) examples could end up in the test set
- Split data (randomly) into k equal-sized parts, called *folds*
 - Create k splits, each time using a different fold as the test set
- Compute k evaluation scores, aggregate afterwards (e.g. take the mean)
- Examine the score variance to see how *sensitive* (unstable) models are
- Reduces sampling bias by testing on every point exactly once
- Large k gives better estimates (more training data), but is expensive



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



Can you explain this result?

```
kfold = KFold(n_splits=3)
print("Cross-validation scores KFold(n_splits=3):\n{}".format(
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

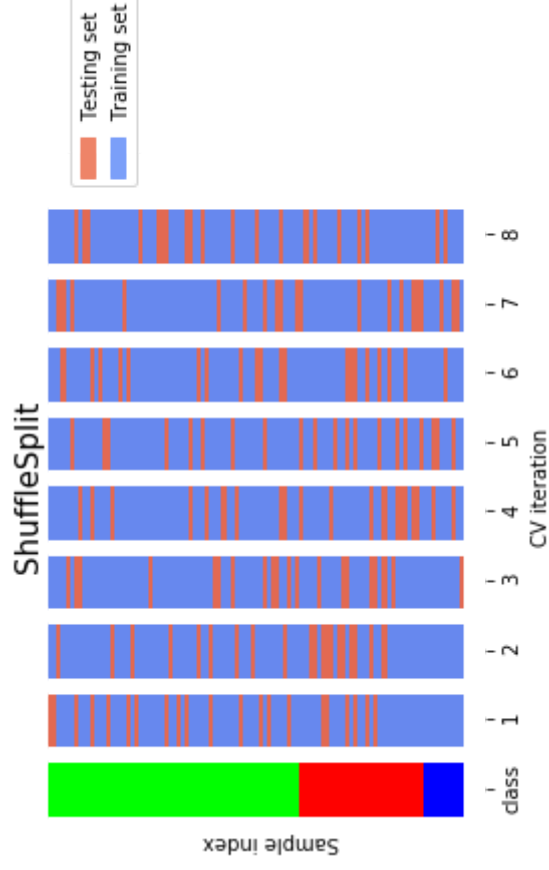
```
Cross-validation scores KFold(n_splits=3):
[0. 0. 0.]
```

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

Shuffle-Split cross-validation

- Additionally shuffles the data (only do this when the data is not ordered)
- Samples a number of samples (`train_size`) randomly as the training set
- Can also use a smaller (`test_size`), handy when using very large datasets

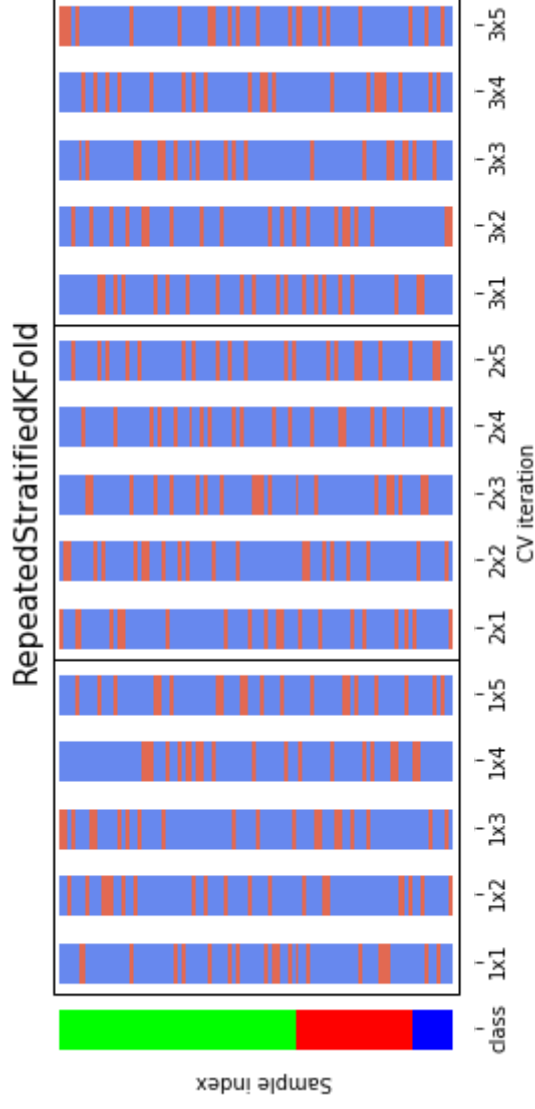


Note: this is related to *bootstrapping*:

- Sample n (total number of samples) data points, with replacement, as training set (the bootstrap)
- Use the unsampled (out-of-bootstrap) samples as the test set
- Repeat n_iter times, obtaining n_iter scores
- Not supported in scikit-learn, only approximated
 - Use Shuffle-Split with `train_size=0.66, test_size=0.34`
 - You can prove that bootstraps include 66% of all data points on average

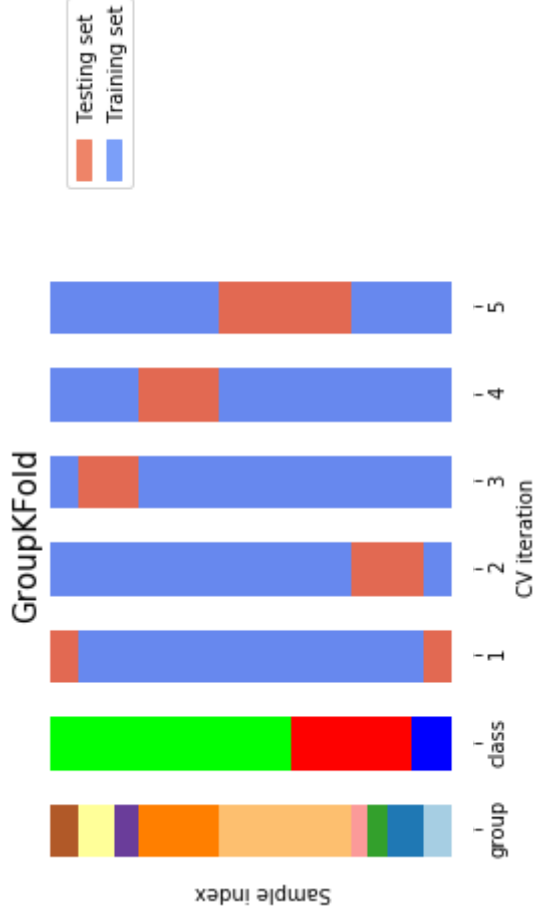
Repeated cross-validation

- Cross-validation is still biased in that the initial split can be made in many ways
- Repeated, or n-times-k-fold cross-validation:
 - Shuffle data randomly, do k-fold cross-validation
 - Repeat n times, yields n times k scores
- Unbiased, very robust, but n times more expensive



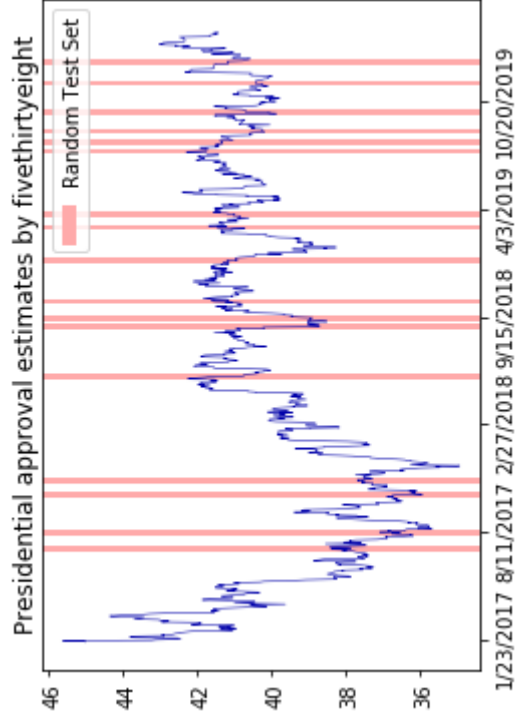
Cross-validation with groups

- Sometimes the data contains inherent groups:
 - Multiple samples from same patient, images from same person,...
- With normal cross-validation, data from the same person may end up in the training *and* test set
- We want to measure how well the model generalizes to *other* people
- Make sure that data from one person are in *either* the train or test set
 - This is called *grouping* or *blocking*
 - Leave-one-subject-out cross-validation: test set for each subject



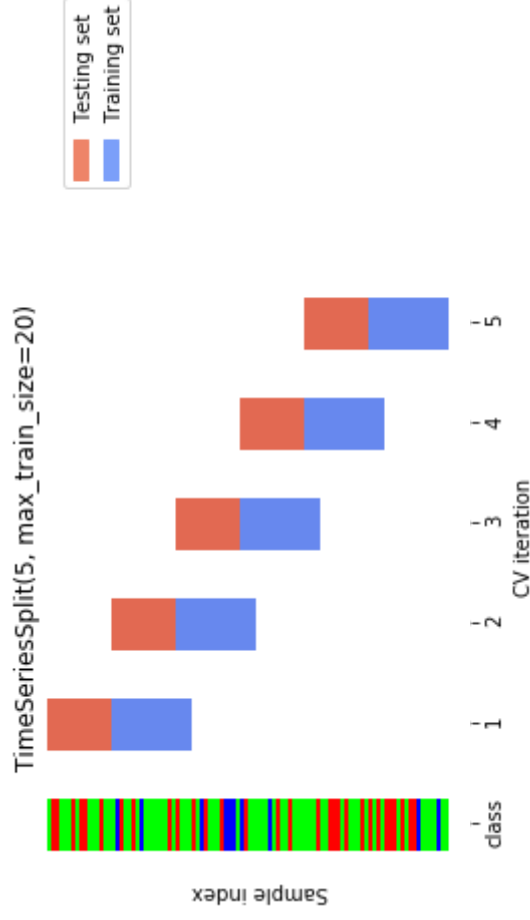
Time series

When the data is ordered, random test sets are not a good idea



Time series

- Test-then-train (prequential evaluation)
 - Every new sample is evaluated on once, then added to the training set
 - Can also be done in batches (of n samples at a time)
- TimeSeriesSplit
 - In the k th split, the first k folds are used as the train set and the $(k+1)$ th fold as the test set
 - Can also be done with a maximum training set size: more robust against concept drift



Choosing a performance estimation procedure

No strict rules, only guidelines:

- Always use stratification for classification (sklearn does this by default)
- 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
- Use train-then-test for time series

Evaluation Metrics and scoring

Keep the end-goal in mind

Evaluation vs Optimization

- Each algorithm optimizes a given objective function (on the training data)

- E.g. remember L2 loss in Ridge regression

$$\mathcal{L}_{ridge} = \sum_i (y_i - \sum_j x_{i,j} w_j)^2 + \alpha \sum_i w_i^2$$

- The choice of function is limited by what can be efficiently optimized
 - E.g. gradient descent requires a differentiable loss function
- However, we *evaluate* the resulting model with a score that makes sense **in the real world**
 - E.g. percentage of correct predictions (on a test set)
- We also tune the algorithm's hyperparameters to maximize that score

Binary classification

- We have a positive and a negative class
- 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 not always equally important
 - Which side do you want to err on for a medical test?

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:

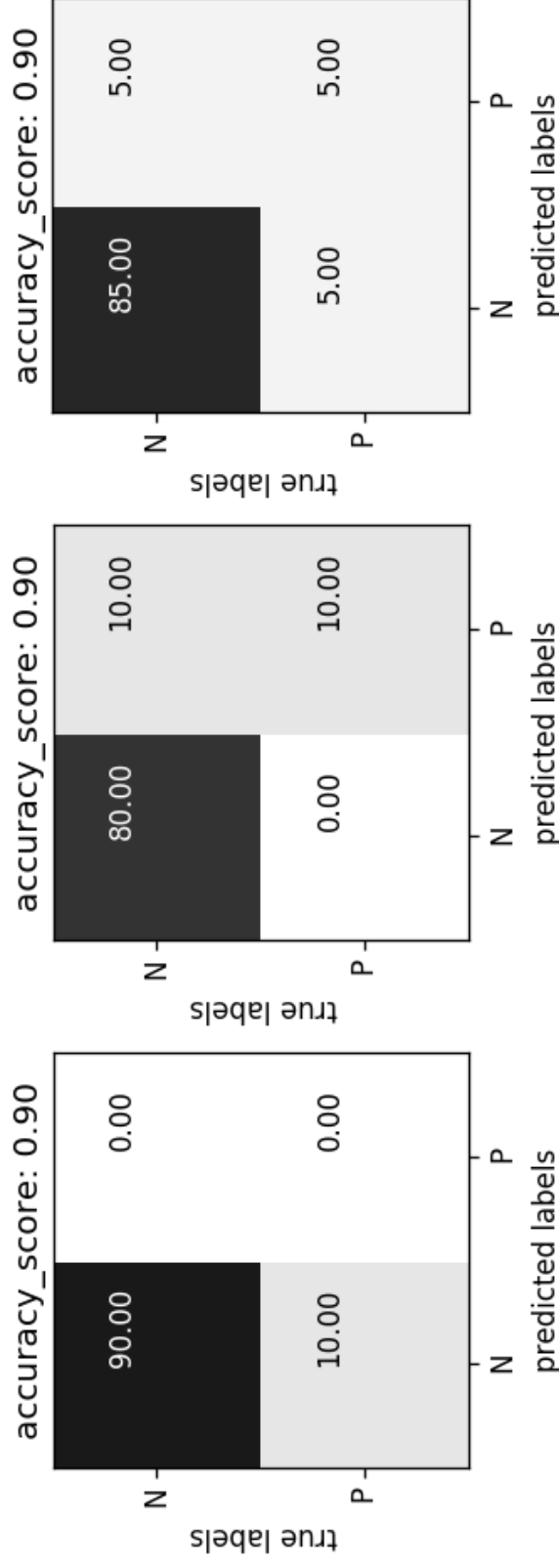
$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- In sklearn: use `confusion_matrix` and `accuracy_score` from `sklearn.metrics`.
