Lecture Notes for
Machine Learning in Python

Professor Eric Larson

Evaluation and Cross Validation, Video Lecture

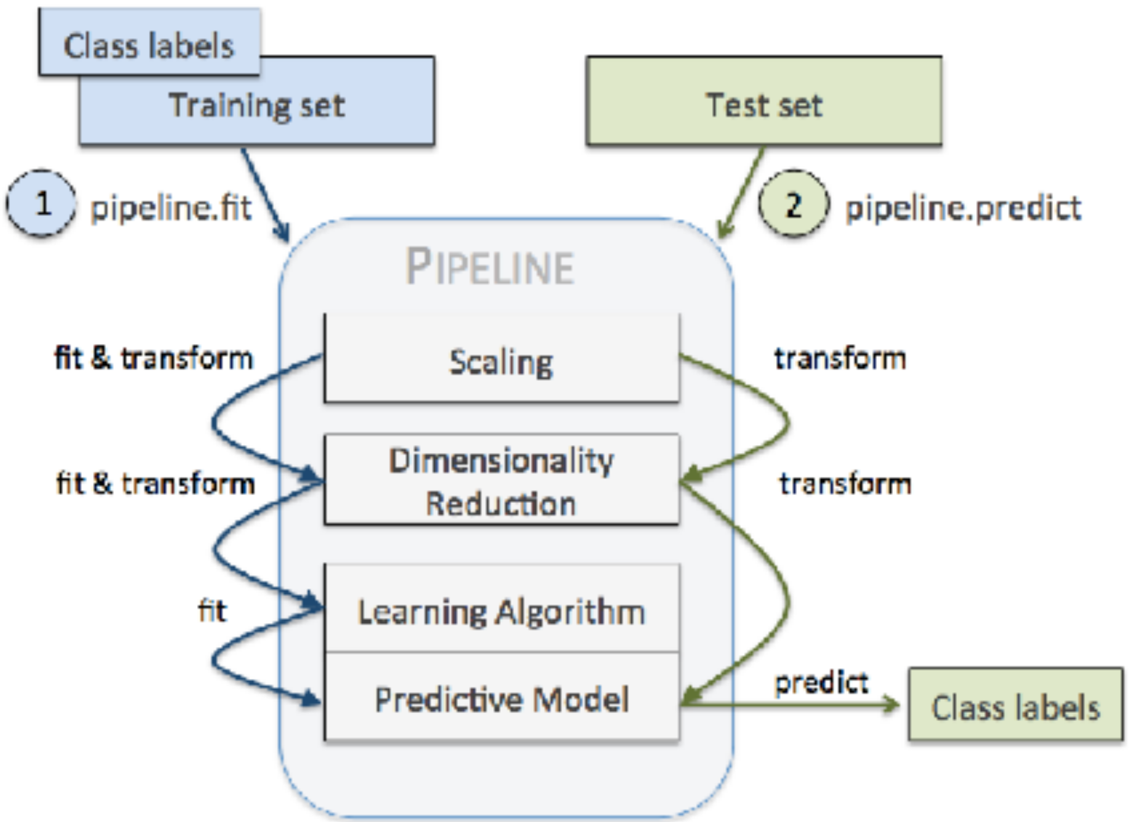
Model Evaluation Best Practices

Model Evaluation

- How reliable are our estimates of performance?
- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

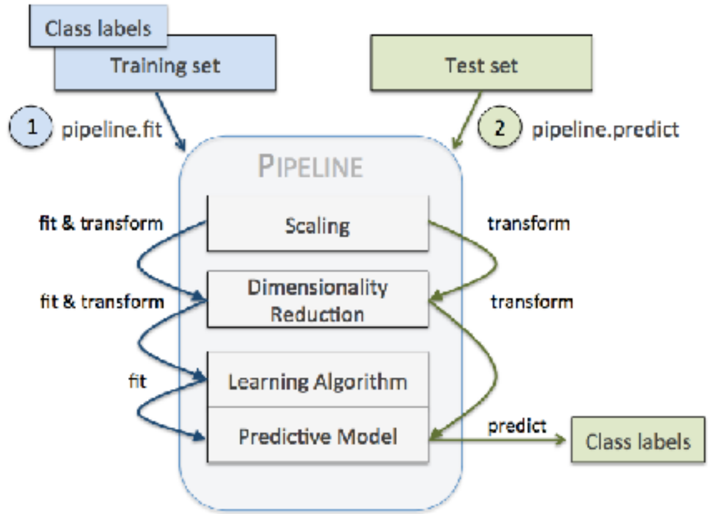
Best Practice: Setup Pipelines

- Combine pre- and post-processing into stages
- Excellent way to prevent “data snooping”
  - guarantees separation of testing and training sets



https://github.com/rasbt/python-machine-learning-book/

Best Practice: Setup Pipelines




```
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', LogisticRegression(random_state=1))])

pipe_lr.fit(X_train, y_train)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
y_pred = pipe_lr.predict(X_test)
```

```
Test Accuracy: 0.947
```


https://github.com/rasbt/python-machine-learning-book/

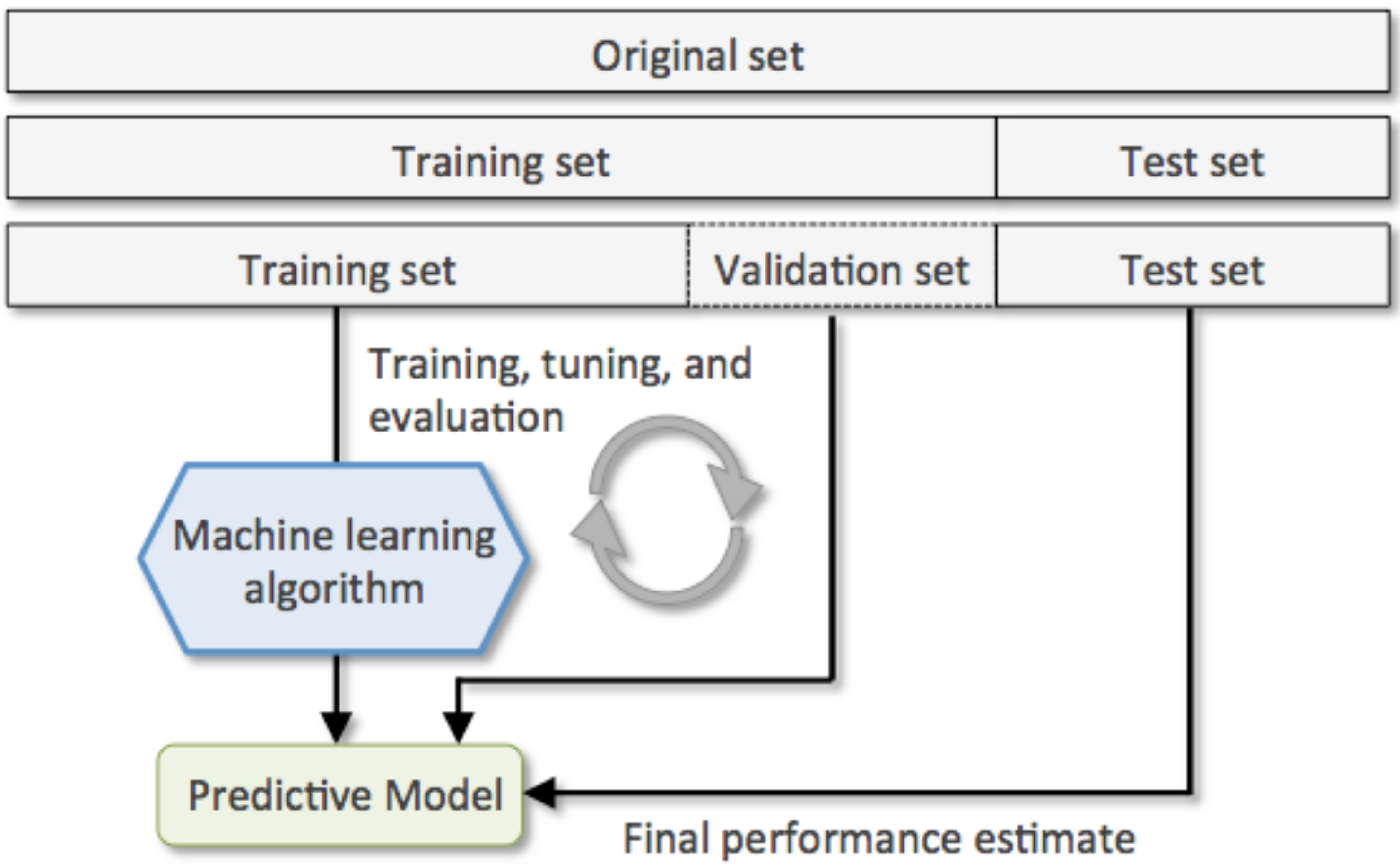
Best Practice: Use Pipelines in Validation Loops

use testing set, and *never*, *never*, *never* let the model see it, Many different strategies:

- Holdout
  - Reserve x% for training and (1-x)% for testing
- Random subsampling
  - Repeated holdout, with replacement
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=M
- Stratified Cross Validation
  - Select samples, keeping overall class distribution same for each fold

