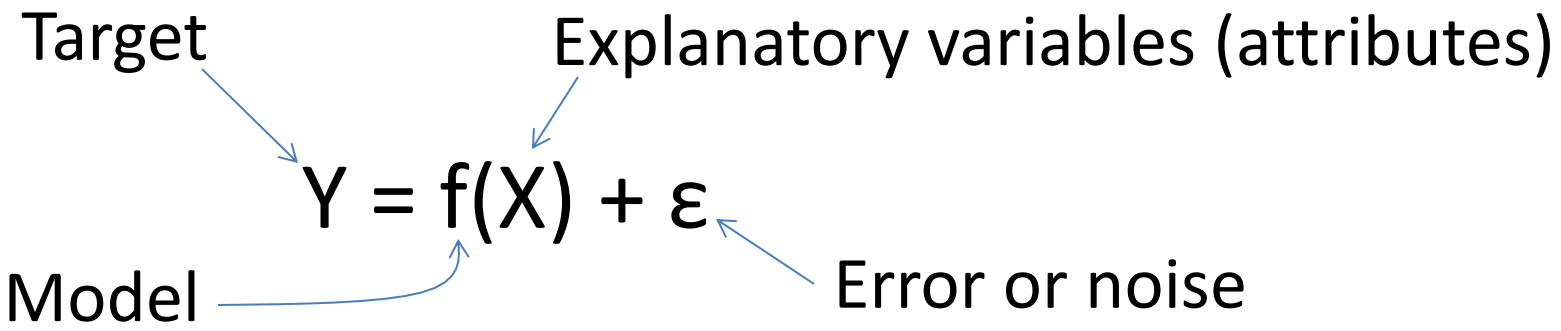
Machine Learning

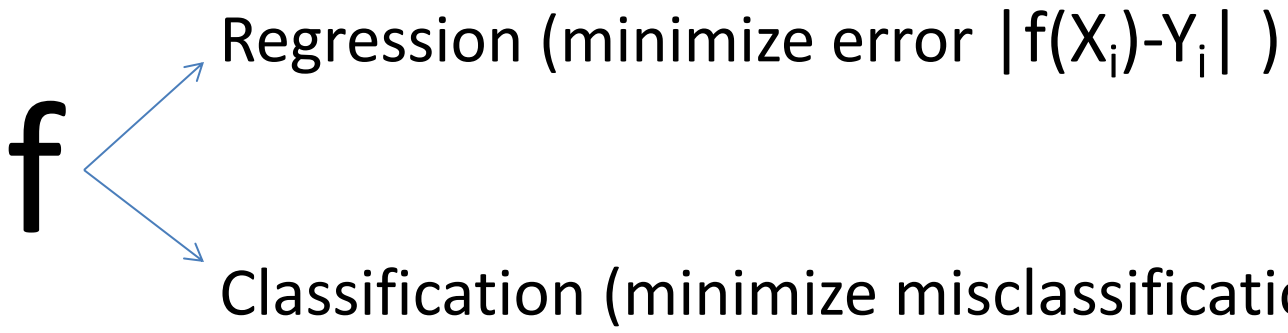
Lecture 03

Model Accuracy and Bias-Variance tradeoff

Model fitting process



- Having the model fit, the next step is to assess the accuracy of the model



- We need to know how well the model will perform in predicting new data

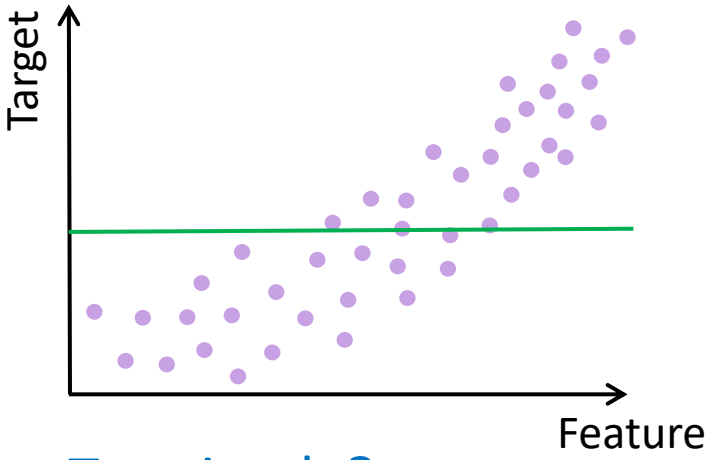
Model accuracy

- What indicators do we use to assess the quality of classification models?
- Quantitative quality indicators
  - Statistical measures (RMSE, variance, R^2, confidence levels, p-values, etc.)
  - Classification accuracy (how good is the model in predicting the unseen data)
- Graphical indicators
  - Confusion matrix (False positives, false negatives)
  - ROC curves
  - LIFT charts

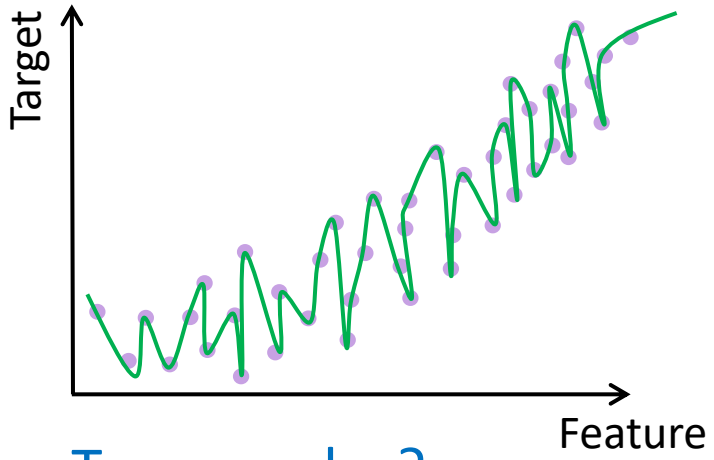
Bias-Variance Tradeoff

Challenges in estimating predictive accuracy

- Below is a regression example for the target and a feature on a given data set



Too simple?



Too complex?

- Which one is better in predicting new data?
- One of the common problems in Machine Learning is **overfitting** (modeling intrinsic noise in the data instead of the true signal)

What is overfitting?

- Let's say you attend a symphony and want to get the clearest, most faithful sound possible. So you buy a super-sensitive microphone and hearing aid to pick up all the sounds.
- Then you start "**overfitting**," hearing the noise on top of the symphony. You hear your neighbors shuffling in their seats, the musicians turning their pages, and even the swishing of the conductor's coat jacket.
- When you're at a concert, there's both the symphony and the random noise. Fitting a perfect model is only listening to the symphony. Overfitting is when you hear more noise then you need to, or worse, letting the noise drown out the symphony.
- Technically speaking, in any dataset we have the signal and the noise. When we do prediction, we want to identify the signal. Our predictions will be the most accurate if we can model as much signal as possible and as little noise as possible.

From: William Chen @ https://www.quora.com/What-is-an-intuitive-explanation-of-overfitting

What is overfitting?

- Imagine a dataset that states whether it was raining or not, *Rain/NotRain* and whether I went outside or stayed inside *(Out/In)*, we'll also include the day of the week.

Day of week	Weather	Stay
Monday	Rain	In
Tuesday	Rain	In
Wednesday	Rain	Out
Thursday	NotRain	In
Friday	NotRain	Out
Saturday	NotRain	Out
Sunday	Rain	In

Good Model: Stay inside when its raining and go outside when it's not.

Overfit Model: Stay inside when it's raining except on Wednesday and go outside when its not raining, except when it was raining the day before

What is overfitting?

How do you fight Godzilla?

UNDERFITTING

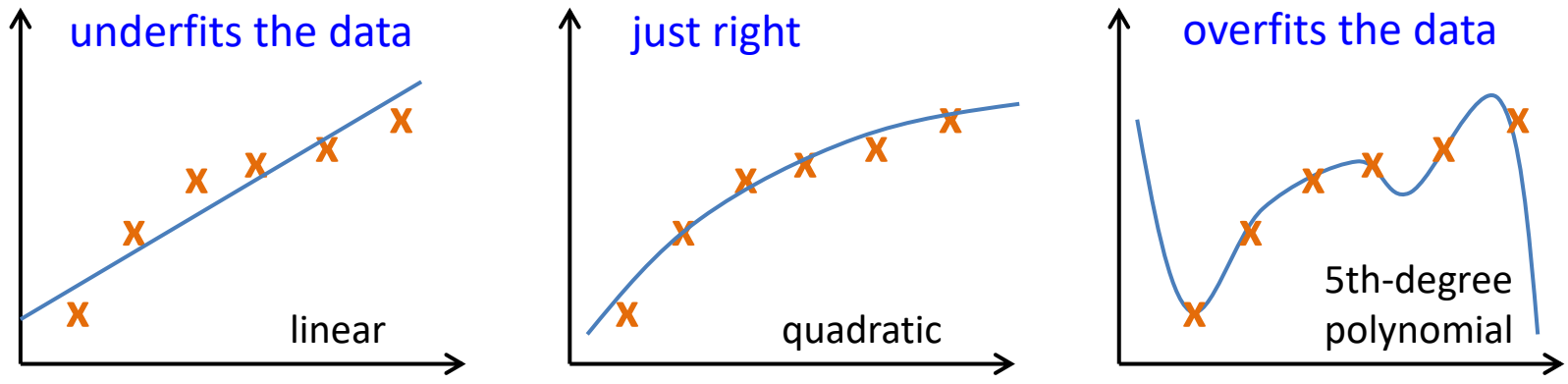
How do you fight a fly?

OVERFITTING

Underfitting and Overfitting

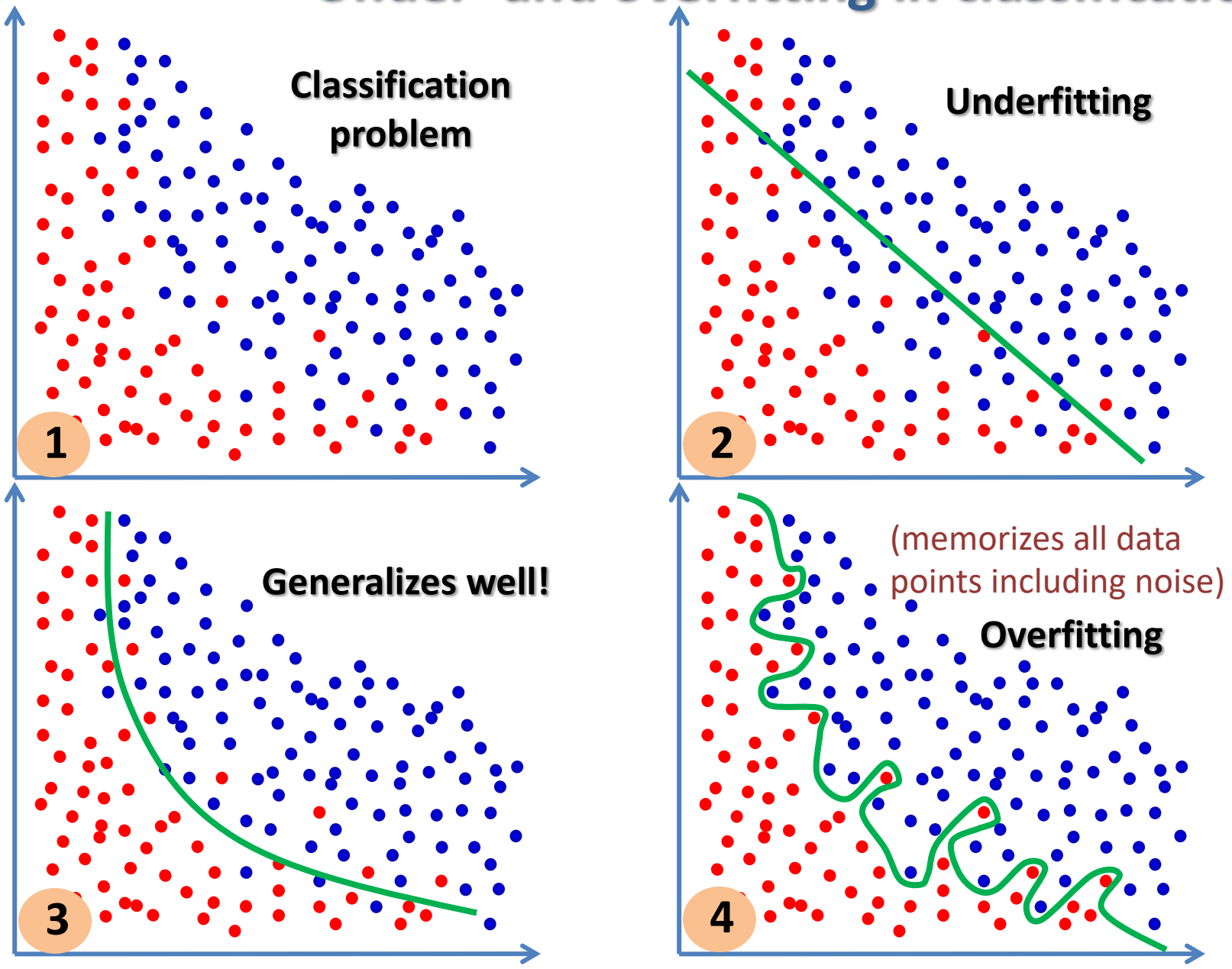
- Let's say we have a set of data pairs $(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)$
- Our goal is to find a function f(x) that closely "fits" the data such that the error $|f(x_i)-y_i|$ is minimum. This type of learning problem is called regression.
- Over- and under-fitting also apply to classification but are easier to visualize in case of regression
- Let us assume we try to fit a model to our data set using a 1-degree (linear), a second degree (quadratic) and a fifth degree polynomial:

Under- and Overfitting in regression



- Ideally, we would like to choose a model that both accurately captures the regularities in the training data, but also generalizes well to unseen data
- Left: No matter how curvy the data set is, we'll only be able to draw a straight line with this linear model leading to an inaccurate representation of data => **underfitting**
- Center: perfect fit (generalizes well)
- Right: Model is obsessively accurate (memorizes all data points but trades away any ability to generalize to unseen data sets) => **overfitting**

Under- and overfitting in classification



Bias and variance

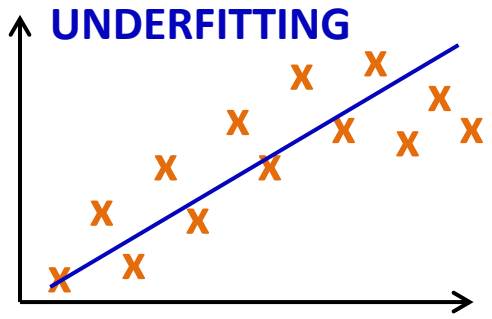
- What are bias and variance?
  - Suppose you are conducting a survey
    - Having a sampling space that is not a representative of the population creates bias
    - Having a way too limited sampling space creates variance
- **Bias**: How much we ignore the data
  - A systematic error in the model (typically caused by an inability to express the full complexity of the data)
- **Variance**: How dependent our model is on data
  How much does the estimate change for different data sets?
  - It's all about the variability in the fitted model and learning all tiny accidental variations of the samples (caused by having a model that is too complex for the amount of data)

Bias-variance vs over-and-underfitting

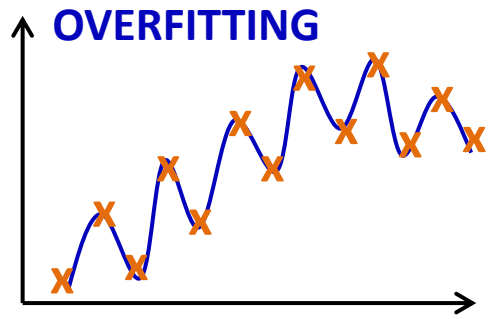
- When the model **underfit**s the data set, we have a **high bias** model.
- When the model **overfit**s the data set, we have a **high variance** model
- Ideally, we would like to see low bias and low variance. Generally speaking, the more complex the model, the lower the bias and higher the variance
- What is model complexity?
  - A linear model is less complex than a quadratic one
  - Extra attributes lead to more complexity

Bias-variance vs over-and-underfitting – cont'd

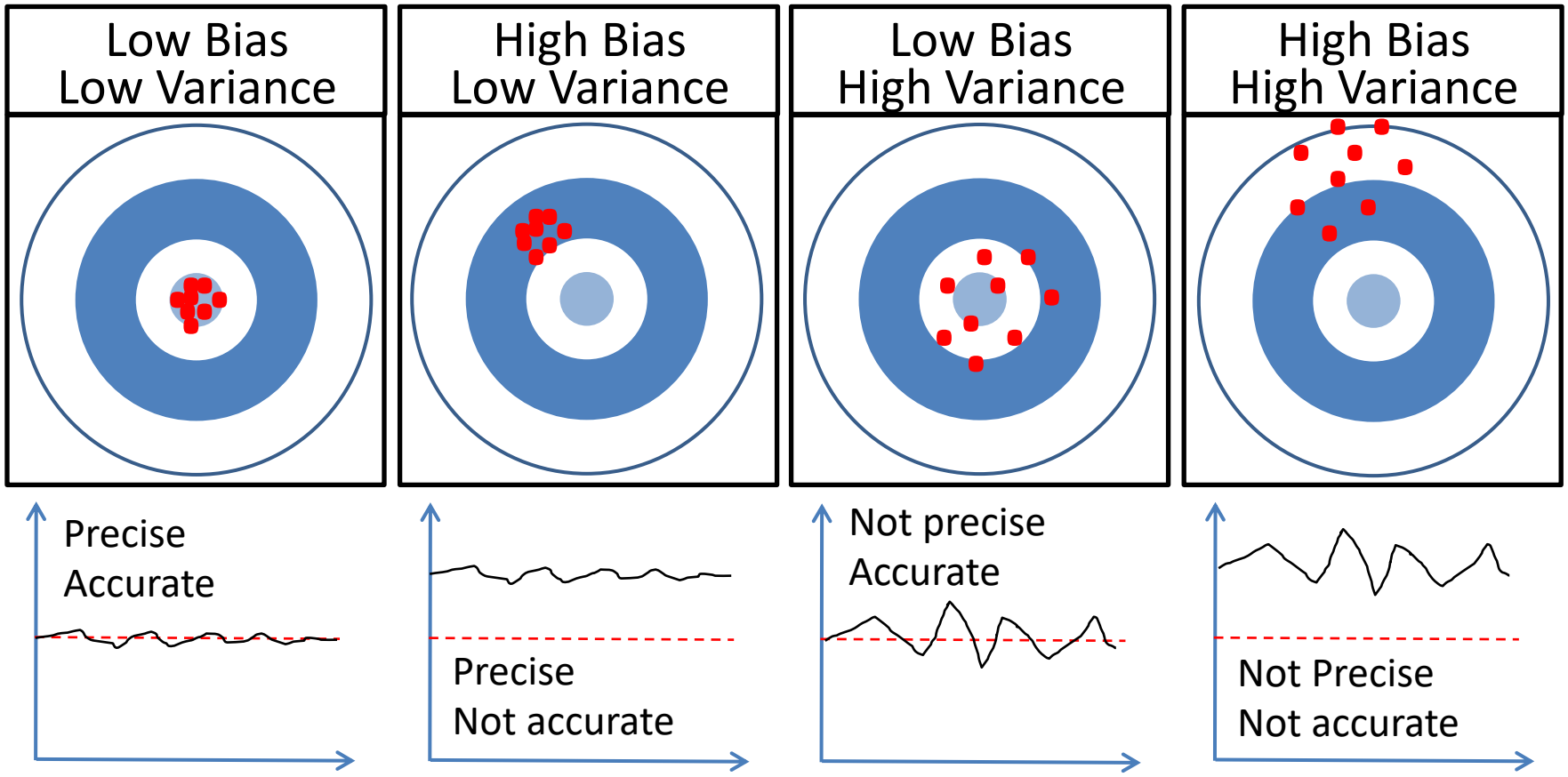
- If we make the model more complicated, then bias decreases but variance increases. These are two opposing forces, and in most cases, you can only decrease one at the expense of the other, thus the concept bias-variance tradeoff.
- A model with high bias is inflexible.