- Accuracy is also the default evaluation measure for classification

```
confusion_matrix(y_test, y_pred):  
[[48  5]  
 [ 5 85]]  
accuracy_score(y_test, y_pred):  0.9300699300699301  
model.score(X_test, y_test):  0.9300699300699301
```


The problem with accuracy: imbalanced datasets

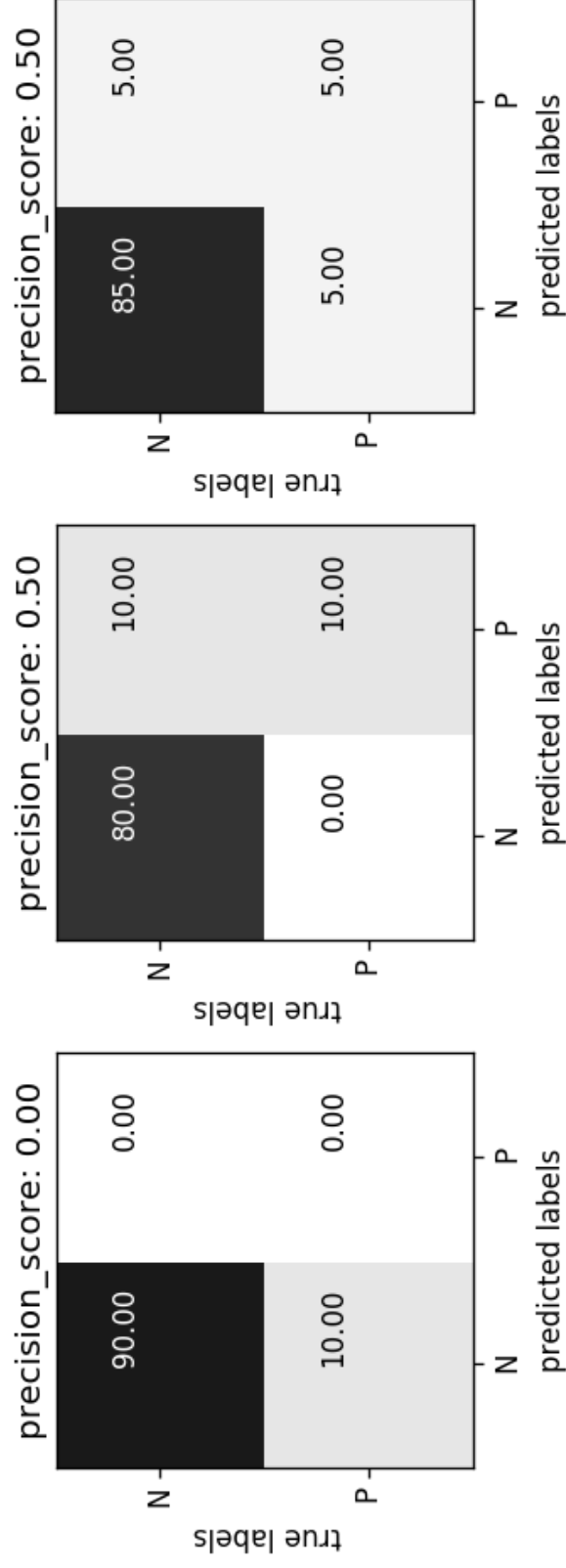
- The type of error plays an even larger role if the dataset is imbalanced
 - One class is much more frequent than the other, e.g. credit fraud
 - Is a 99.99% accuracy good enough?
- Are these three models really equally good?



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

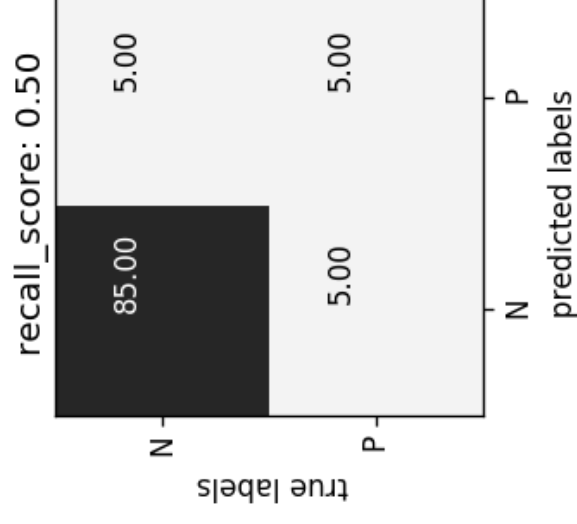
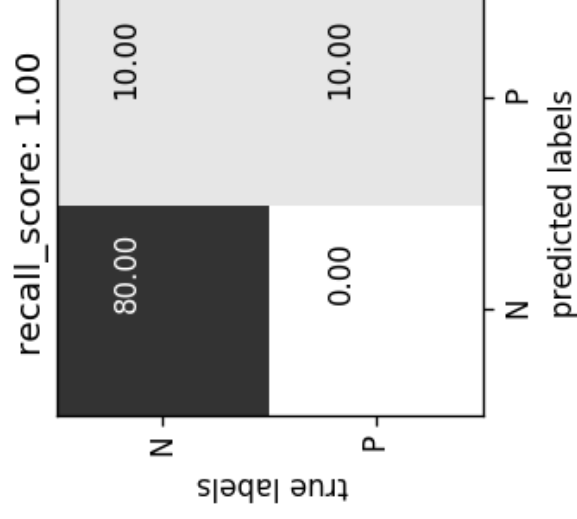
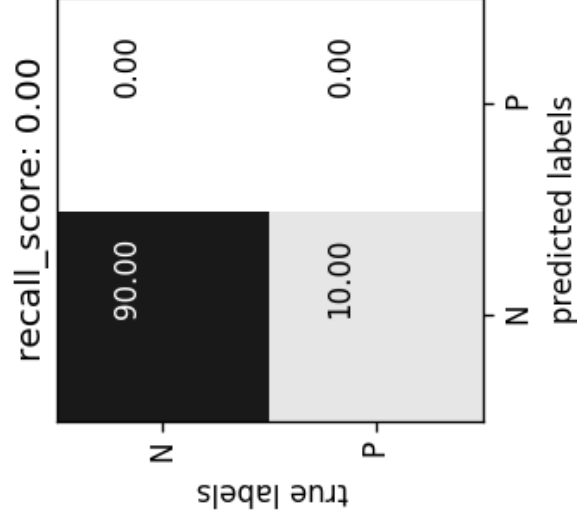
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$



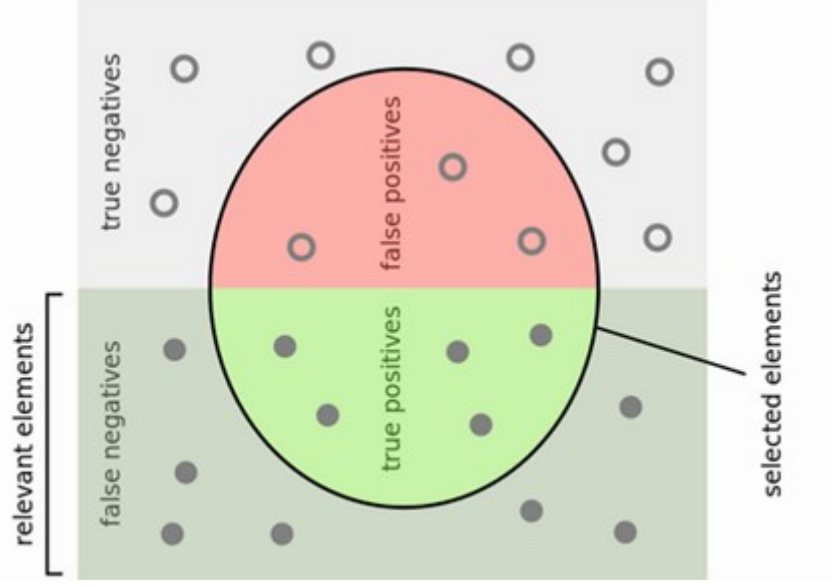
Recall is used when the goal is to limit FNs

- 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)

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



Comparison



How many selected items are relevant?

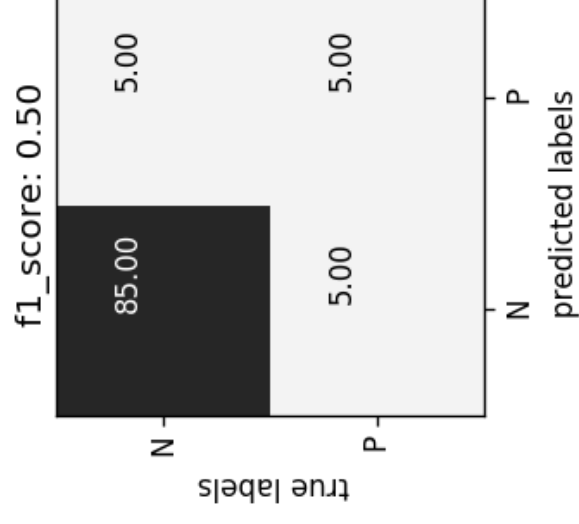
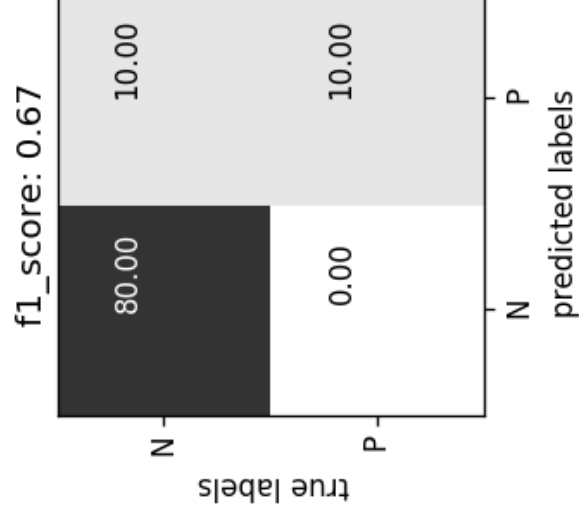
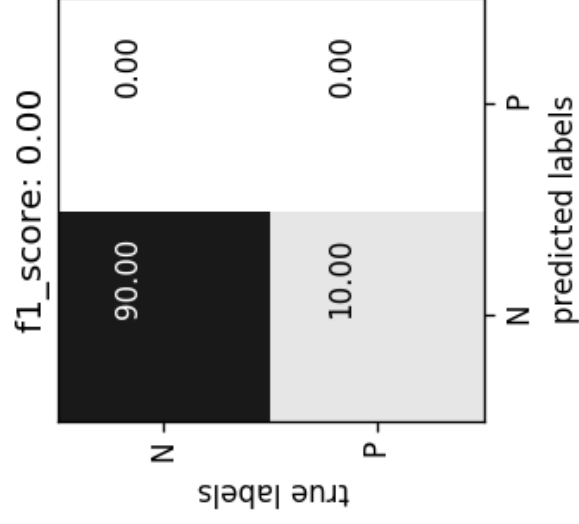
$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

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

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



Classification measure Zoo

		True condition		
		Condition positive	Condition negative	
Total population				$\text{Prevalence} = \frac{\sum \text{Condition positive}}{\sum \text{Total population}}$ $\text{Accuracy (ACC)} = \frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
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	$\text{False omission rate (FOR)} = \frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$ $\text{Negative predictive value (NPV)} = \frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		$\text{True positive rate (TPR), Recall, Sensitivity, probability of detection} = \frac{\sum \text{True positive}}{\sum \text{Condition positive}}$ $\text{False negative rate (FNR), Miss rate} = \frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	$\text{False positive rate (FPR), Fall-out, probability of false alarm} = \frac{\sum \text{False positive}}{\sum \text{Condition negative}}$ $\text{Specificity (SPC), Selectivity, True negative rate (TNR)} = \frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	$\text{Diagnostic odds ratio (DOR)} = \frac{\text{LR}^+}{\text{LR}^-}$ $\text{F}_1 \text{ score} = \frac{1}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$

[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)),

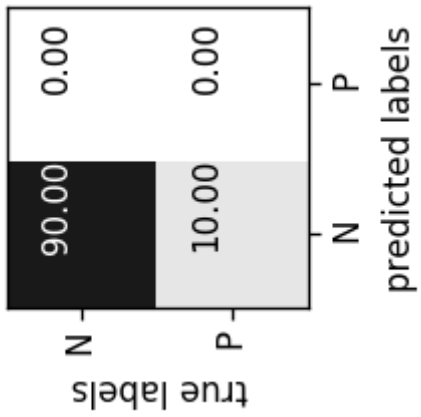
Averaging scores per class

- Study the scores *by class* (in scikit-learn: `classification_report`)
 - One class viewed as positive, other(s) als negative
 - Support: number of samples in each class
 - Last line: weighted average over the classes (weighted by number of samples in each class)
- Averaging for scoring measure R across C classes (also for multiclass):
 - micro: count total number of TP, FP, TN, FN
 - macro

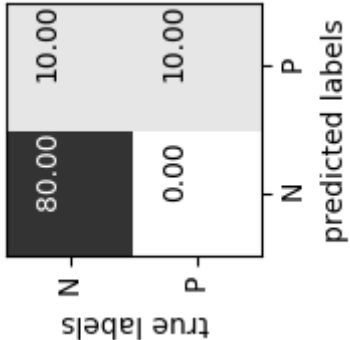
$$\frac{1}{C} \sum_{c \in C} R(y_c, \hat{y}_c)$$

- weighted (w_c : ratio of examples of class c)
$$\sum_{c \in C} w_c R(y_c, \hat{y}_c)$$

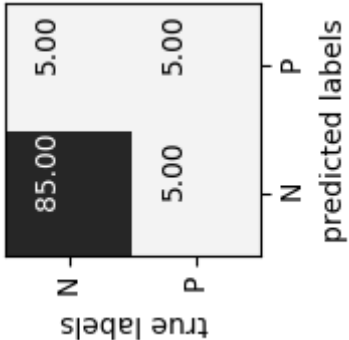
	precision	recall	f1-score	support
0	0.90	1.00	0.95	90
1	0.00	0.00	0.00	10
accuracy			0.90	100
macro avg	0.45	0.50	0.47	100
weighted avg	0.81	0.90	0.85	100



	precision	recall	f1-score	support
0	1.00	0.89	0.94	90
1	0.50	1.00	0.67	10
accuracy			0.90	100
macro avg	0.75	0.94	0.80	100
weighted avg	0.95	0.90	0.91	100



	precision	recall	f1-score	support
0	0.94	0.94	0.94	90
1	0.50	0.50	0.50	10
accuracy			0.90	100
macro avg	0.72	0.72	0.72	100
weighted avg	0.90	0.90	0.90	100



Uncertainty estimates from classifiers

- Classifiers can often provide uncertainty estimates of predictions.