Validation Loop Strategies

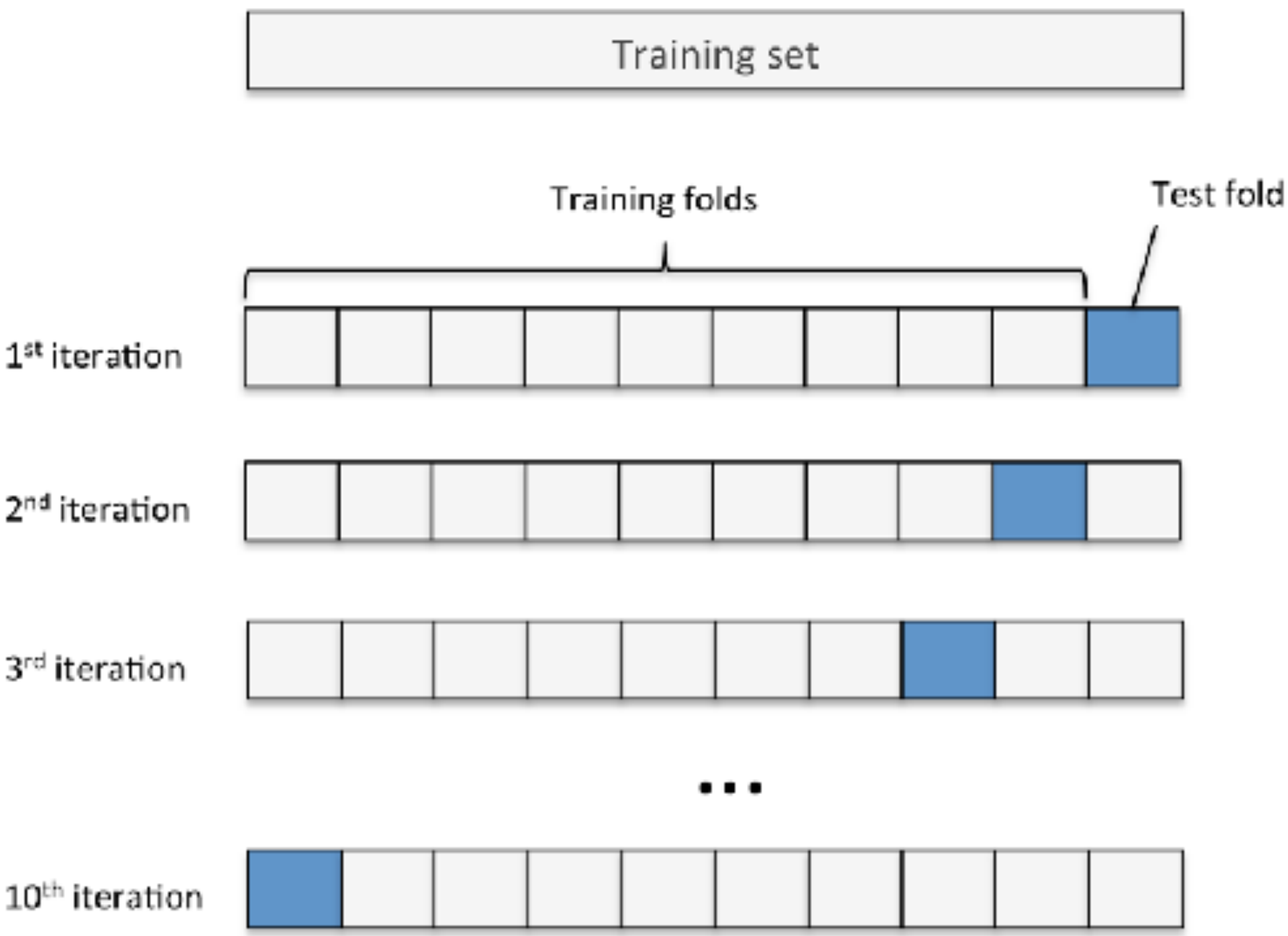
Holdout



https://github.com/rasbt/python-machine-learning-book/

Validation Loop Strategies

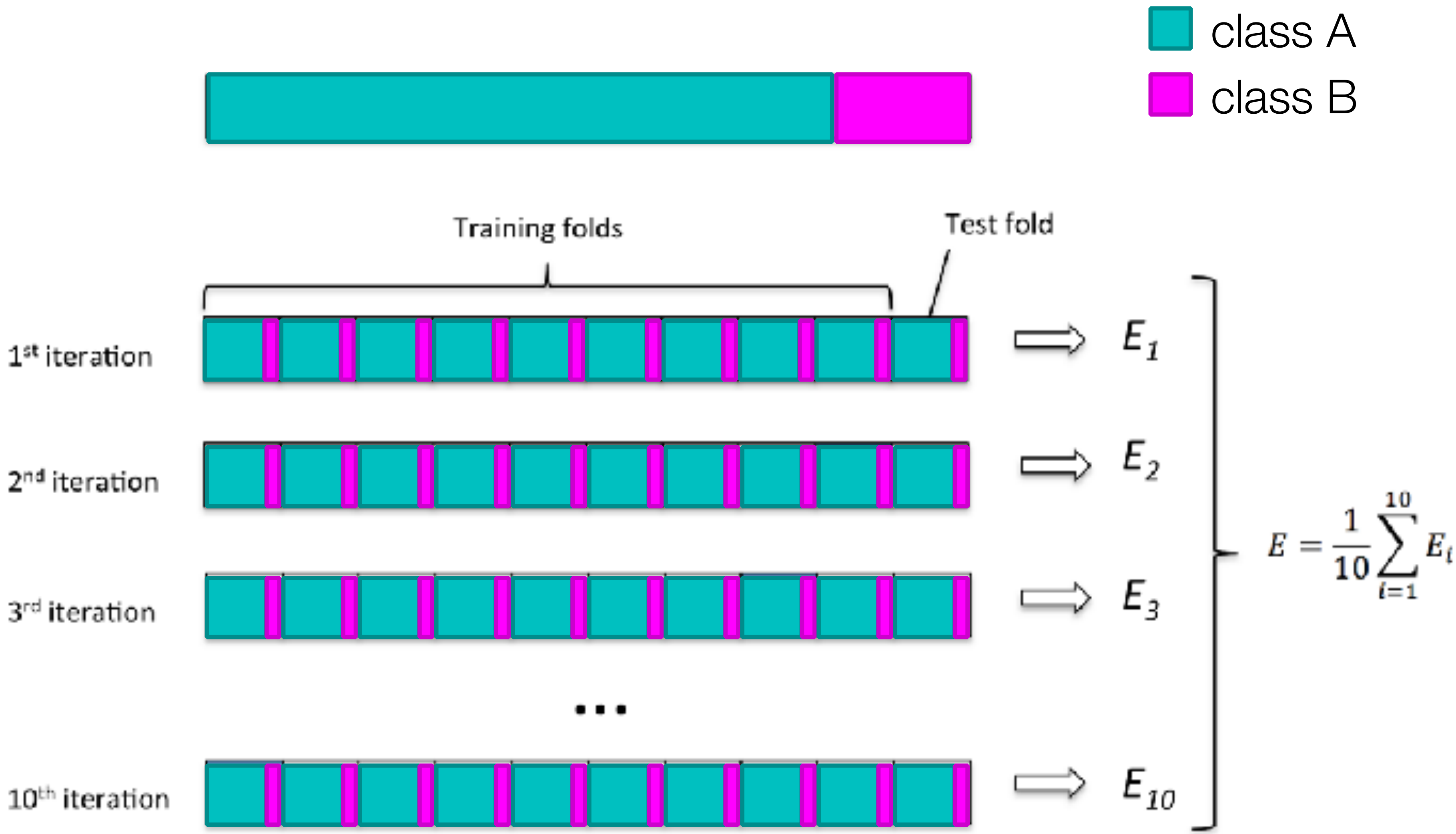
k-Fold Cross Validation



https://github.com/rasbt/python-machine-learning-book/

Validation Loop Strategies

Stratified *k*-Fold Cross Validation



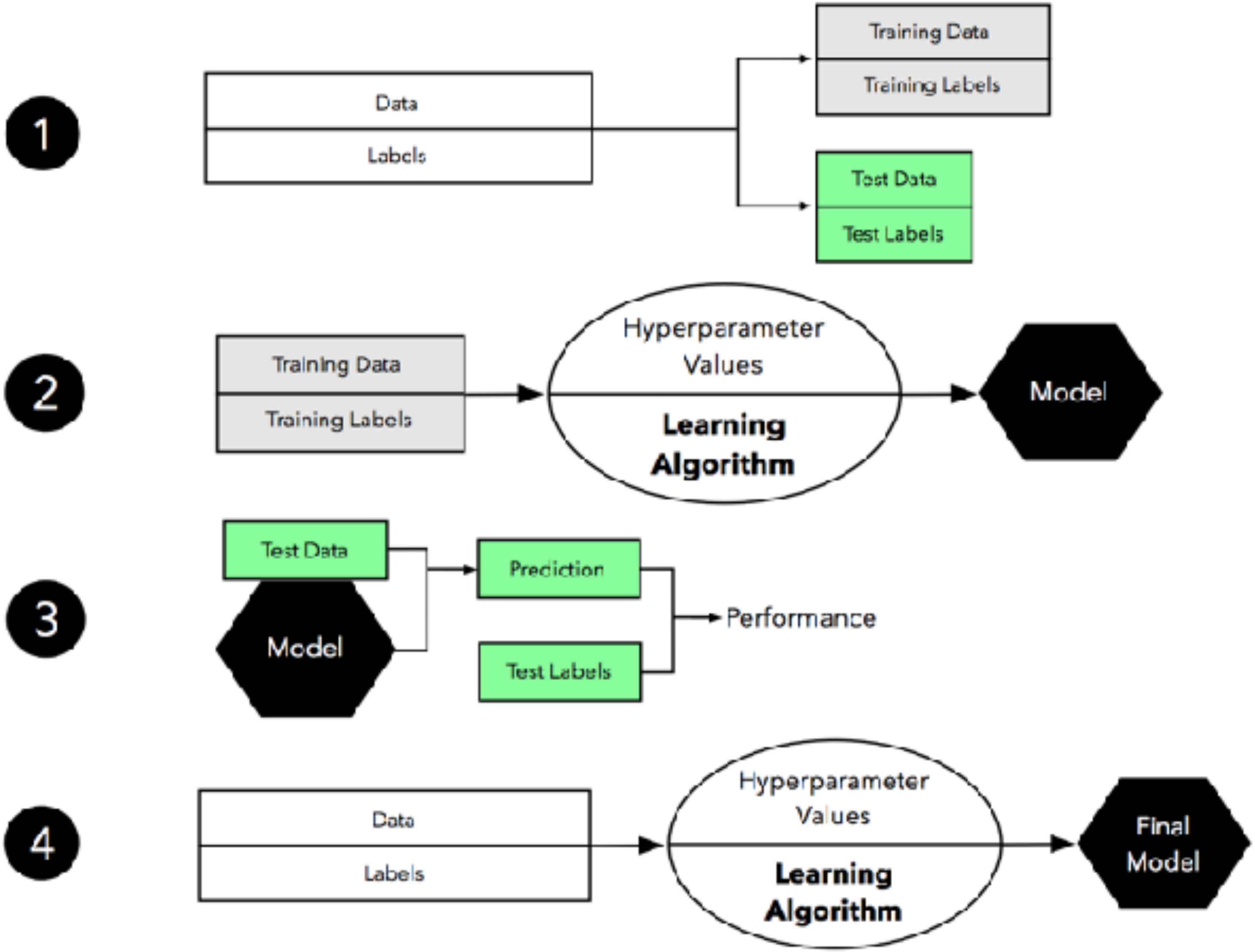
https://github.com/rasbt/python-machine-learning-book/

Cross Validation: Reality Check

- What is the point of cross validation?