- A model with high variance, however, is so flexible that it even models the noise in the training set.



Graphical representation of bias & variance



If either bias or variance is high, dart throw (our model prediction) can be very far off. When we talk about bias and variance in Machine Learning, we're indeed talking about bias and variance of a type of model.

Certain model characteristics

Some of the different but related model characteristics in case of underfitting and overfitting:

	MODEL				
	Bias	**Variance**	**Complexity**	**Flexibility**	**Generalizability**
Underfitting: An overly simple model	**High**	**Low**	**Low**	**Low**	**High**
Overfitting: Modeling the noise as well	**Low**	**High**	**High**	**High**	**Low**

Ref: https://cambridgecoding.wordpress.com/2016/03/24/misleading-modelling-overfitting-cross-validation-and-the-bias-variance-trade-off/

Model Selection and Validation: Cross Validation

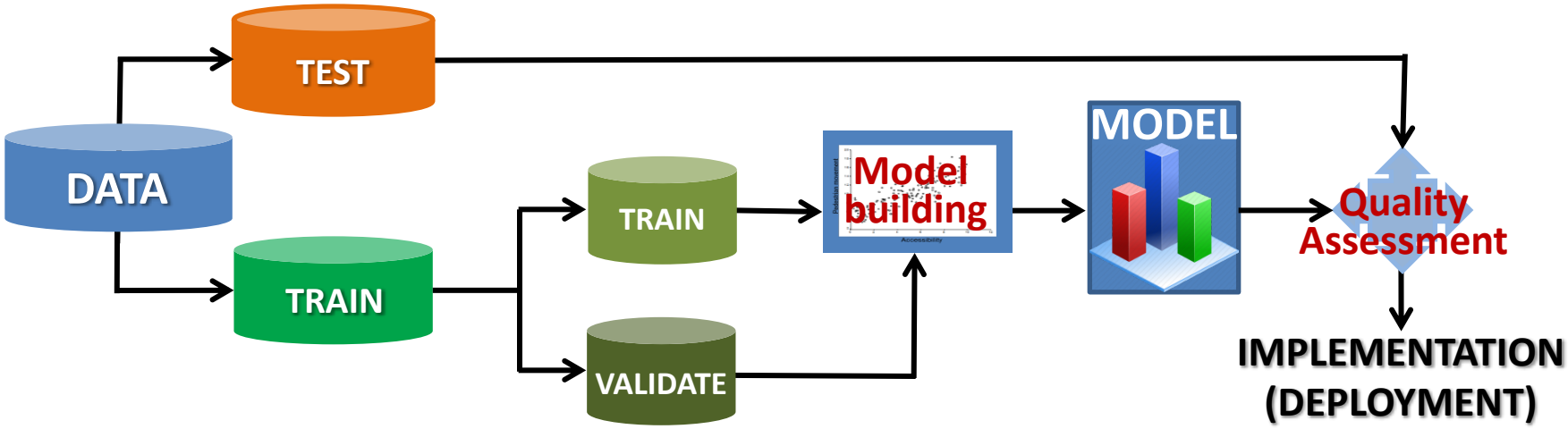
Model accuracy

- We want to build a model that works best on the available data and perform well on the new (unseen) data: Good prediction accuracy
- How do we decide that the model we use is accurate?
- For regression problems
  - Prediction accuracy:
    - Mean square error (MSE), Root MSE, etc.
- For classification problems
  - Classification accuracy (ACC):

$$\text{ACC} = \frac{\text{Number of correct decisions made}}{\text{Total number of decisions made}} = 1 - \text{Error rate}$$

Model selection and validation

- **Model selection**: How do we select the optimal parameters for a given classification problem?
  - All Machine Learning models have one or more parameters that need to be tuned for the best outcome
- **Validation**: How do we estimate the true error rate (or classification accuracy) of the chosen model?



- We need to use our finite set of data points for model selection and model validation.
- How do we do this?

Model selection and validation – cont'd

- What is our data set like?
  - If we had access to the entire population (unlimited number of data points), we would choose the model with the smallest error rate (= true error rate).
  - Reality, however, is different. We have a limited set of data points available for use. Thus, care must be taken in assessing the accuracy of the chosen model.
- One may be tempted to use the entire data set for training the model (choosing the optimal classifier) and then estimate the true error rate based on the same data set.
- **Training accuracy**: This is the accuracy obtained when you train and test the model on the same data

Double—dipping is a terrible idea!

Model selection and validation – cont'd

- Testing on the same data we used to train the model is invitation for trouble. Why?
  - The error rate estimate will be overly optimistic (smaller than the true error rate). 100% correct classification is common in such cases (overfitting)
  - We want to estimate the likely performance of the model on previously unseen data. By maximizing training accuracy, the model will not generalize well to unseen data (overfitting).
- There are 4 approaches in use: (for model selection, model evaluation, and model selection+evaluation:
  1. **Evaluation with hold-out data**
  2. **Hold-out validation with a test set**
  3. **k-fold Cross validation**
  4. **Nested Cross Validation**

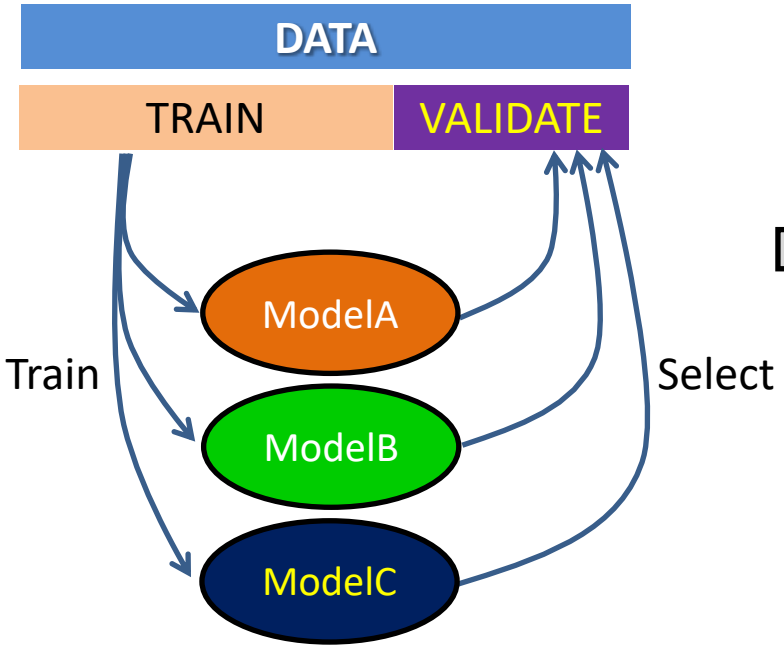
Train-Validation-Test data

- The underlying idea is that:

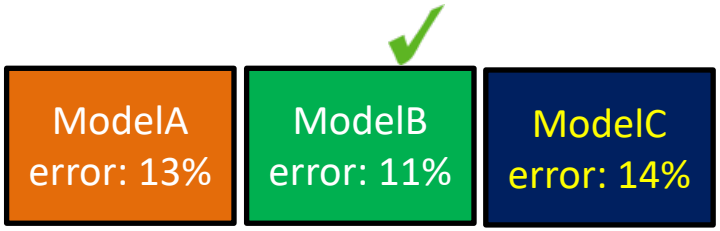
1. The training set is used to train a given model.
2. The validation set is used to choose between models For instance:
   - Do you want a polynomial regression of 3rd degree, 4th degree or 5th degree? Do you want a random forest with 40 trees or 50 trees? => **Select a model**
3. The test set tells you how you've done. If you've tried out a lot of different models, you may get one that does well on your validation set just by chance, and having a test set helps make sure that this is not the case => **Evaluate (estimate) model performance**

1. Evaluation with Hold-out data

1. Split the data into train and hold-out (validation) sets



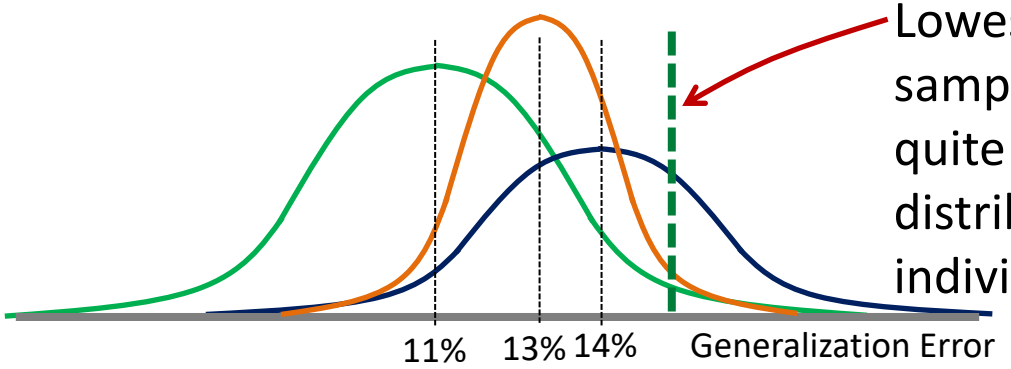
Select the best model with the lowest error based on validation data



2. For each parameter combination, train a model on the training set and compute the metric (RMSE, classification accuracy etc.) on the validation set
3. Choose the hyperparameters with the best metric
4. Re-train on the whole data with chosen hyperparameters

1. Evaluation with Hold-out data

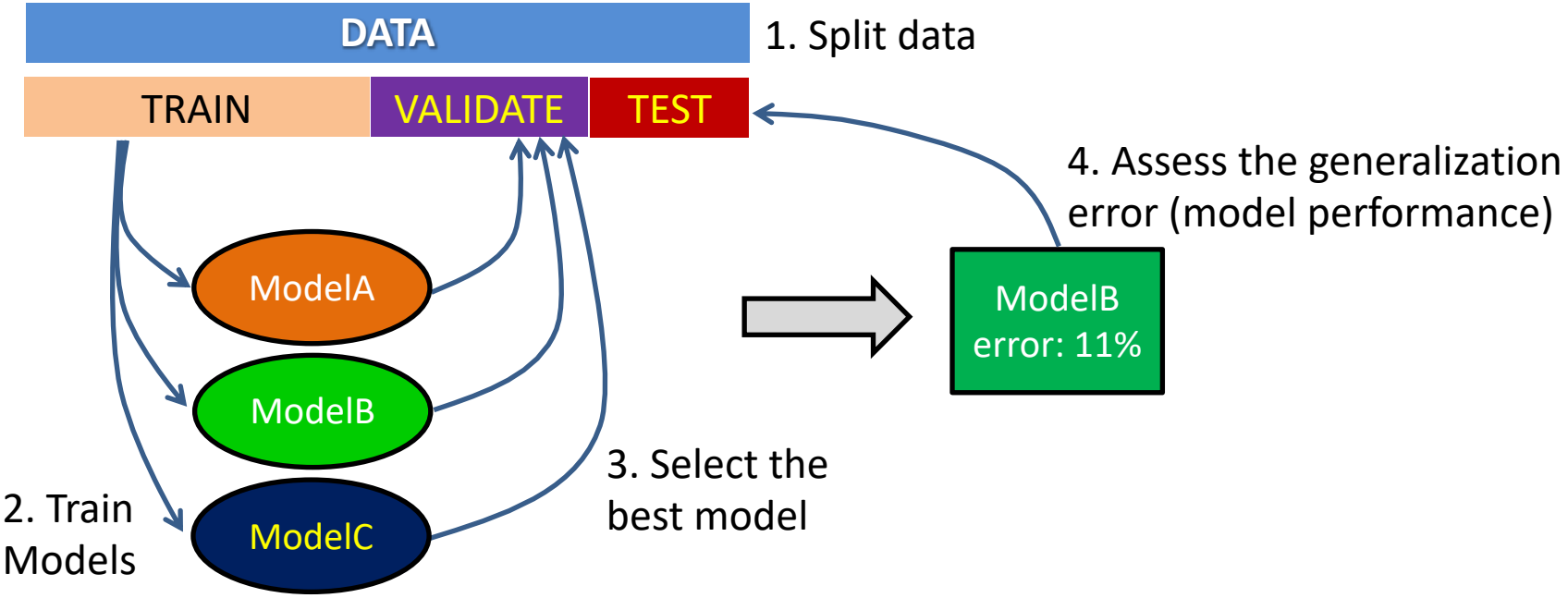
- While a "holdout validation" indeed provides a relatively better generalization performance, how do we know that:
  - Our selection of split did not yield a lucky pair, or the validation part is not fundamentally different from the rest of the data (what if there is a seasonal trend)?



Lowest error for ModelB on a random sample taken from the data. This is quite likely as we don't know the true distribution and variance for the individual model.

- Pros and Cons of holdout evaluation
  - Simple, fast (run only once), accurate given enough data
  - Dependent on how the data is split
- Solution? Hold-out validation with a test set

2. Hold-out evaluation using test data



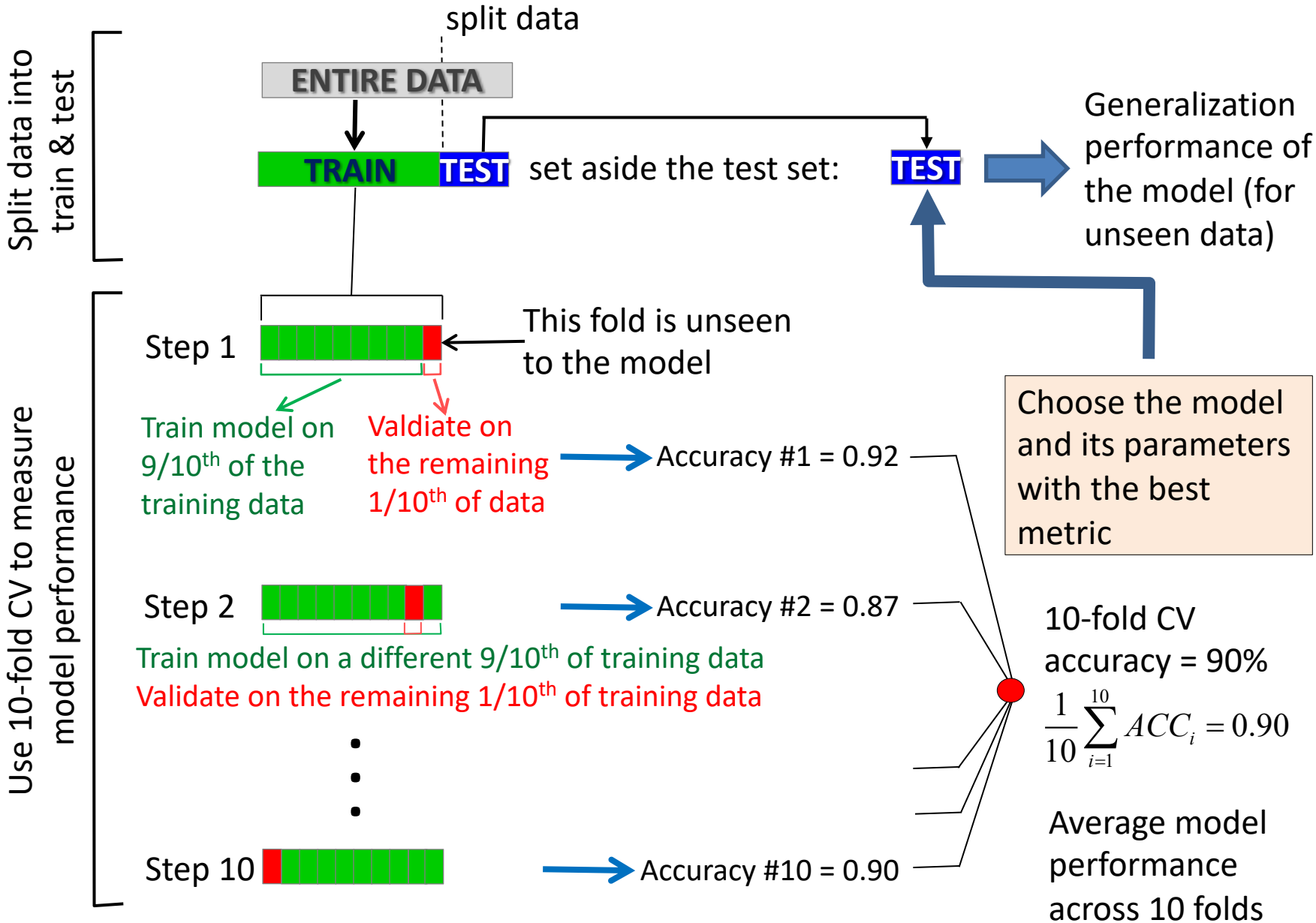
Now we have a completely legitimate way of selecting a model (via a validation set) and assessing its generalization performance (via a test set). But what about any variance in the validation set?

