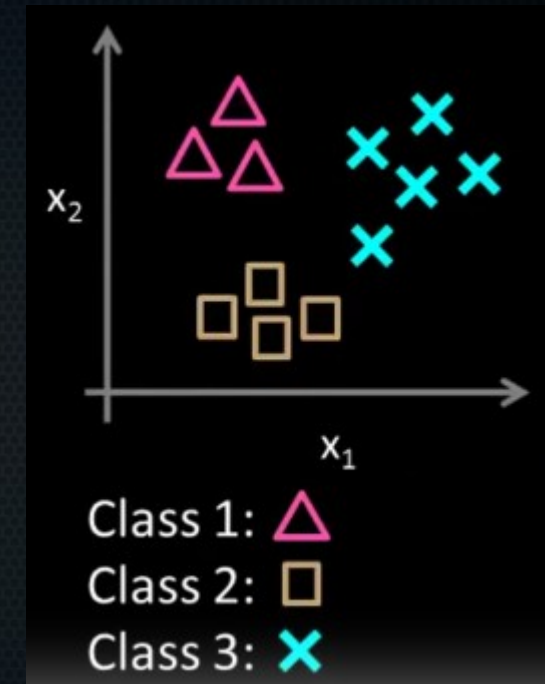


This presentation

- Dealing with > 2 classes
- ROC curves and ROC AUC
- Regularisation
- Getting hyperparameters

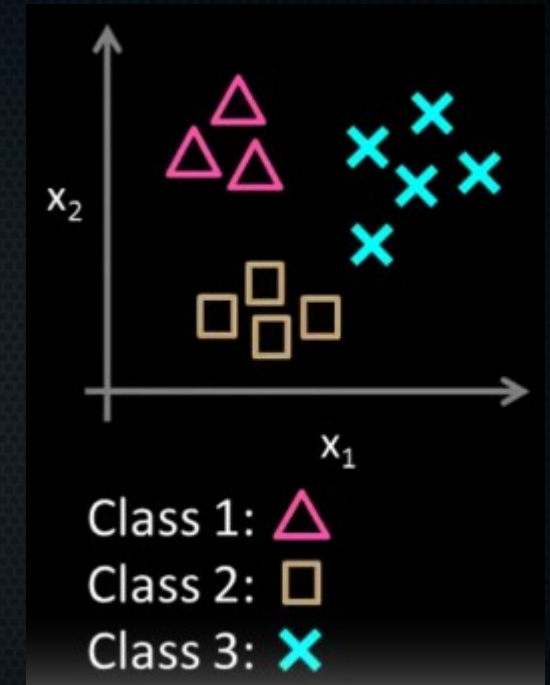
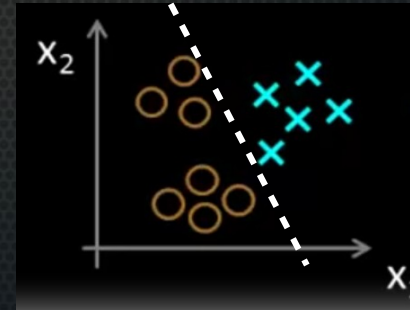
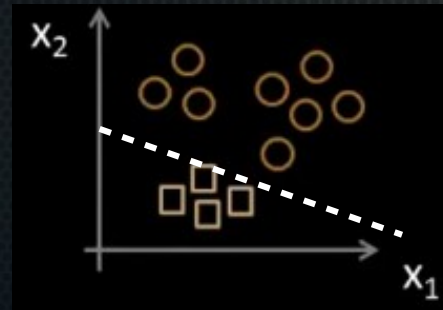
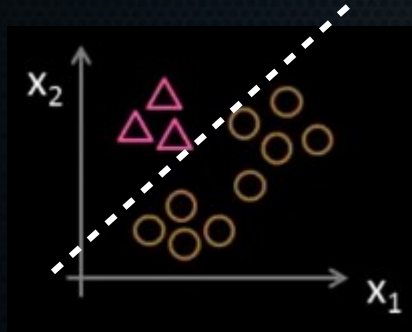
What if we have more than one class?

- What do you think we could do?



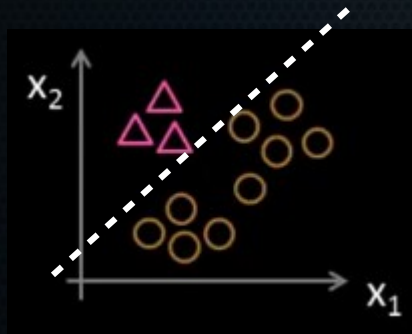
What if we have more than one class?

- What do you think we could do?
- Train a separate binary classifier for each instance:



What if we have more than one class?

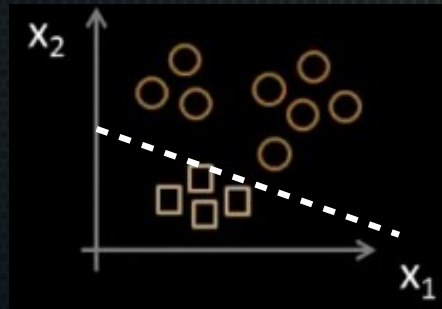
- What do you think we could do?
- Train a separate binary classifier for each instance:



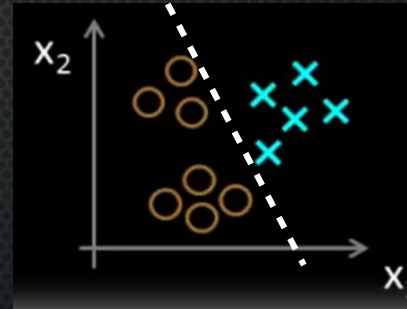
$$h_{\theta}^{(1)}(x)$$

$$P(y=i|x;\theta)$$

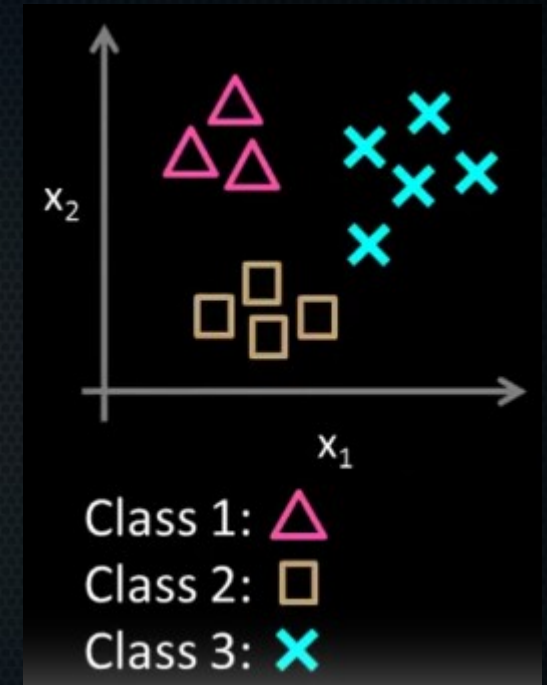
For i classes, train i binary classifiers to predict that the point is class i given the data



$$h_{\theta}^{(2)}(x)$$



$$h_{\theta}^{(3)}(x)$$



How well are we doing?

- We can't use an R-squared here. So what can we use?
- Idea: accuracy.

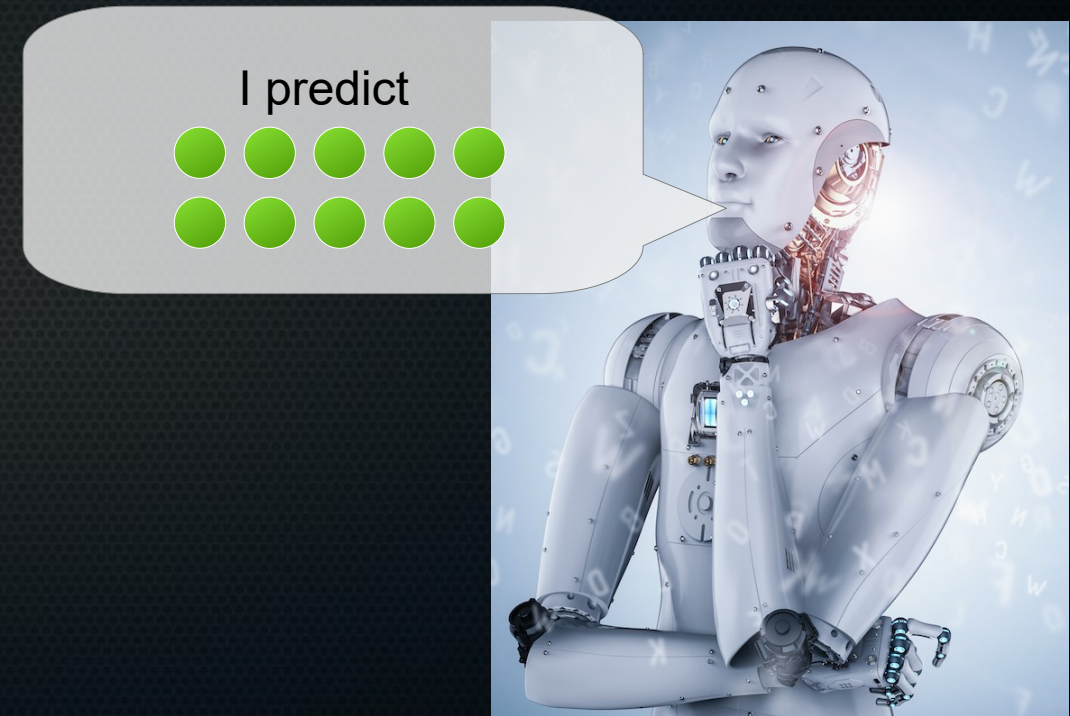
How well are we doing?