- Remember that linear models actually return a numeric value.
 - When $\hat{y} < 0$, predict class -1, otherwise predict class +1
- In practice, you are often interested in how certain a classifier is about each class prediction (e.g. cancer treatments).

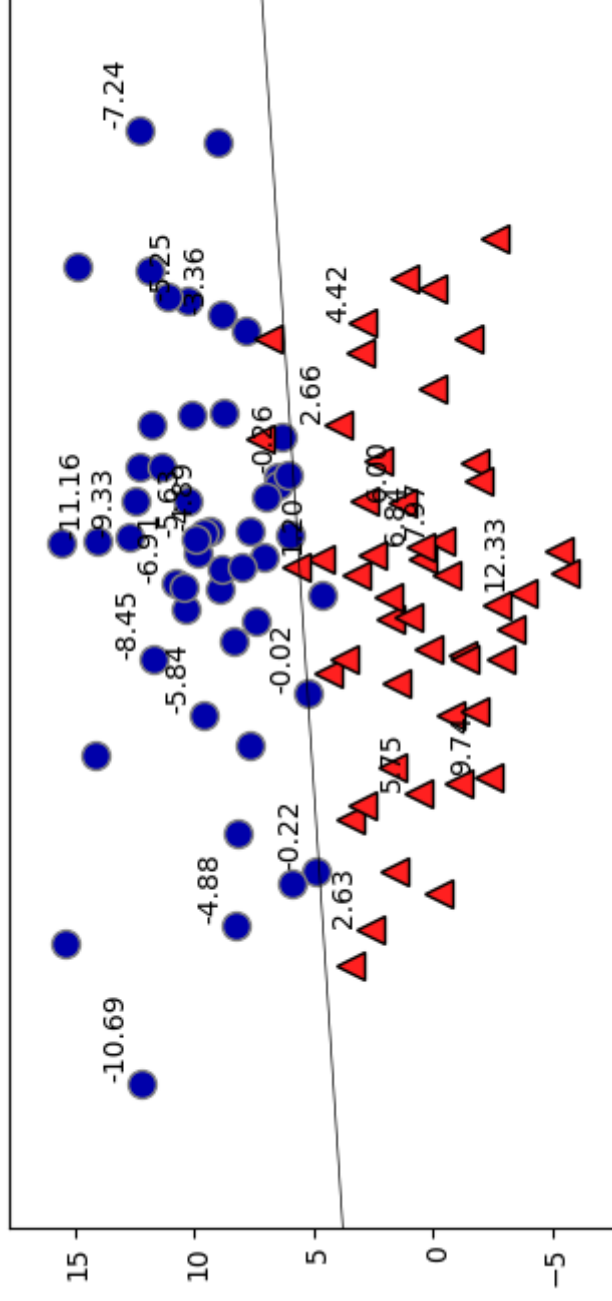
Scikit-learn offers 2 functions. Often, both are available for every learner, but not always.

- `decision_function`: returns floating point value for each sample
- `predict_proba`: return probability for each class

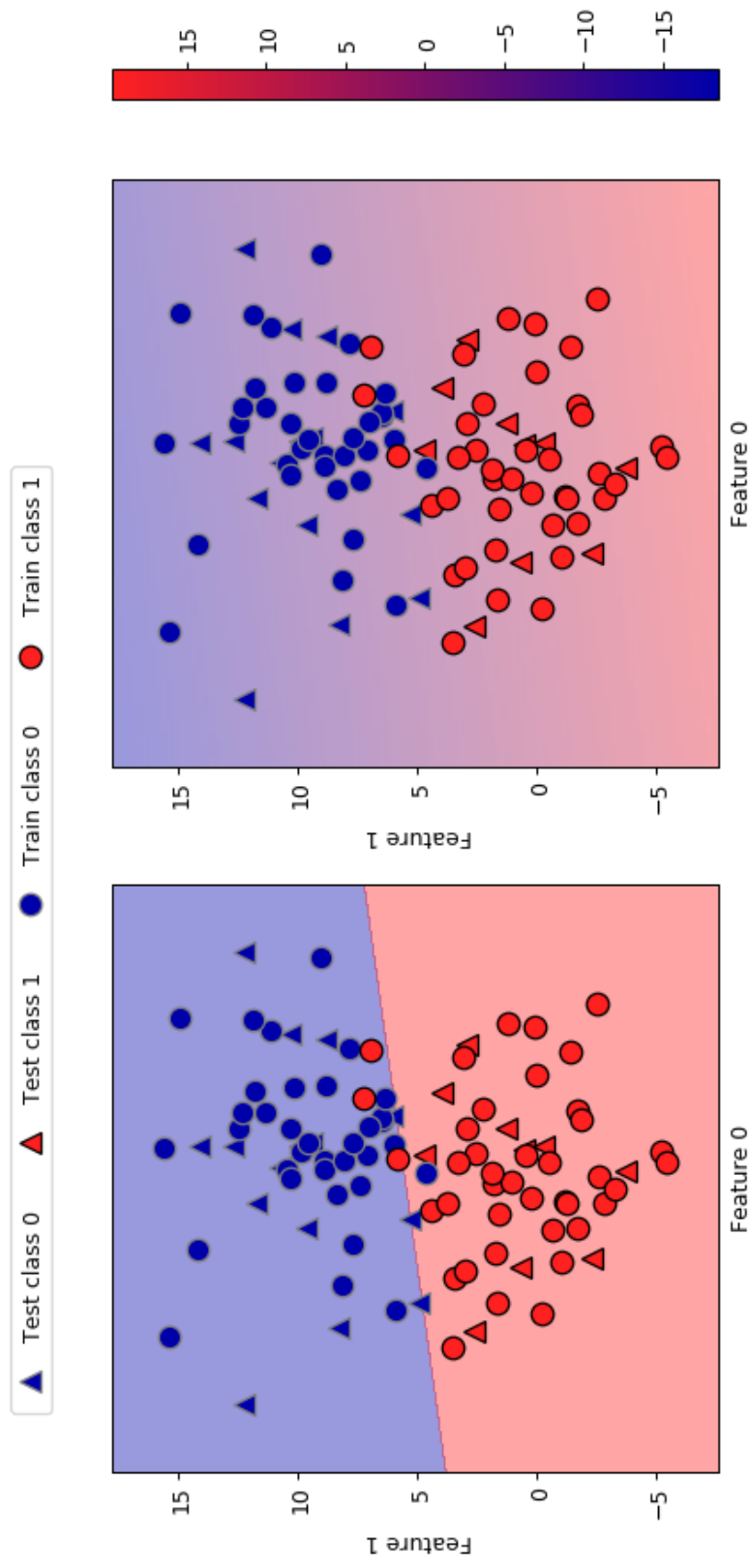
The Decision Function

In the binary classification case, the return value of `decision_function` encodes how strongly the model believes a data point to belong to the “positive” class.

- Positive values indicate a preference for the “positive” class
- Negative values indicate a preference for the “negative” (other) class



- The range of `decision_function` can be arbitrary, and depends on the data and the model parameters. This makes it sometimes hard to interpret.
- We can visualize the decision function as follows, with the actual decision boundary left and the values of the decision boundaries color-coded on the right.
- Note how the test examples are labeled depending on the decision function.

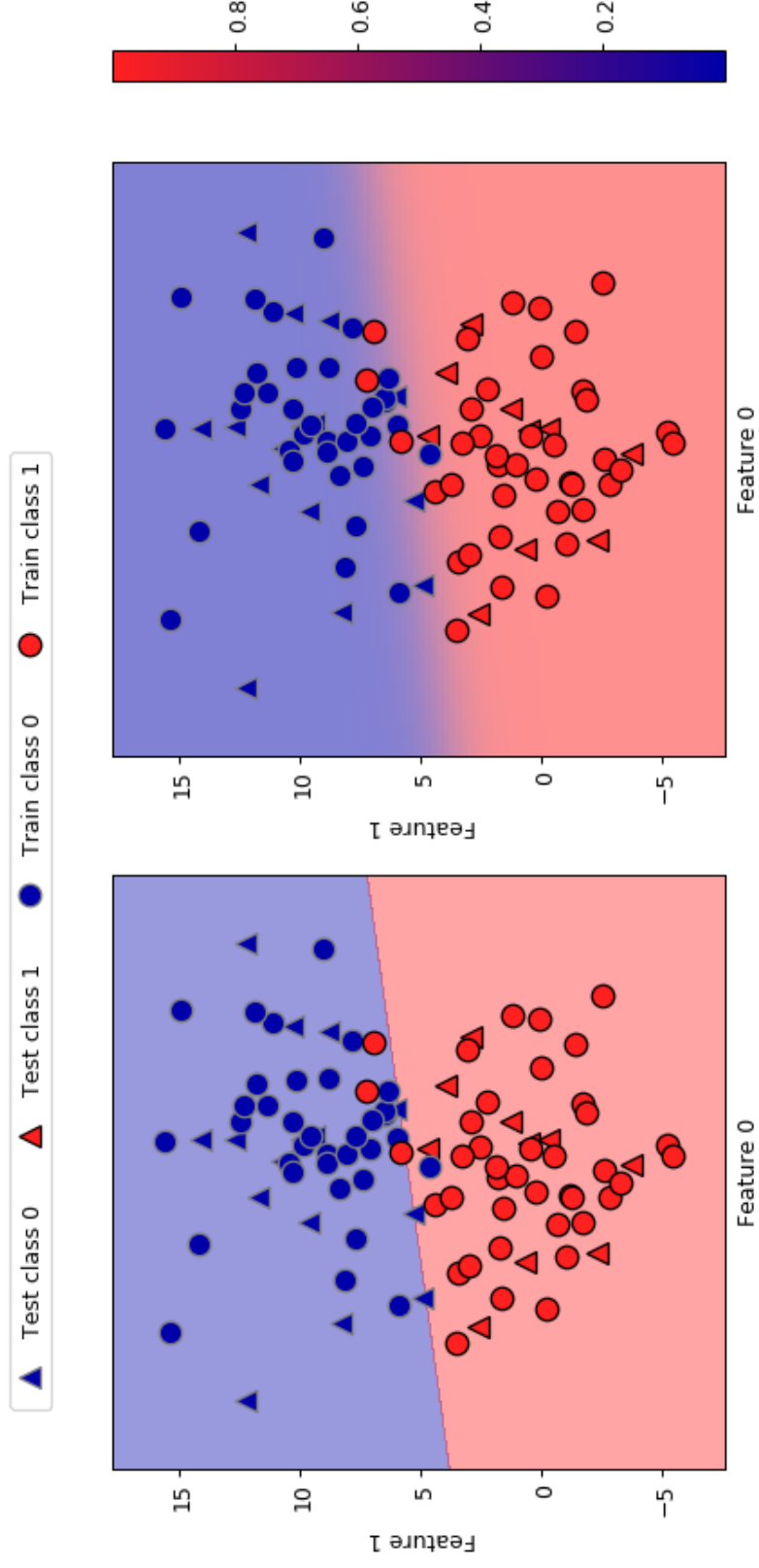


Predicting probabilities

The output of predict_proba is a *probability* for each class, with one column per class. They sum up to 1.

```
Shape of probabilities: (25, 2)
Predicted probabilities:
[[0.232 0.768]
 [0.002 0.998]
 [0.    1.    ]
 [0.003 0.997]
 [0.001 0.999]
 [1.    0.    ]]
```

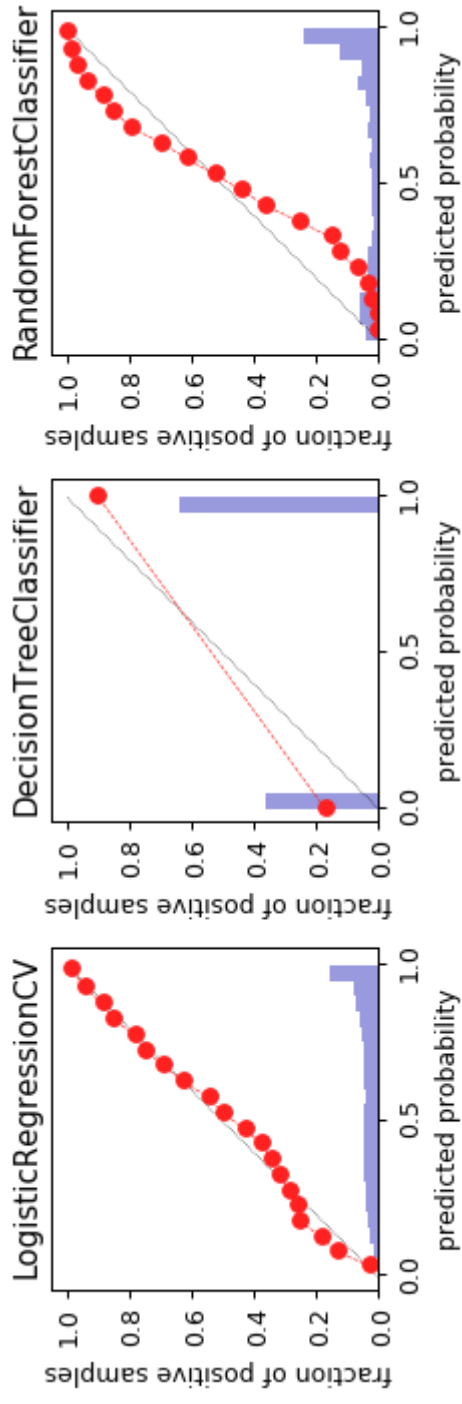
We can visualize them again. Note that the gradient looks different now.



Interpreting probabilities

- The class with the highest probability is predicted.
- How well the uncertainty actually reflects uncertainty in the data depends on the model and the parameters.
 - An overfitted model tends to make more certain predictions, even if they might be wrong.
 - A model with less complexity usually has more uncertainty in its predictions.
- A model is called *calibrated* if the reported uncertainty actually matches how correct it is — A prediction made with 70% certainty would be correct 70% of the time.
 - LogisticRegression returns well calibrated predictions by default as it directly optimizes log-loss
 - Linear SVM are not well calibrated. They are *biased* towards points close to the decision boundary.
- Calibration techniques (<http://scikit-learn.org/stable/modules/calibration.html>) can calibrate models in post-processing.