What if?

Solution? Use **Cross Validation** with multiple validation splits

Could we end up with a different model with a different split? Perhaps. This is ok for really large data sets (few million samples) where all splits are representative of the whole data.

3. k-fold Cross Validation



3. k-fold Cross Validation

- **Pros**
  - Given enough data, highly accurate
  - The whole dataset is used for both training & validation (therefore less variance)
  - k-fold CV is a more sophisticated procedure and overcomes the limitations of the holdout method by systematically repeating the train-split process many times and averaging the results

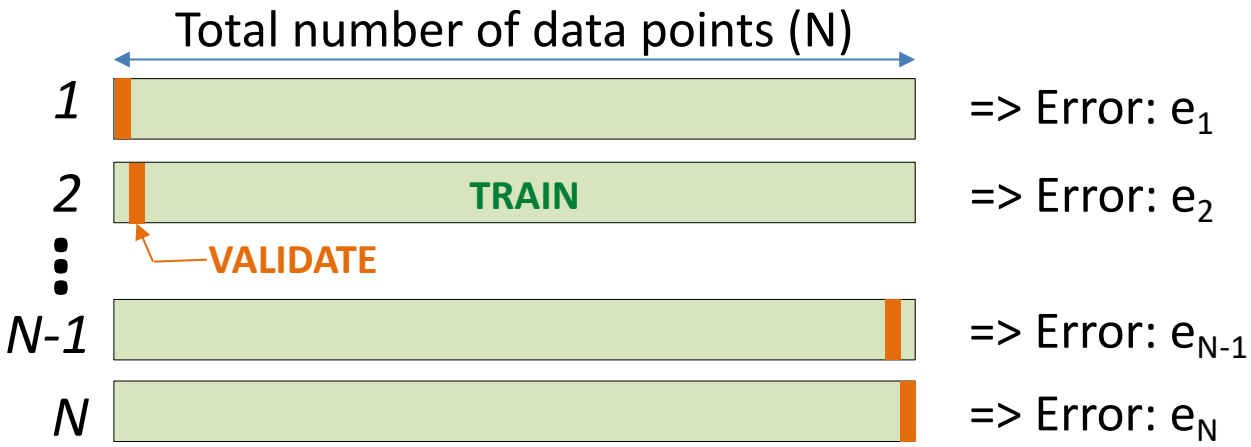
    Multiple rounds of k-fold CV can be performed using different partitions and averaging the validation results over the rounds to further reduce variability (more computational challenge).
- **Cons**
  - Computationally expensive (depending on k)

3. k-fold Cross Validation

- **What is a good value for k?**
- With a large k (smaller portion of data for testing):
  - Small estimator bias, large estimator variance
  - Computationally expensive
- With a small k (larger portion of data for testing):
  - Large estimator bias, small estimator variance
- In practice, **5-fold** or **10-fold** CV are usually effective
  - For very large data sets, even a 2-fold CV is good enough
  - For very sparse data sets, LOOCV (k=N-fold) might be a good choice to train on as many examples as possible
- Used for
  - Picking variables to include in a model
  - Picking the type of prediction function to use
  - Picking the parameters in the prediction function
  - Comparing different predictors

3. A special case of k-fold Cross Validation

- **Leave-One-Out Cross Validation (LOOCV):**
- This is a special case for k-fold CV where k = N

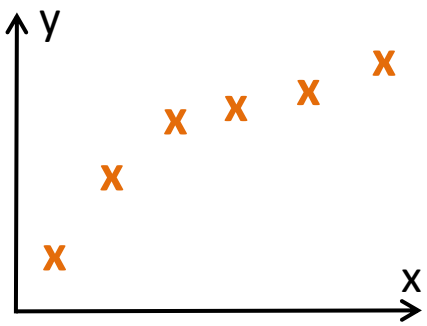


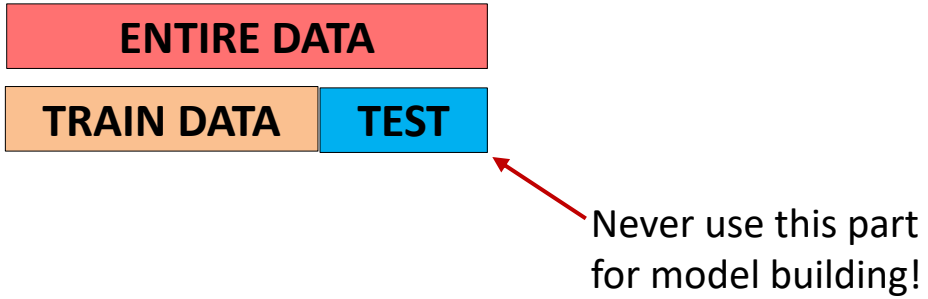
This is a k-fold CV with k=N

- Error rate is the average over N samples: $Error = \frac{1}{N}\sum_{i=1}^{N} e_i$
- Pros and Cons
  - Suitable for very small datasets
  - We make use of all data points, hence low bias
  - Can have high variance for the errors in the test set as there is only 1 point used for testing each time (outlier problem)
  - Computationally expensive

k-fold Cross Validation (Example)

- Linear Regression problem on a data set D:

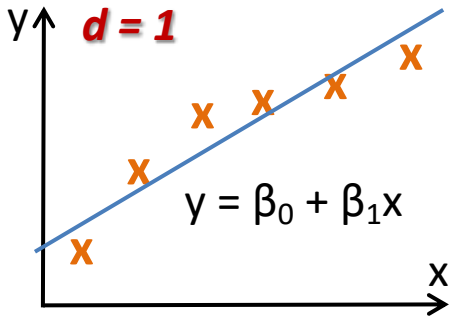




- We want to fit a polynomial regression model to our training data with no over/under-fitting.
- We need to try regression models of varying degree of polynomial (d) and select the best one based on a cross validation scheme.
- We repeat the cross-validation calculation for each model of degree (d) we use during training. We then select the model with the minimum Cross Validation error.

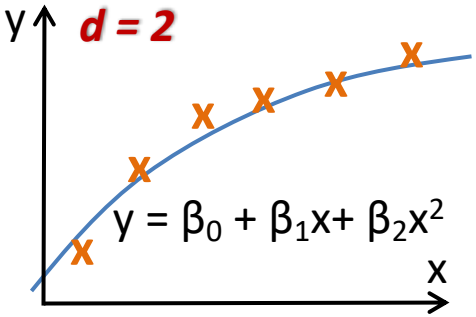
k-fold Cross Validation (Example)

TRAIN DATA 3-fold validation =>



Train	V		$(\beta_0, \beta_1)_{fold1}$, $RMSE_1$
Train	V	Train	$(\beta_0, \beta_1)_{fold2}$, $RMSE_2$
V	Train		$(\beta_0, \beta_1)_{fold3}$, $RMSE_3$

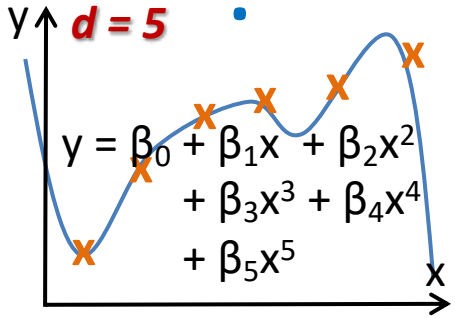
$$RMSE^{(d=1)} = \frac{1}{3}\sum_{i=1}^{3} RMSE_i$$



Train	V		$(\beta_0, \beta_1, \beta_2)_{fold1}$, $RMSE_1$
Train	V	Train	$(\beta_0, \beta_1, \beta_2)_{fold2}$, $RMSE_2$
V	Train		$(\beta_0, \beta_1, \beta_2)_{fold3}$, $RMSE_3$

$$RMSE^{(d=2)} = \frac{1}{3}\sum_{i=1}^{3} RMSE_i$$

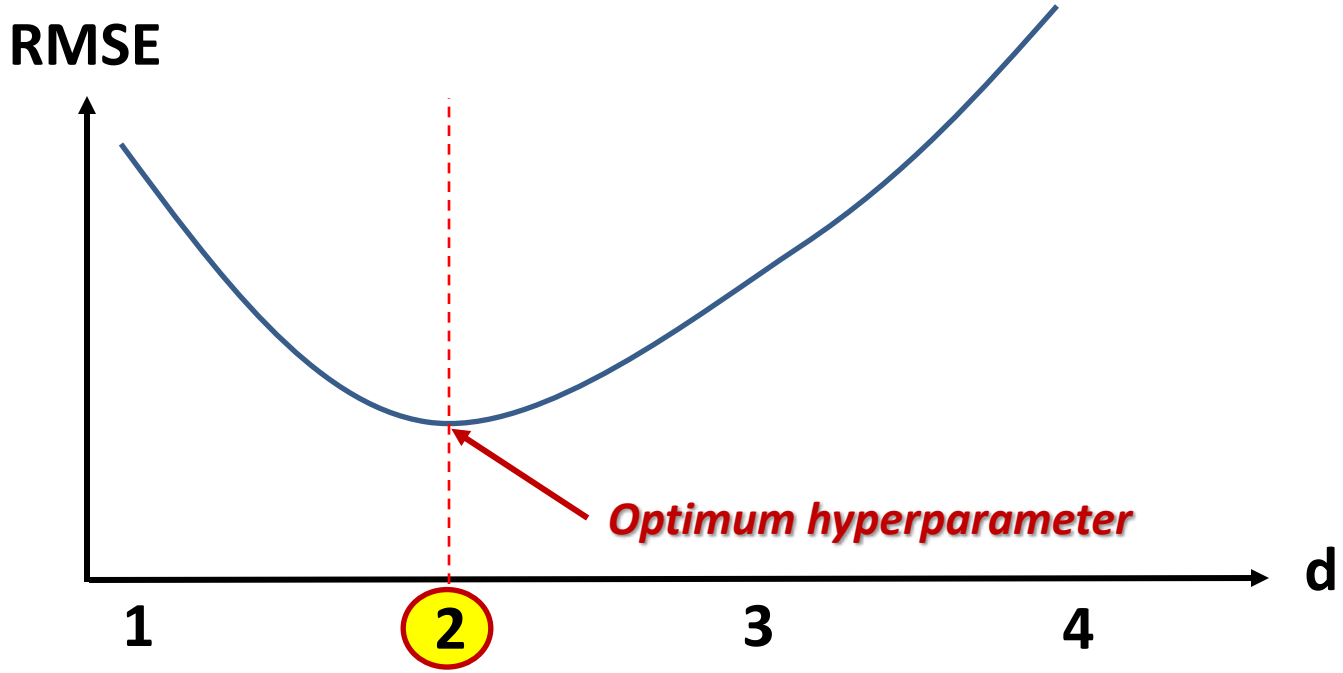
We repeat the CV calculation for each method we select during training and then select the model with the minimum CV error



Train	V		$(\beta_0, \beta_i)_{fold1}$, $RMSE_1$
Train	V	Train	$(\beta_0, \beta_i)_{fold2}$, $RMSE_2$
V	Train		$(\beta_0, \beta_i)_{fold3}$, $RMSE_3$

$$RMSE^{(d=5)} = \frac{1}{3}\sum_{i=1}^{3} RMSE_i$$

k-fold Cross Validation (Example)

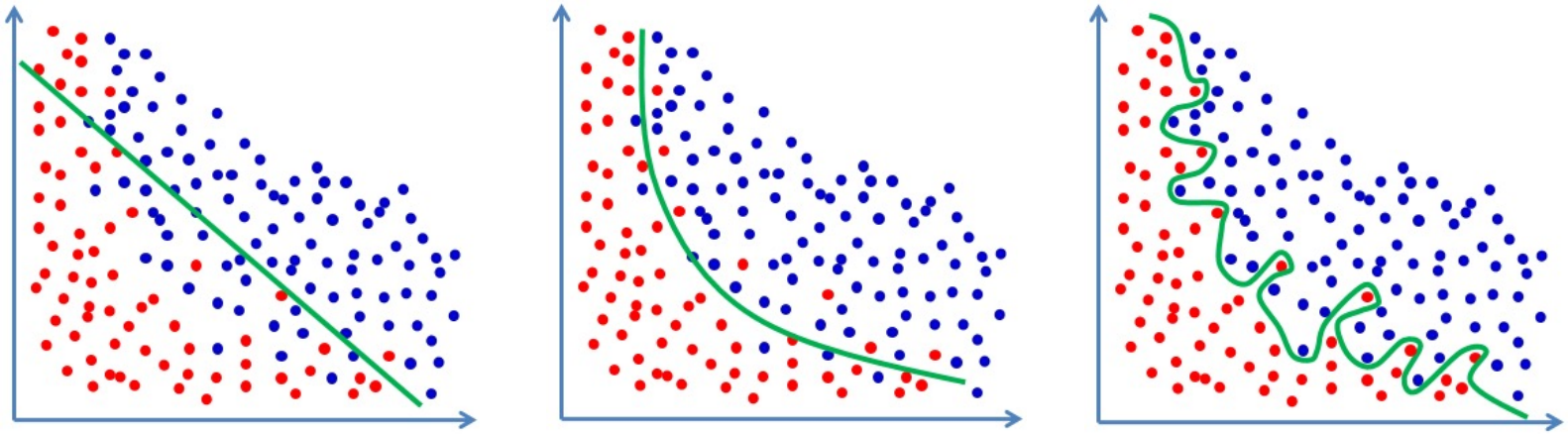


$y = \beta_0 + \beta_1 x + \beta_2 x^2$ ⇒ **TRAIN DATA**

- A quadratic regression $y = \beta_0 + \beta_1 x + \beta_2 x^2$ is re-trained on the entire training set one last time to find the coefficients to be used in the final model.
- Find the generalization performance of the model by running this model on the test data **TEST**

3. k-fold Cross Validation (Example)

- Suppose we're working on a classification problem.
- We repeat the cross-validation calculation for each method we select during training. We then select the model with the minimum Cross Validation error:



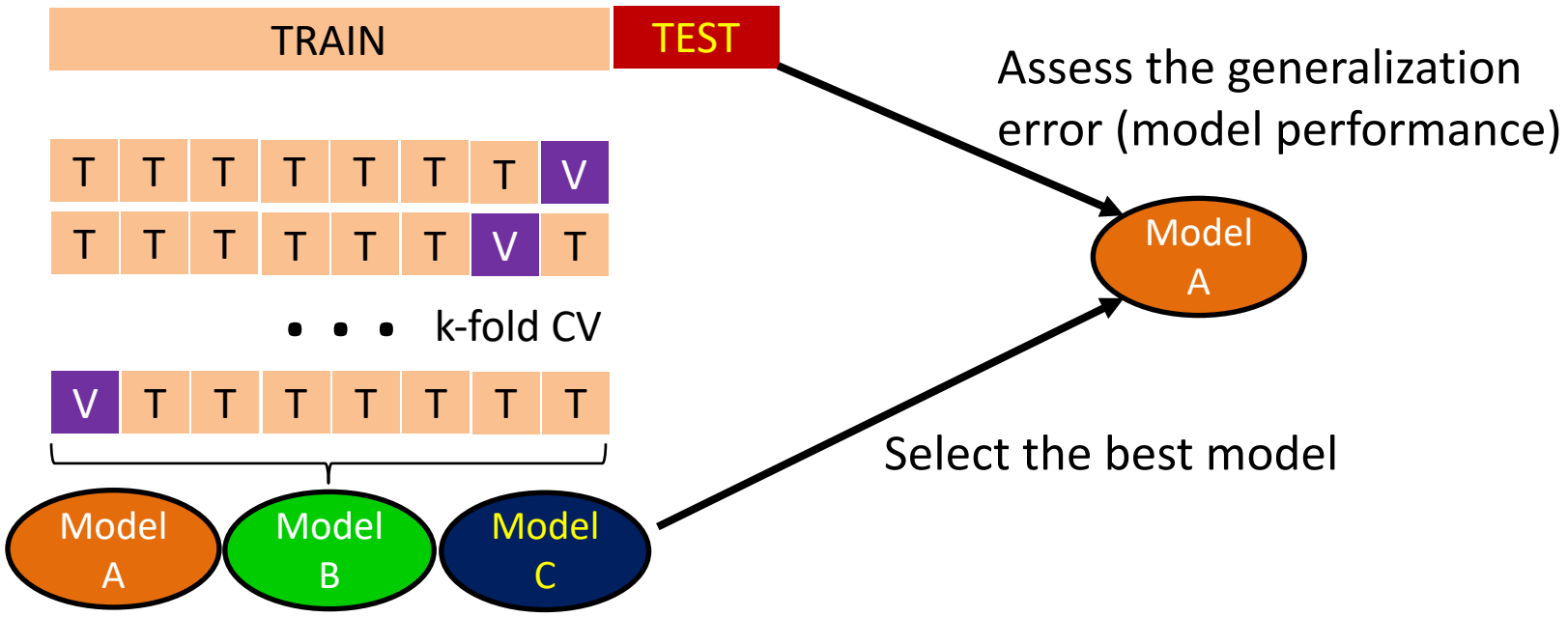
$$\frac{1}{k}\sum_{i=1}^{k} e_i = 19.7\%$$

$$\frac{1}{k}\sum_{i=1}^{k} e_i = 14.1\%$$

$$\frac{1}{k}\sum_{i=1}^{k} e_i = 24.6\%$$

- Now that we choose the best model (middle one), we can evaluate it on the test data.