- What can we use?
- Idea: accuracy.



How well are we doing?

- What can we use?
- Idea: accuracy.

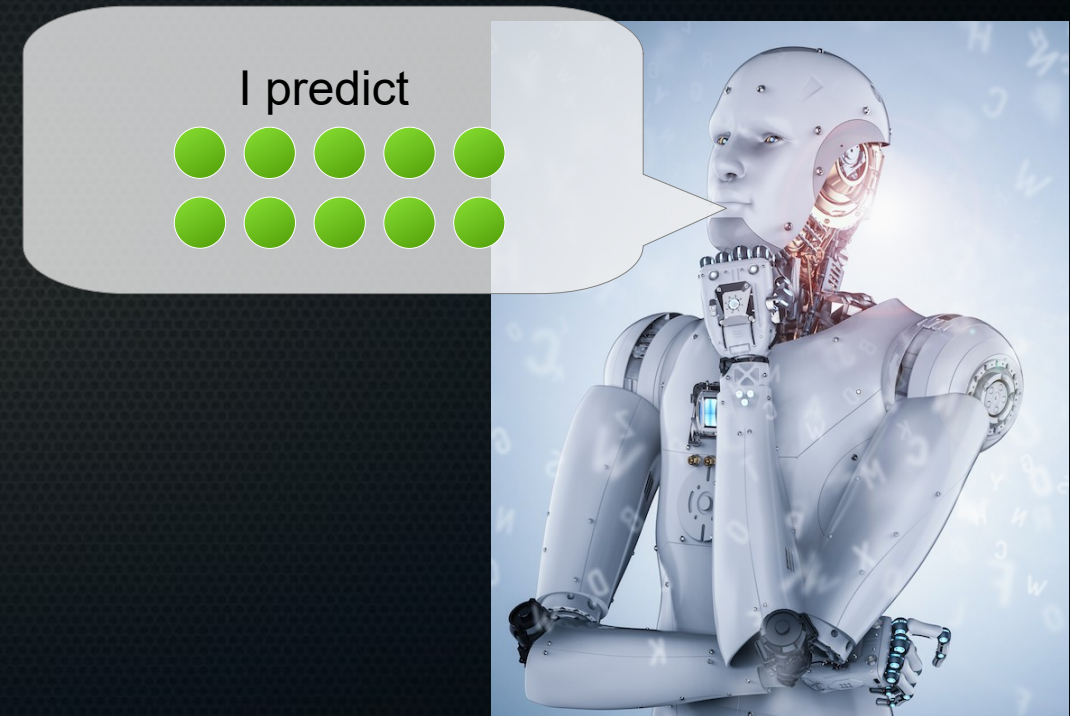


How well are we doing?

- What can we use?
- Idea: accuracy.



- Accuracy = 90% (9/10 correct).
→ seems pretty good!

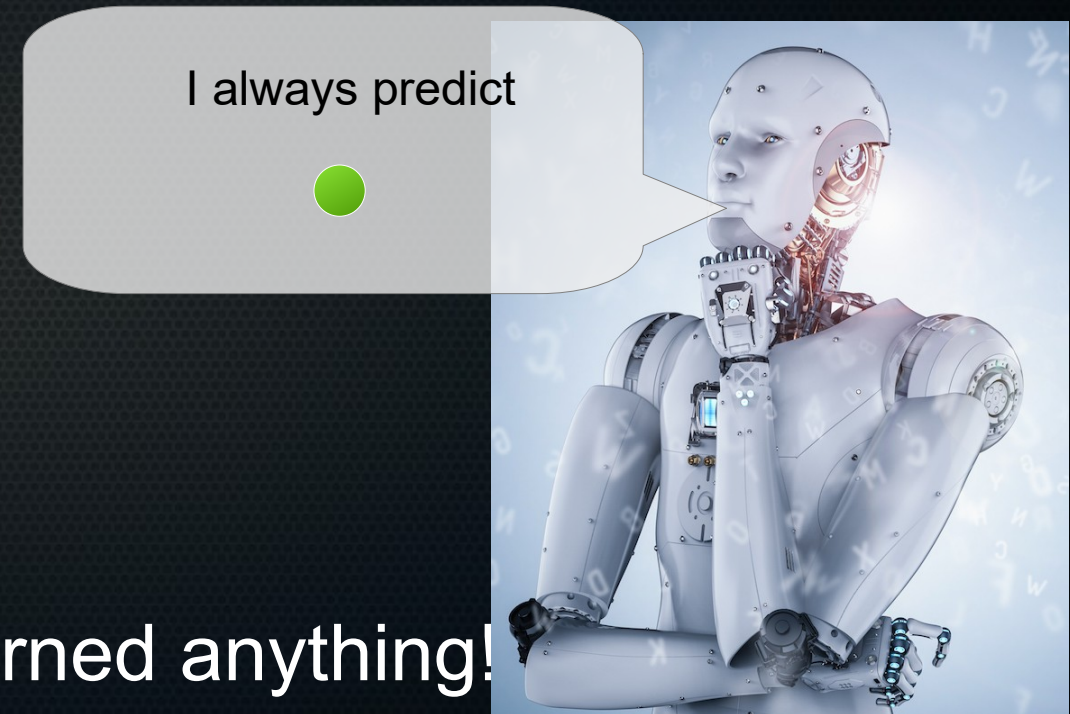


How well are we doing?

- What can we use?
- Idea: accuracy.



- Accuracy = 90% (9/10 correct).
→ Lucky break! Classifier hasn't learned anything!



Need something else to measure performance

- Have 4 types of predictions:

		<u>Reality</u>	
		Cancer	Not Cancer
<u>Model Prediction</u>	Cancer	True Positive	False Positive
	Not Cancer	False Negative	True Negative

Need something else to measure performance

- Have 4 types of predictions:
- We want to know both how many true positives we pick out from the test data (sensitivity) and how many true negatives we correctly classify as negative (specificity).

		<u>Reality</u>	
		Cancer	Not Cancer
<u>Model Prediction</u>	Cancer	True Positive	False Positive
	Not Cancer	False Negative	True Negative

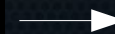
Need something else to measure performance

- Have 4 types of predictions:
- We want to know both how many true positives we pick out from the test data (sensitivity) and how many true negatives we correctly classify as negative (specificity).

		<u>Reality</u>	
		Cancer	Not Cancer
<u>Model Prediction</u>	Cancer	True Positive	False Positive
	Not Cancer	False Negative	True Negative

Sensitivity (true positive rate)

$$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



What proportion of positives in the data do we correctly predict?

Specificity (true negative rate)

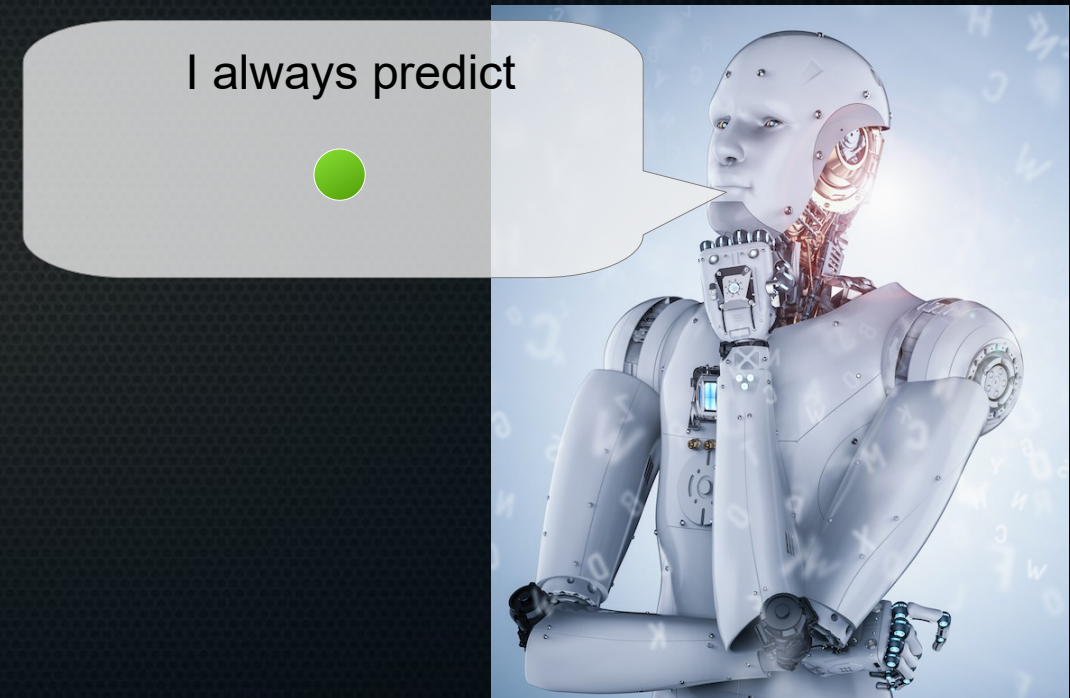
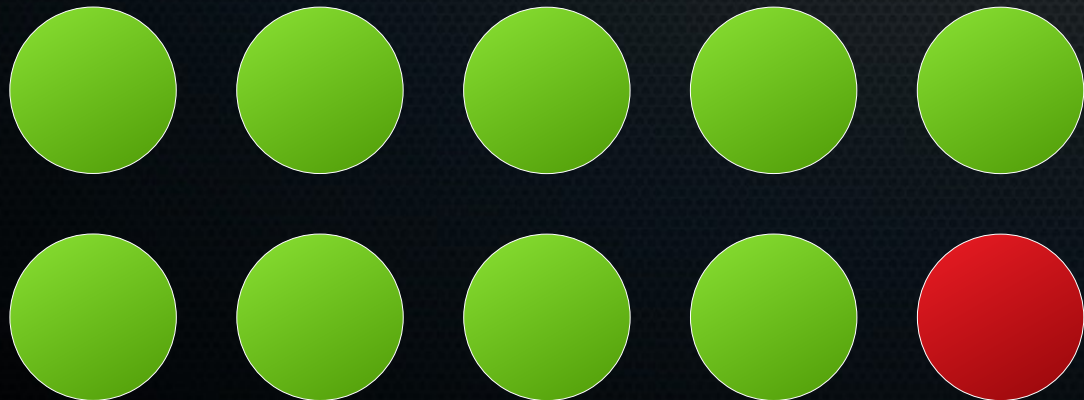
$$\frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$



What proportion of negatives in the data do we correctly predict?