Model calibration



Model calibration

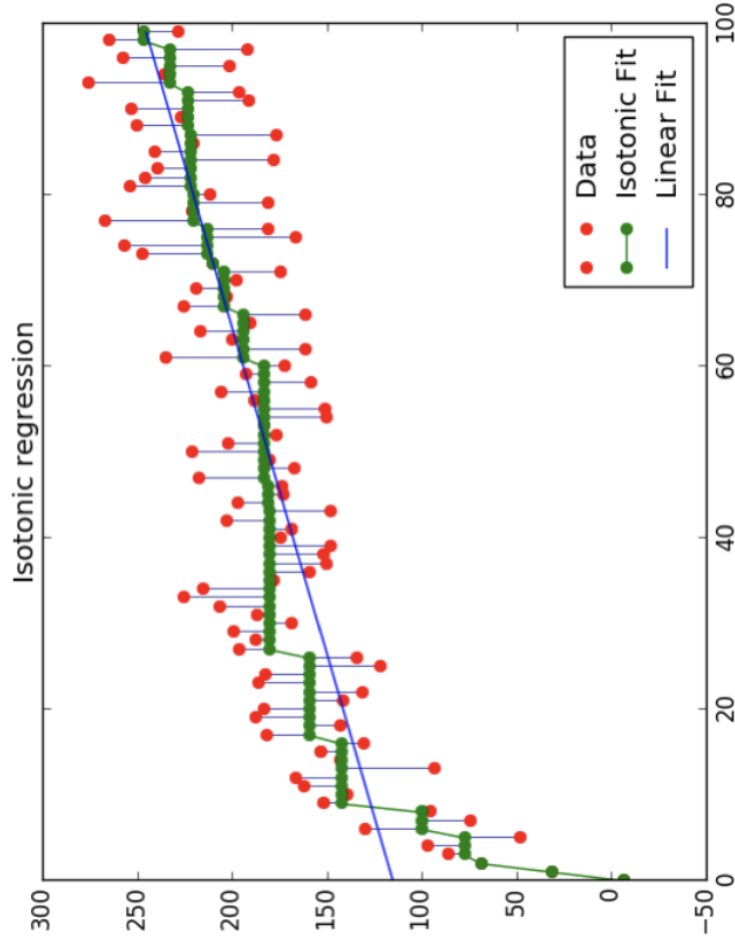
- Build another model, mapping classifier probabilities to better probabilities!
- 1d model! (or more for multi-class)
$$f_{calib}(s(x)) \approx p(y)$$
- $s(x)$ is score given by model, usually
- Can also work with models that don't even provide probabilities! Need model for $f_{\{calib\}}$, need to decide what data to train it on.
- Can train on training set, causes overfit
- Can train using cross-validation, slower

Platt Scaling

- Use a logistic sigmoid for f_{calib}
$$f_{platt} = \frac{1}{1 + \exp(-ws(x) - b)}$$
- Basically learning a 1d logistic regression (+ some tricks)
- Works well for SVMs

Isotonic regression

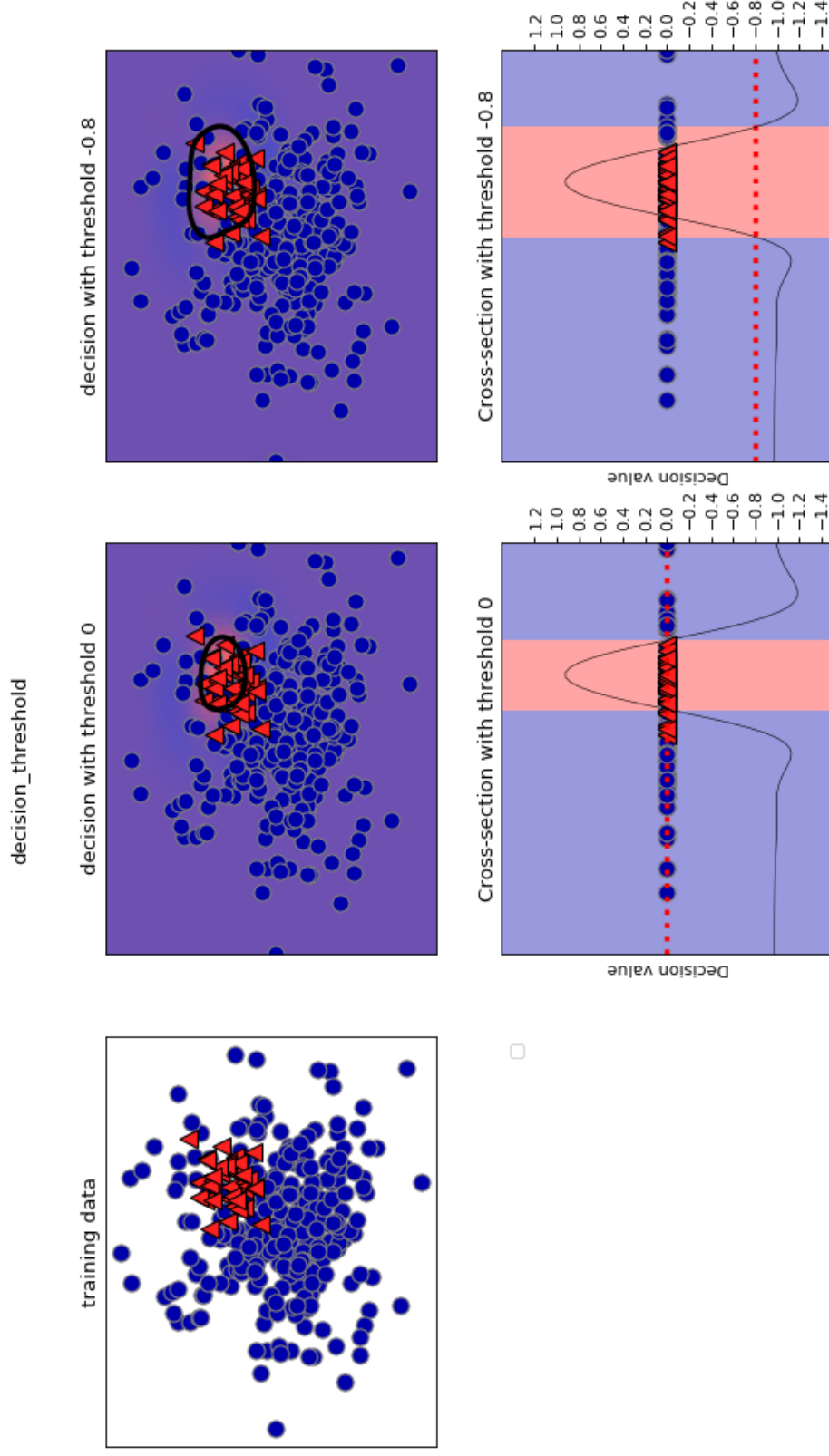
- Very flexible way to specify f_{calib}
- Learns arbitrary monotonically increasing step-functions in 1d.
- Groups data into constant parts, steps in between.
- Optimum monotone function on training data (wrt MSE)



Taking uncertainty into account

- Remember that many classifiers actually return a probability per class
 - We can retrieve it with `decision_function` and `predict_proba`
- For binary classification, we threshold at 0 for `decision_function` and 0.5 for `predict_proba` by default
- However, depending on the evaluation measure, you may want to threshold differently to fit your goals
 - For instance, when a FP is much worse than a FN
 - This is called *threshold calibration*

- Imagine that we want to 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



- Studying the classification report, we see that lowering the threshold yields:
 - higher recall for class 1 (we risk more FPs in exchange for more TP)
 - lower precision for class 1
- We can often trade off precision for recall

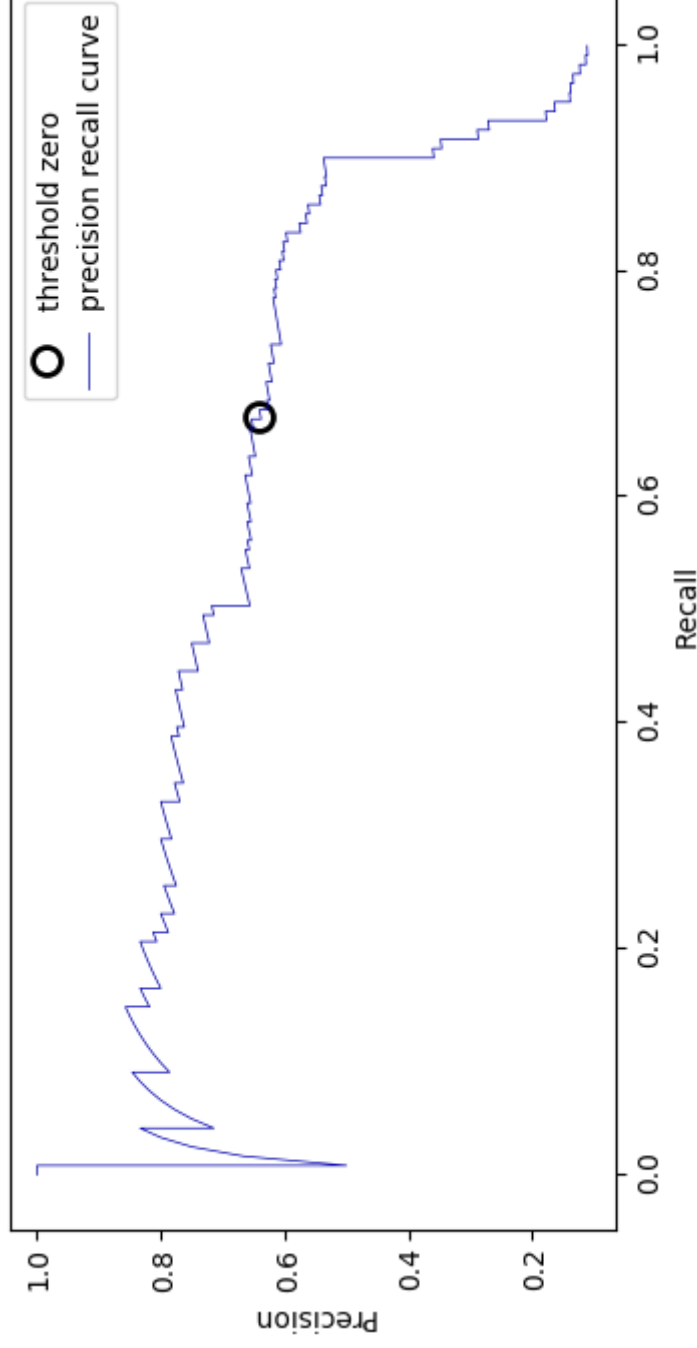
Threshold 0				
	precision	recall	f1-score	support
0	0.91	0.96	0.93	96
1	0.67	0.47	0.55	17
accuracy			0.88	113
macro avg	0.79	0.71	0.74	113
weighted avg	0.87	0.88	0.88	113

Threshold -0.8				
	precision	recall	f1-score	support
0	0.98	0.92	0.95	96
1	0.65	0.88	0.75	17
accuracy			0.91	113
macro avg	0.81	0.90	0.85	113
weighted avg	0.93	0.91	0.92	113

Precision-Recall curves

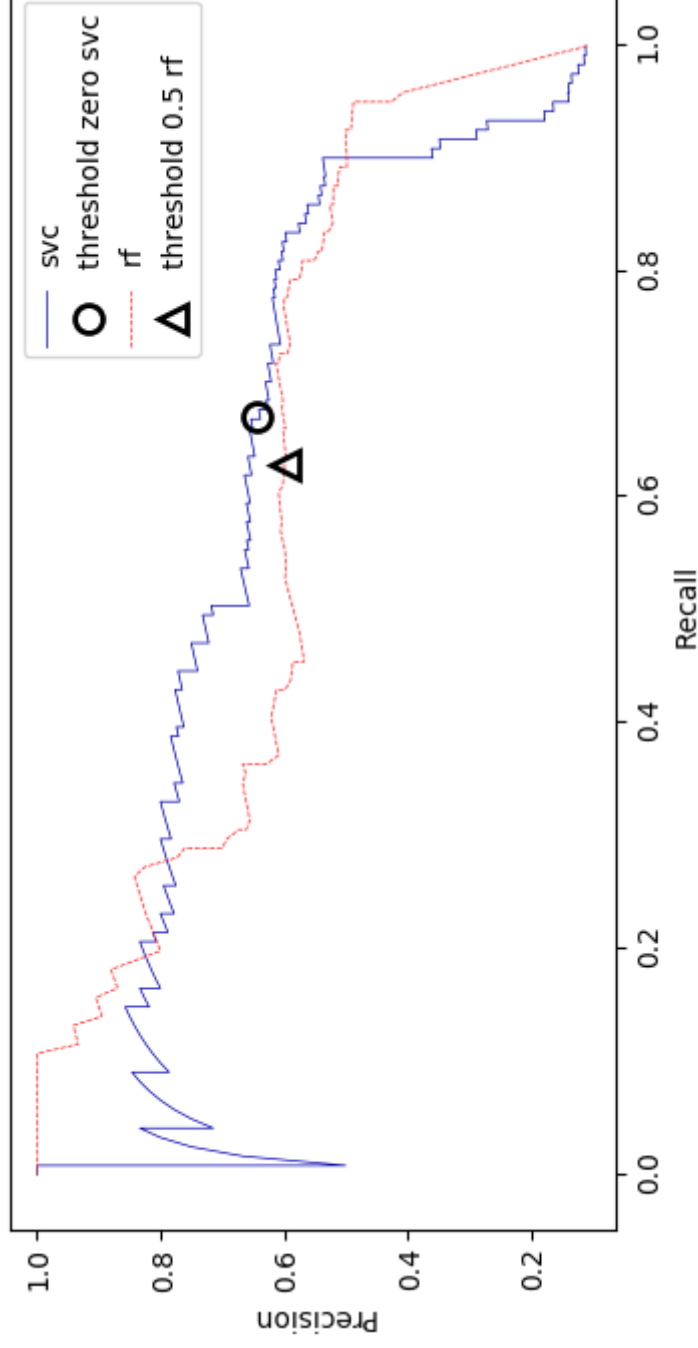
- As we've seen, you can trade off precision for accuracy by changing the decision threshold
- The best trade-off depends on your application, driven by real-world goals.
 - You can have arbitrary high recall, but you often want reasonable precision, too.