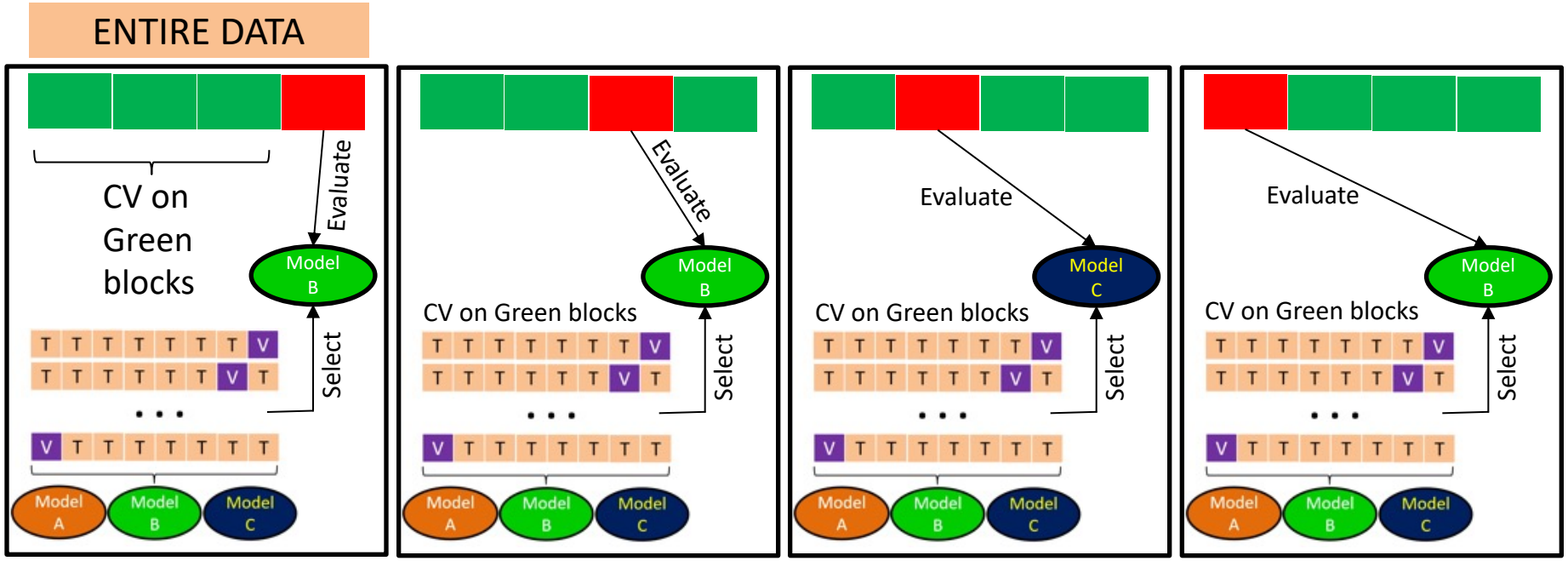
Any shortcomings of k-fold Cross Validation?



- We use the CV procedure to choose the best model (one with the lowest error) and use the test data to assess its generalization error.
- We fixed the variance in the validation set by using k-fold CV
- But we have a single test set that is randomly chosen. So, what about the variance in the test set? What would happen with a different test set? Do we get a different model? Different generalization performance?
- Solution? **Nested Cross Validation**

Evaluation via Nested Cross Validation

- 4-fold Outer Cross Validation
- k-fold Inner Cross Validation



- This could be computationally expensive. For n-fold outer CV, k-fold inner cv and trying out m models, you will end up doing n*k*m fits altogether. But this scheme easily lends itself to parallelization.

Evaluation via Nested Cross Validation

Outer Loop: CV for performance evaluation

Train	Test	C	Acc	Avg. Acc
D1,D2	D3	1	89%	83%
D1,D3	D2	2	84%	
D2,D3	D1	1	76%	

Inner Loop: CV for model selection/parameter tuning

Train	Validation	C	Acc	Avg. Acc
D1	D2	1	86%	85%
D2	D1		84%	
D1	D2	2	75%	82%
D2	D1		89%	

Choose C=1 as it maximizes accuracy

- The data are split into mutually exclusive data sets, say: D1, D2, and D3.
- The inner loop is used to determine the optimal value of the classifier's hyperparameter C.
- The model performance is estimated in the outer loop by training on all sample sets but one (test set).
- Repeat the entire procedure

Reference: journals.sagepub.com/doi/full/10.1177/117693510600200004

Reference: www.philipscurve.com/post/2020-10-24-nested-cross-validation/

Nested Cross Validation – cont'd

- What did we gain?
  - With a single CV, we got a single best model with a single estimate of generalization error.
  - With nested CV, we may have multiple best models. Some outer CV runs may yield different best models.
  - The nested CV does a better job in **assessing the generalization performance** and **its variance**.
- What to do with multiple models?
  - **We use nested CV to select the most stable algorithm** (one with the least amount of variation in test errors).
  - If the model stability (large variations in test errors) is not an issue, an ensemble of models could be the best option for a good generalization performance.

Which validation and when?

- Depending on the size and dimension of the data set:
- **Small data**: # of observations up to 2000 with 50 to 100 features
  - Use nested CV.
- **Medium data**: 100,000 observations / up to 1000 features
  - Use nested CV if you can afford it (computational and time constraints). Use k-fold CV otherwise.
- **Big Data**: Millions of observations
  - Use a Single train-validation-test split (no CV at all). A single set will suffice for validation purposes as the sample is large enough to capture the variability in the true data set (representative of the population).

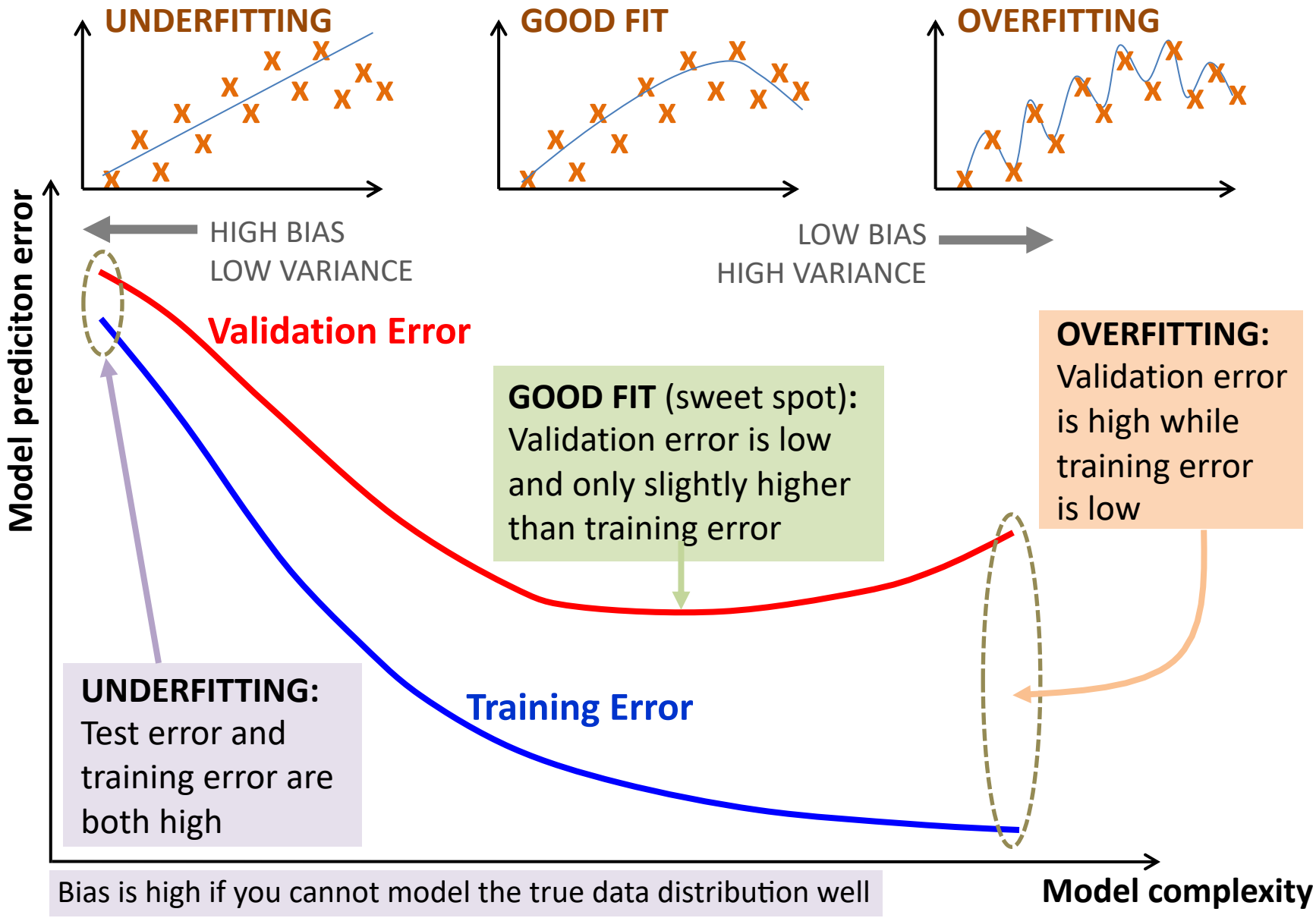
Reference: You Should Probably Be Doing Nested Cross-Validation | PyData Miami 2019

Repeated Cross Validation

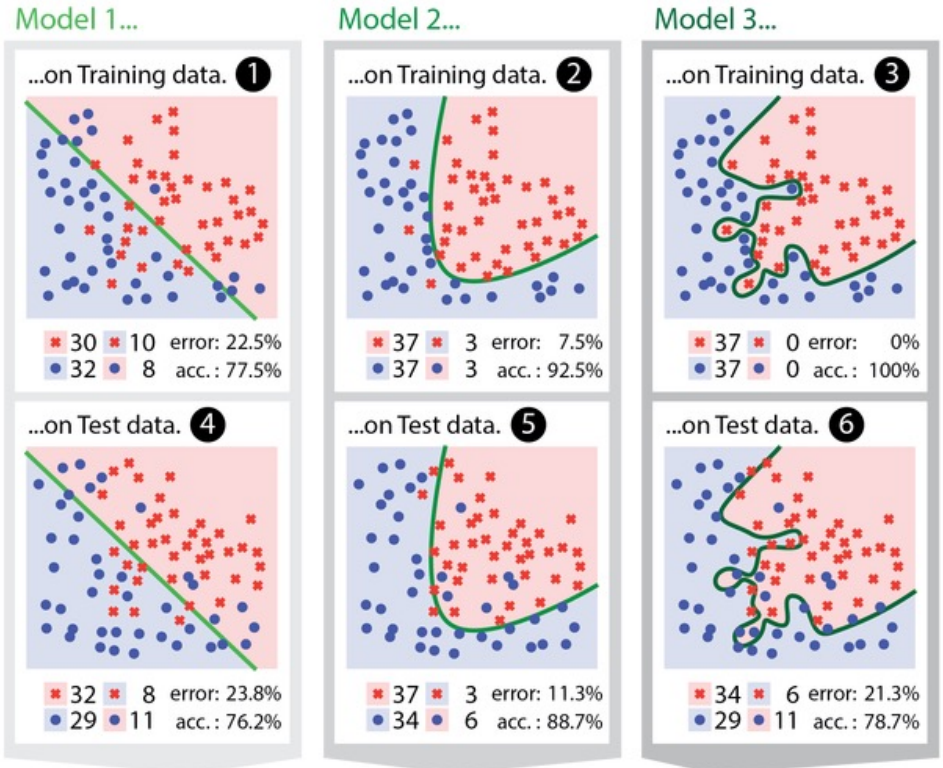
- With a single cross validation (a single run over all folds) cannot distinguish between model stability and variance due to variance in the validation folds.
- To measure the effect of the model stability (on predictions), one can run repeated cross validations on the same data set (shuffling the data before every iteration).
- If the model (with its parameters) is stable, irrespective of small changes in the data for different folds, we would get the same predictions (error rate).
- If the model is sensitive to small changes in the data, then the model predictions would differ across the iterations.
- Repeated cross validation helps assess the model stability.

```
RepeatedKFold(n_splits=5, n_repeats=10)
RepeatedStratifiedKFold(n_splits=5, n_repeats=10)
```

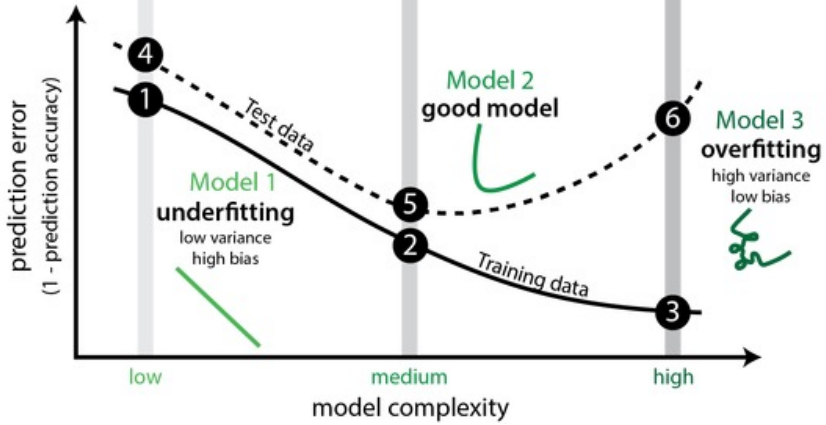
Bias-variance vs model accuracy: Regression



Bias-variance vs model accuracy: Classification



From cambridgecoding.wordpress.com/2016/03/24/misleading-modelling-overfitting-cross-validation-and-the-bias-variance-trade-off/



How to detect under/over-fitting?

- **Detecting overfitting**

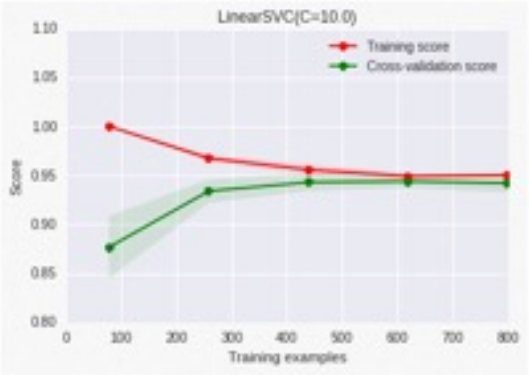
Upon plotting the accuracy (score) obtained for the training and validation sets, we notice that there is a large gap between error on training and validation data.

How to address overfitting?

i. Reduce complexity (decrease number of features or use regularization)

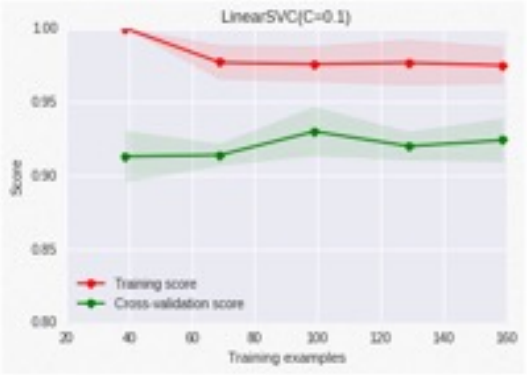
ii. Increase number of training instances

Using more data points



Using less # of features

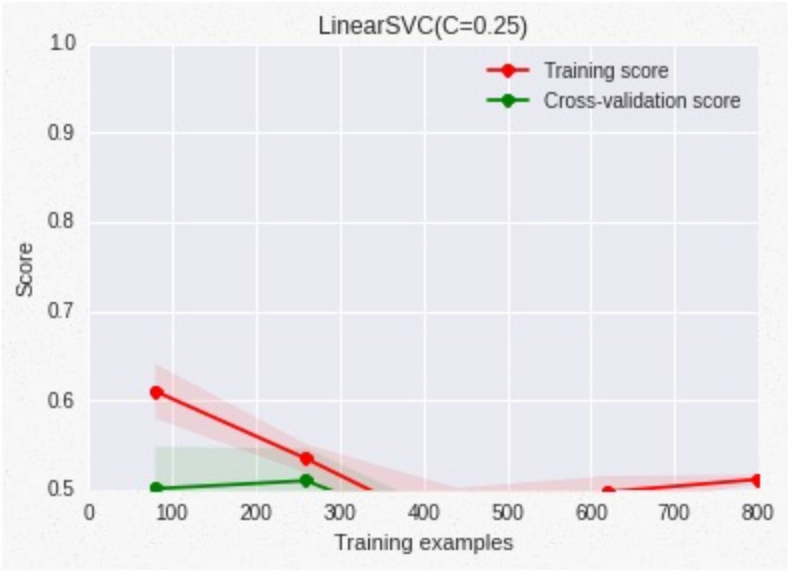
Increasing regularization



Ref: http://jmetzen.github.io/2015-01-29/ml_advice.html

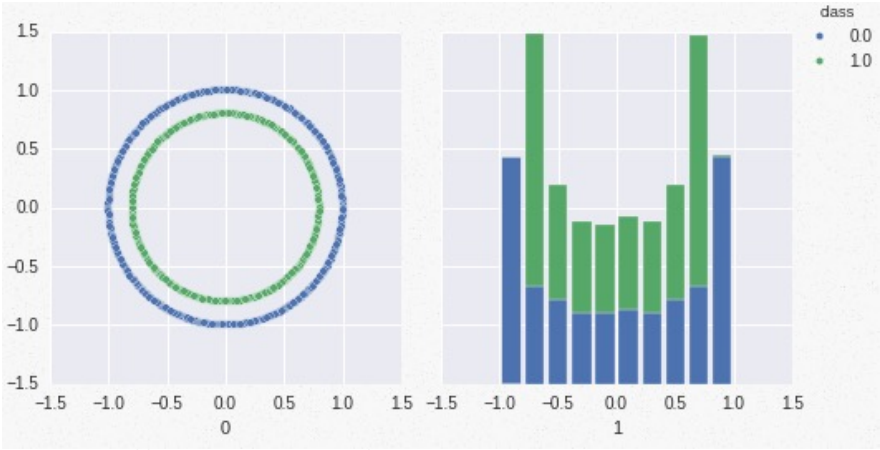
How to detect under/over-fitting? – cont'd

- **Detecting underfitting**



Upon plotting the accuracy (score) obtained for the training and validation sets, we realize that the result is horrible. Even the training error is awful. What helped us before in the previous case will not be helpful as we have an entirely different problem now. We're underfitting!