  - **Primary:** To realistically estimate how your classifier will perform on data it has not seen
    - situation must be ***plausible*** scenario
  - **Secondary**: hyperparameter tuning
    - what parameters should I set when training the model?
      - attain average performance from CV
      - use parameters to obtain final model

Cross Validation: Reality Check

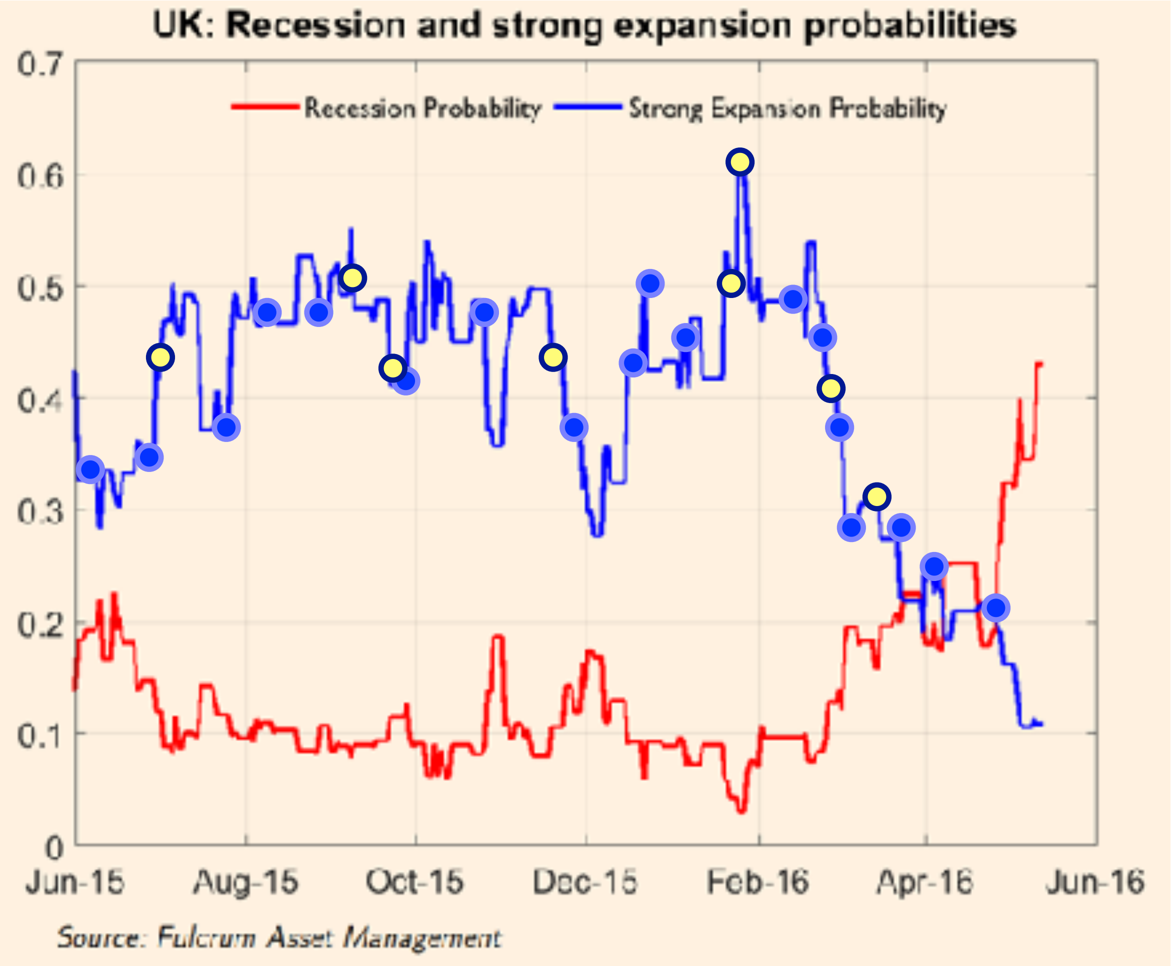


https://github.com/rasbt/python-machine-learning-book/

Cross Validation: Reality Check

- Folds must be **plausible**, **representative** of the **actual use case** for the classifier

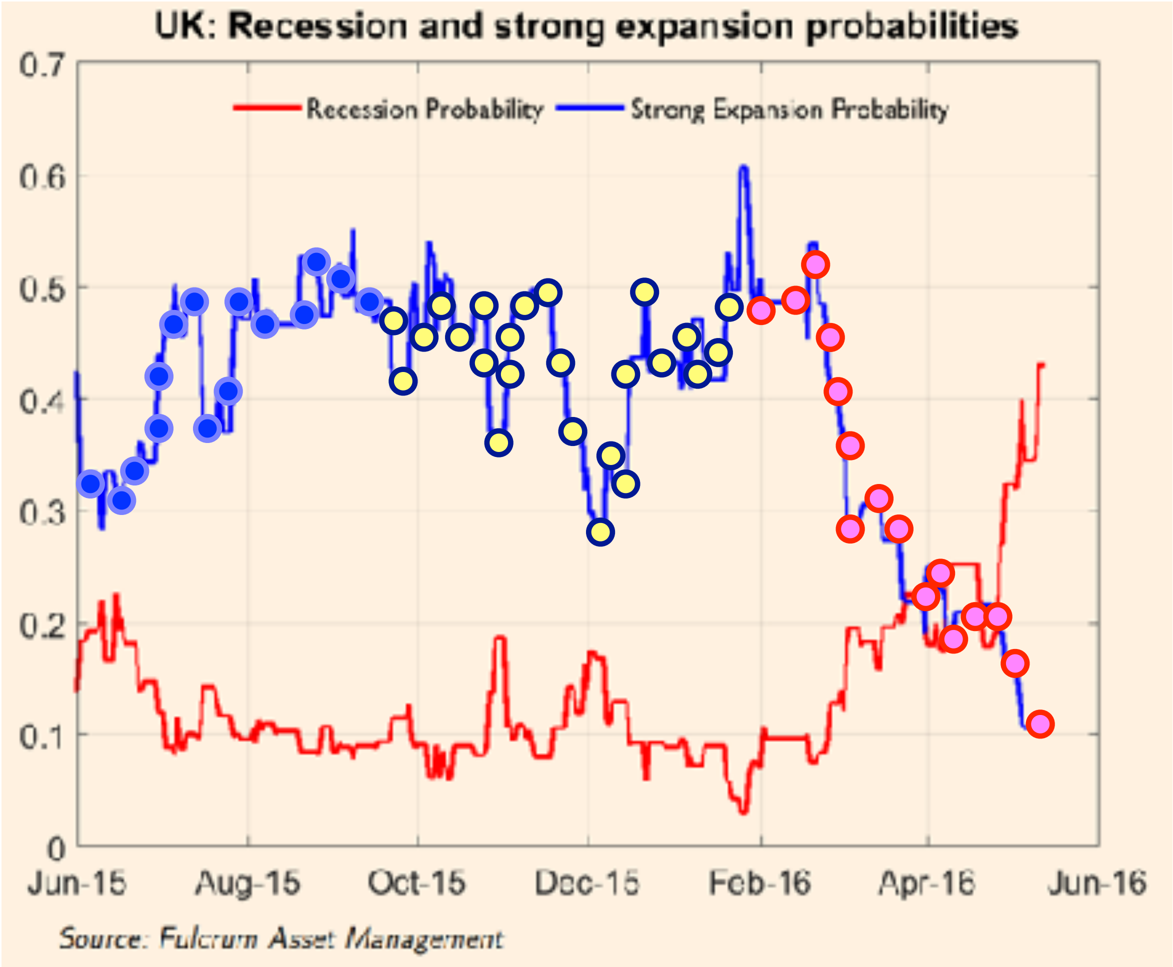
- Time series: **Cannot** apply stratified cross validation



Cross Validation: Reality Check

- Folds must be **plausible**, **representative** of the **actual use case** for the classifier