- Plotting precision against recall for all possible thresholds yields a **precision-recall curve**
- It helps answer multiple questions:
 - Threshold calibration: what's the best achievable precision-recall tradeoff?
 - How much more precision can I gain without losing too much recall?
 - Which models offer the best trade-offs?

- The default threshold (*threshold zero*) gives a certain precision and recall
 - Lower the threshold to gain higher recall (move right)
 - Increase the threshold to gain higher precision (move left)
- The curve is often jagged: increasing the threshold leaves fewer and fewer positive predictions, so precision ($\frac{TP}{TP+FP}$) can change dramatically
- The closer the curve stays to the upper-right corner, the better
- Here, it is possible to still get a precision of 0.55 with recall 0.9



Model selection

- Different classifiers offer different trade-offs
- RandomForest (in red) performs better at the extremes, SVM better in center
- In applications we may only care about a specific region (e.g. very high recall)



AUPRC

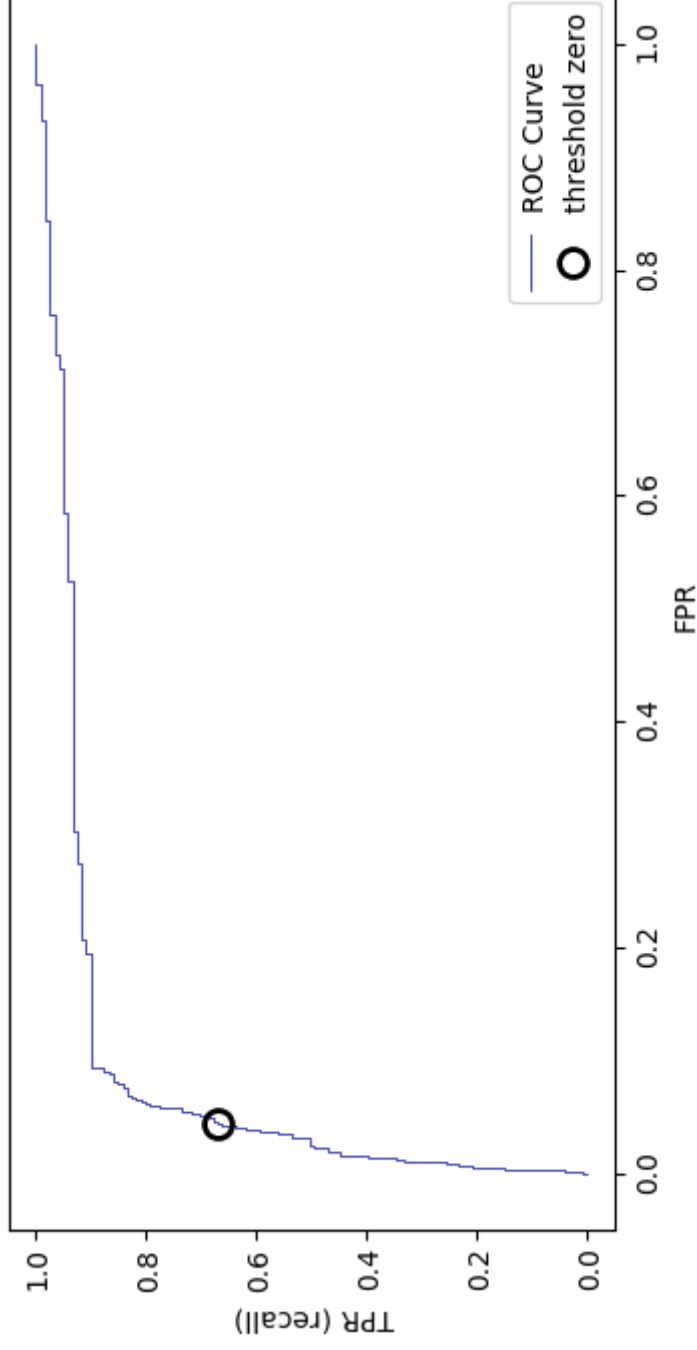
- The area under the precision-recall curve (AUPRC) is often used as a general evaluation measure
- This is a good general measure, but also hides the subtleties we saw in the curve

Average precision of random forest: 0.660

Average precision of svc: 0.666

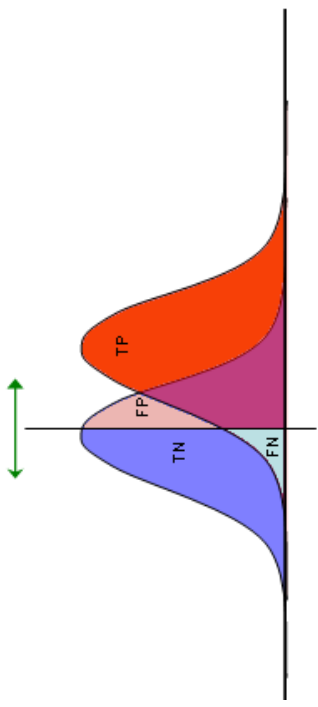
Receiver Operating Characteristics (ROC)

- We can also trade off recall (or true positive rate) $TPR = \frac{TP}{TP+FN}$ with *false positive rate* $FPR = \frac{FP}{FP+TN}$
- Varying the decision threshold yields the ROC curve
 - Lower the threshold to gain more recall (move right)
 - Increase the threshold to reduce FPs (move left)

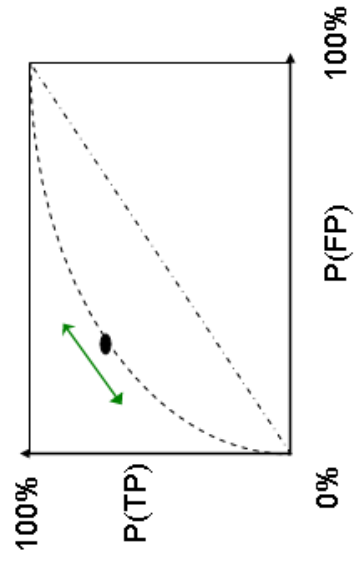


Visualization

- Horizontal axis represents the decision function. Every predicted point is on it.
- The blue probability density shows the actual negative points. The red one is for the positive points.
- Vertical line is the decision threshold: every point to the left is predicted negative (TN or FN) and vice versa (TP or FP).
- Increase threshold: fewer FP and TP: point on ROC curve moves leftward

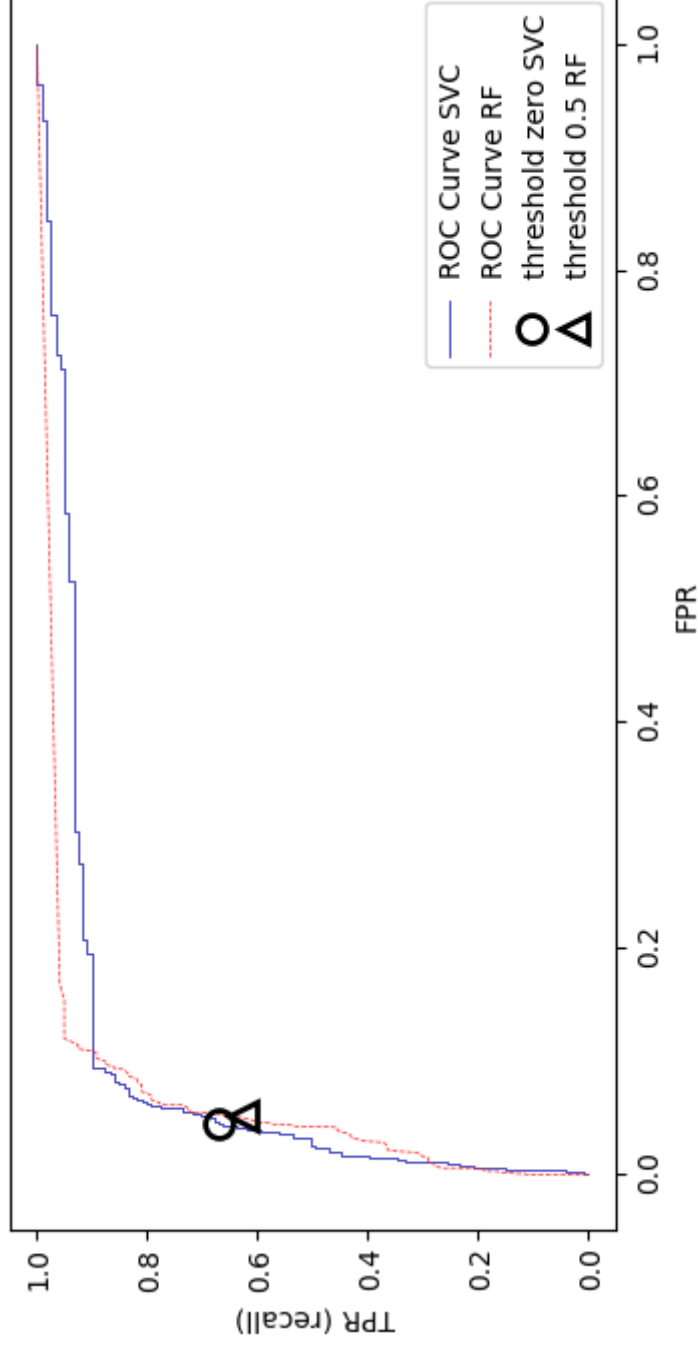


TP	FP
FN	TN
1	1



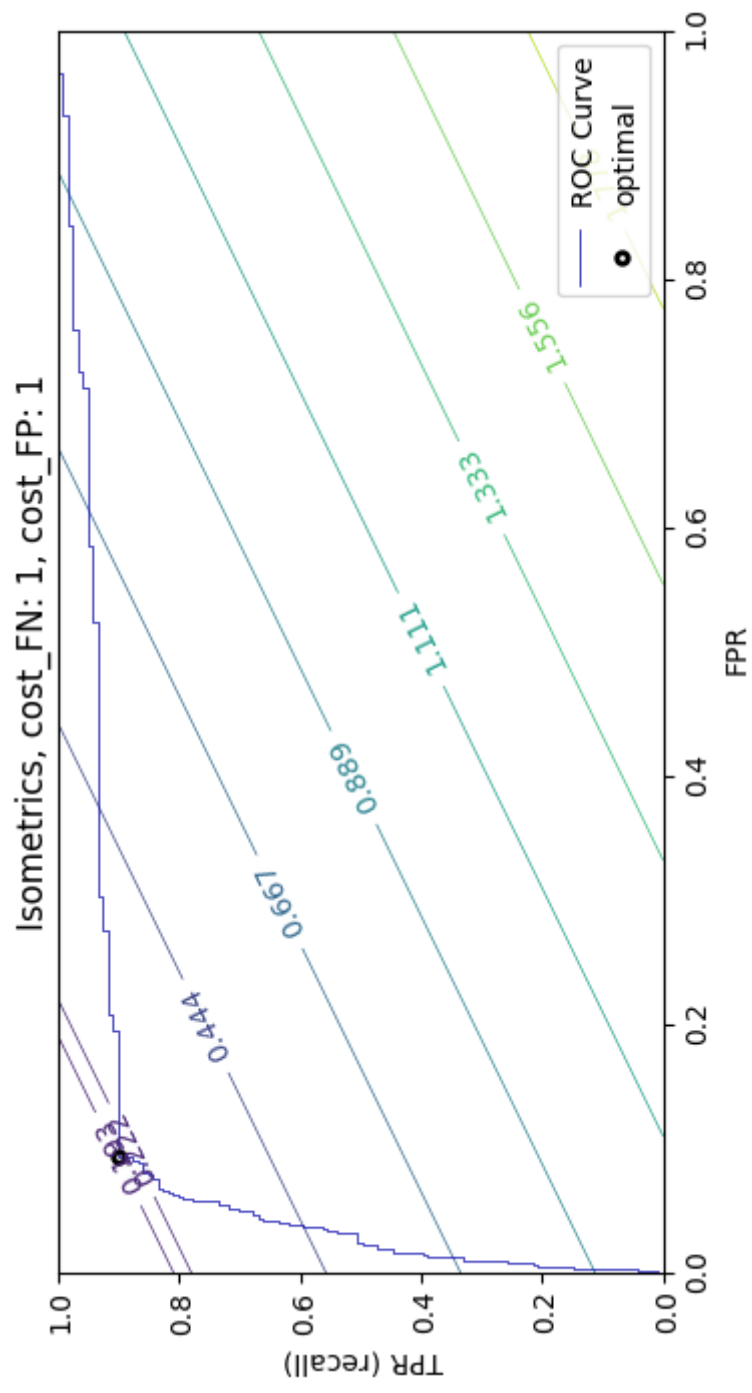
Model selection

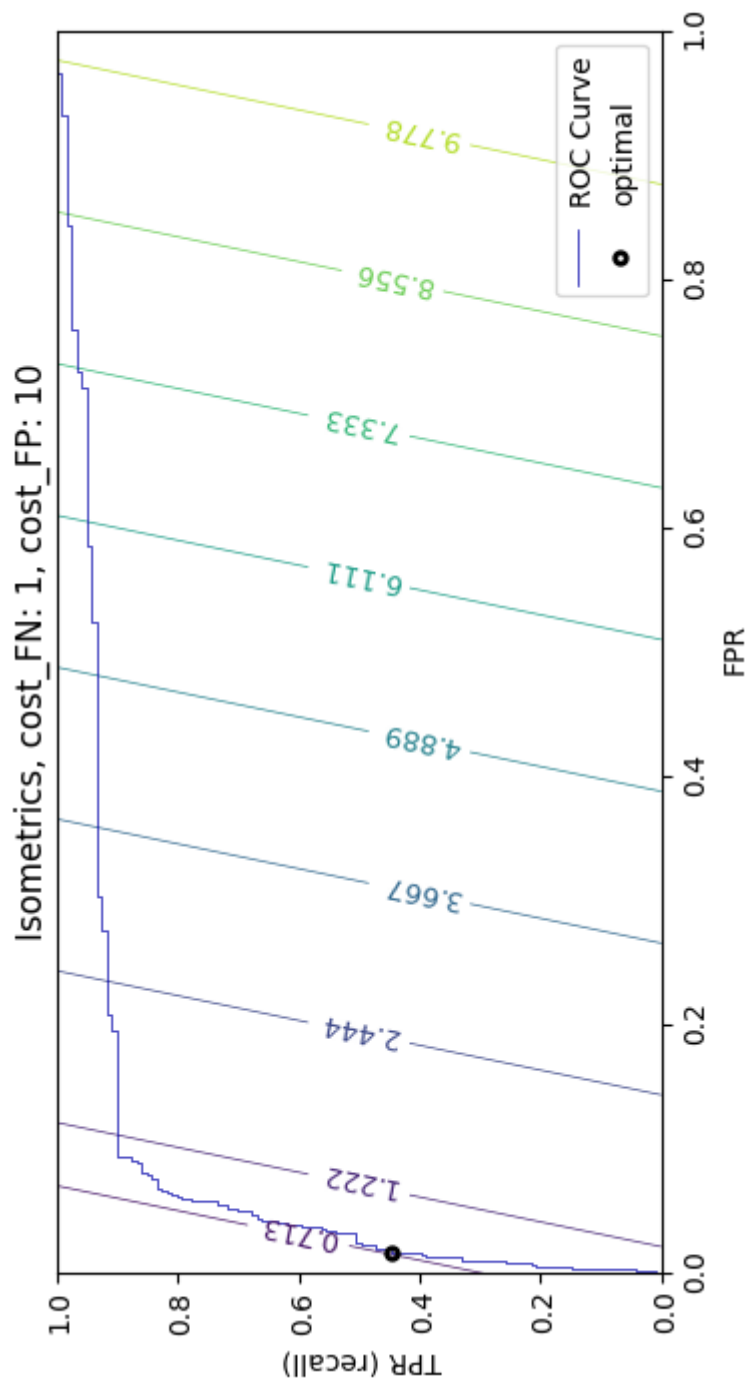
- Again, we can compare multiple models by looking at the ROC curves
- We can calibrate the threshold depending on whether we need high recall or low FPR