How to address underfitting?

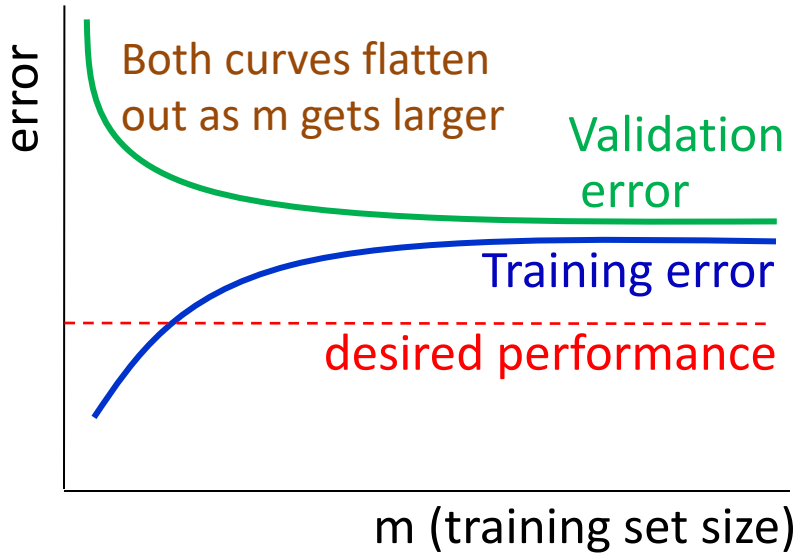


This plot shows us that the data is clearly not separable. So more data or less features won't help. We need a more complex model that can deal with nonlinear decision boundaries.

Ref: http://jmetzen.github.io/2015-01-29/ml_advice.html

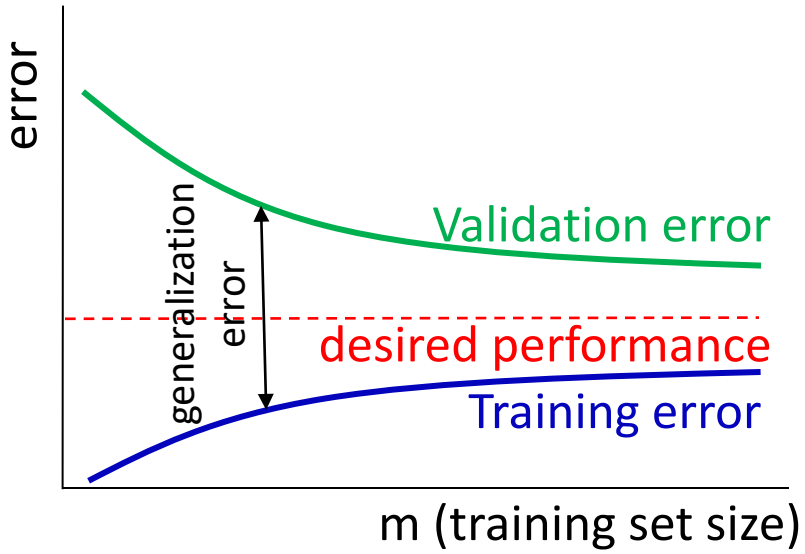
Learning curves – cont'd

Learning curve for high bias



- Even training error is unacceptably high
- Small gap between training and test error (and narrow down as m increases)
- If a learning algorithm is suffering from high bias, adding more training data will not help much

Learning curve for high variance



- Test error still decreasing as m increases suggesting that larger training set will help
- Large gap between training and test error, a sign for poor generalization

Ref: http://www.holehouse.org/mlclass/10_Advice_for_applying_machine_learning.html
https://see.stanford.edu/materials/aimlcs229/ML-advice.pdf (Andrew Ng)
https://www.kdnuggets.com/2018/01/learning-curves-machine-learning.html

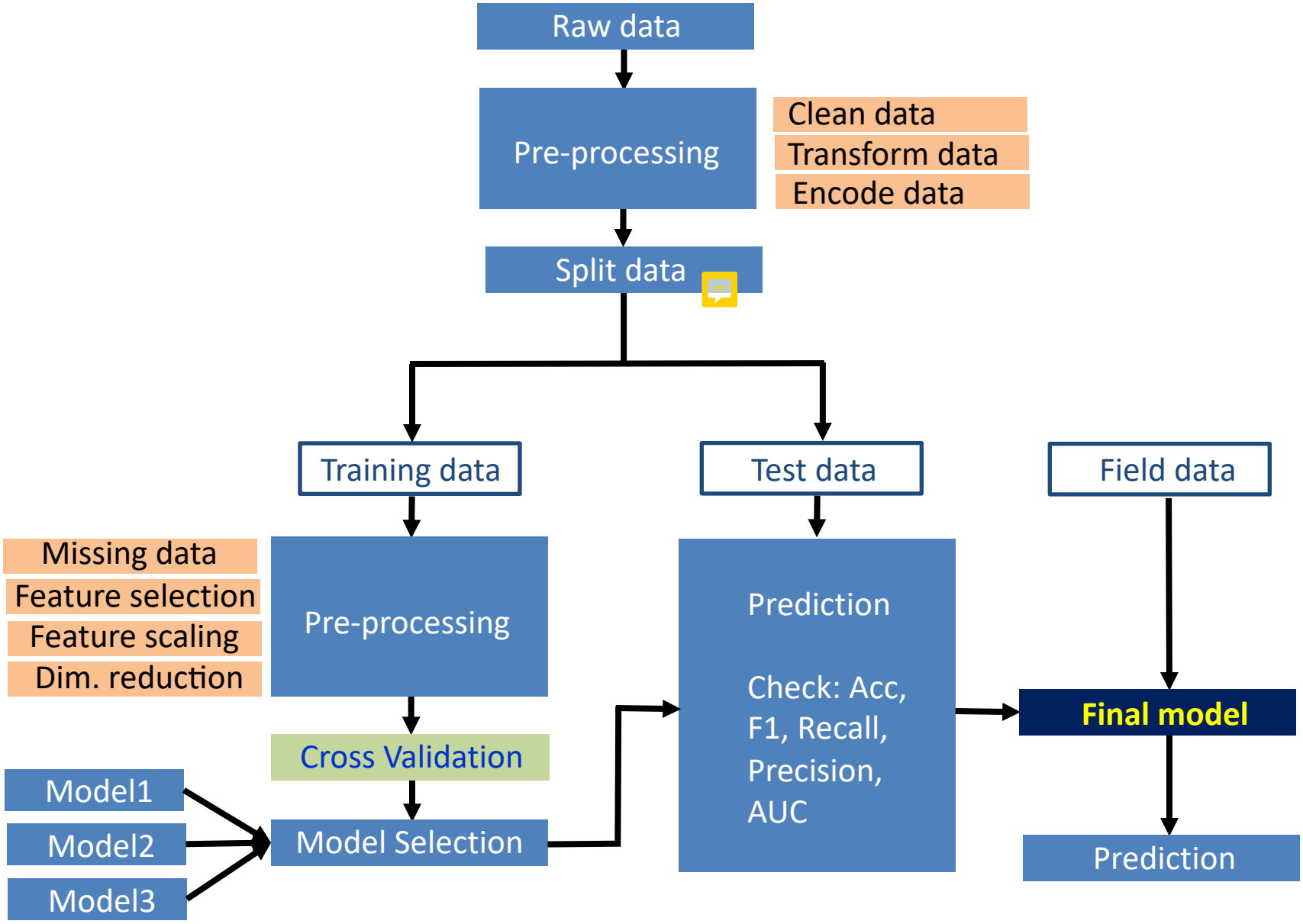
How to deal with over– and under–fitting

- We evaluate the training and validation errors to conclude whether our algorithm overfits or underfits.
- In case of high bias (underfitting)
  - Add more features (attributes)
  - Use a more sophisticated model (e.g. add more polynomial features)
  - Using more training data won't help (bias doesn't depend on training data size)
- In case of high variance (overfitting)
  - Reduce number of features
  - Reduce the complexity of the model (choose a simpler classifier if nothing else works)
  - Cross-validation keeps the variance low
  - Get more training data (might help)
  - Regularize the parameters (panelize parameters causing overfitting)
  - Early stopping, pruning (in decision trees)
- If your training set is small, you should use a high-bias/low-variance classifier (e.g. Naive Bayes).
- Low bias/high variance classifiers are more appropriate if you have a large training set as they have a smaller asymptotic error.

Bias/variance characteristics of ML models

- Parametric or linear machine learning algorithms often have a high bias but a low variance.
- Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.
- High bias-Low variance models:
  - Naïve-Bayes, Perceptron
- Low bias-high variance models:
  - Logistic regression, kNN, Decision trees, SVM, Neural Networks
  - For kNN, bias ↗ as k ↗ and variance ↗ as k ↘
  - For SVM, as C ↗, bias ↗ and variance ↘

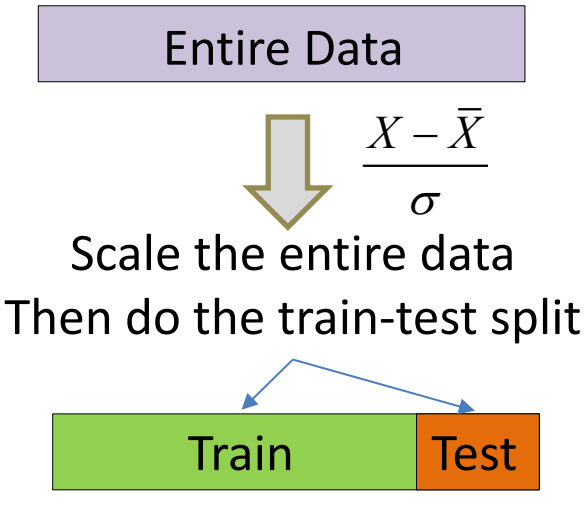
A roadmap on Supervised Learning

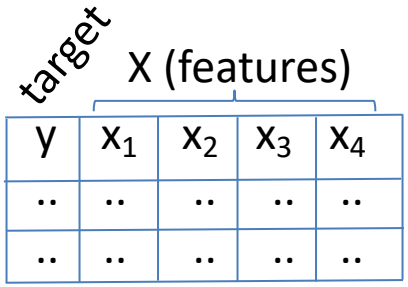


Data Leakage issues with Cross Validation

Issues with CV: Scaling

- In the presence of features with disparate scale, scaling the data is of importance for certain models such as kNN, SVM, ANN, or when applying regularization.
- So how do we scale data?

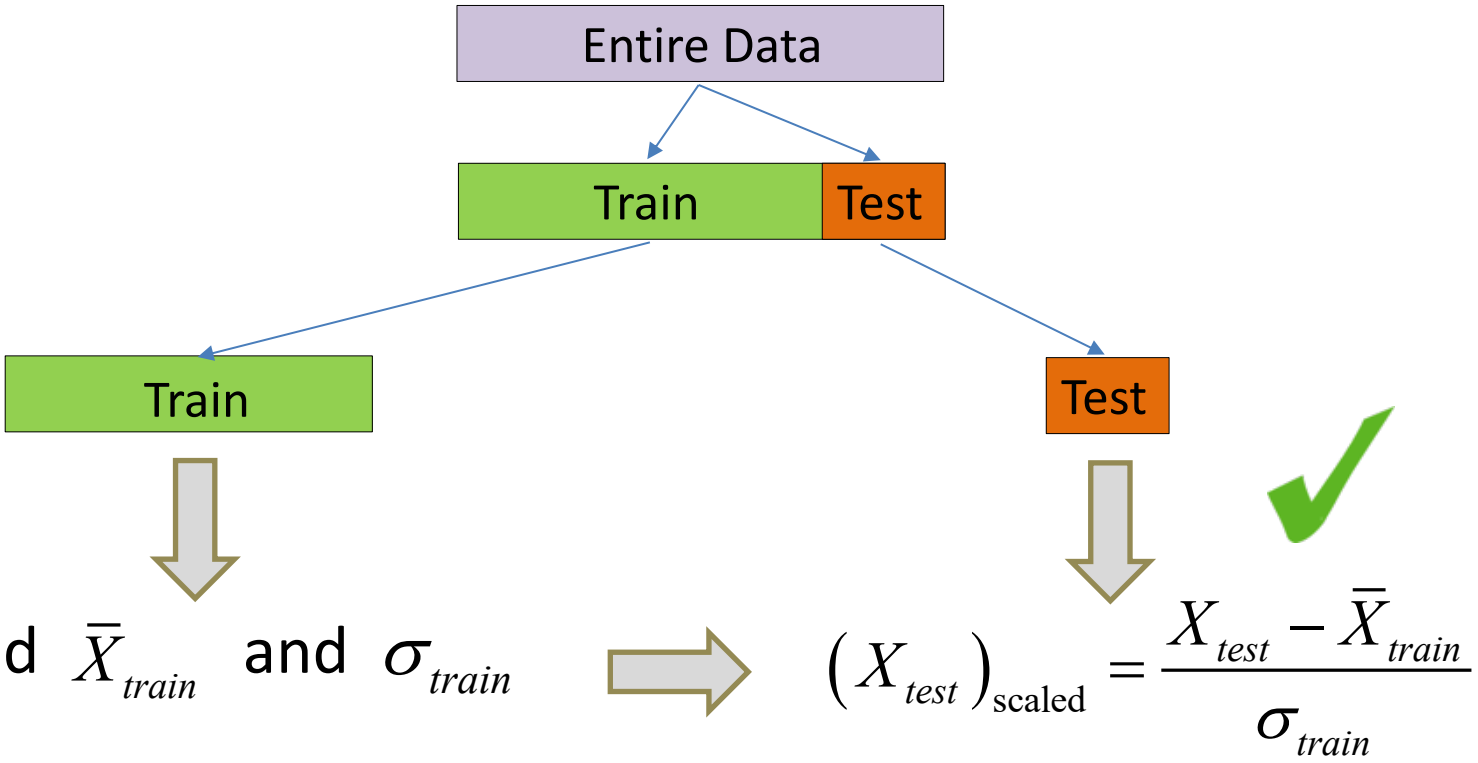




- Via CV done this way, **we're using a variable transformer ($\bar{x}$) which contains information from the test part**. So the information from the test data leaks into the training data (aka data leakage) leading to optimistic model accuracies.

Issues with CV: Scaling – cont'd

- We need to apply scaling after we split the data into train and test parts:



- If you need to do Cross Validation, this procedure needs to be applied in the same way between the train and validation folds.
- Using **`pipeline`** is the safest/easiest method.

Issues with CV: Scaling – cont'd

- **Pipelines**: They help implement proper transformations (scaling, imputing, encoding etc.) by
  - enforcing desired order of steps by encapsulating multiple different transformers making the procedure error free

  - creating a convenient work-flow (good for reproducibility) with clean code writing standards
  - facilitating easy, fast, and proper use of critical tools such as cross validation and grid search.
- Pipelines glue together the transformations and the estimator to ensure that the transformation is fitted only to the training folds during CV procedure. This prevents any data leakage from the validation fold.