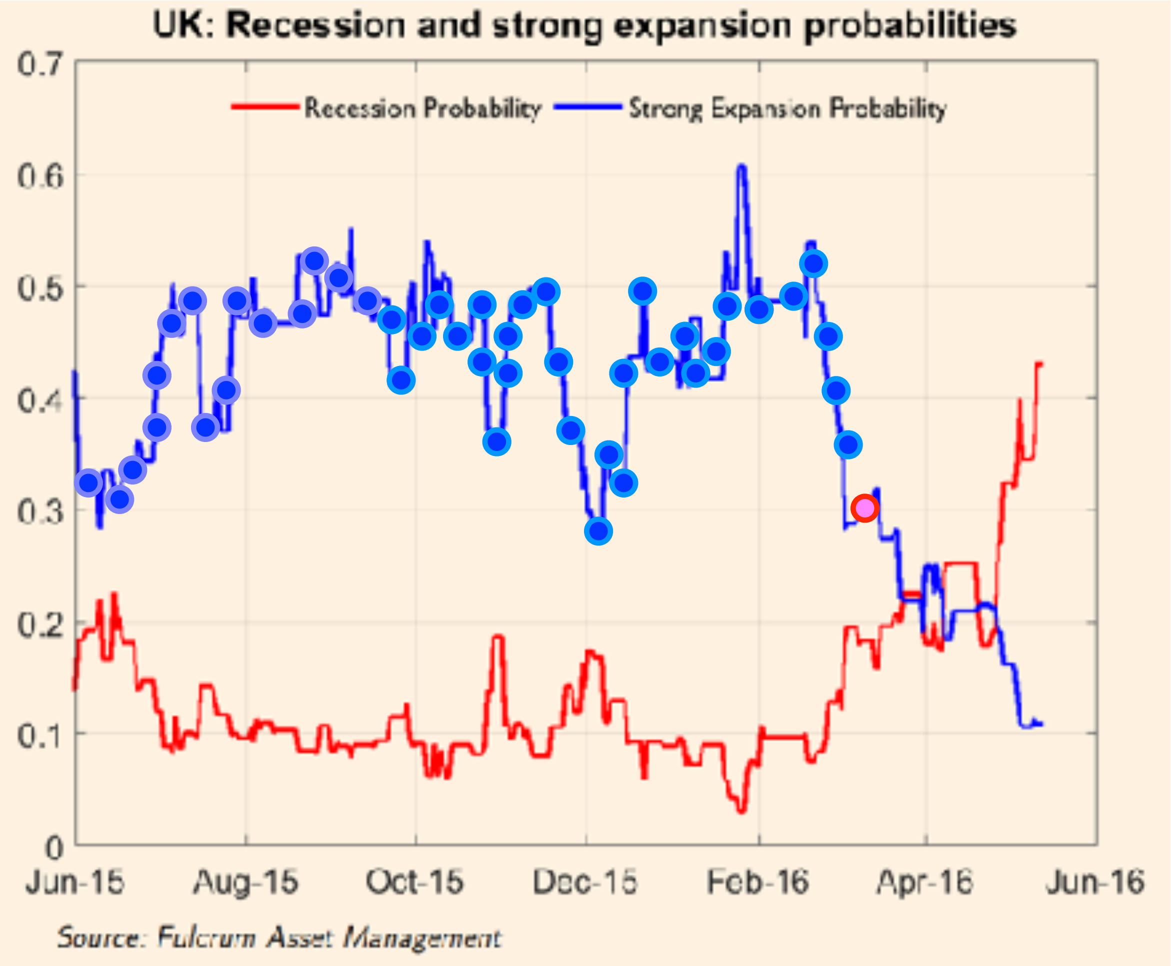
- Time series: **Cannot** apply stratified cross validation
- **Cannot** apply folding, actually



Cross Validation: Reality Check

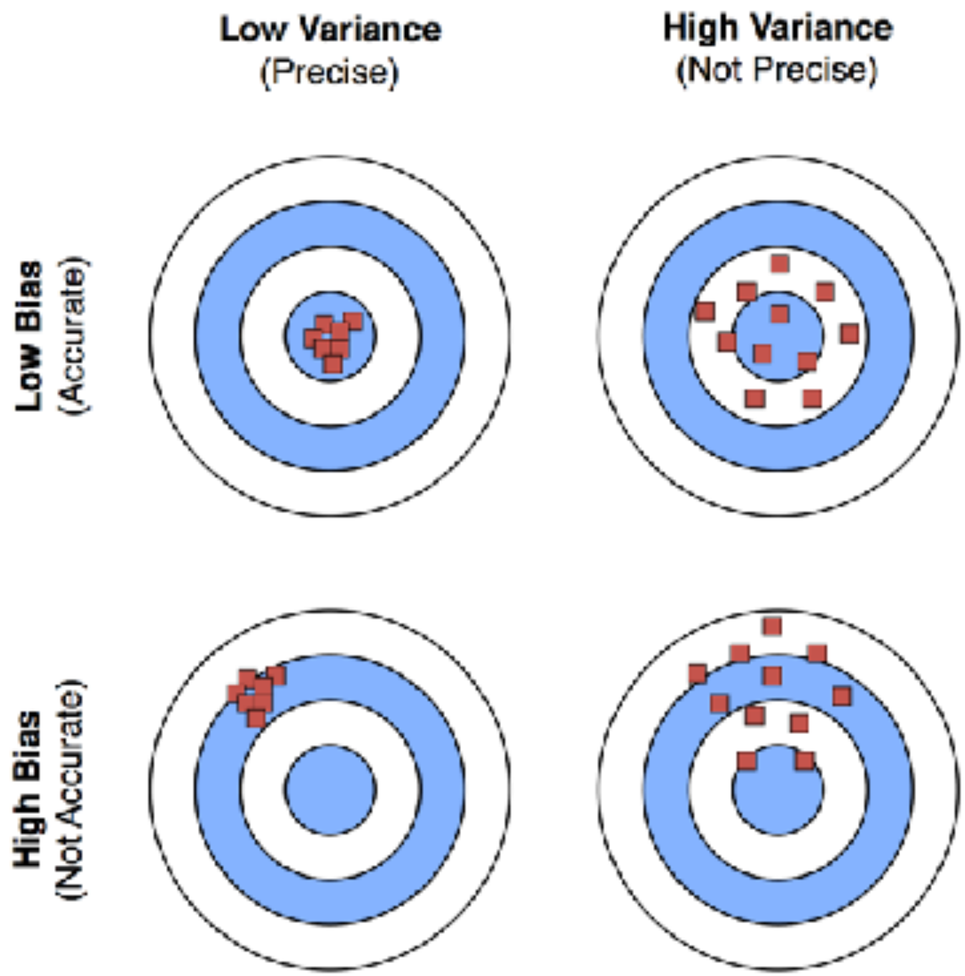
- Folds must be **plausible**, **representative** of the **actual use case** for the classifier

- Time series: **Cannot** apply stratified cross validation
- **Cannot** apply folding, actually
- Even better: **Mirror** real life use case



Beyond cross validation

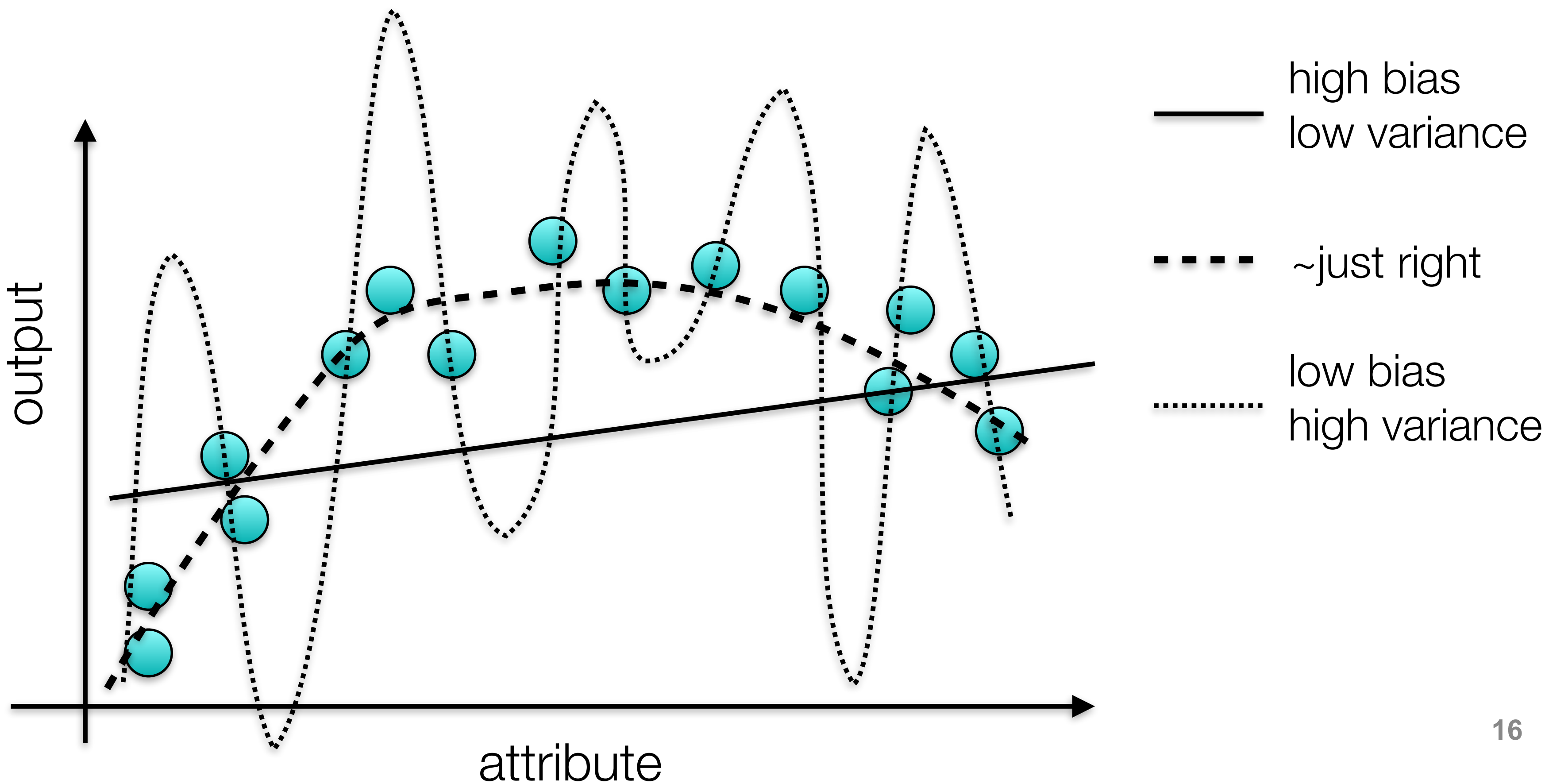
- So … we separated out the data
- How do we know a model is actually good?