- We can select between algorithms (or hyperparameters) depending on the involved *costs*



Calculating costs

- A certain amount of FP and FN can be translated to a *cost*:
total cost = $FPR * cost_{FP} + (1 - TPR) * cost_{FN}$
- 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





Area under the ROC curve

- A useful summary measure is the area under the ROC curve (AUROC or AUC)
- Key benefit: 'sensitive' to class imbalance
 - Random guessing always yields $TPR=FPR$
 - All points are on the diagonal line, hence an AUC of 0.5

AUC for Random Forest: 0.937

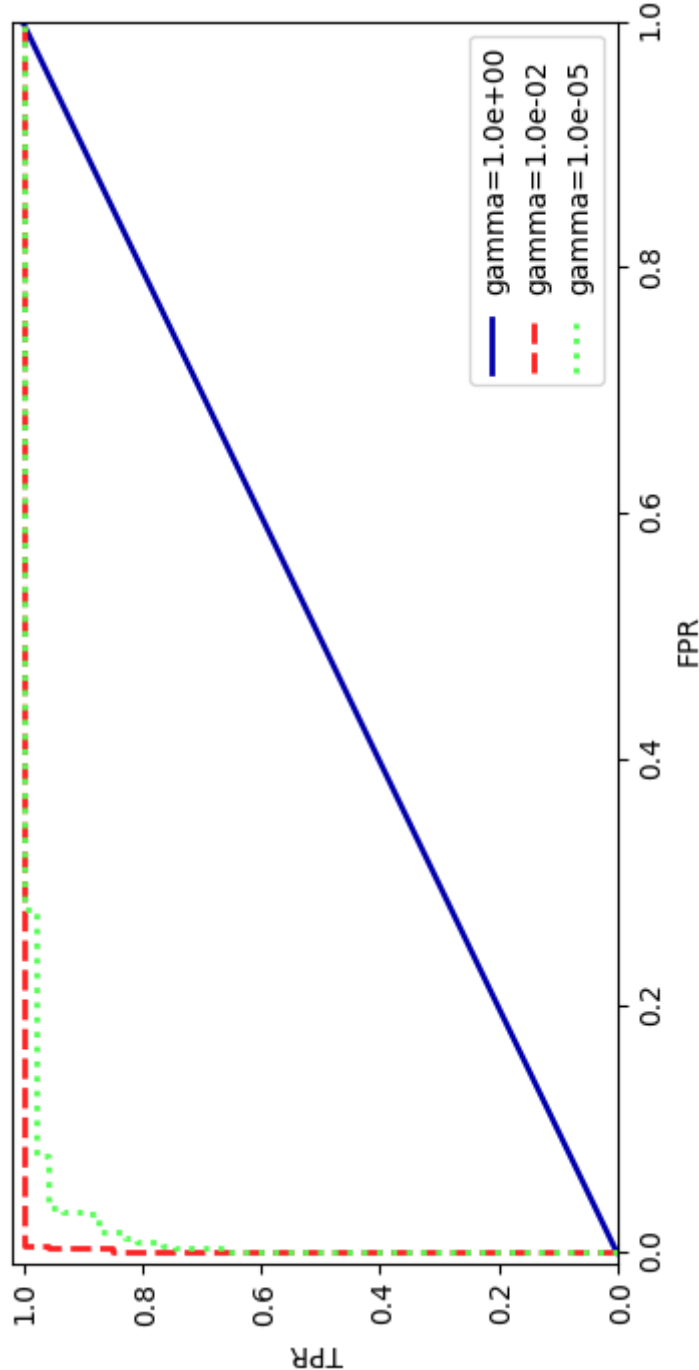
AUC for SVC: 0.916

AUC for dummy classifier: 0.514

Example: unbalanced dataset (10% positive, 90% negative):

- 3 models: overfitting ($\gamma = 1.0$), good ($\gamma = 0.01$), underfitting ($\gamma=1e-5$)
- ACC is the *same* (we might be random guessing), AUC is more informative

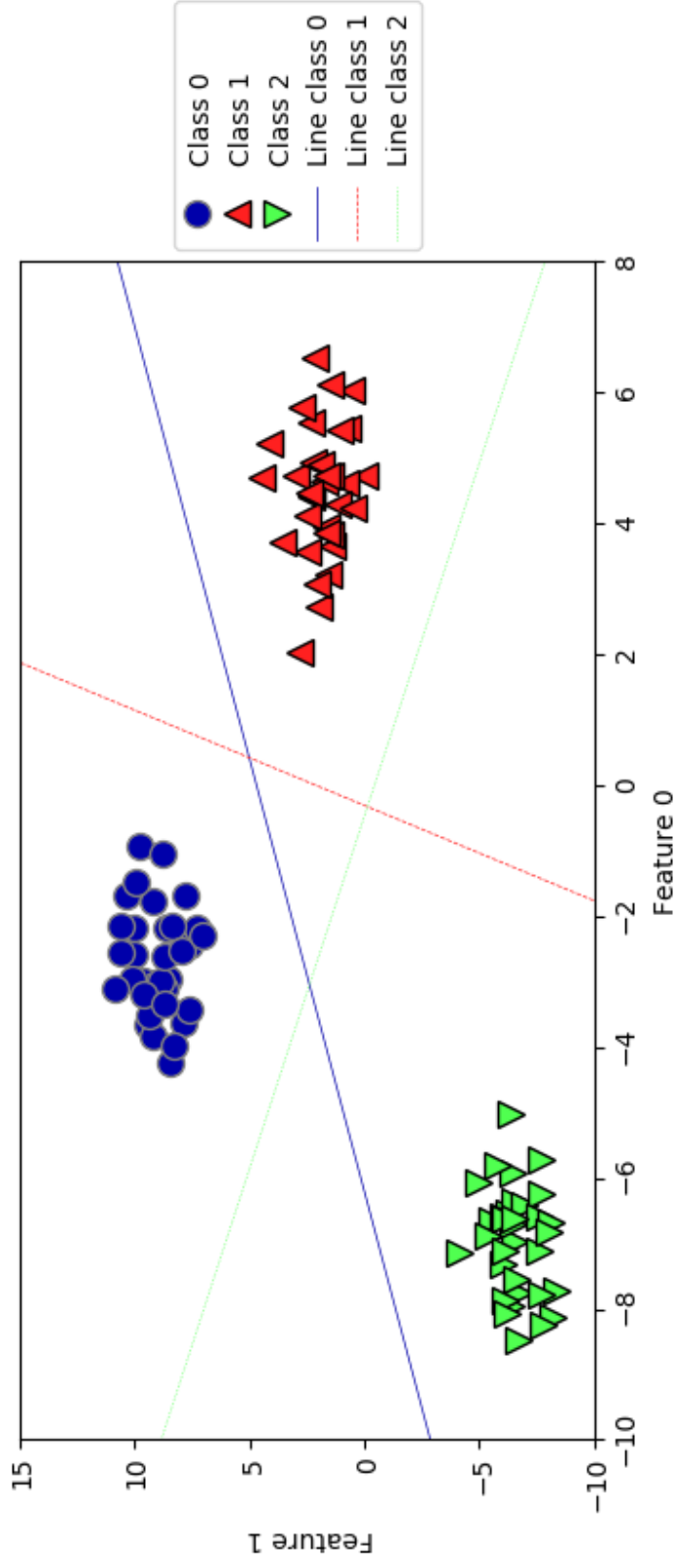
gamma = 1.0e+00 ACC = 0.90 AUC = 0.5000
gamma = 1.0e-02 ACC = 0.90 AUC = 0.9995
gamma = 1.0e-05 ACC = 0.90 AUC = 0.9882



Multiclass classification

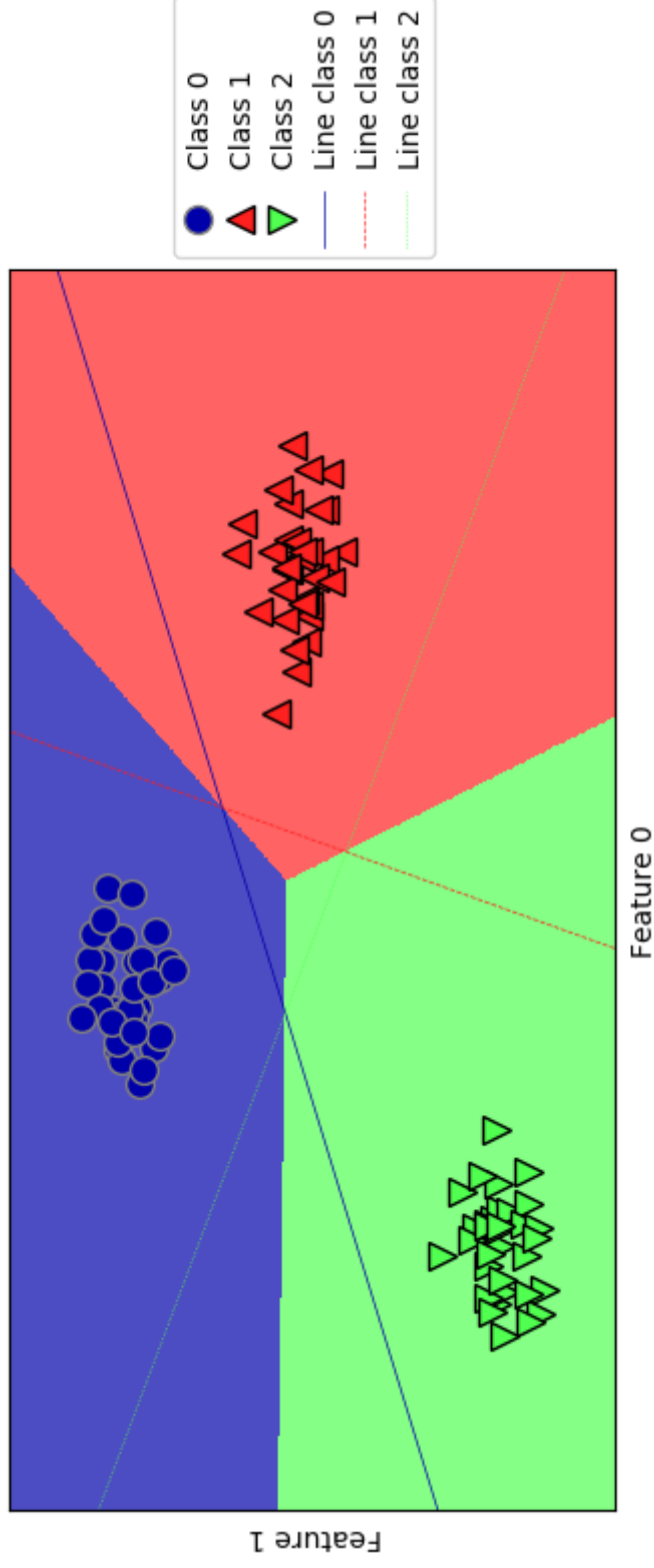
Common technique: one-vs-rest approach:

- A binary model is learned for each class vs. all other classes
- Creates as many binary models as there are classes



Every binary classifiers makes a prediction

- The confidence (decision score) of that prediction is the confidence in that class
- The class with the highest decision score (>0) wins
- Decision boundaries visualized below



Uncertainty in multi-class classification

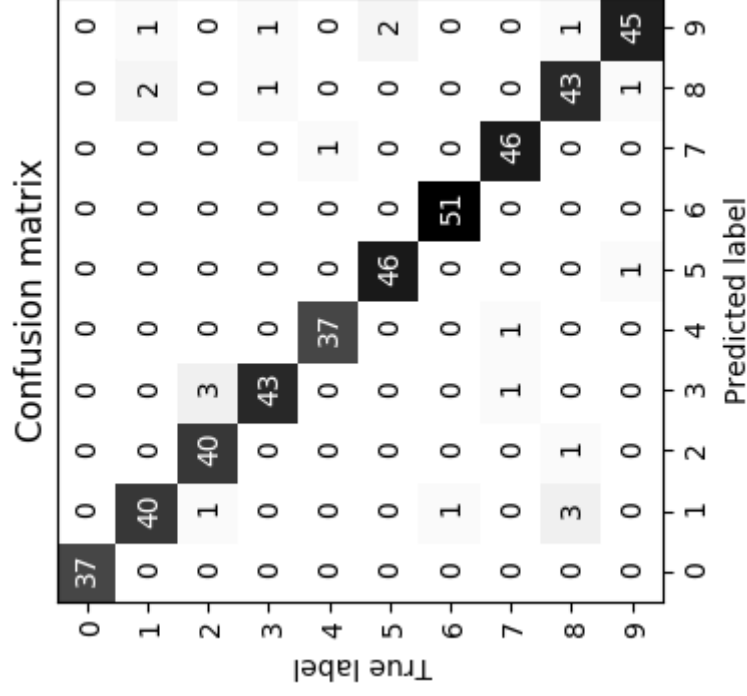
- `decision_function` and `predict_proba` also work in the multiclass setting
 - always have shape `(n_samples, n_classes)`
 - Example on the Iris dataset, which has 3 classes:

```
Decision function:
[[ -3.035   2.294   0.741]
 [  5.919   3.091  -9.01 ]
 [-10.052   1.875   8.177]
 [ -2.733   2.036   0.697]
 [ -3.737   2.476   1.262]
 [  6.036   3.035  -9.07 ]]

Predicted probabilities:
[[0.004 0.822 0.174]
 [0.944 0.056 0.   ]
 [0.   0.002 0.998]
 [0.007 0.787 0.206]
 [0.002 0.77  0.229]
 [0.953 0.047 0.   ]]
```


Multi-class metrics

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



Precision, recall, F1-score now yield 10 per-class scores

	precision	recall	f1-score	support
0	1.00	1.00	1.00	37
1	0.89	0.93	0.91	43
2	0.98	0.91	0.94	44
3	0.91	0.96	0.93	45
4	0.97	0.97	0.97	38
5	0.98	0.96	0.97	48
6	1.00	0.98	0.99	52
7	0.98	0.96	0.97	48