Issues with CV: Scaling – cont'd

- Pipeline examples:

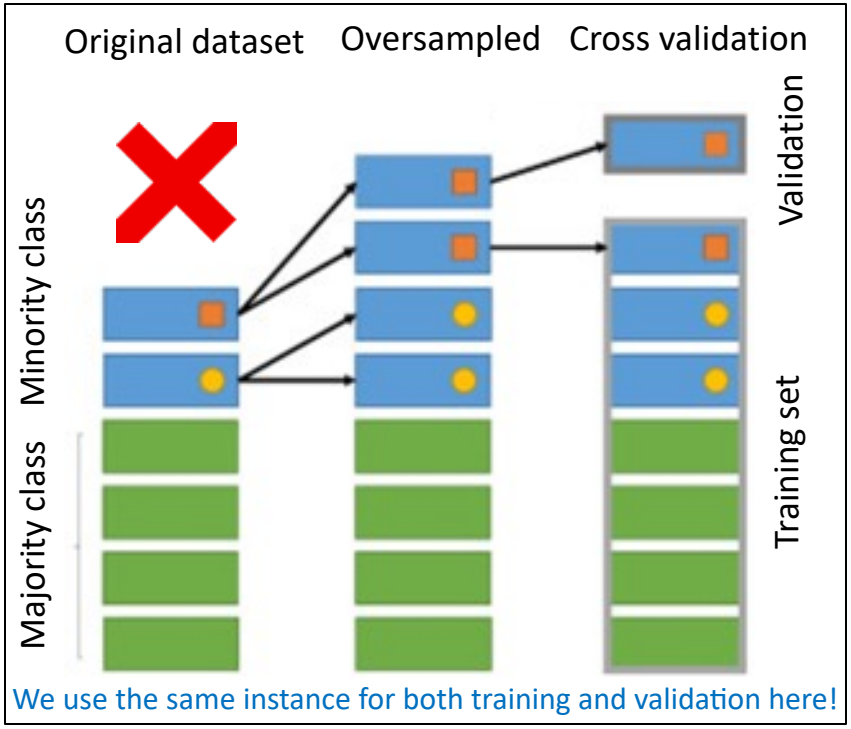
```
from sklearn.pipeline import Pipeline
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
grid = GridSearchCV(LogisticRegression(),
                    param_grid=parameters, cv=5)
grid.fit(X_train_scaled, y_train)
```

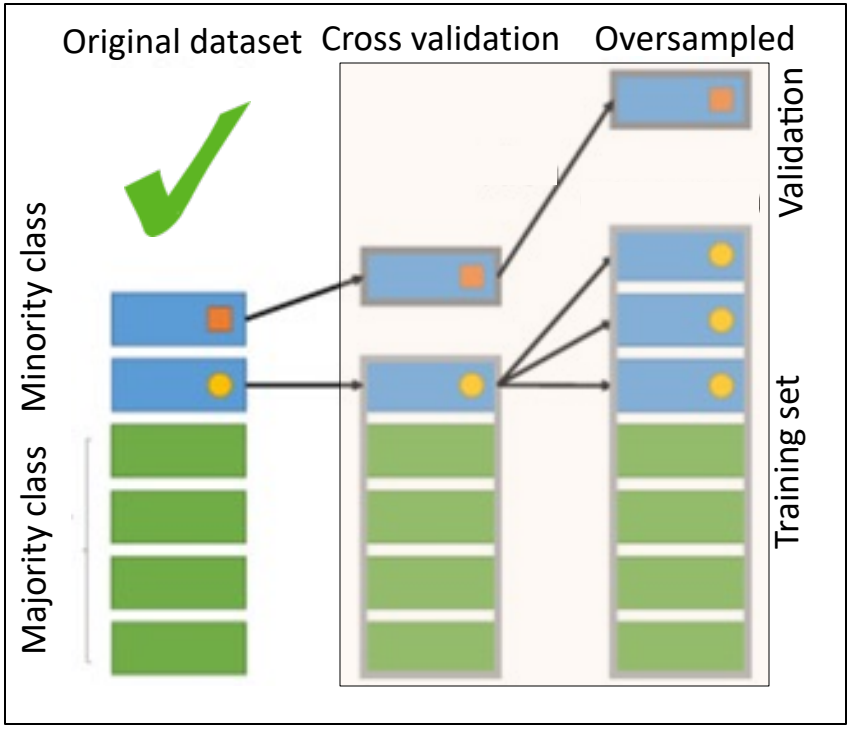
- What's wrong with this approach? When grid.fit() is done using cross-validation, the features already include info from the test-fold as StandardScaler.fit() was done on the whole training set (data leakage). Here is the proper one:

```
pipe = Pipeline([('scaler'    , StandardScaler()),
                 ('classifier', LogisticRegression())])

grid_search =GridSearchCV(pipe,
                          param_grid=parameters, cv=5)
grid_search.fit(X_train, y_train)
```

Issues with CV: Imbalanced data sets

- For imbalanced data sets, the classifier turns out to be biased towards the dominating class. So the sensitivity is ~ 0, and specificity is ~ 1 (see next chapter for definitions).
- When using **oversampling**, be careful with the CV procedure: At each iteration, first exclude the sample to use as validation set, and then oversample the remaining of the minority class.





From www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation

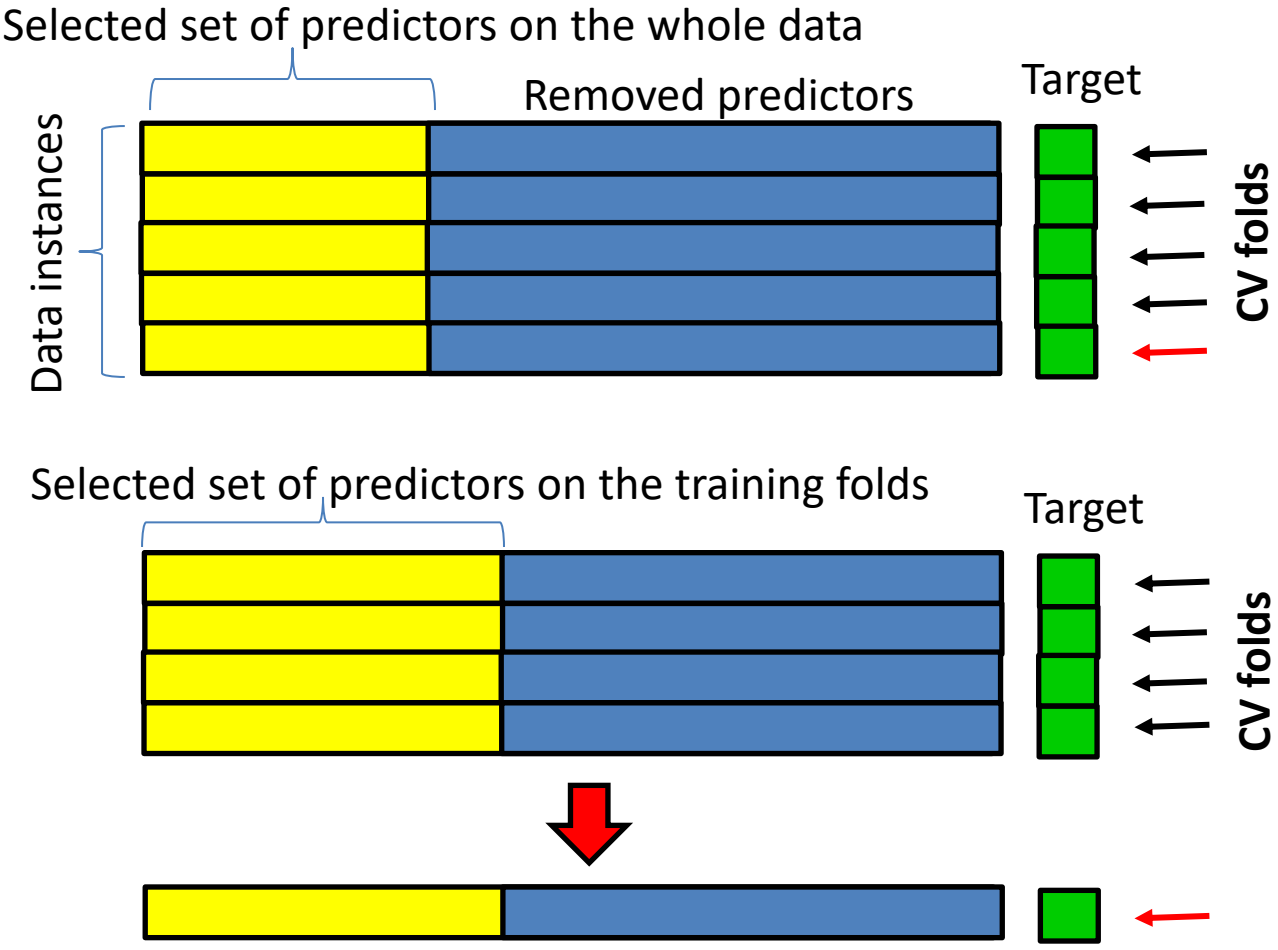
Issues with CV: Imbalanced data sets – cont'd

- Scikit-learn implementation

```
from imblearn.pipeline import Pipeline

pipe = Pipeline([('imputer', IterativeImputer())
                 ('scaler', StandardScaler()),
                 ('sampling', SMOTE()),
                 ('classifier', LogisticRegression())
                ])

params = [{'classifier__C': [1, 10, 100, 1000],
           'classifier__solver': ['lfgbs', 'saga'],
          }]

grid = GridSearchCV(estimator=pipe,
                    param_grid=params,
                    cv=5)
grid.fit(X,y)
```

Issues with CV: Feature selection

- A similar issue surfaces during feature selection: If you want to remove features that are weakly correlated with the target (say, features whose correlations with the target < 0.5)



Other Performance Metrics

Other performance measures

- **Example**: Spam detection in emails
  - Training data: 1300 instances with 150 spam, 1150 ham (non-spam)
  - You trained a classifier to do spam filtering and detected 100 spams and 1000 hams correctly
  - Classification accuracy: Ratio of correctly labeled instances to all instances
  - Classification accuracy = 1100/1300 = 84.6%
  - You may be thinking that this classifier is not so bad as it detected almost 85% of cases correctly.
  - What about a dumb classifier without a model that classifies everything as "no spam". How accurate is it?
  - Classification accuracy = 1150/1300=88.5% ???

Other performance measures (cont'd)

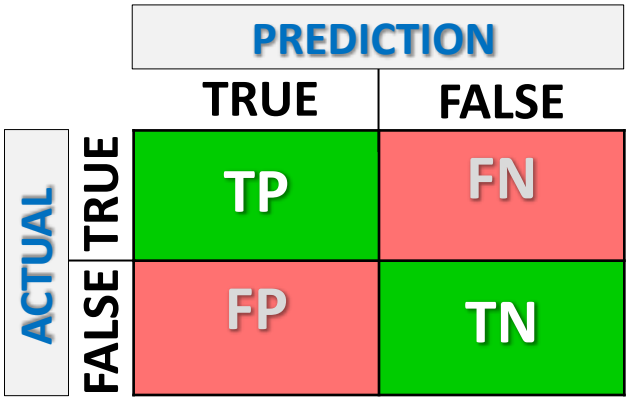
- A completely useless classifier that has no predictive power has a better accuracy?
- This is called the "**accuracy paradox**"
- This could happen in classification problems with imbalanced data (such as spam and fraud cases) where one class dominates the other.
- If a single class contains most of the data, classifying all cases belonging to this majority class will produce an accurate result.
- The classification accuracy doesn't tell us much about the performance of the classifier for this case.
- We clearly need something else as a performance measure. So, what is there to consider?

Other performance measures (cont'd)

- Measures of performance:
- We have other measures of performance focusing on misclassifications, correctly predicted positives, incorrectly predicted negatives, etc. depending on the type of problem or the risk involved in the decision.
- These are:
  - False Positives / False Negatives (confusion matrix)
  - Precision / Recall, PR-curve-AUC
  - F1 score (weighted F1 score)
  - Cohen's Kappa score
  - ROC-AUC

Confusion Matrix

- Separates the decisions made by the classifier taking into account how one class is being confused for the other:



True Positives (TP): + examples predicted as +

False Positives (FP): - examples predicted as +

True Negatives (TN): - examples predicted as -

False Negatives (FN): + examples predicted as -

- Total number of examples = TP + TN + FP + FN
- For a perfect classifier: FP = FN = 0

Classification ACCURACY:
(How often is it correct?)

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

Misclassification rate: ERROR:
(How often is it wrong?)

$$\frac{FP+FN}{TP+TN+FP+FN} = 1 - \text{Accuracy}$$

Example

- **Case**: Classify documents as **urgent** or **non-urgent**
- **Dataset**: A pool of 10 urgent , 90 non-urgent documents
  - What do you optimize for? Accuracy?
  - If your model's accuracy is 90%, what would this mean?
- **Output I**: Suppose your model classified
  - 80 non-urgent docs correctly
  - 20 as urgent
  - **Result**: No missed urgent documents + 90% accuracy
- **Output II**: Your model marked everything as non-urgent
  - 90 non-urgent docs correctly identified
  - 10 urgent docs (all of them) are misclassified!!!
  - **Result**: Missed all urgent documents + 90% accuracy ?
- So 90% accuracy tells us nothing!

Precision and Recall

- We see that accuracy is not a good metric when the actual cost assigned to every error (FP or FN) is not equal!
- We need other metrics: **Precision** and **Recall**

$$\text{Precision} = \frac{\text{TP}}{\text{Predicted}(+)} = \frac{\text{TP}}{\text{TP+FP}}$$

How many of predicted urgent documents are actually urgent

want less false positives for better precision

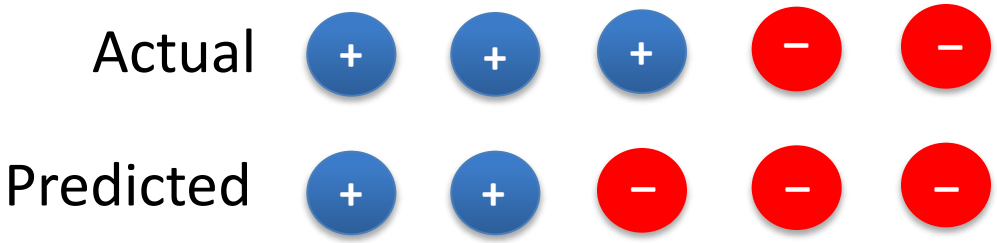
$$\text{Recall} = \frac{\text{TP}}{\text{Actual}(+)} = \frac{\text{TP}}{\text{TP+FN}}$$

How many of the total urgent documents are predicted correctly as urgent

want less false negatives for better recall

- **Precision**: How many of the (+) predictions are relevant? (measure of a classifier's exactness)
- **Recall** (aka **Sensitivity**): How good a test is at detecting the positives? (measure of a classifier's completeness)

Precision and Recall – cont'd



		Predicted +	Predicted −
Actual	+	2 (TP)	1 (FN)
	−	0 (FP)	2 (TN)

A measure of exactness

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{2}{2+0} = 100\%$$

How many predicted blues were really blue?

A measure of completeness (sensitivity)

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{2}{2+1} = 67\%$$

How many blues are predicted correctly?

So Precision is only part of the story!

How about precision and recall for the "red" class?

Precision and Recall – cont'd

- For a given class, the different combinations of recall and precision have the following meanings :

- **high recall + high precision** :
  The model handles the class well

- **low recall + high precision** :
  The model can't detect the class well but is highly trustable when it does

- **high recall + low precision** :
  The class is well detected but the model also include points of other classes in it

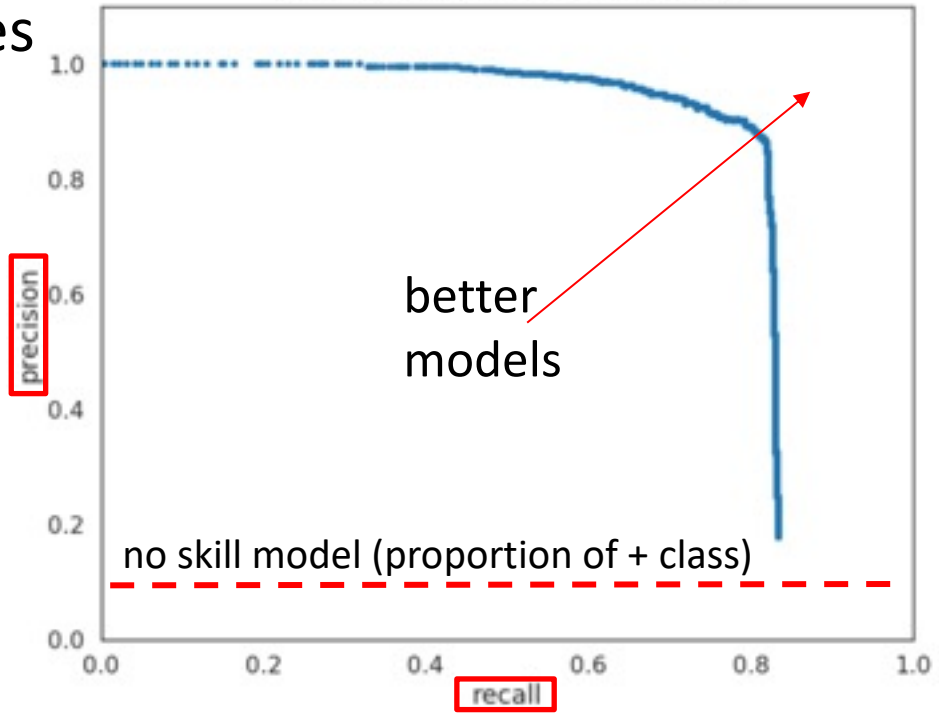
- **low recall + low precision** :
- The class is poorly handled by the model

Precision and Recall – cont'd

- Accuracy is a choice of performance metric for well balanced (not skewed) problems. For the document classification problem, we need to optimize for Recall.
- Do we always optimize for Recall?
- **Examples**:
- Detection of cancerous cells: Minimize (avoid) False Negatives (due to high cost for FN) => Recall
- Spam detection for e-mail: Minimize FP => Precision
  - FP: Marked as spam, but it isn't! => Could miss a critical mail
  - FN: Marked as NOT spam, but it is => Inconvenient/annoying
- If your prediction is True for all instances, you'll get a perfect Recall score (100%). But is this a good thing? How about Precision?

Precision and Recall – cont'd

- Problem: You want to catch all criminals. and identify
- Solution: Identify everyone as criminal => No FN,
  So, Recall = $\frac{TP}{TP+FN} = 1$
- On the other hand, we'll have many FP's, therefore a very low value of Precision.
- High values of Recall causes low Precision & vice versa. So, there is a tradeoff.
- See the Precision-Recall curve on the right:
- We can use the area under the PR-curve as a metric (more later).



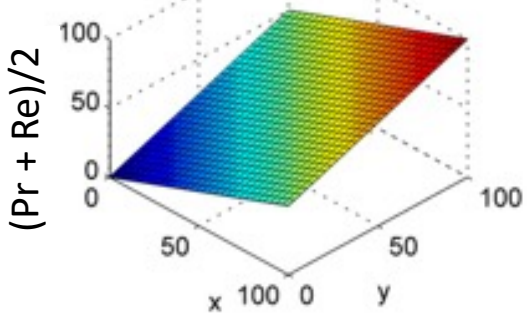
F-measure

- F-measure is a useful metric used for imbalanced classes involving both Precision and Recall. For an optimal mix of both, we can combine the two as a harmonic average:

$$F1 = 2\frac{\text{Precision*Recall}}{\text{Precision} + \text{Recall}}$$

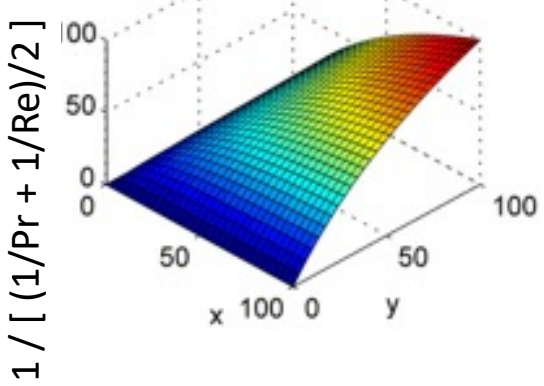
Why not a simple average? The denominator is not the same in Precision and Recall. So, the arithmetic mean won't be in the same scaled units.