https://github.com/rasbt/python-machine-learning-book/

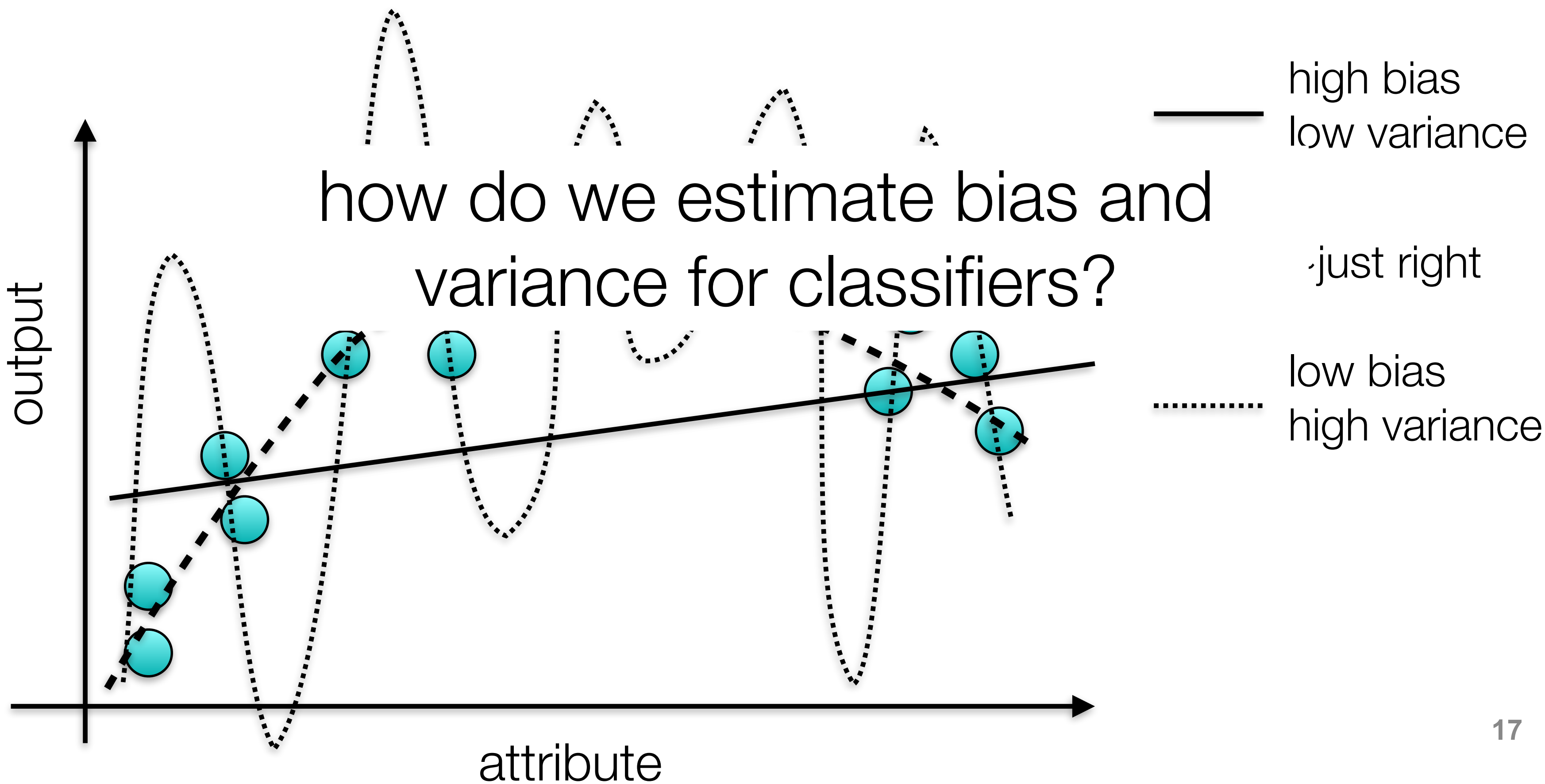
Bias Variance Tradeoff

- **Complex** models can really fit the training data, giving **lower bias**
- **Simpler** models have trouble fitting data, resulting in **higher bias**
- But complex models can have **high variance** in their decision!



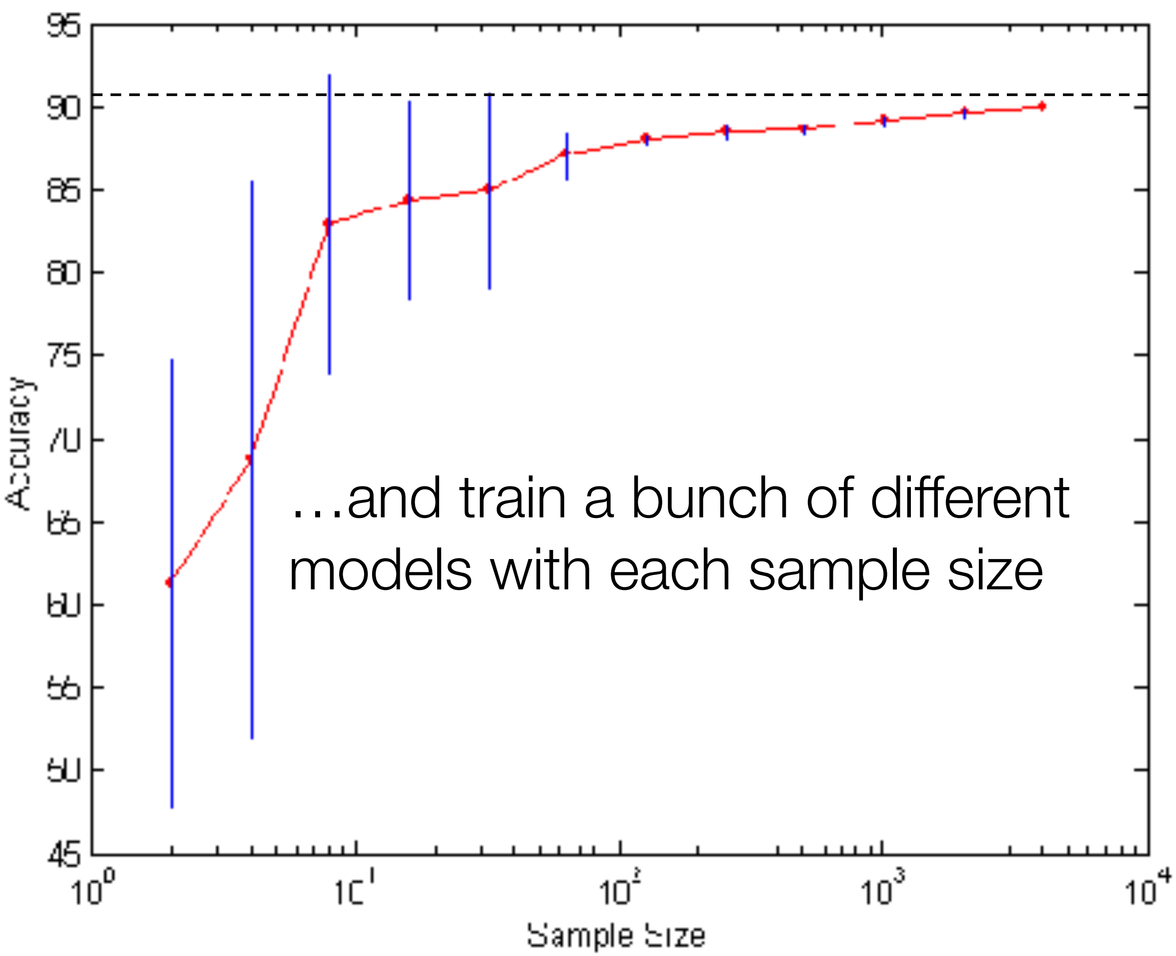
Bias Variance Tradeoff

- **Complex** models can really fit the training data, giving **lower bias**
- **Simpler** models have trouble fitting data, resulting in **higher bias**
- But complex models can have **high variance** in their decision!



how do we estimate bias and variance for classifiers?

The Learning Curve: Number of Samples



randomly get this number of samples …

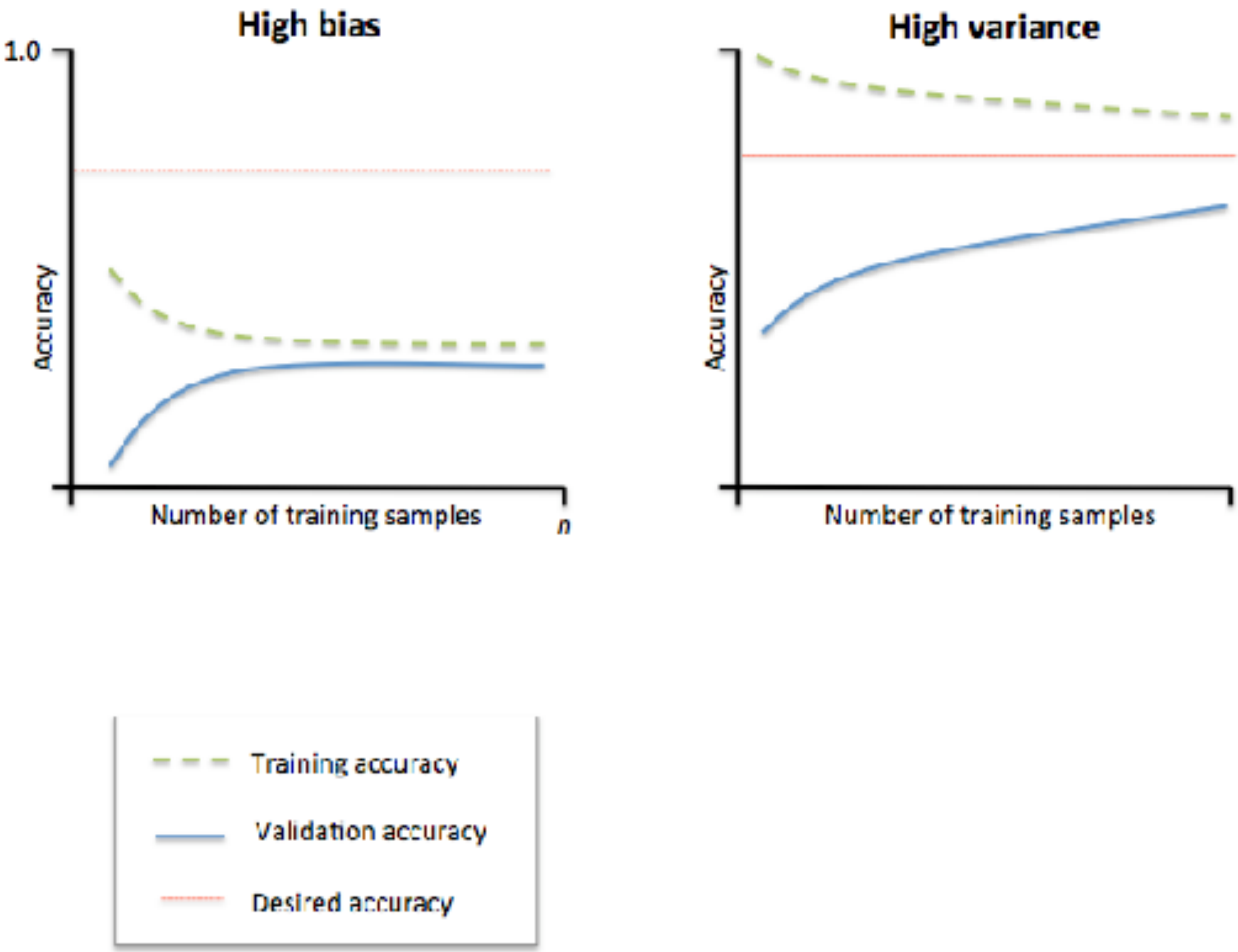
Learning curve shows how accuracy changes with varying sample size