8	0.91	0.90	0.91	48
9	0.90	0.96	0.93	47
accuracy			0.95	450
macro avg	0.95	0.95	0.95	450
weighted avg	0.95	0.95	0.95	450

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): $\frac{\sum_{i=0}^n score_i \cdot weight_i}{n}$
 - 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

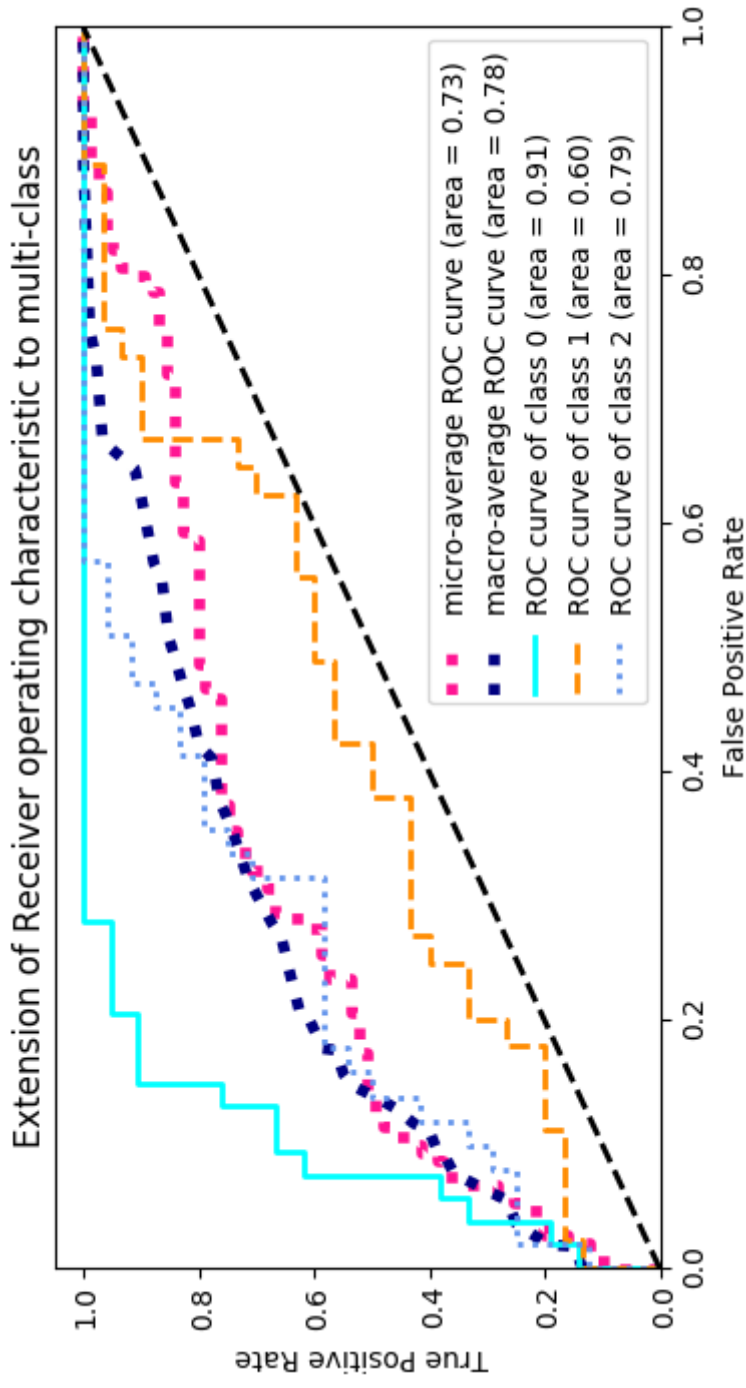
Micro average f1 score: 0.951

Weighted average f1 score: 0.951

Macro average f1 score: 0.952

Multi-class ROC

- To use AUC in a multi-class setting, you need to choose whether you use a micro- or macro average TPR and FPR.
- Depends on the application: is every class equally important?
 - SKlearn currently doesn't have a default option



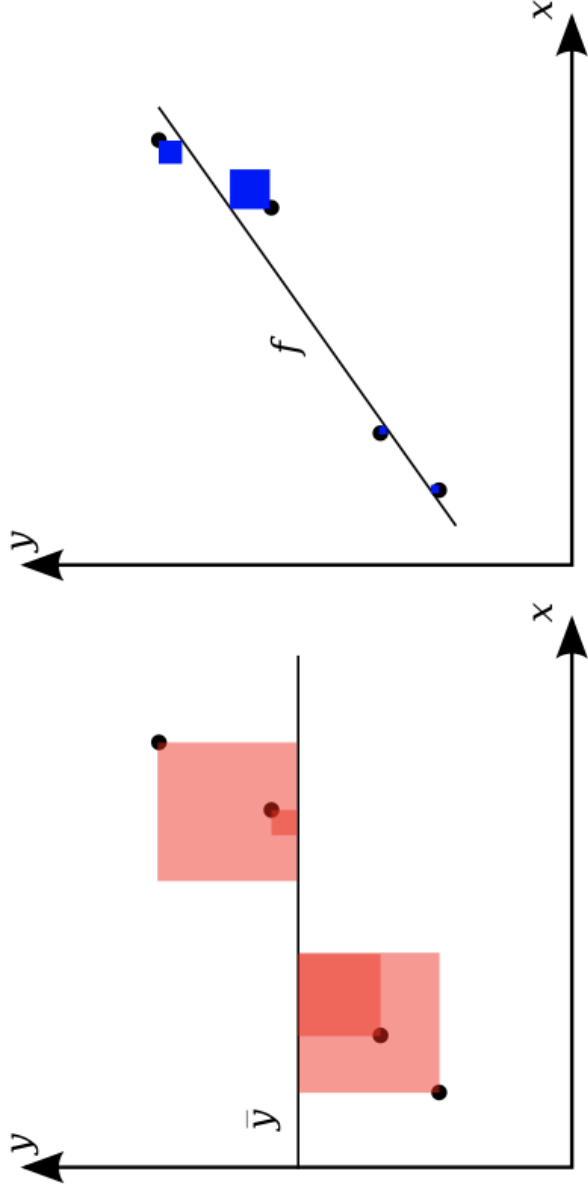
Regression metrics

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

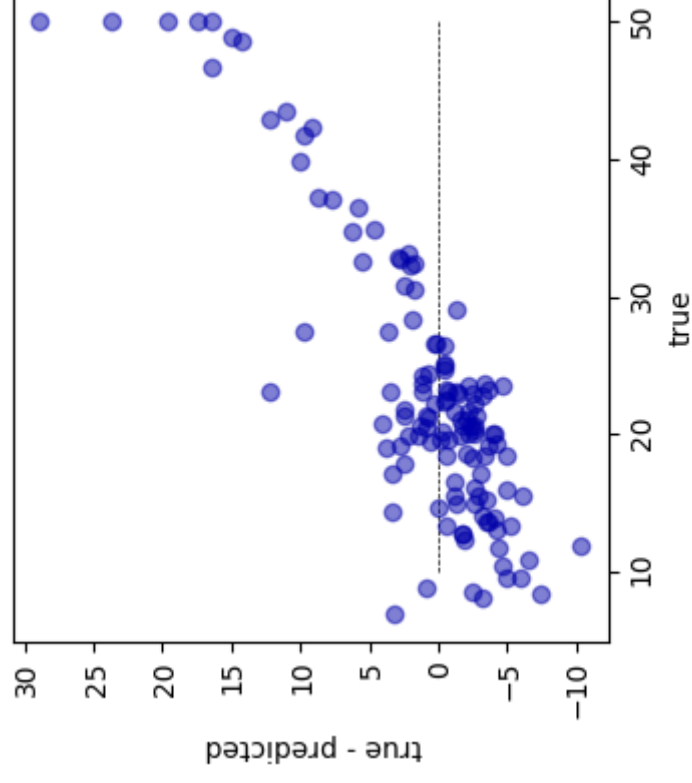
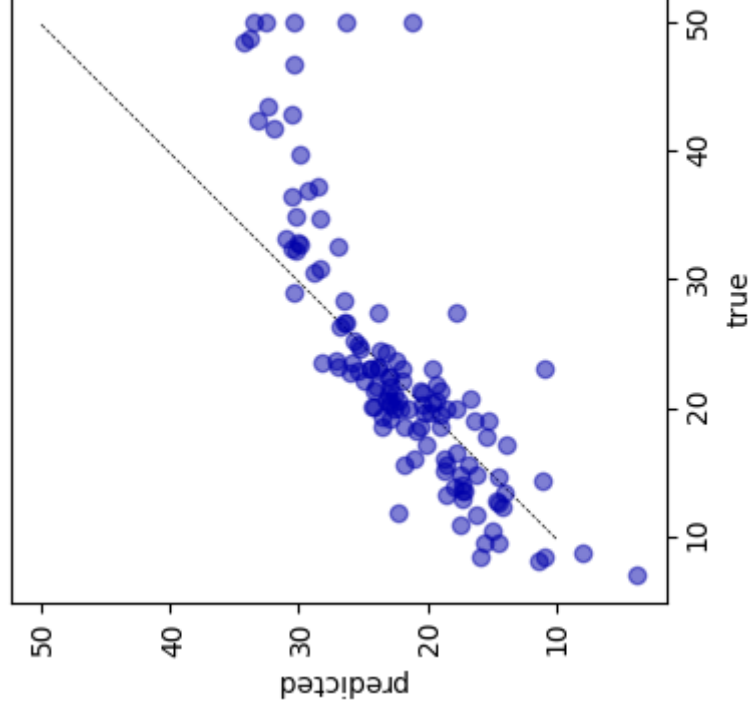


- R squared (r2): $1 - \frac{\sum_i (\mathcal{Y}_{pred_i} - \mathcal{Y}_{actual_i})^2}{\sum_i (\mathcal{Y}_{mean} - \mathcal{Y}_{actual_i})^2}$
 - Ratio of variation explained by the model / total variation
 - Between 0 and 1, but *negative* if the model is worse than just predicting the mean
 - Easier to interpret (higher is better).



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



Other considerations

- There exist techniques to correct label imbalance (see next lecture)
 - Undersample the majority class, or oversample the minority class
 - SMOTE (Synthetic Minority Oversampling TEchnique) adds artificial *training* points by interpolating existing minority class points
 - Think twice before creating 'artificial' training data
- Cost-sensitive classification (not in sklearn)
 - *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)

- There are many more metrics to choose from
 - Balanced accuracy: accuracy where each sample is weighted according to the inverse prevalence of its true class
 - Identical to macro-averaged recall
 - 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:

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

- Matthews correlation coefficient: another measure that can be used on imbalanced data
 - 1: perfect prediction, 0: random prediction, -1: inverse prediction

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

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

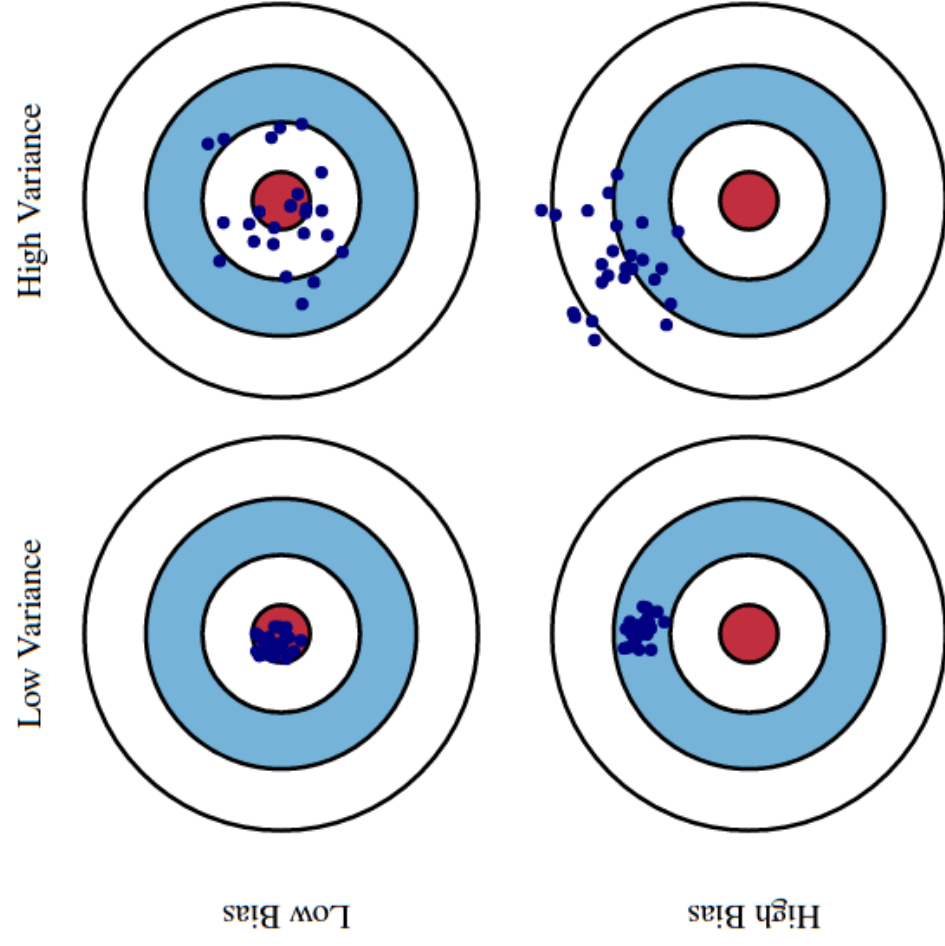


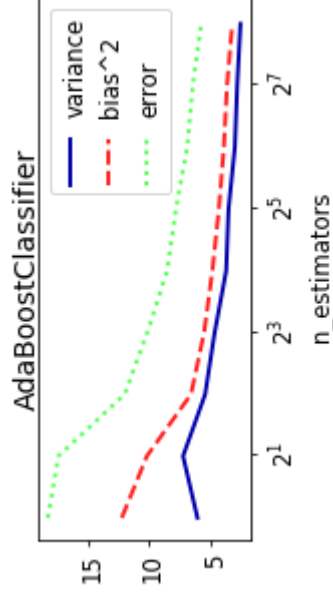
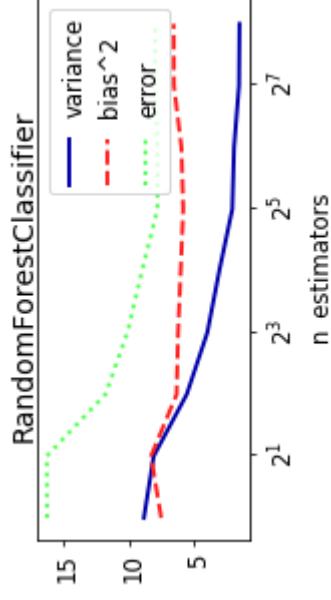
Fig. 1 Graphical illustration of bias and variance.