Arithmetic mean



Harmonic mean penalizes extreme values

Harmonic mean



- How is F1 different from Accuracy?
  - Accuracy might be dominated by the large number of TN's. So F1 score is probably a better measure to use if we need to search for a balance between Precision and Recall, and there is an uneven class distribution (actual negatives >> actual positives).

Precision and Recall – cont'd

- F1 is not of much use for balanced data sets. In case of balanced classes, accuracy is a reasonable performance metric.
- When the class is unbalanced (highly skewed):
  - If both classes are important, optimize F1 score on both classes
  - If one class is more important, then choose the classifier with a good F1 score on the important class (F1 is available for both)
- When Precision & Recall are used together, you get all the pieces of available information. There is more to F-score:
- F-beta measure:

$$F_\beta = (1+\beta)^2 \frac{\text{Precision*Recall}}{(\beta^2 \text{Precision}) + \text{Recall}}$$

Are FN's & FP's equally costly?	Use F1-measure
Are FN's more costly?	Use F2-measure ($F_{\beta=2}$)
Are FP's more costly?	Use F0.5-measure ($F_{\beta=0.5}$)

Summary

Classification ACCURACY

How often is it correct?

(TP + TN) / Total

Misclassification rate (ERROR)

How often is it wrong?

1 – Accuracy = (FP + FN) / Total

RECALL (Sensitivity)

TPR: True Positive rate

When it's actually (+), how often does it predict (+)?

$$\frac{\mathbf{TP}}{\mathbf{Actual\,(+)}} = \frac{\mathbf{TP}}{\mathbf{TP + FN}}$$

How good a test is at detecting the positives?

A measure of a classifier's completeness

PRECISION

When it predicts (+), how often is it (+)?

$$\frac{\mathbf{TP}}{\mathbf{Predicted\,(+)}} = \frac{\mathbf{TP}}{\mathbf{TP + FP}}$$

A measure of a classifier's exactness

How many of the (+) predictions are relevant?

FPR: False Positive rate

When it's actually (-), how often does it predict (+)?

$$\frac{\mathbf{FP}}{\mathbf{Actual\,(-)}} = \frac{\mathbf{FP}}{\mathbf{FP + TN}}$$

Specificity (1-FPR)

How good a test is at avoiding false alarms?

When it's actually (-), how often does it predict (-)?

$$\frac{\mathbf{TN}}{\mathbf{Actual\,(-)}} = \frac{\mathbf{TN}}{\mathbf{FP + TN}}$$

- **ROC** (Receiving Operator Characteristic)
  - ROC curve was first developed by electrical and radar engineers during World War II for detecting enemy objects.
  - ROC curve is a graphical way to show the connection or trade-off between the True Positive Rate (TPR, recall or sensitivity) and the False Positive Rate (FPR).

(recall)

$$TPR = \frac{TP}{\text{Actual } (+)} = \frac{TP}{TP + FN}$$

When it's actually (+), how often does it predict (+)?

$$FPR = \frac{FP}{\text{Actual } (-)} = \frac{FP}{FP + TN}$$

When it's actually (-), how often does it predict (+)?

  - Provides a measure of separability between the (+) and (-) classes in a binary classification problem.

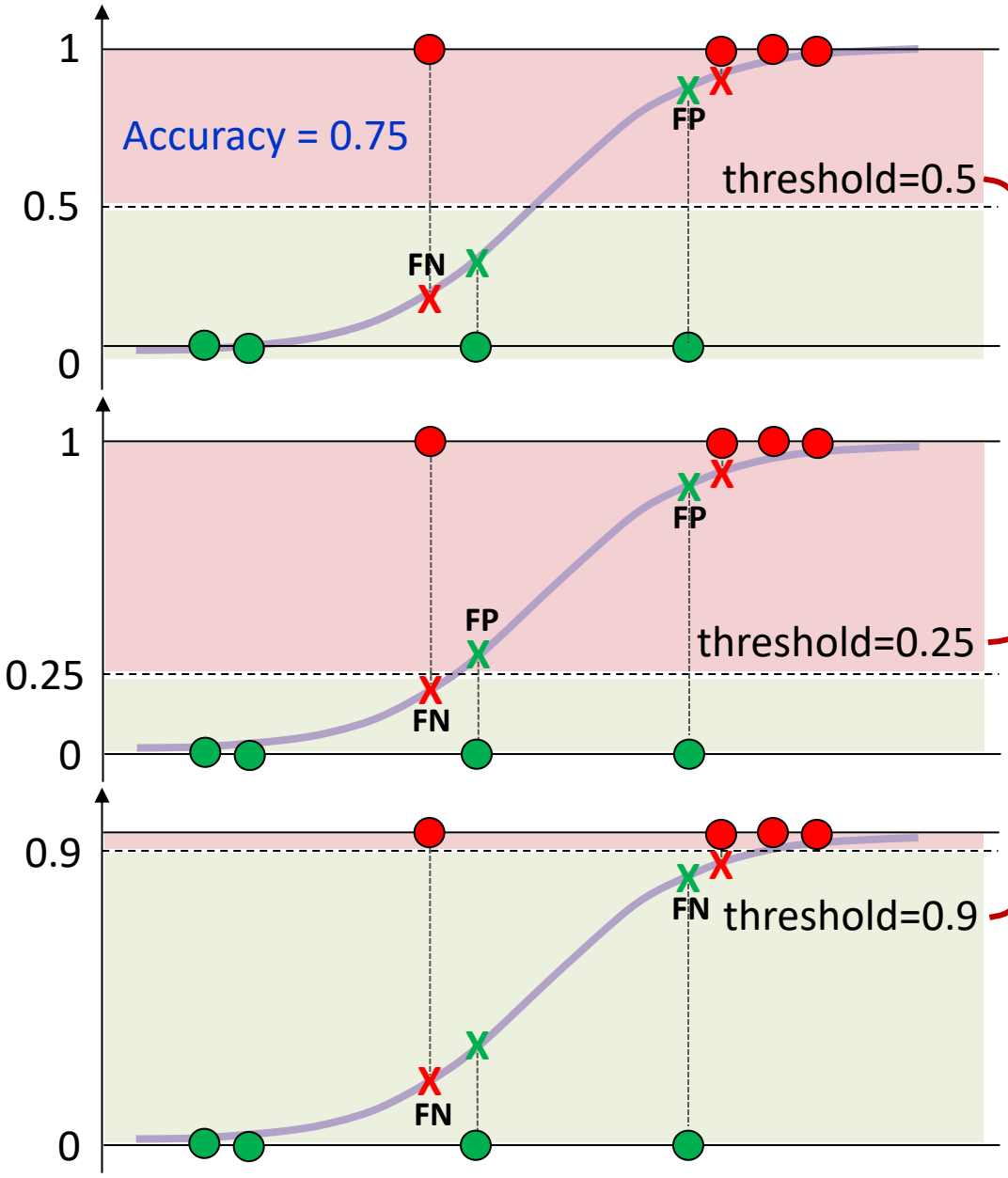
ROC – cont'd

- We got the probabilities from our classifier. Logistic Regression, for example, classifies an instance depending on the probability by choosing a threshold:
- For the i^{th} instance, if $p_i \geq 0.5$ => class is (+), (-) otherwise.

$$p_i \geq 0.5 \Rightarrow \text{class is (+)}$$
$$p_i < 0.5 \Rightarrow \text{class is (-)}$$

- 0.5 is the default threshold here for the class assignment. Our classification accuracy depends on the threshold chosen.
- If we change the threshold, we'll end up with different values of TPR and FPR for every threshold. ROC curve is what we got for TPR and FPR as threshold is varying.
- The best threshold has the highest true positive rate together with the lowest false positive rate.

How do we get ROC?



y : 1=Heart Attack / 0=No HA
x : Cholesterol level (mg/dl)

$$TPR=\frac{TP}{TP+FN} \qquad FPR=\frac{FP}{FP+TN}$$

TP=3 FN=1 FP=1 TN=3
TPR=3/4=0.75 FPR=1/4=0.25

TP=3 FN=1 FP=2 TN=2
TPR=3/4=0.75 FPR=2/4=0.5

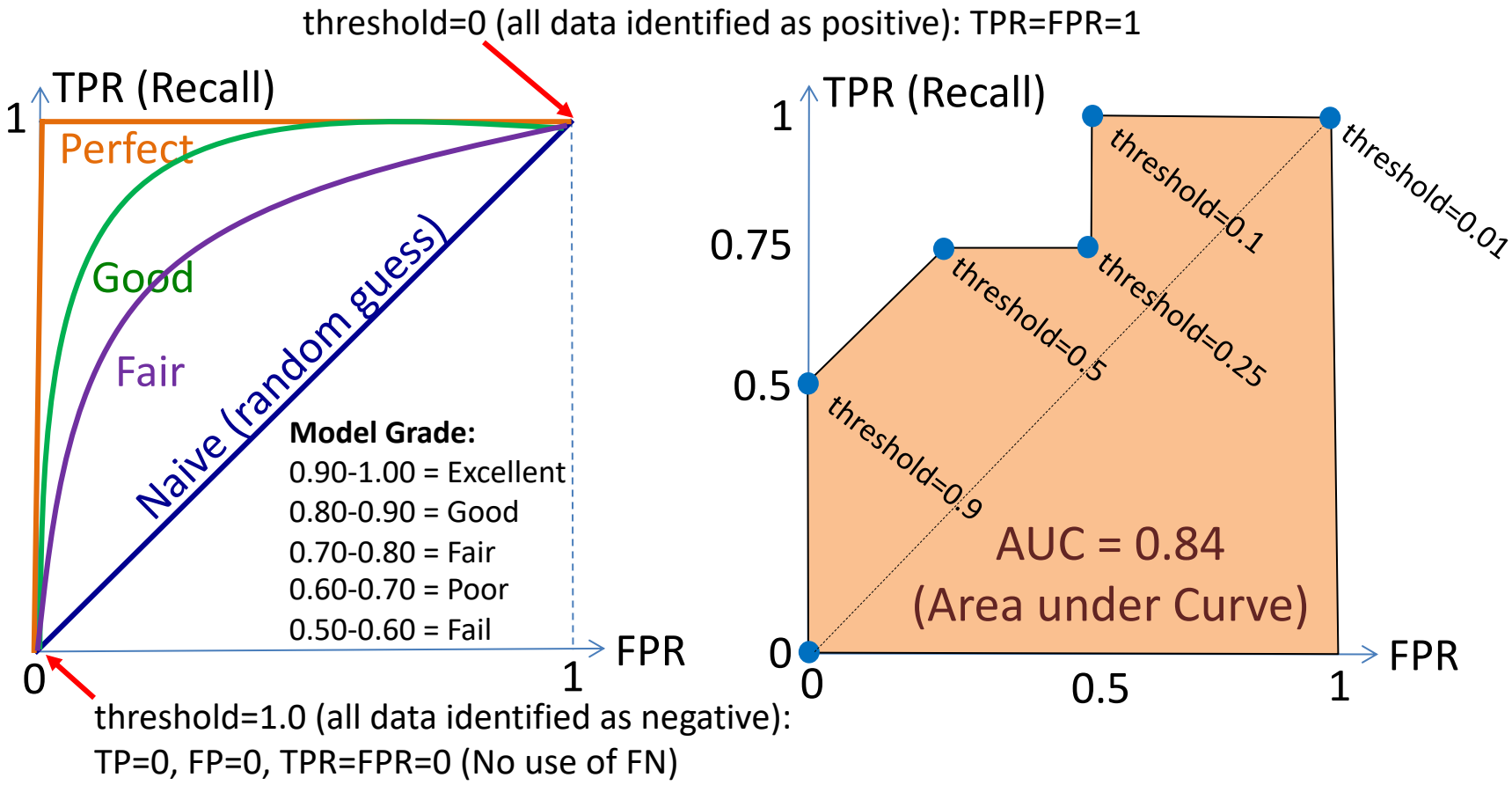
TP=2 FN=2 FP=0 TN=4
TPR=2/4=0.5 FPR=0

threshold=0.1
TP= 4 FN=0 FP=2 TN=2
TPR=4/4=1 FPR=2/4=0.5

threshold=0.01
TP=4 FN=0 FP=4 TN=0
TPR=4/4=1 FPR=4/4=1

ROC curve and AUC

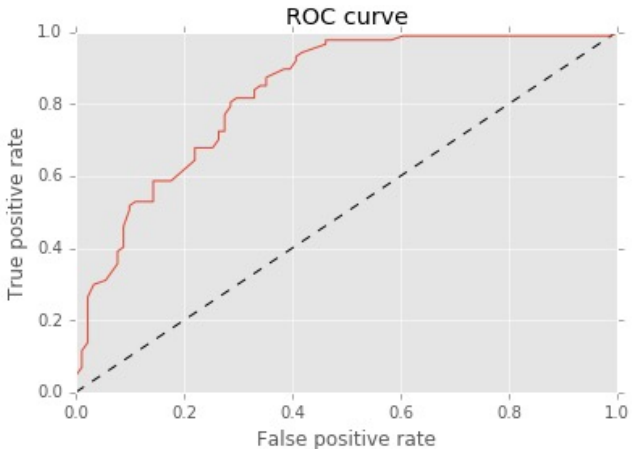
- An ROC curve shows the tradeoff between true and false positives as a function of threshold (cut-off):



- Area underneath the ROC curve is known as AUC (Area Under Curve) and can be used as a classifier performance metric.

Interpretation of ROC

- ROC is really the tradeoff between cost and benefit.
- Let's assume we have the following confusion matrix:



ACTUAL	PREDICTED: Stock UP	PREDICTED: Stock DOWN
Stock UP	30 (TP)	10 (FN)
Stock DOWN	20 (FP)	40 (TN)

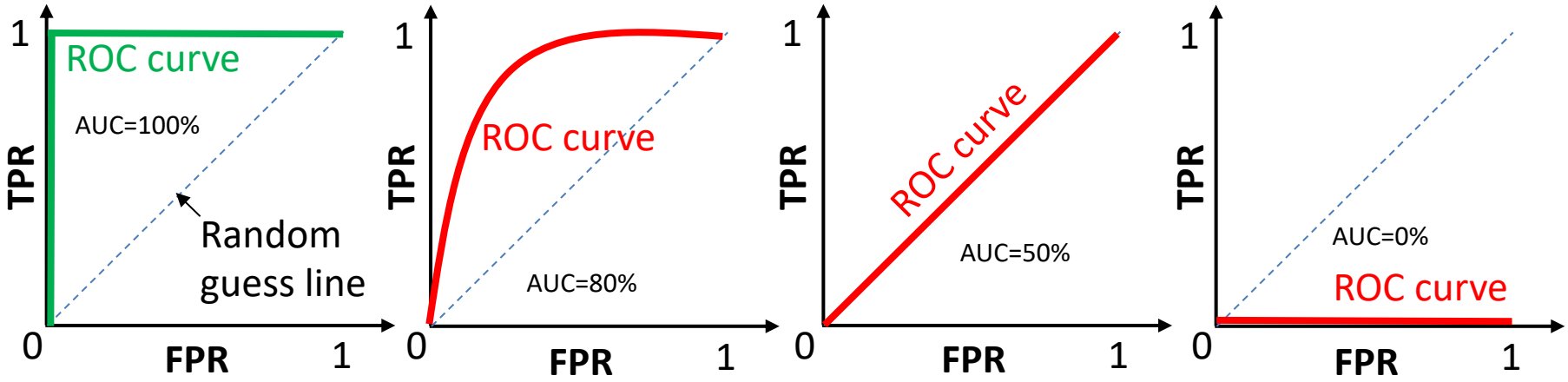
(BENEFIT) $$TPR = \frac{TP}{Actual(+)} = \frac{TP}{TP + FN} = \frac{30}{30 + 10} = 0.75$$

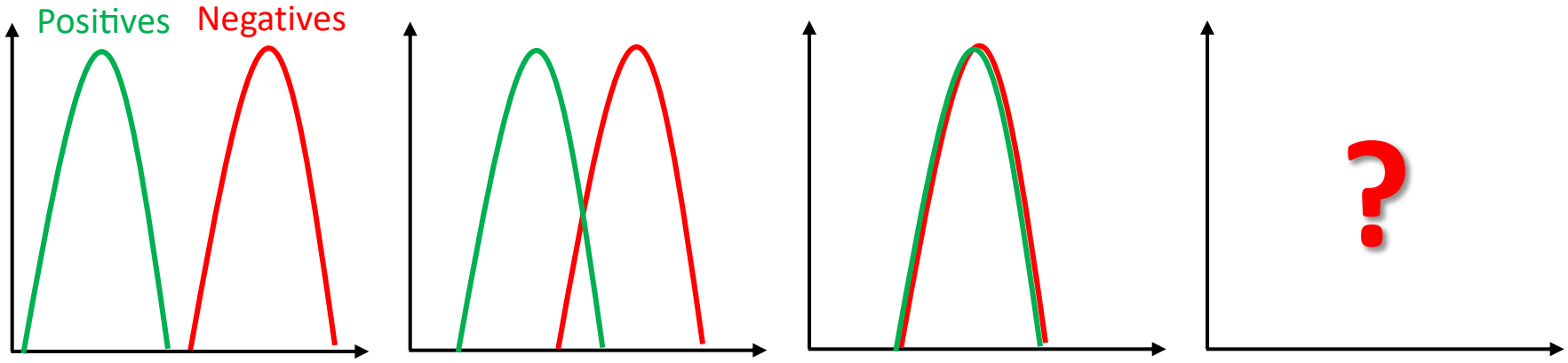
There are 40 good investments and we capitalized on 30 (75%) of them (Recall). But we missed 10, that's our opportunity cost.

(COST) $$FPR = \frac{FP}{Actual(-)} = \frac{FP}{FP + TN} = \frac{20}{20 + 40} = 0.33$$

There are 60 bad investments and sadly we're stuck with 20 (33%) of them This is our real loss, not a missed opportunity.

ROC and class separability





Perfect separability!