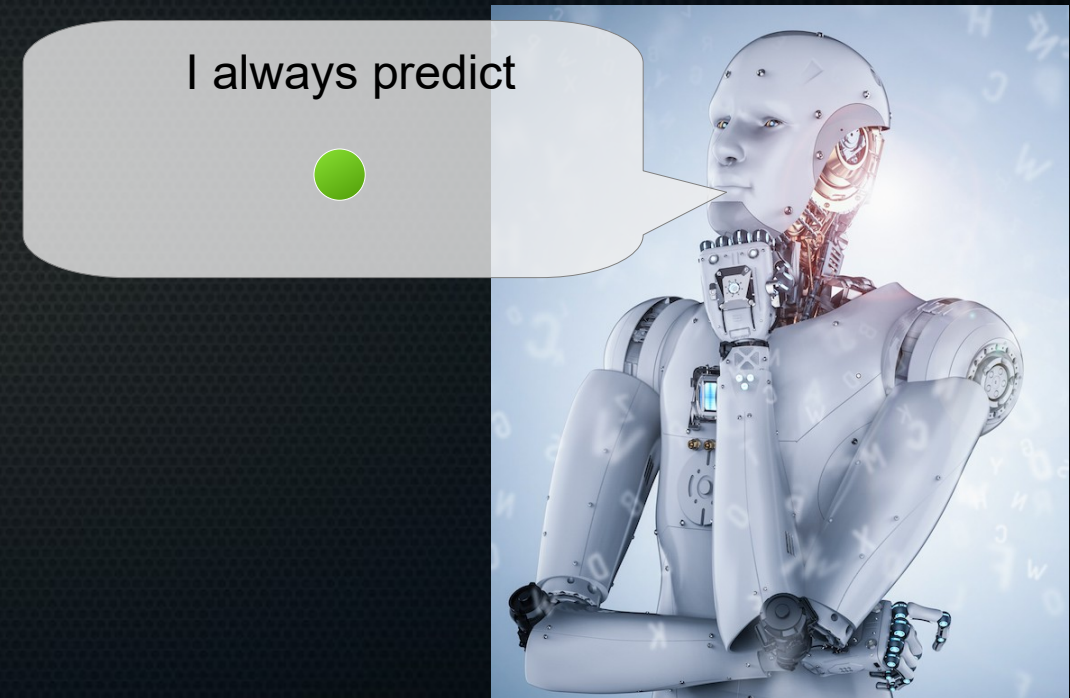
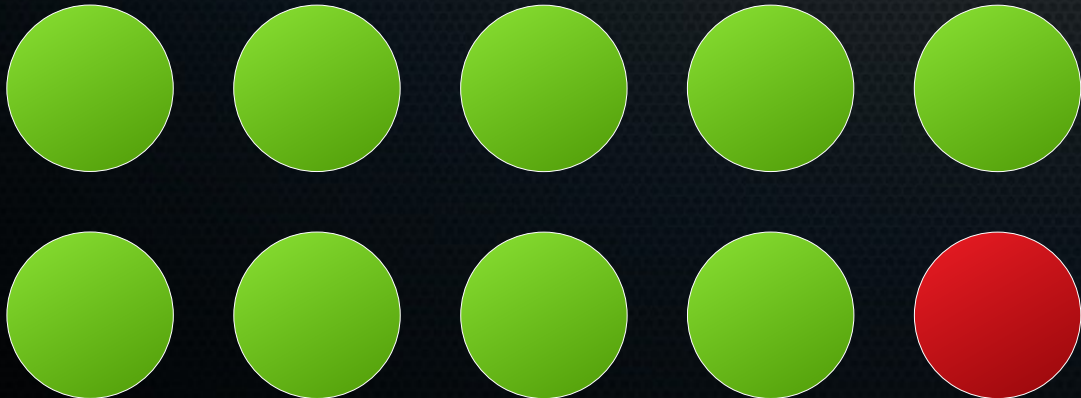
How well are we doing?

- Accuracy = 90% (9/10 correct).
→ Lucky break! Classifier hasn't learned anything!
- Sensitivity = 100%
- Specificity = 0%



How well are we doing?

- Accuracy = 90% (9/10 correct).
→ Lucky break! Classifier hasn't learned anything!
- Sensitivity = 100% } Found all positives
- Specificity = 0% } By having 0 discerning ability



ROC curve

- What is the best balance between sensitivity and specificity?
 - Depends on your application:

Sensitivity (true positive rate)	$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$
Specificity (true negative rate)	$\frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$

Patient data



**What do you care about
most in each case?**



No chemo

Chemotherapy

Patient data



No follow-up
screening

Follow-up
diabetes

ROC curve

- What is the best balance between sensitivity and specificity?

- Depends on your application:

Sensitivity (true positive rate)	$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$
Specificity (true negative rate)	$\frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$

Patient data



Don't want to give devastating chemo treatments unnecessarily: care most about specificity!



No chemo

Chemotherapy

Patient data



Don't want to miss early signs diabetes if follow-up tests will confirm or deny: care most about sensitivity!



No follow-up screening

Follow-up diabetes

ROC curve

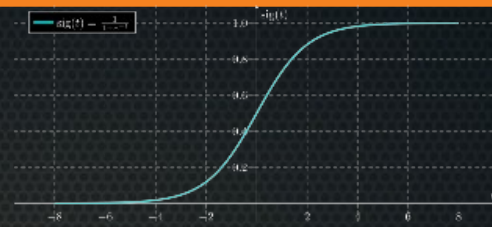
- How do you implement a focus on specificity or sensitivity?

- How do we work with this? $h_{\theta}(x) = \frac{1}{1 + e^{-(\theta^T \cdot x)}}$

- Interpret outcome of $h_{\theta}(x)$ as probability that class = 1 given the features. Example:

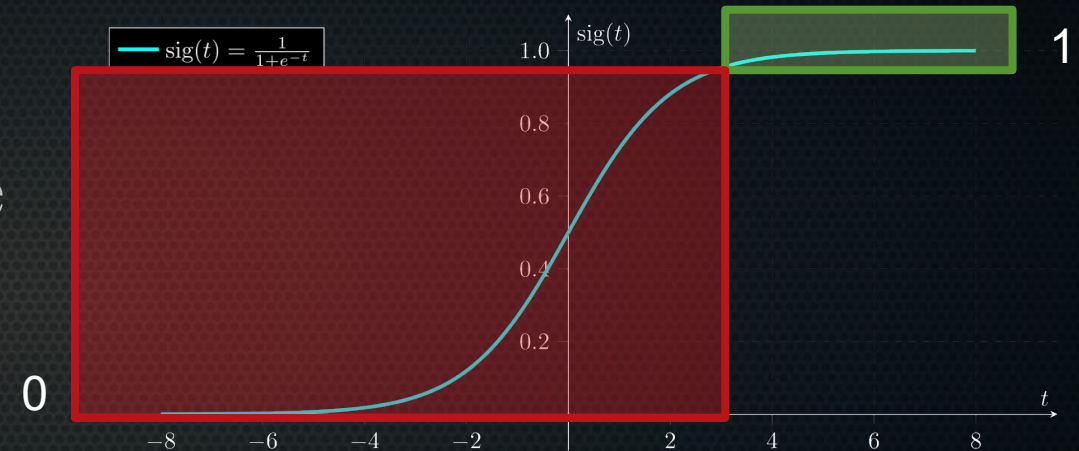
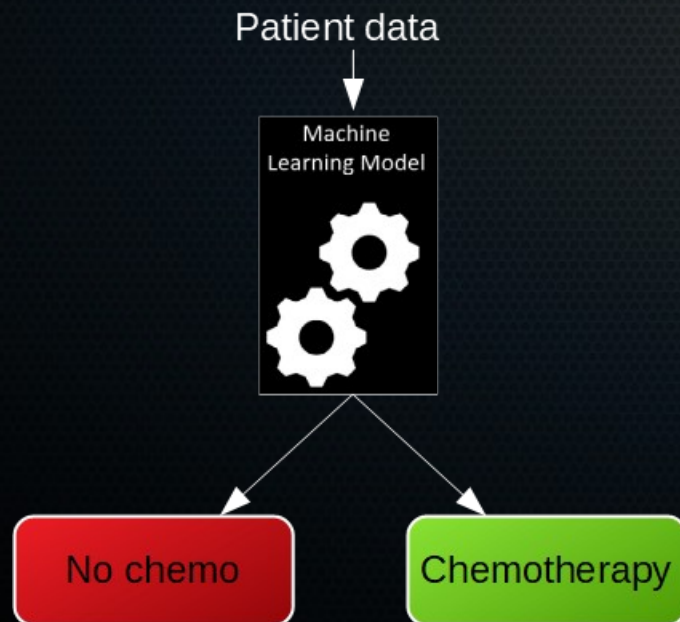
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{Tumor size} \\ \text{Neovascularisation level} \end{bmatrix}$$