Effect of small sample size:

- Bias in the estimate
- Variance of estimate

You cannot estimate this curve without **collecting the data**. Some **bounds exist**, but they are **too loose** to be **useful**!!!!

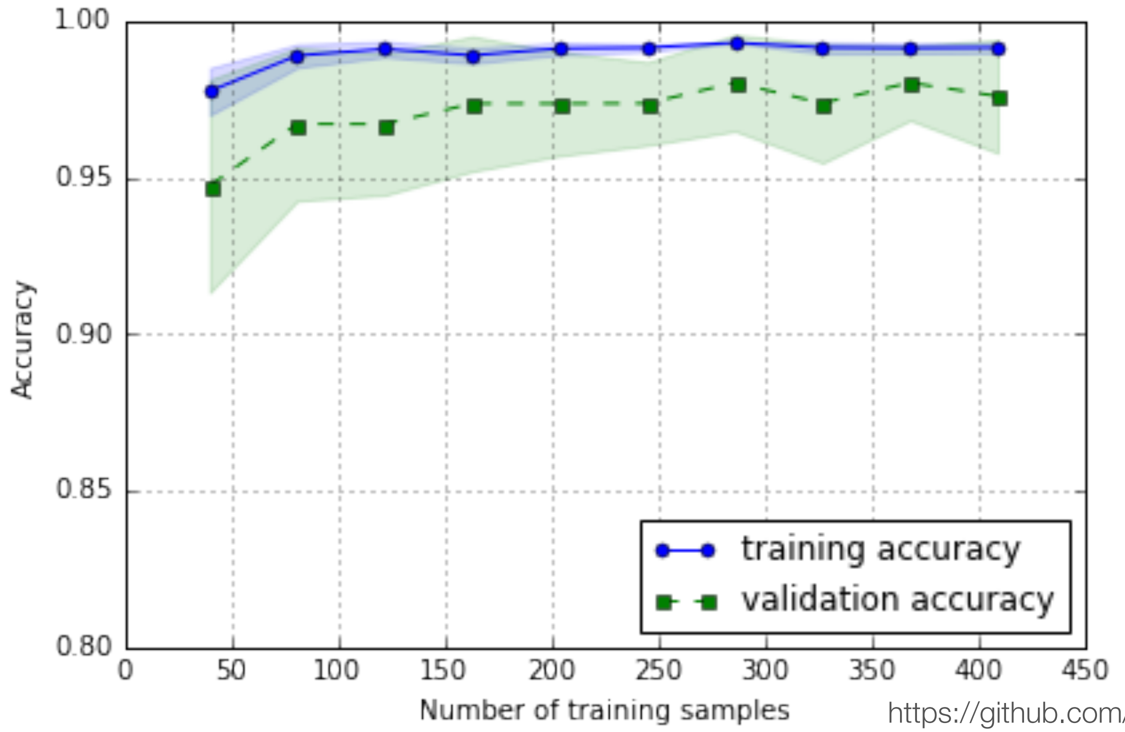
The Learning Curve: Number of Samples



https://github.com/rasbt/python-machine-learning-book/

The Learning Curve: Number of Samples

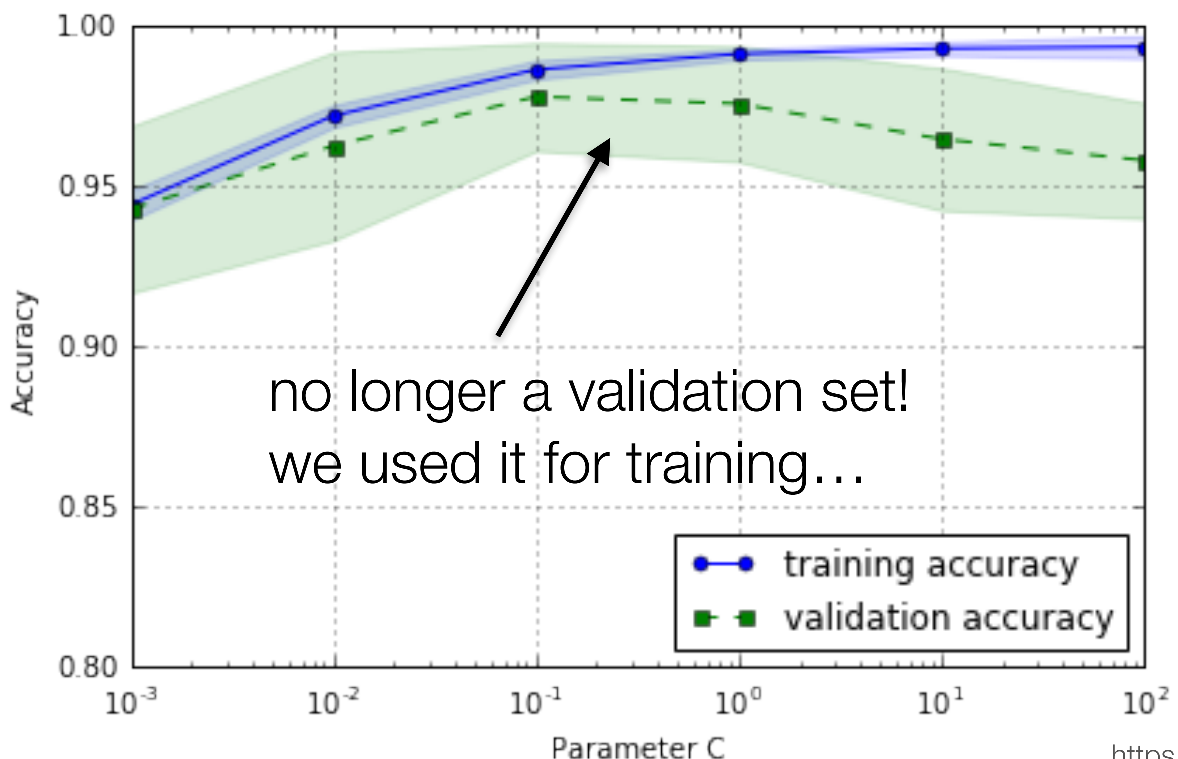
```
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LogisticRegression(penalty='l2', random_state=0))])
```



https://github.com/rasbt/python-machine-learning-book/

The Validation Curve: Hyper-parameters

```
param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
train_scores, test_scores = validation_curve(
                estimator=pipe_lr,
                X=X_train,
                y=y_train,
                param_name='clf__C',
                param_range=param_range,
                cv=10)
```