AUC (Area Under Curve)

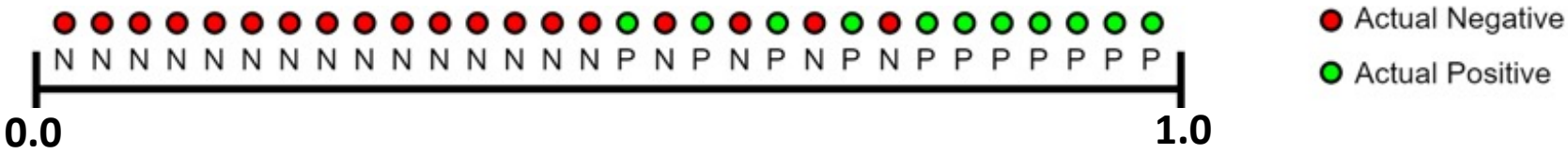
- **What is it?**
  - It's **a measure of the ability of a classifier to distinguish between classes**. It's a performance metric and is used as a summary of the ROC curve irrespective of what classification threshold is chosen (classification-threshold invariant).
- **How does it differ from accuracy?**
  - Accuracy measures the percentage of correctly classified instances given a fixed threshold. ROC-AUC considers all possible thresholds and therefore provides a broader view of the classifier's performance (a best summary of the performance of a classifier).
  - On the other hand, we end up using a single threshold for our classification problem anyways. So it's not clear how useful this property of ROC is.

AUC (Area Under Curve)

- **How to use it?**
  - The **model with the highest AUC** is the model that dominates all others in terms of **delivering the most benefit for the amount of cost incurred**.
- **When to use?**
  - As the ROC curves can present an overly optimistic view of model performance for imbalanced data, it's recommended to use ROC when the (+) and (-) classes are roughly equal in size.
- **AUC might be desirable for 2 properties:**
  - **Scale-invariant**: It measures how well predictions are ranked, rather than their absolute values (see next). This may not be accurate for classifiers that need well calibrated probabilities.
  - **Threshold-invariant**: It measures the quality of model predictions independent of the threshold chosen. AUC won't be useful, however, when the cost of FP and FN are different. If you need to minimize FN's for example, AUC isn't a useful metric for this task.

AUC (Area Under Curve)

- **How to interpret AUC?**
  - The AUC of a classifier represents the probability that a randomly chosen positive example has a higher score (rank) than a randomly chosen negative example.
  - In the following, AUC = probability that a random positive (green) example is positioned to the right of a random negative (red) example.



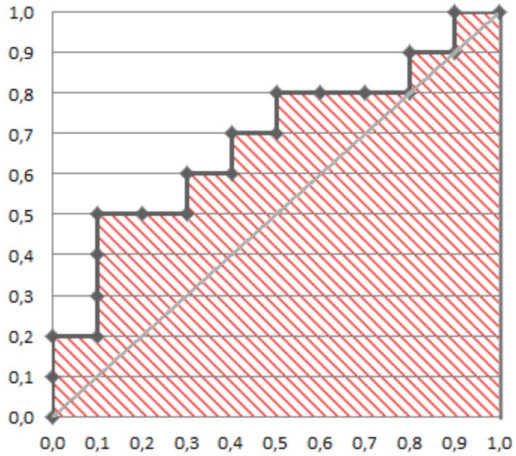
  - **Example**: Remember AUC was 84% for our heart attack problem. This means that, if you take 1 random TP individual and 1 random TN individual, 86% of the time a TP individual will have a higher level of cholesterol than a TN individual (see next slide).

AUC (Area Under Curve)

- The AUC of a classifier is equal to the probability that

$$P[\ score(x^+) > score(x^-)\ ]$$

- **Simulation**: Consider the following example with an AUC = 0.68
- Let's draw random (+) and (-) instances and then calculate the ratio of cases when (+)'s have greater score than (-)'s:




```
cls = ['P', 'P', 'N', 'P', 'P', 'P', 'N', 'N', 'P', 'N',
       'P', 'N', 'P', 'N', 'N', 'N', 'P', 'N', 'P', 'N']
scr = [0.9, 0.8, 0.7, 0.6, 0.55, 0.51, 0.49, 0.43, 0.42, 0.39,
       0.33, 0.31, 0.23, 0.22, 0.19, 0.15, 0.12, 0.11, 0.04, 0.01]
```

```
df= pd.DataFrame()
cls = ['P', 'P', 'N', 'P', 'P', 'P', 'N', 'N', 'P', 'N',
       'P', 'N', 'P', 'N', 'N', 'N', 'P', 'N', 'P', 'N']
scr = [0.9, 0.8, 0.7, 0.6, 0.55, 0.51, 0.49, 0.43, 0.42, 0.39,
       0.33, 0.31, 0.23, 0.22, 0.19, 0.15, 0.12, 0.11, 0.04, 0.01]
df['cls'] = cls
df['scr'] = scr
```

```
counter = 0
for i in range(5000):
    posindx = random.choice(df[df.cls=='P'].index)
    negindx = random.choice(df[df.cls=='N'].index)
    if df.loc[posindx,:]['scr'] > df.loc[negindx,:]['scr']: counter += 1
print(counter/5000)
```

```
0.6816
```

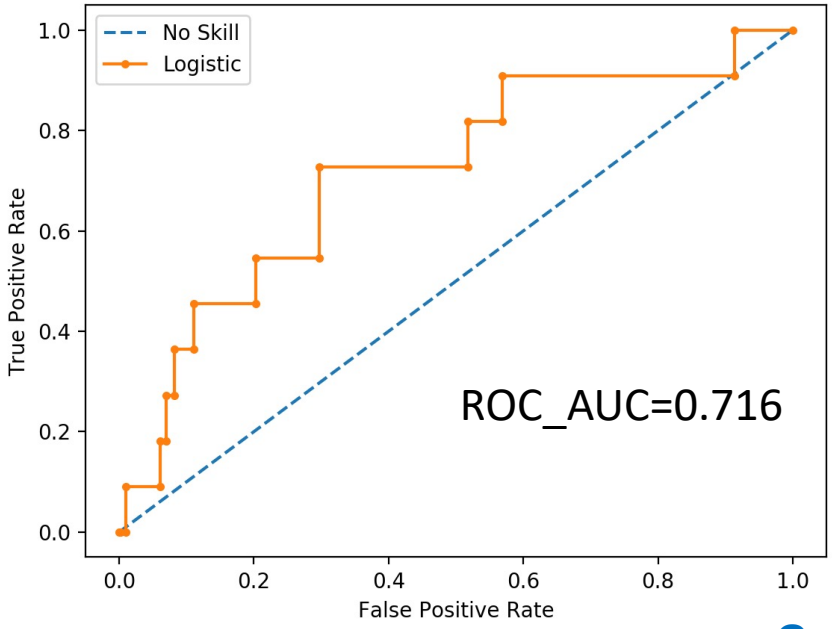
Very close to 0.68!

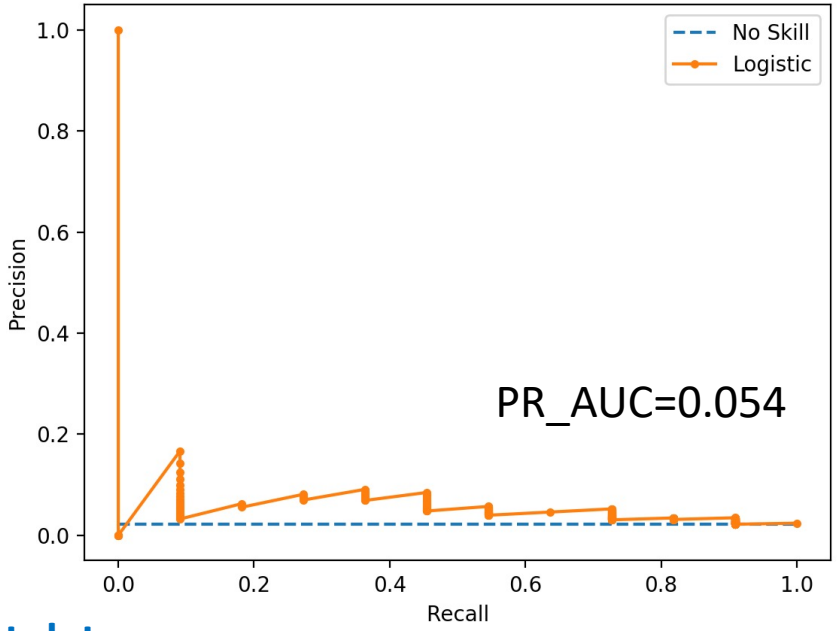
ROC-plot vs PR-plot

- **Example:** For an artificial classification problem with a highly imbalanced data set, we'll compare ROC vs PR plots.

```
X, y = make_classification(n_samples=1000, n_classes=2,
                           weights=[0.99,0.01], random_state=1)
```

```
trainX, testX, trainy, testy = train_test_split(X, y,
                                   test_size=0.5, random_state=2)
```





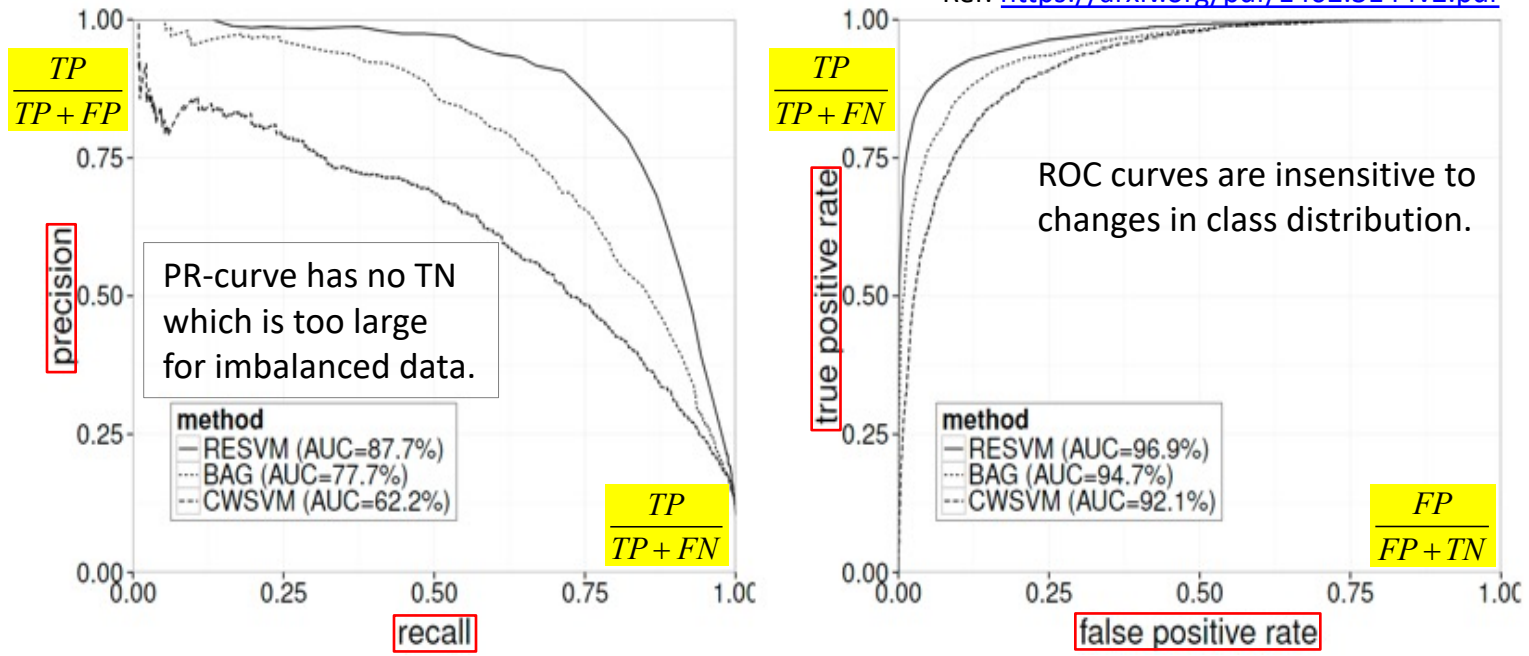
On test data

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

ROC vs PR curves

- Area under the ROC curve (ROC-AUC) summarizes performance based on all thresholds, and is therefore quite informative. But they are insensitive to class imbalance.

Ref: https://arxiv.org/pdf/1402.3144v2.pdf



For ROC curves, the differences between classifiers tend to be small (but present!) for highly imbalanced data. PR curves are better to highlight differences between classifiers for highly imbalanced data sets. As shown above, a difference of 4.5% in ROC space corresponds to 25% in PR space.

ROC vs PR curves – cont'd

- ROC curves can present an overly optimistic view of an algorithm's performance if there is a large skew in the class distribution.
  - *Ref:dl.acm.org/citation.cfm?id=1143874*
- The main reason for this optimistic picture is because of the use of TN in the False Positive Rate in the ROC Plot and the careful avoidance of this rate in the PR-plot.
- **ROC-AUC** is more appropriate if you care equally about the positive and negative classes or the dataset is quite balanced.
- **PR-AUC** is a better metric if you care more about the positive class.
- **When evaluating binary classifiers on imbalanced data, the PR-plot is more informative than the ROC-plot**.
  - *Ref:www.ncbi.nlm.nih.gov/pmc/articles/PMC4349800*

ROC vs PR curves – cont'd

- We have seen that the accuracy is not a good metric for problems with imbalanced (highly skewed) data sets.
- To handle a problem with an imbalanced data set, you have to ask ourselves:
  1. **Which question do you want to address?**
  2. **What is a good metric for this question?**
- These should be answered before you decide which model to use.
- **Key takeaway**: Selecting the right metric for the business problem is a critical element for success in Machine Learning.

Probability vs Class

- Probabilities are much more informative than class labels:
- Suppose we have the following information:
  - **The model predicted you don't have cancer** versus
  - **The model predicted you're 45% likely to have cancer**
- Would you take comfort in just hearing that you don't have cancer while the probability of having he condition is 45%?
- Please read Frank Harrell's blog article on "**Why classification accuracy is a terrible metric?**":
  - Damage Caused by Classification Accuracy and Other Discontinuous Improper Accuracy Scoring Rules, by F. Harrell: https://www.fharrell.com/post/class-damage/
- And this stackexchange article
  - https://stats.stackexchange.com/questions/367855/predictive-distribution-what-can-we-say-about-the-prediction/367880#367880

Log-Loss (Binary Crossentropy)

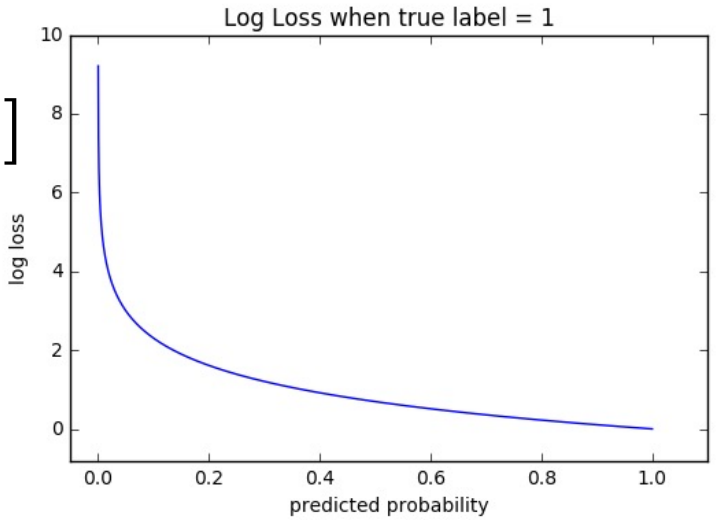
- **Log-Loss** is a good evaluation metric for binary classifiers. It's often used as an optimization function for classifiers like Logistic Regression and Neural Networks that produce a probability associated with a classification.
- Logistic Regression, for example, classifies an instance with a probability value. Depending on the probabilistic value, we classify the instance as a TRUE or FALSE class.
- As we end up converting probability to a binary class decision, using a Log-Loss function as the accuracy metric is a good idea for problems of such nature.
- Log-Loss is a "soft" accuracy measurement that incorporates the concept of probabilistic confidence. It quantifies the classification accuracy by penalizing false positives. Minimizing the Log-Loss is maximizing the accuracy, but there is a subtle detail:

Log-Loss – cont'd

- For calculating the Log-Loss, classifier assigns a probability to each class rather than giving the most likely class:

$$LL = -\frac{1}{N}\sum_{i=1}^{N}\left[(y_i \log p_i + (1-y_i)\log(1-p_i)\right]$$

p_i: Probability of predicting "1"
N: Number of instances



- Note that for each instance only the term for the correct class actually contributes to the sum.
- When the output of a classifier is prediction probabilities. Log Loss takes into account the uncertainty of your prediction based on how much it varies from the actual label.

Brier Score

- The Brier score (sum of squared errors of the class-wise probability estimates) is gentler than log loss but still penalizes proportional to the distance from the expected value.

$$\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{p}_i - y_i\right)^2$$

- where N is the number of samples
  $\hat{p}$ is the predicted probability for data instance "i"
  y_i is the true label (class) for data instance "i"

```
from sklearn.metrics import brier_score_loss
# predict probabilities
probs = model.predict_proba(X_test)
# keep the predictions for class 1 only
probs = probs[:, 1]
# calculate Brier score
loss = brier_score_loss(y_test, probs)
```

References

- References on cross validation
  - Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning: https://sebastianraschka.com/pdf/manuscripts/model-eval.pdf
  - A "short" introduction to model selection: https://towardsdatascience.com/a-short-introduction-to-model-selection-bb1bb9c73376
  - Sample code on nested CV
  - https://github.com/esentri/datascience_blog_resources/blob/master/model_selection.ipynb
  - https://github.com/rasbt/model-eval-article-supplementary/blob/master/code/nested_cv_code.ipynb
- References on performance metrics
  - https://medium.com/x8-the-ai-community/understanding-ml-evaluation-metrics-precision-recall-2b3fb915b666
  - https://medium.com/bcggamma/beyond-the-roc-auc-toward-defining-better-performance-metrics-b11f5d35adda