$h_{\theta}(x) = 0.8 \longrightarrow$ 80% chance of tumor being malignant (class 1)
100% - 80% \rightarrow 20 % chance of being benign (class 0)



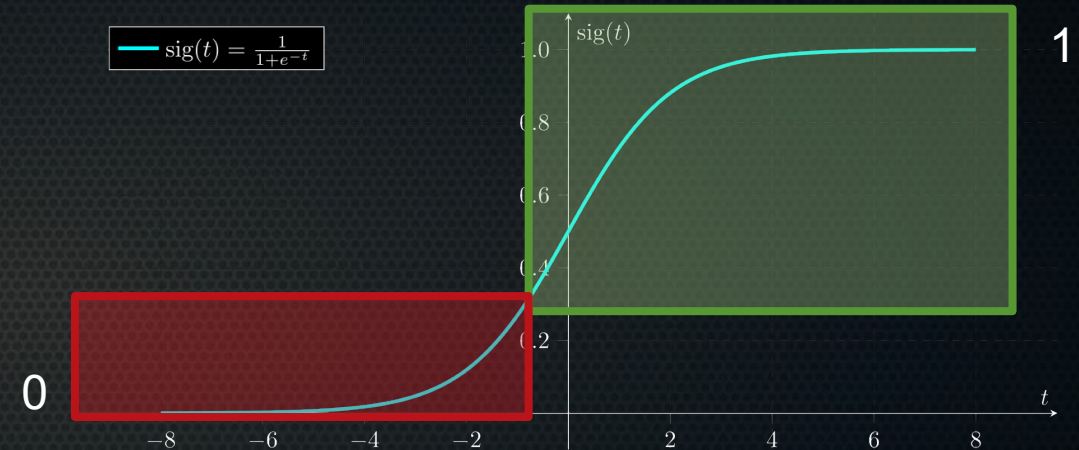
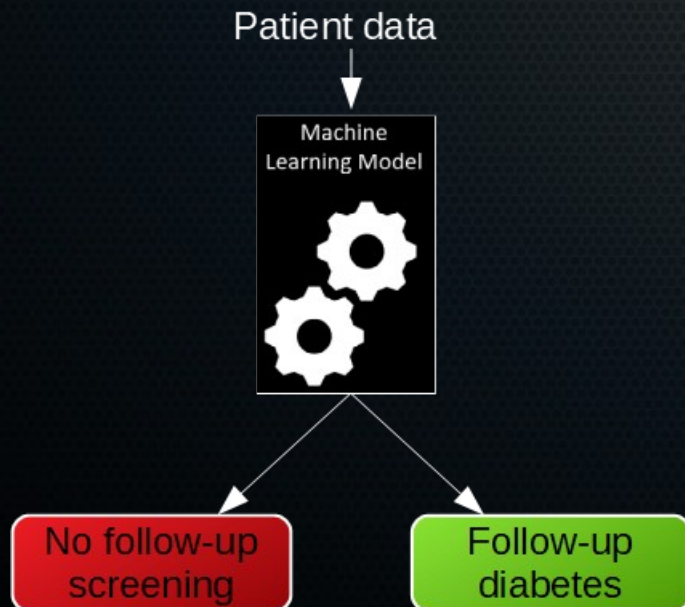
ROC curve

- How do you implement a focus on specificity or sensitivity?
- We could say:
if $h_{\theta}(x) \geq 0.95$ classify as positive
else classify as negative



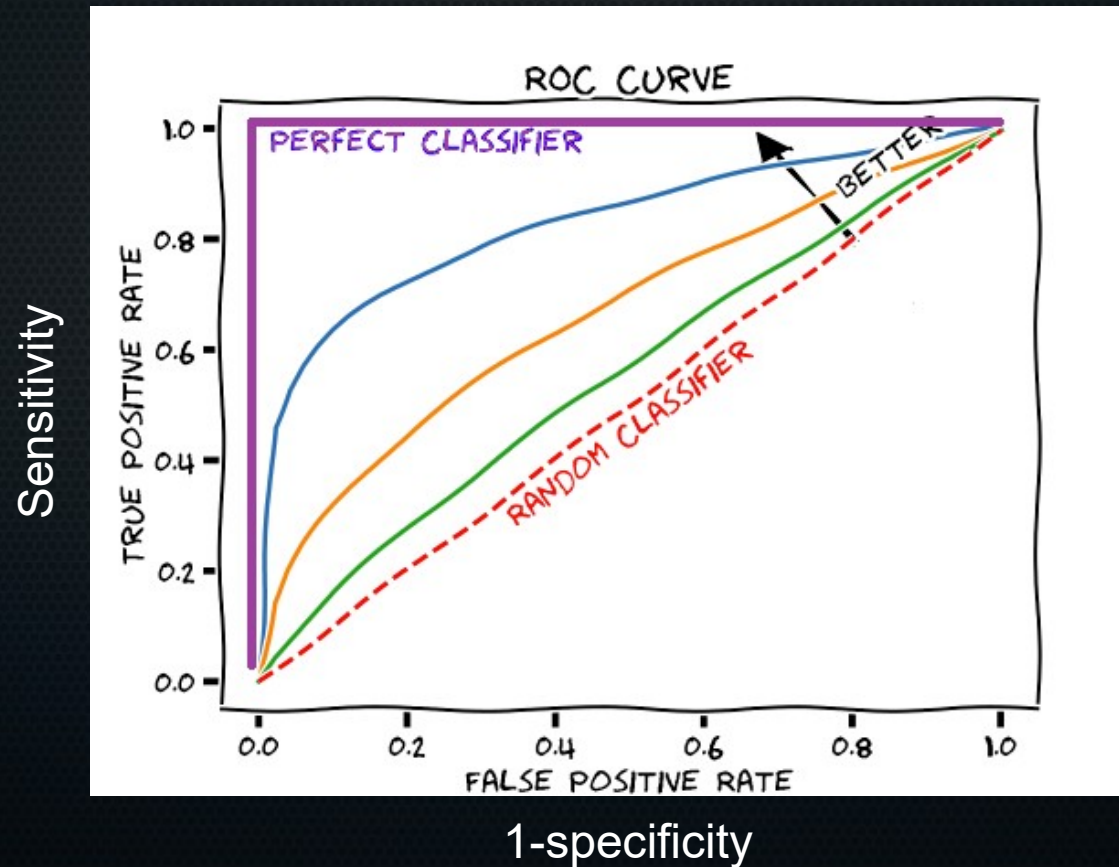
ROC curve

- How do you implement a focus on specificity or sensitivity?
- We could say:
if $h_{\theta}(x) \geq 0.3$ classify as positive
else classify as negative



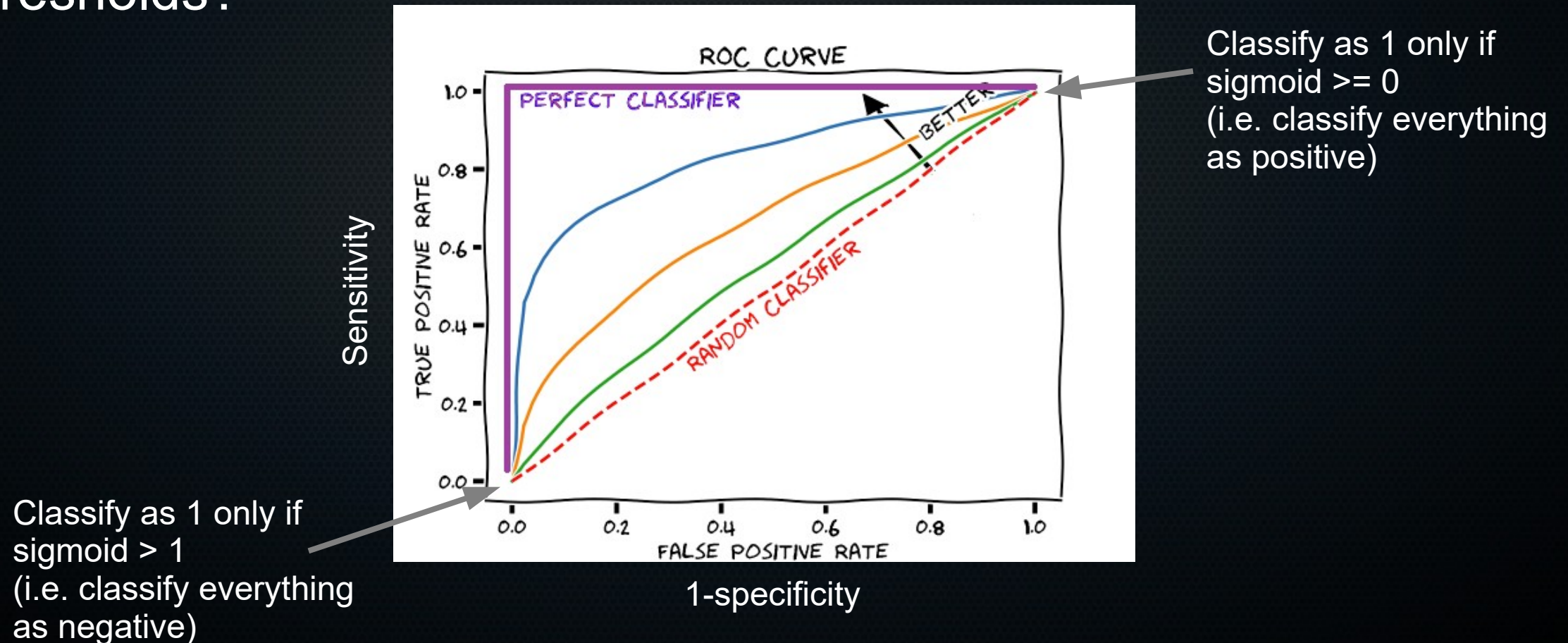
ROC curve

- What if we see how our classifier performs for all possible thresholds?