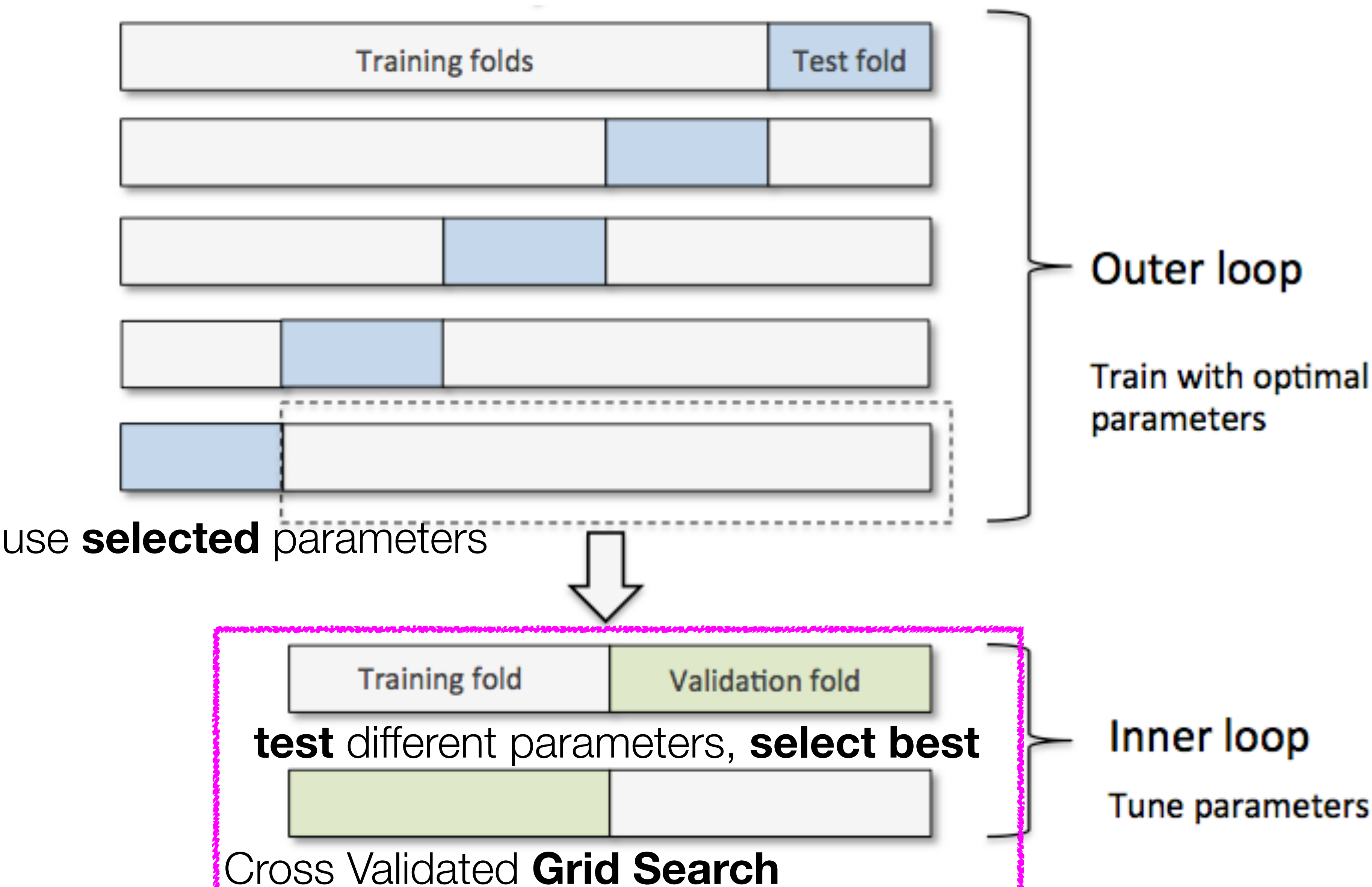
Similar to learning curve, but sweeping a hyper-parameter of the learning algorithm

Look for sweet spot in the parameter for the validation accuracy

But, this can introduce data snooping…

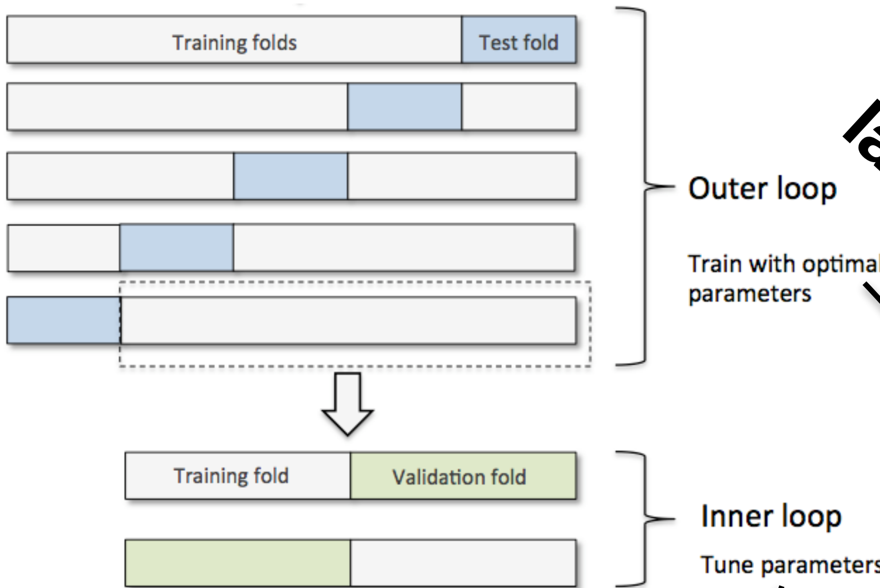
https://github.com/rasbt/python-machine-learning-book/

Nested Cross Validation: Hyper-parameters



https://github.com/rasbt/python-machine-learning-book/

Nested Cross Validation: Hyper-parameters



We will **revisit** this **later** in semester

```
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=2)

# Note: Optionally, you could use cv=2
# in the GridSearchCV above to produce
# the 5 x 2 nested CV that is shown in the figure.

scores = cross_val_score(gs, X_train, y_train, scoring='accuracy', cv=5)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

https://github.com/rasbt/python-machine-learning-book/

Model Evaluation Metrics/Measures

Metrics for Performance Evaluation

- Focus on the **predictive capability** of a model
- **Not** how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

	PREDICTED CLASS		
ACTUAL CLASS		Class=No	Class=Yes
	Class=No	d	c
	Class=Yes	b	a

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Metrics for Performance Evaluation…

	PREDICTED CLASS	
ACTUAL CLASS	Class=No	Class=Yes
Class=No	d (TN)	c (FP)
Class=Yes	b (FN)	a (TP)

- Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Limitations of Accuracy

- Ignores the **cost** of misclassifications

- Consider an **imbalanced** 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10

- If model **predicts everything to be class 0**, accuracy is 9990/10000 = 99.9 %
  - Accuracy is **misleading** because model does not detect any class 1 example

Other evaluation metrics: Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	$C(i\,|\,j)$	**Class=No**	**Class=Yes**
	Class=No	C(No|No)	C(Yes|No)
	Class=Yes	C(No|Yes)	C(Yes|Yes)

Define a cost function based on your expertise with problem:

$C(i \mid j)$: Cost of misclassifying class j example as class i

Cost Matrix Examples

Lower cost means "better"

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	-	+
	-	0	1
	+	100	-1

i.e., medical diagnosis costs?

Cost Matrix	PREDICTED CLASS				
ACTUAL CLASS	C(i|j)	**USA**	**CAN**	**AUS**	**NZ**
	USA	0	1	10	10
	CAN	1	0	10	10
	AUS	10	10	0	20
	NZ	10	10	20	0

i.e., predicting travel locations?

Cost-Sensitive Measures

	PREDICTED CLASS		
CLASS	Class=Yes	b (FN)	a (TP)

```python
from sklearn.metrics import precision_score, recall_score, f1_score

print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

$$\text{Precision (p)} = \frac{a}{a+c}$$

Higher Precision ==
Lower false positives

$$\text{Recall (r)} = \frac{a}{a+b}$$

Higher Recall ==
Lower false negatives

$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

Higher F1 ==
Lower FN & FP

Cost-Sensitive Measures: Multi-Class

```
pre_scorer = make_scorer(score_func=precision_score,
                         pos_label=1,
                         greater_is_better=True,
                         average='micro')
```

$$\text{Precision}_{micro} = \frac{TP_1+\ldots+TP_k}{TP_1+\ldots+TP_k + FP_1+\ldots+FP_k}$$

$$\text{Recall}_{micro} = \frac{TP_1+\ldots+TP_k}{TP_1+\ldots+TP_k + FN_1+\ldots+Fn_k}$$

weight all instances equally

$$F1_{micro} = \frac{2(TP_1+\ldots+TP_k)}{2(TP_1+\ldots+TP_k)+ FP_1+\ldots+FP_k + FN_1+\ldots+Fn_k}$$

$$X_{macro} = \frac{X_1+\ldots+X_k}{k}$$

weight all classes equally

The Receiver Operating Characteristic

ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize the trade-off between positive hits and false alarms

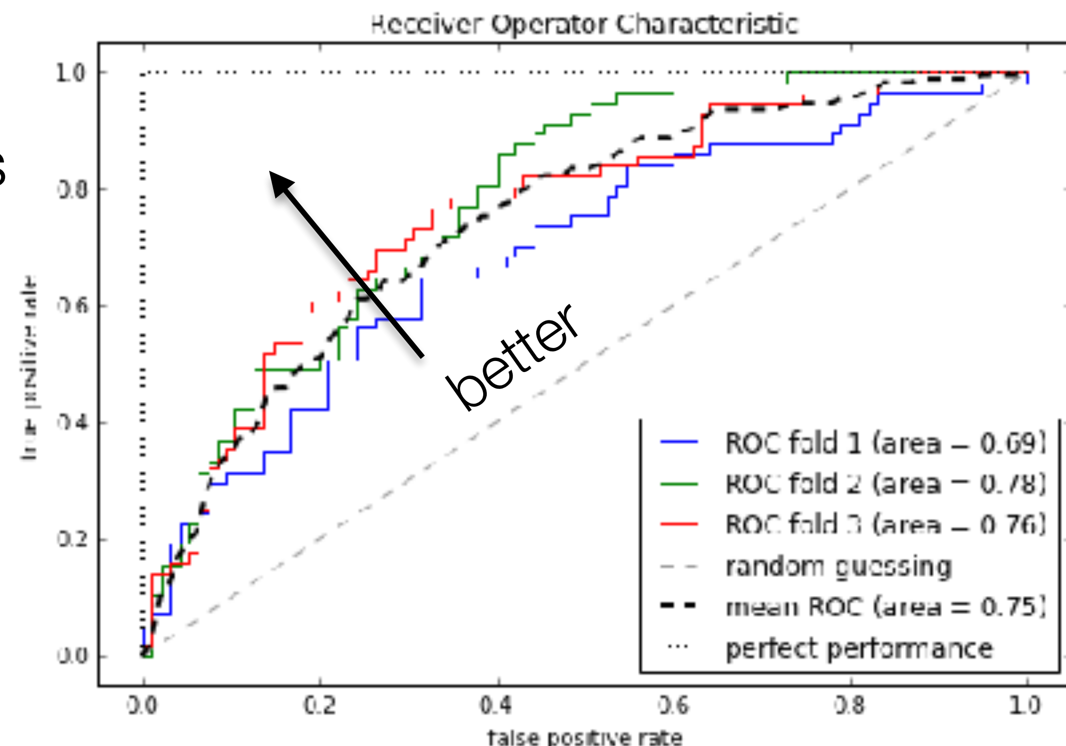
- ROC curve plots TP (on the y-axis) against FP (on the x-axis)