Computing bias-variance

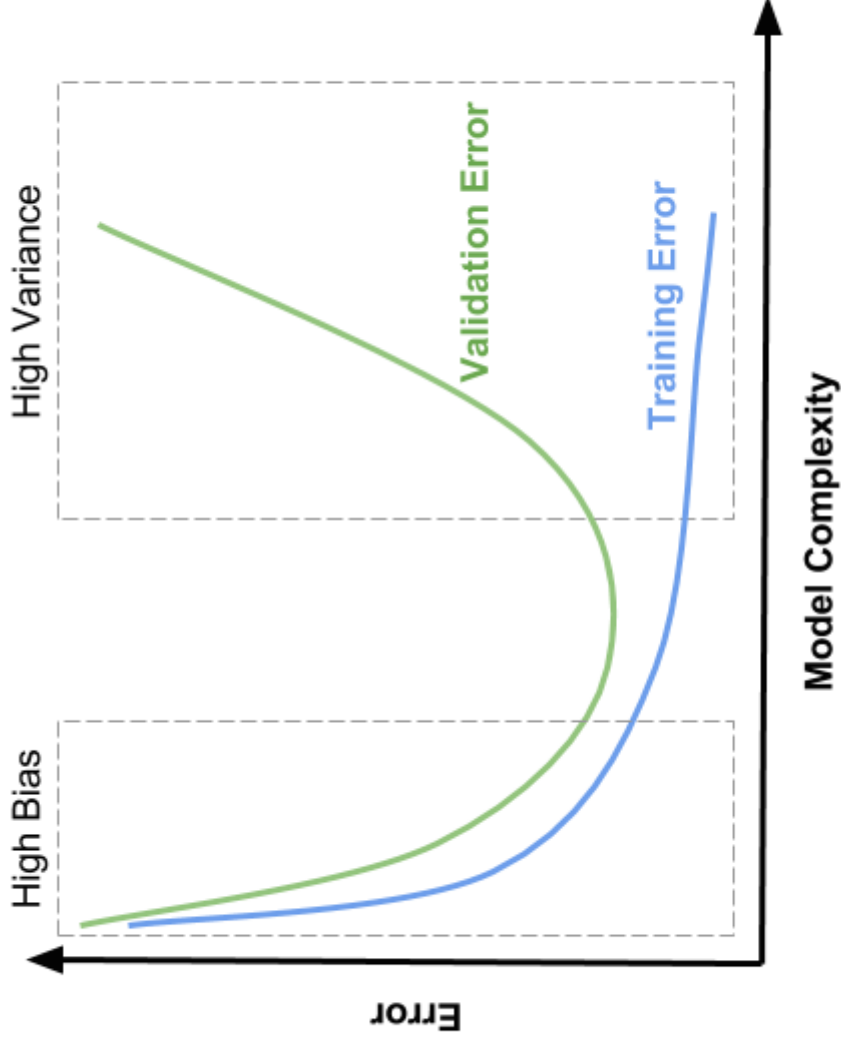
- Take 100 or more bootstraps (or shuffle-splits)
- Regression: for each data point x :
 - $bias(x)^2 = (x_{true} - mean(x_{predicted}))^2$
 - $variance(x) = var(x_{predicted})$
- Classification: for each data point x :
 - $bias(x)$ = misclassification ratio
 - $variance(x)$
$$= (1 - (P(class_1)^2 + P(class_2)^2))/2$$
 - $P(class_i)$ is ratio of class i predictions
- Total bias: $\sum_x bias(x)^2 * w_x$, with w_x the ratio of x occurring in the test sets
- Total variance: $\sum_x variance(x) * w_x$

Bias and variance reduction

- Bagging (RandomForests) is a variance-reduction technique
- Boosting is a bias-reduction technique

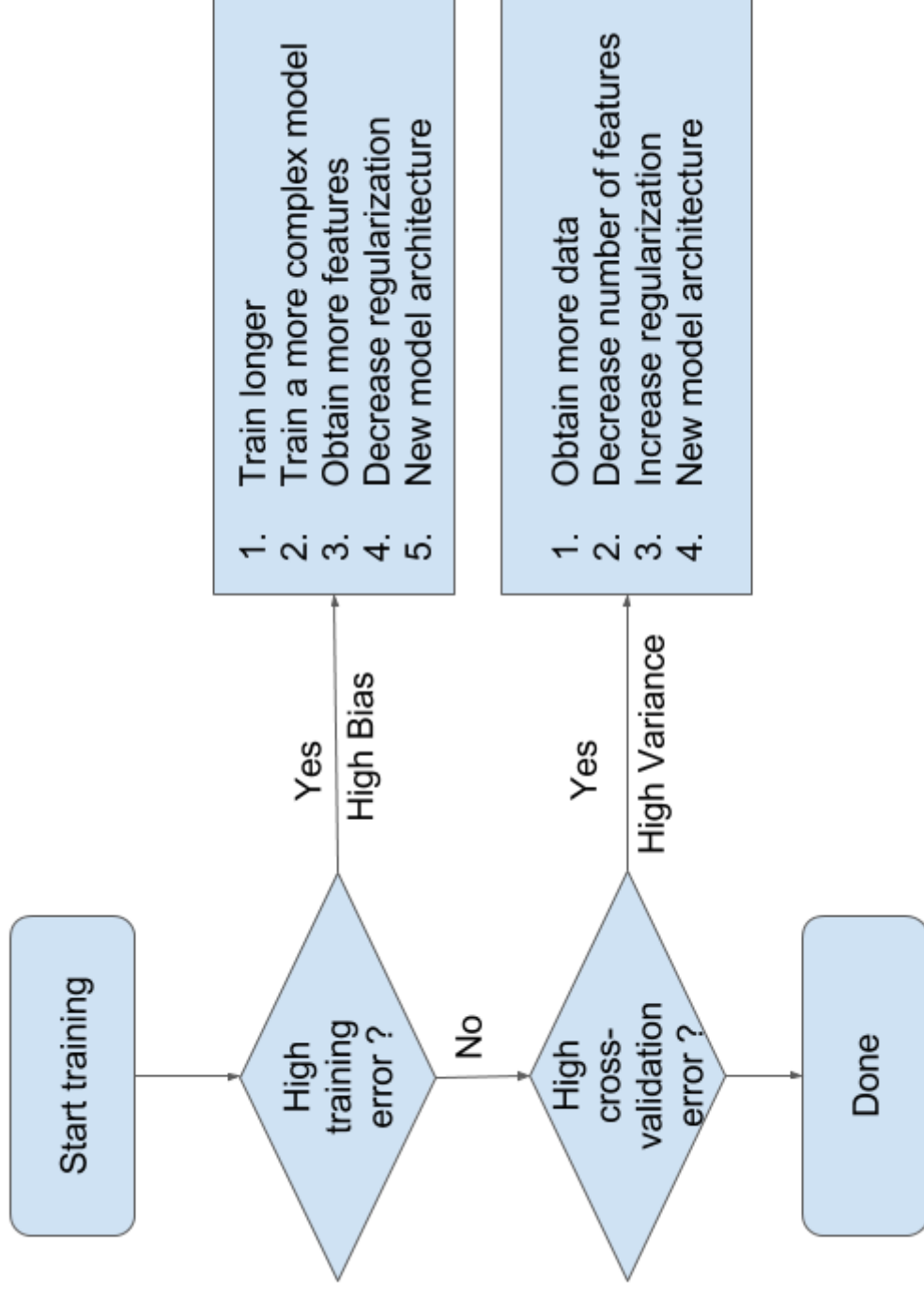


Bias-variance and overfitting



- 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)
- 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)

Bias-Variance Flowchart (Andrew Ng, Coursera)



Hyperparameter tuning

We can basically use any optimization technique to optimize hyperparameters:

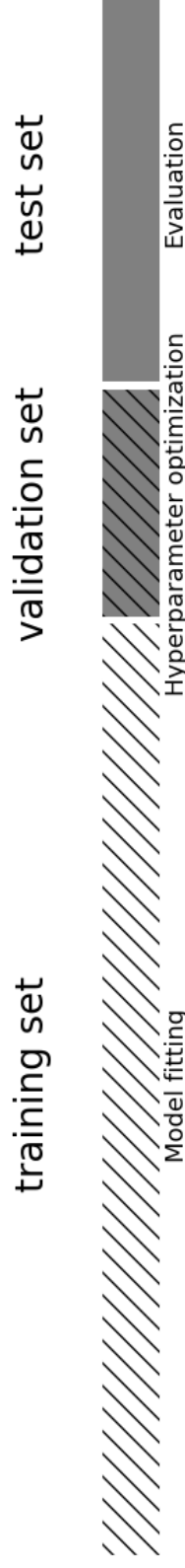
- **Grid search**
- **Random search**

More advanced techniques (see lecture 7):

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

Overfitting on the test set

- Simply taking the best performing hyperparameters yields optimistic results
- We've already used the test data to evaluate each hyperparameter setting!
 - Information 'leaks' from test set into the final model
- 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

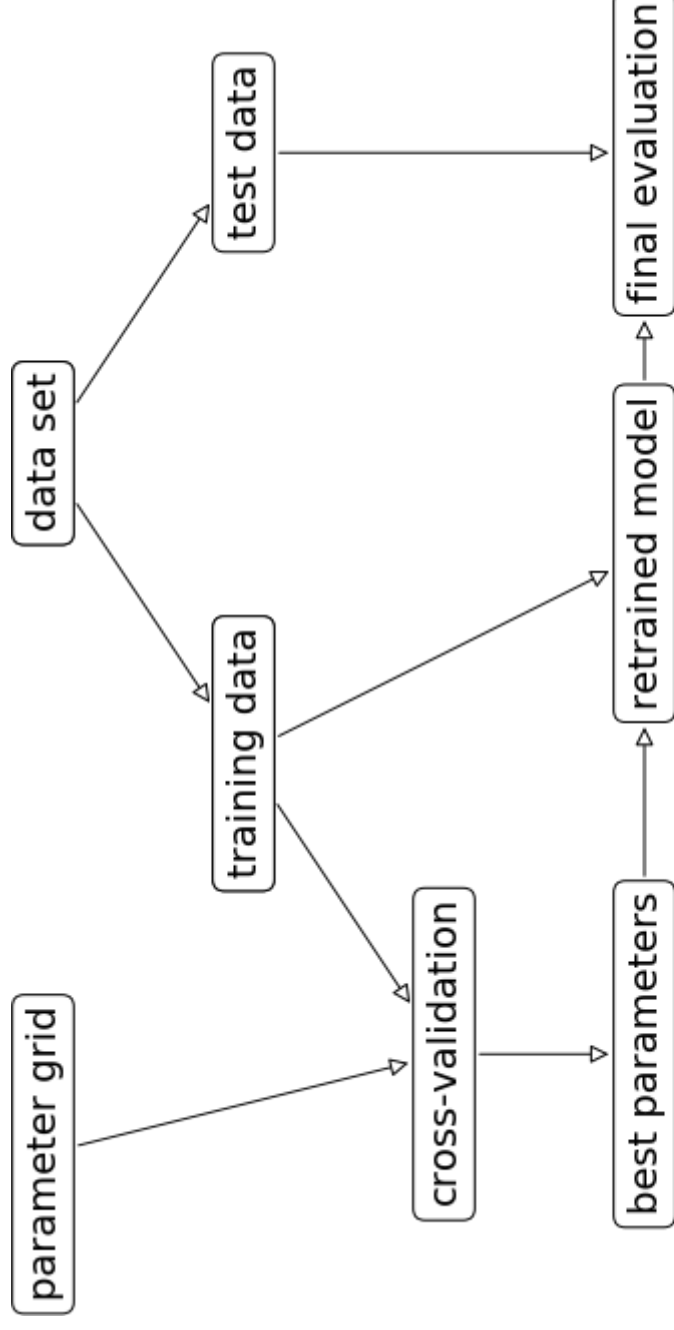


Size of training set: 84 size of validation set: 28 size of test set: 38

Best score on validation set: 0.96
Best parameters: {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92

Grid-search with cross-validation

- The way that we split the data into training, validation, and test set may have a large influence on estimated performance
- We need to use cross-validation again (e.g. 3-fold CV), instead of a single split



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 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

```
scores = cross_val_score(GridSearchCV(SVC(), param_grid, cv=5),  
                           iris.data, iris.target, cv=5)
```

```
Cross-validation scores: [0.967 1.    0.9   0.967 1.    ]  
Mean cross-validation score: 0.9666666666666666
```