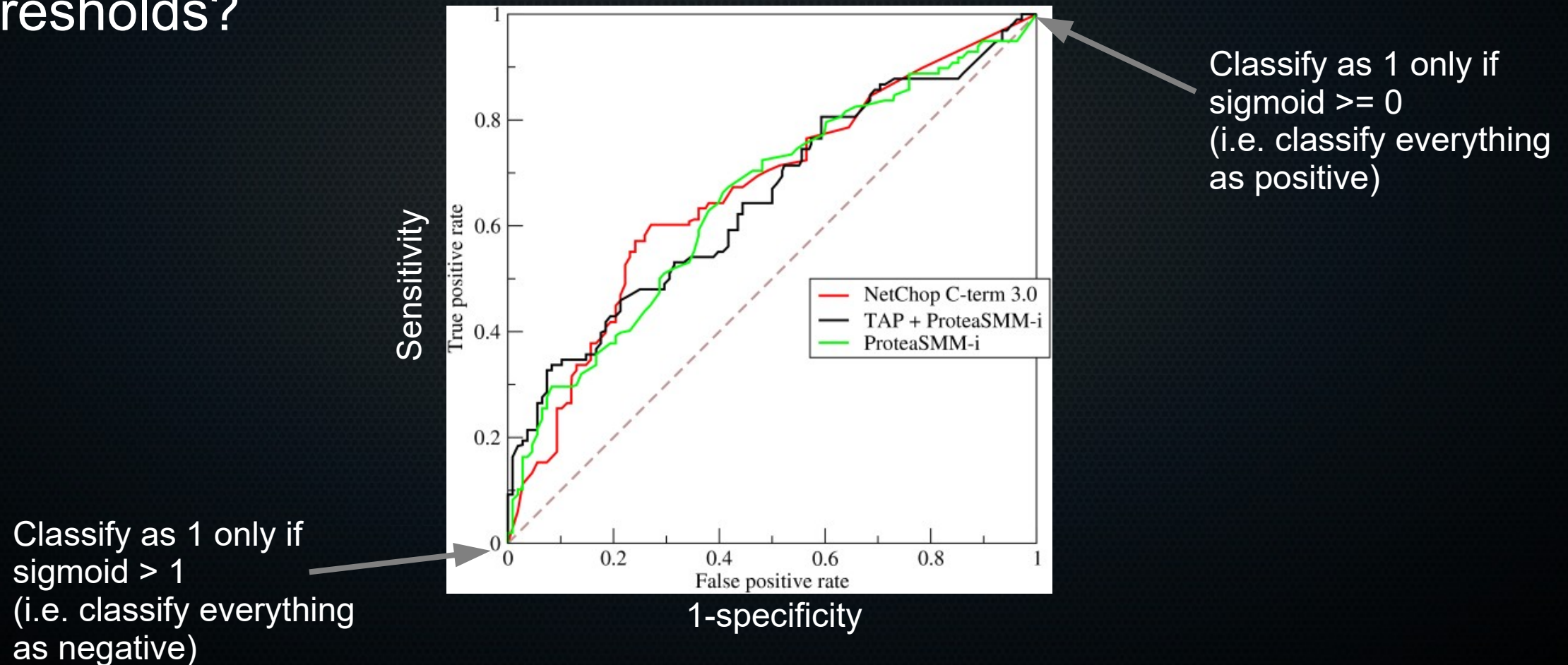
ROC curve

- What if we see how our classifier performs for all possible thresholds?



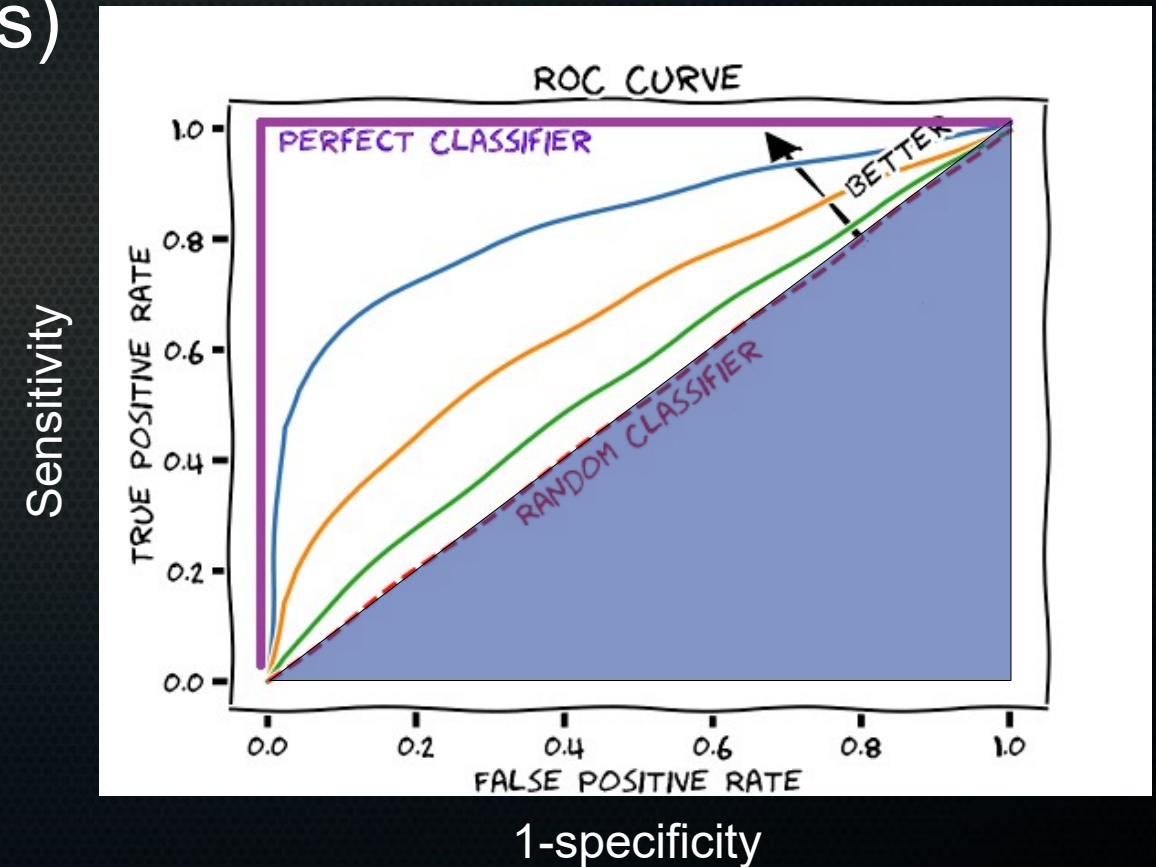
ROC curve

- What if we see how our classifier performs for all possible thresholds?



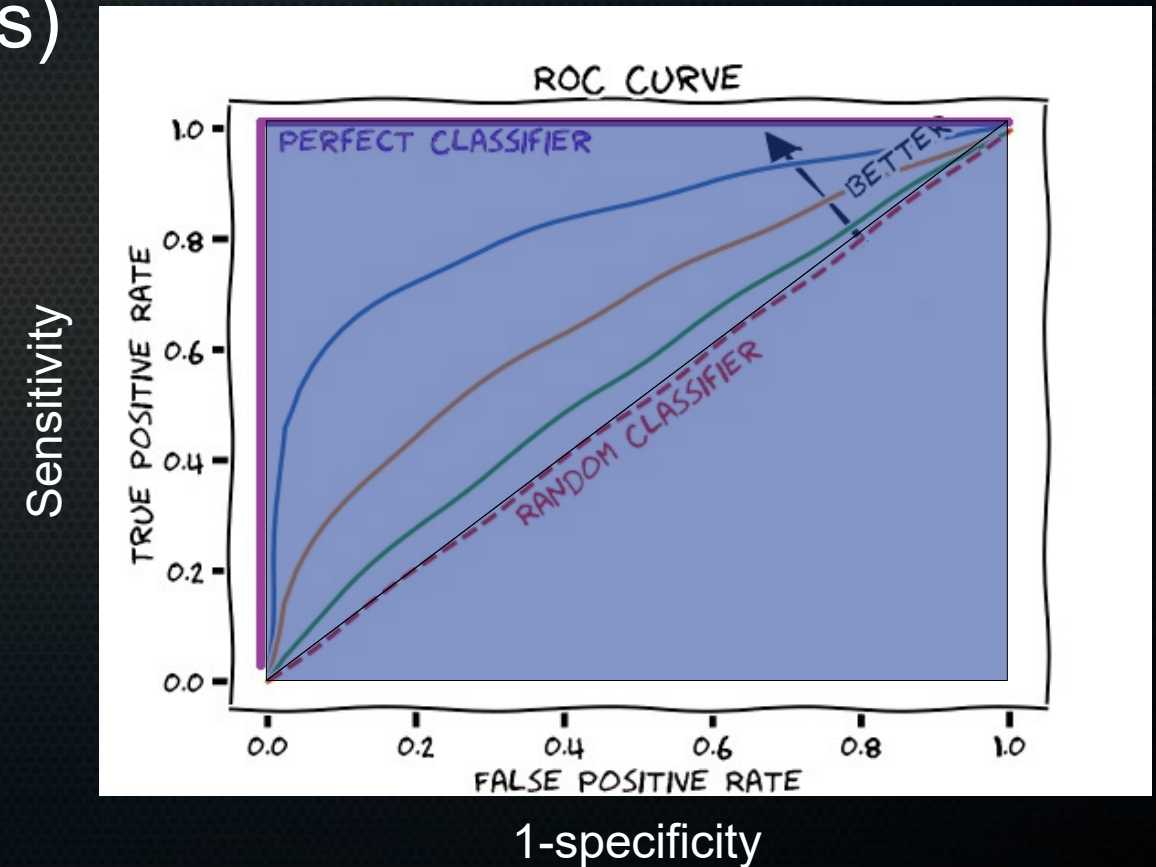
Area under the ROC curve (AUC)

- How to compare classifiers numerically?
- Coin-flip classifier (random guess)
AUC = 0.5



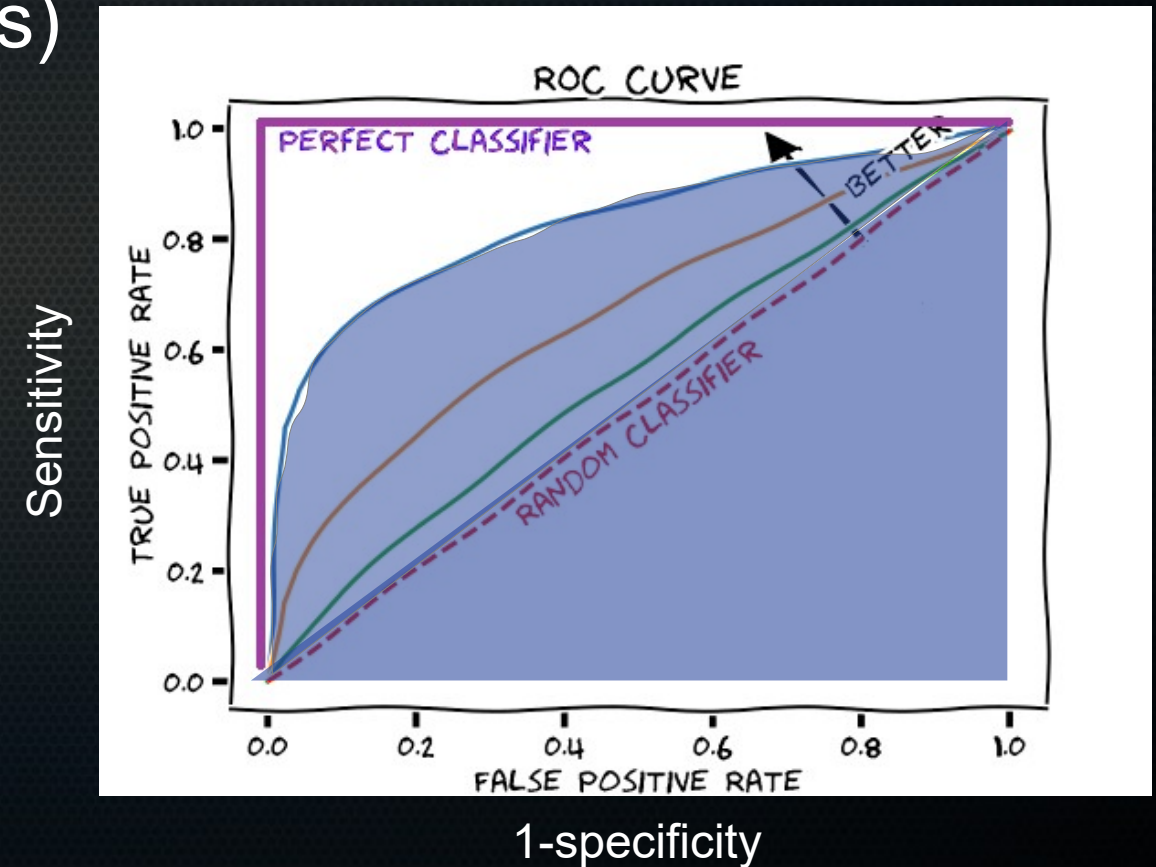
Area under the ROC curve (AUC)

- How to compare classifiers numerically?
- Coin-flip classifier (random guess)
AUC = 0.5
- Best possible classifier (positive cases all predicted 1)
AUC = 1



Area under the ROC curve (AUC)

- How to compare classifiers numerically?
- Coin-flip classifier (random guess)
AUC = 0.5
- Best possible classifier (positive cases all predicted 1)
AUC = 1
- In-between: ~0.8, for instance



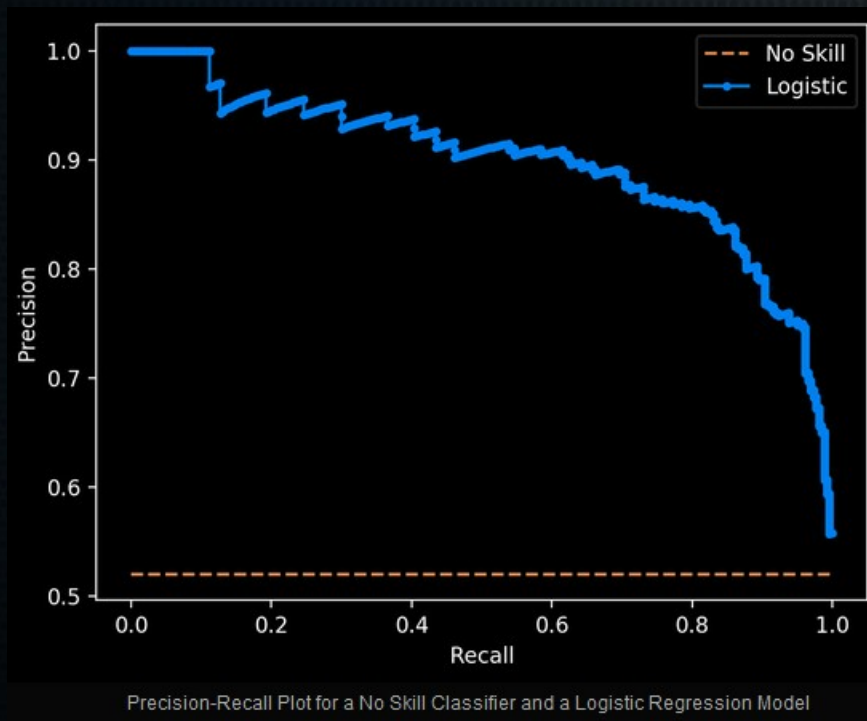
Pitfall AUC (of ROC curve)

- All these values are determined by ratios
- If data is sampled from general population, $N_{\text{negative}} \gg N_{\text{positive}}$ for a disease
- For specificity, because true negatives is a *huge number*, more false positives matter much less. Specificity becomes overly optimistic (especially if, later, you run your classifier in a clinical setting where N_{negative} is much smaller!)

		Reality	
		Cancer	Not Cancer
Model Prediction	Cancer	True Positive	False Positive
	Not Cancer	False Negative	True Negative
Sensitivity (true positive rate)		$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$	
Specificity (true negative rate)		$\frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$	

Pitfall AUC (of ROC curve)

- Instead, should then look at precision-recall curve.



Source: <https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/>

- Or make sure to train on balanced data!

		Reality	
		Cancer	Not Cancer
Model Prediction	Cancer	True Positive	False Positive
	Not Cancer	False Negative	True Negative

Recall =

$$\text{Sensitivity (true positive rate)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{Specificity (true negative rate)} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$

$$\frac{\text{True Positives}}{(\text{True Positives} + \text{False Positives})}$$

Precision

Regularisation

- Change the cost function to apply a cost for complexity

Regularisation

- Change the cost function to apply a cost for complexity

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m -y^{(i)} \cdot \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \cdot \log(1 - h_{\theta}(x^{(i)}))$$

Regularisation

- Change the cost function to apply a cost for complexity

$$J(\theta_0 \dots \theta_n) = \frac{1}{m} \sum_{i=1}^m -y^{(i)} \cdot \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \cdot \log(1 - h_{\theta}(x^{(i)}))$$

$$J(\theta_0 \dots \theta_n) = \frac{1}{m} \sum_{i=1}^m (-y^{(i)} \cdot \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \cdot \log(1 - h_{\theta}(x^{(i)}))) + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

- Make J a function of both the error of predictions given some parameters *and* the magnitude of those parameters themselves

Regularisation

- Change the cost function to apply a cost for complexity

$$J(\theta_0 \dots \theta_n) = \frac{1}{m} \sum_{i=1}^m -y^{(i)} \cdot \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))$$

$$J(\theta_0 \dots \theta_n) = \frac{1}{m} \sum_{i=1}^m (-y^{(i)} \cdot \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))) + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

- Make J a function of both the error of predictions given some parameters *and* the magnitude of those parameters themselves
- By convention: don't shrink bias/intercept term

Easiest to visualise how it works in linear regression

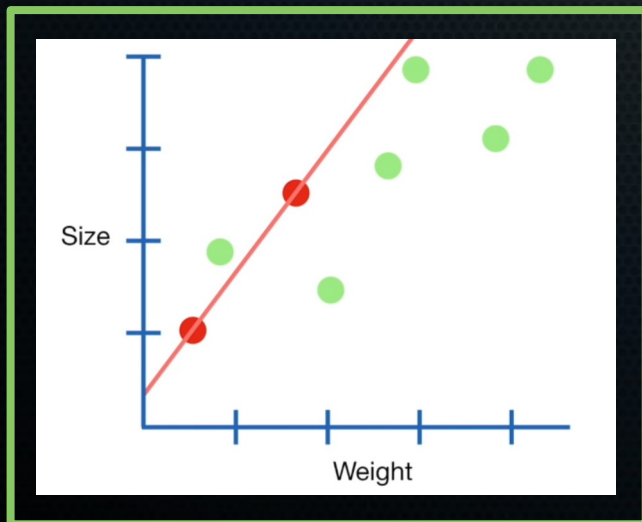
$$J(\theta_0, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x) - y^{(i)})^2 + \lambda \sum_{j=1}^n (\theta_j)^2 \quad h_{\theta}(x) = \theta_0 + \theta_1 x$$

Regularisation: ridge regression

- Change the cost function to apply a cost for complexity

$$J(\theta_0, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x) - y^{(i)})^2 + \lambda \sum_{j=1}^n (\theta_j)^2 \quad h_{\theta}(x) = \theta_0 + \theta_1 x$$

- Add some **bias** (constrain hypothesis to a set with small parameter values) but reduces **variance**:

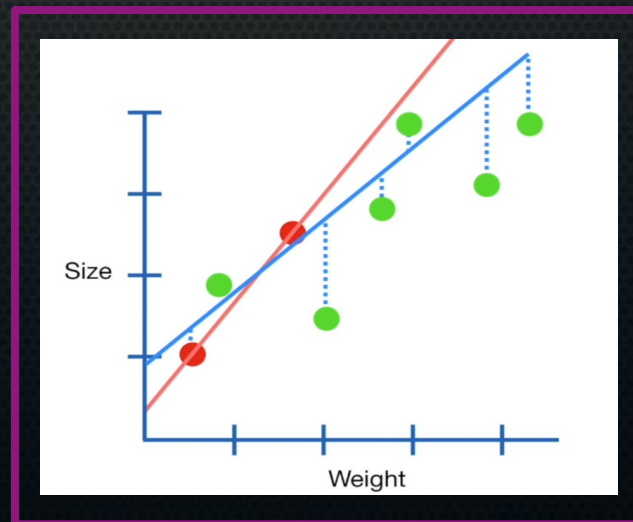
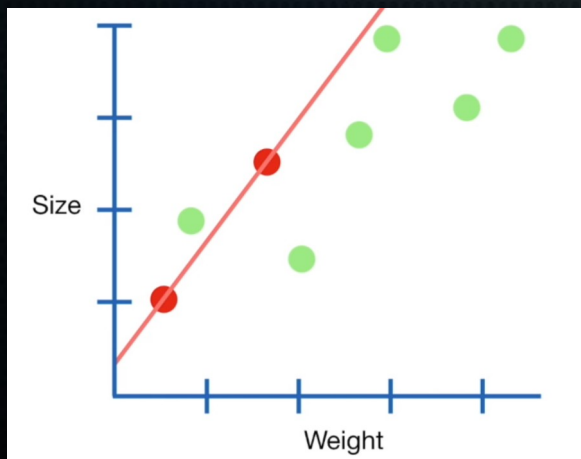


Regularisation: ridge regression

- Change the cost function to apply a cost for complexity

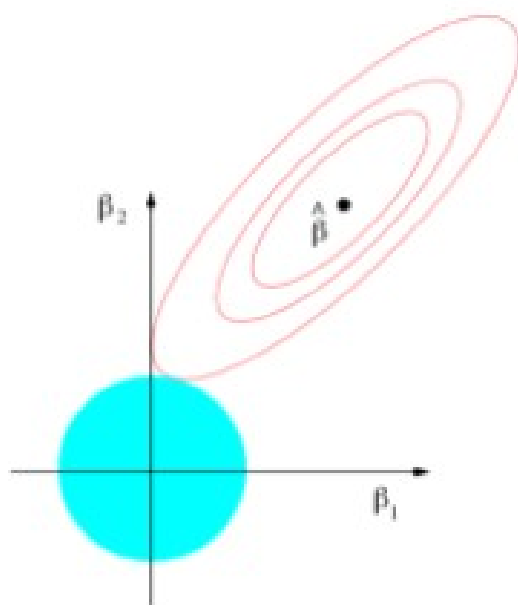
$$J(\theta_0, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x) - y^{(i)})^2 + \lambda \sum_{j=1}^n (\theta_j)^2 \quad h_{\theta}(x) = \theta_0 + \theta_1 x$$

- Add some **bias** (constrain hypothesis to a set with small parameter values) but reduces **variance**:



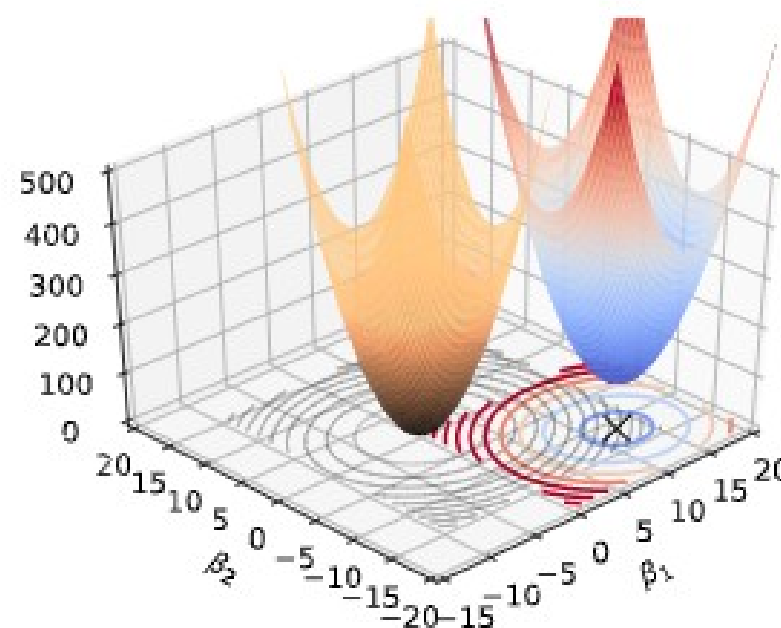
Constrained how much the line may increase with Weight (biased) → generalises better to test set

Regularisation visually



Hard constraint illustration from ESL page

71.



Soft constraint with non-regularized loss function (blue-red) term and penalty term (orange).

Regularisation taken together

- Changes your cost function to add penalty for too large parameters. Slightly increases bias, decreases variance.
- Adds another hyperparameter, λ
- Can be used for linear regression, logistic regression, neural networks, etc.

$$J(\theta_0 \dots \theta_n) = \frac{1}{m} \sum_{i=1}^m \left(-y^{(i)} \cdot \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \cdot \log(1 - h_{\theta}(x^{(i)})) \right) + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

- **Important note: don't use the regularisation term when calculating on validation set!**

Iteration k

k₁

k₂

k₃

k₄

k₅

k₆

k₇

k₈

k₉

k₁₀

What about those hyperparameters?

- We've now encountered λ for regularisation and α , the learning rate.
- How do we pick these hyperparameters?

What about those hyperparameters?

- Back to cross-validation:

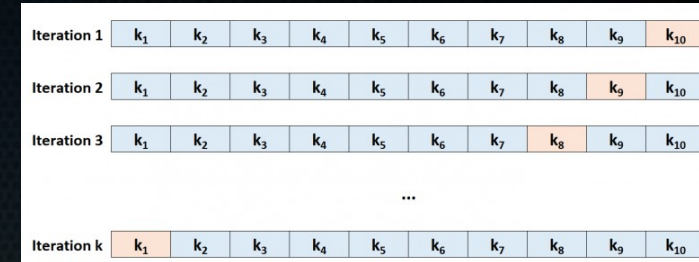


Source: <https://www.statology.org/k-fold-cross-validation/>

- Here, we train our model on 9 folds, and test it on a hold-out set of data to get a feel for the generalisation error (or the *real* performance on unseen data) by averaging the 10 values we get.

What about those hyperparameters?

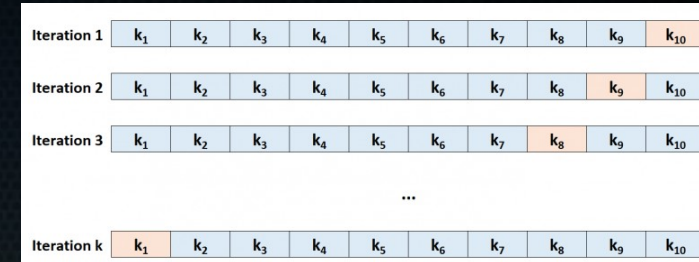
- Naive idea for hyperparameters:
 - Say we want to pick alpha from [0.001, 0.01, 0.1, 1, 10]
 - Take each alpha, train your model with those using cross-validation, and pick whichever set performs best on average out of the 5 * 10 models you made.



Source: <https://www.statology.org/k-fold-cross-validation/>

What about those hyperparameters?

- Naive idea for hyperparameters:
 - Say we want to pick alpha from $[0.001, 0.01, 0.1, 1, 10]$
 - Take each alpha, train your model with those using cross-validation, and pick whichever set performs best on average out of the $5 * 10$ models you made.
- But there's something wrong here:
 - If you do this, you select hyperparameters that give the best score on the validation set → so it's fitted to your data in a way.
 - You want to prevent this leak of information.

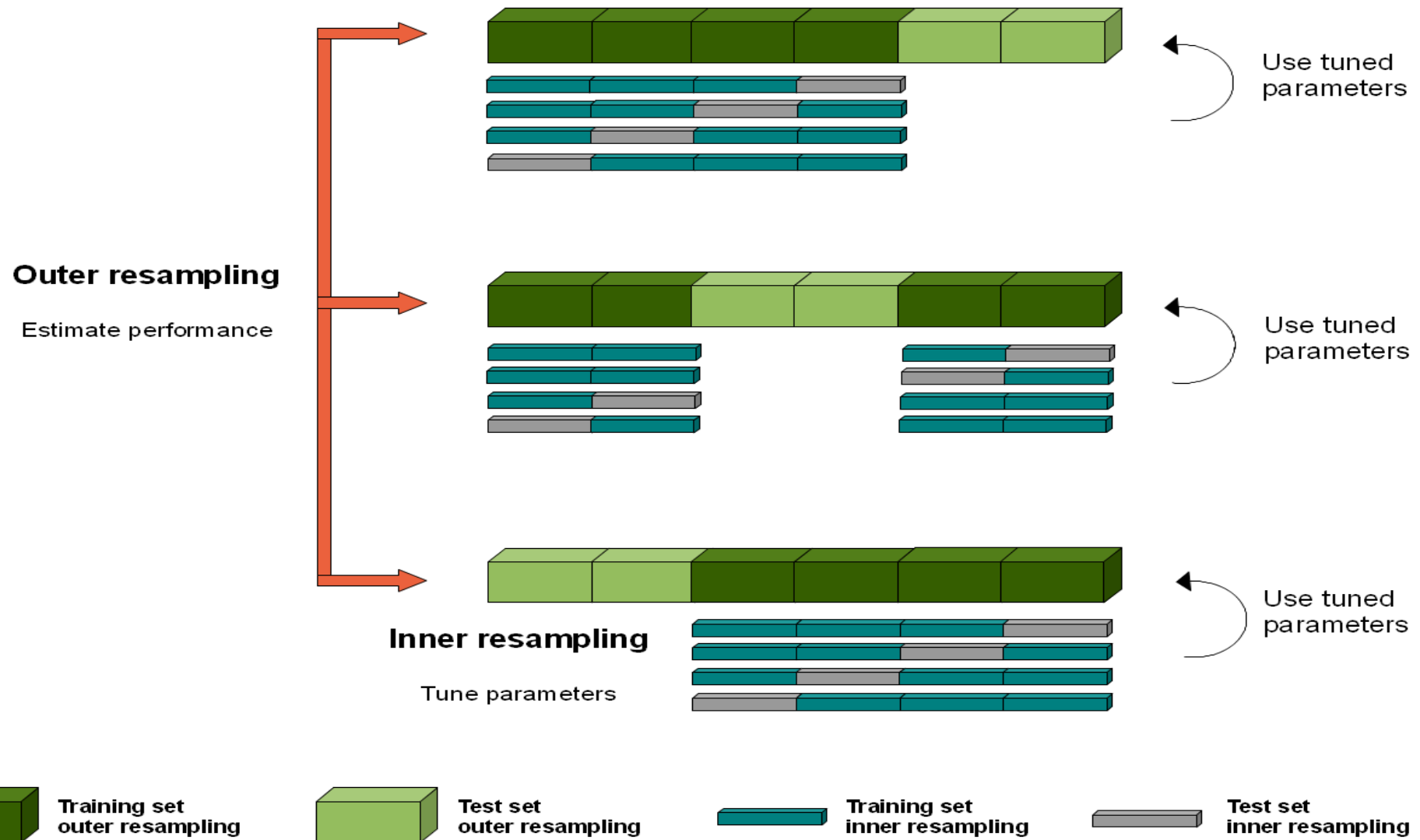


Source: <https://www.statology.org/k-fold-cross-validation/>

Nested cross-validation

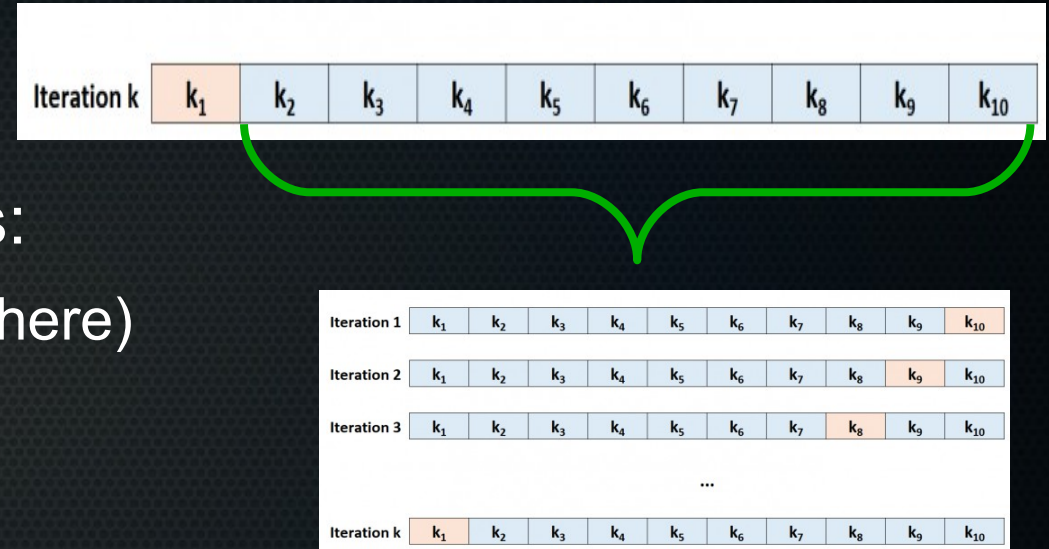
- What we want: pick best hyperparameters automatically *and* independently from validation performance.

Nested cross-validation



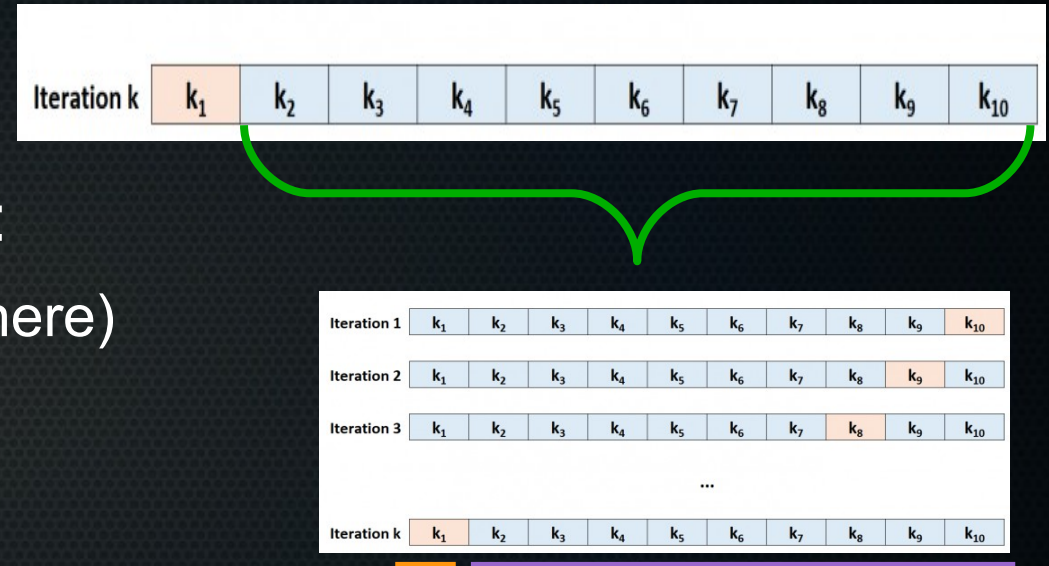
Nested cross-validation

- So: $\alpha = [0.001, 0.01, 0.1, 1, 10]$
 - Make outer folds, say from 1 to k (10)
 - Within each of the 10 training data sets:
 - Make inner folds, say from 1 to l (also 10 here)



Nested cross-validation

- So: $\alpha = [0.001, 0.01, 0.1, 1, 10]$
 - Make outer folds, say from 1 to k (10)
 - Within each of the 10 training data sets:
 - Make inner folds, say from 1 to l (also 10 here)
 - For each of the 10 training data sets:
 - For each of our alpha values:
 - Train a model with those alpha values
 - Test it on the validation set
 - End up with: average performance of 5 alpha values over 10 folds.
 - Pick alpha with best average performance (and record it)
 - For the outer fold, train with that best alpha and predict on held out data



>1 hyperparameters

- Just train models with all possible (or a subset of combinations of them)
- Note: nested cross-validation takes *a lot* of computing power.
For the example on the previous slide:
10 upper folds *
10 lower folds *
5 alpha values
= *500 models!*

Summary

- Multi-class classification: simply train n independent binary logistical regressors for your n classes, run them all on the data, pick for each sample the class with the highest probability over the regressors
- Performance metrics: Accuracy, sensitivity and specificity, ROC curve and ROC AUC (or PRC AUC for imbalanced data)
- Regularisation: add a cost to making parameters too large (i.e. fitting them too precisely to the data). Forces the model to only increase those parameters that really improve the fit (less fine-tuning exactly to the training data)!
- Nested cross-val for finding hyperparameters.

BREAK FOR PRACTICAL