- Performance of each classifier represented as a point on the ROC curve
  - changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point

ROC Curve

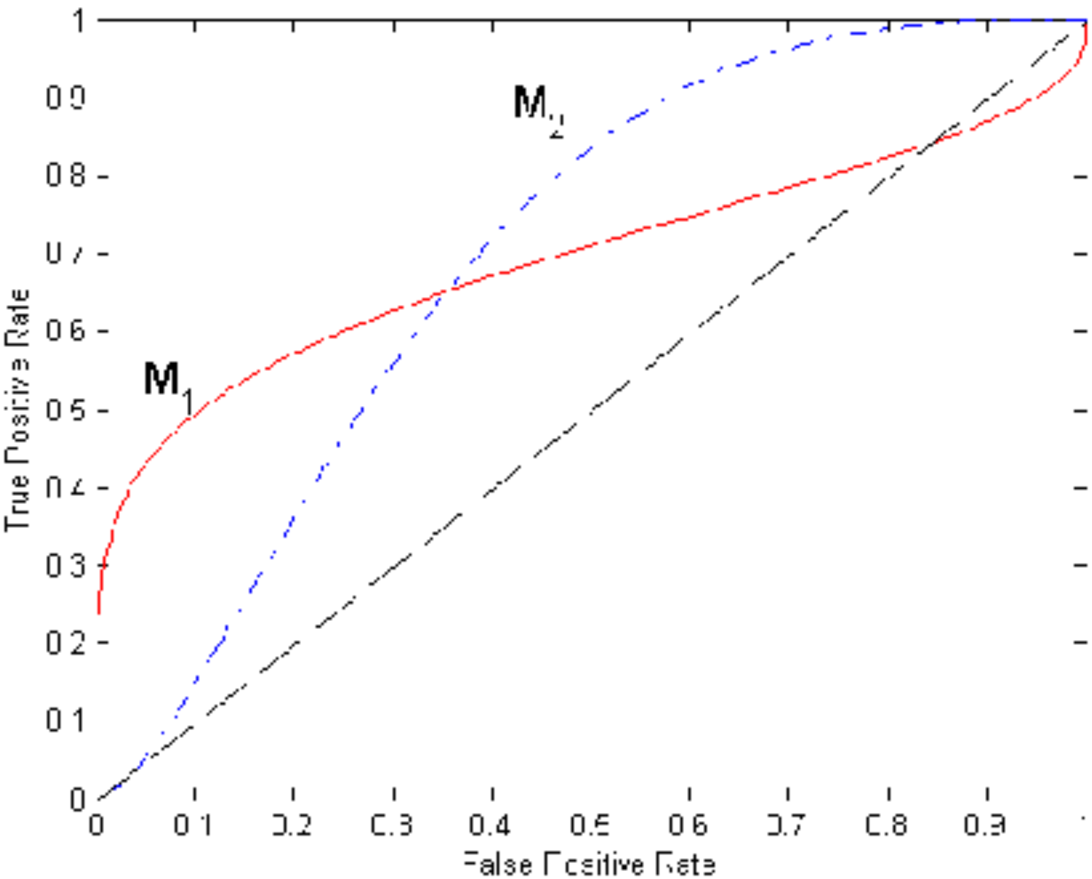
(TP,FP):

- (0,0): declare everything
  to be negative class
- (1,1): declare everything
  to be positive class
- (1,0): ideal



- Diagonal line:
  - Random guessing for equal number of classes
  - Below diagonal line:
    - prediction is opposite of the true class

Using ROC for Model Comparison



- No model consistently outperforms the other
  - M_1 is better for small FPR
  - M_2 is better for large FPR
- Area Under the ROC curve
  - Ideal: Area = 1.0

How to Construct an ROC curve

classifier score

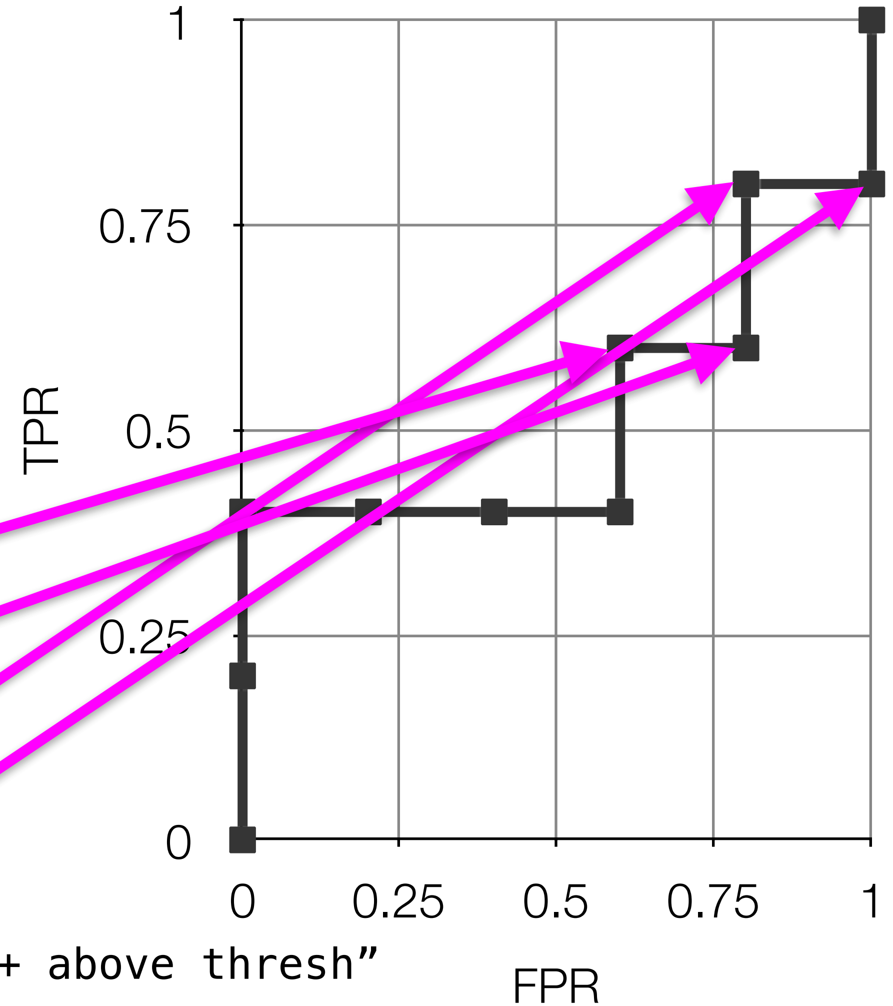
Instance #	P(+|A)	True Class
6	0.95	+
2	0.93	+
5	0.87	-
4	0.85	-
9	0.85	-
1	0.85	+
10	0.76	-
8	0.53	+
3	0.43	-
7	0.25	+

- Use classifier that produces probability score for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold, T, at each unique value of P(+|A)
- P(+|A) < T, is negative class, else it is a positive class
- Count the number of TP, FP, TN, FN at each threshold
- TP rate, TPR = TP/Positives
- FP rate, FPR = FP/Negatives

How to Construct an ROC curve

classifier score

Instance #	P(+|A)	True Class
6	0.95	+
2	0.93	+
5	0.87	-
4	0.85	-
9	0.85	-
1	0.85	+
10	0.76	-
8	0.53	+
3	0.43	-
7	0.25	+



- TP rate, TPR = TP/Positives, `“+ above thresh”`
- FP rate, FPR = FP/Negatives `“- above thresh”`

How to Construct an ROC curve

```
for i, (train, test) in enumerate(cv):
    probas = pipe_lr.fit(X_train2[train],
                         y_train[train]).predict_proba(X_train2[test])

    fpr, tpr, thresholds = roc_curve(y_train[test],
                                     probas[:, 1],
                                     pos_label=1)
```

https://github.com/rasbt/python-machine-learning-book/

Significance Testing

Tests of Significance

- Given two models:
  - Model M_1: accuracy = 85%, tested on 30 instances
  - Model M_2: accuracy = 75%, tested on 5000 instances

- Can we say M_1 is better than M_2?
  - How much confidence can we place on accuracy of M_1 and M_2?
  - Can the difference in performance measure be explained as a result of random fluctuations in the test set?

Comparing Performance of 2 Models

- Given two models, M_1 and M_2, which is better?
  - M1 is tested on D_1 (size=n_1), found error rate = e_1
  - M2 is tested on D_2 (size=n_2), found error rate = e_2
  - Assume D_1 and D_2 are independent
  - If n_1 and n_2 are sufficiently large, then

$$e_1 \sim N(\mu_1, \sigma_1)$$
$$e_2 \sim N(\mu_2, \sigma_2)$$

- Approximate:

$$\hat{\sigma}_i^2 = \frac{e_i(1 - e_i)}{n_i}$$

variance estimate comes from **binomial distribution**, which is approximated well by **normal distribution**

Comparing Performance of 2 Models

- To test if performance difference is statistically significant: $d = e_1 - e_2$ ← **estimate of the mean difference**
    - $d \sim N(d_t, \sigma_t)$ where d_t is the true difference
    - Since D_1 and D_2 are independent, their variance adds up:

$$\sigma_t^2 = \sigma_1^2 + \sigma_2^2 \cong \hat{\sigma}_1^2 + \hat{\sigma}_2^2$$
$$= \frac{e1(1-e1)}{n1} + \frac{e2(1-e2)}{n2}$$

estimate of the variance in subtracted error rates

- At (1-α) confidence level, bounds on d $= d \pm Z_{\alpha/2} \hat{\sigma}_t$

does this interval include zero?

An Illustrative Example

$$= d \pm Z_{\alpha/2} \hat{\sigma}_t$$

- Given: M_1: $n_1 = 30$, $e_1 = 0.15$
  M_2: $n_2 = 5000$, $e_2 = 0.25$
- $d = |e_2 - e_1| = 0.1$ (2-sided test)

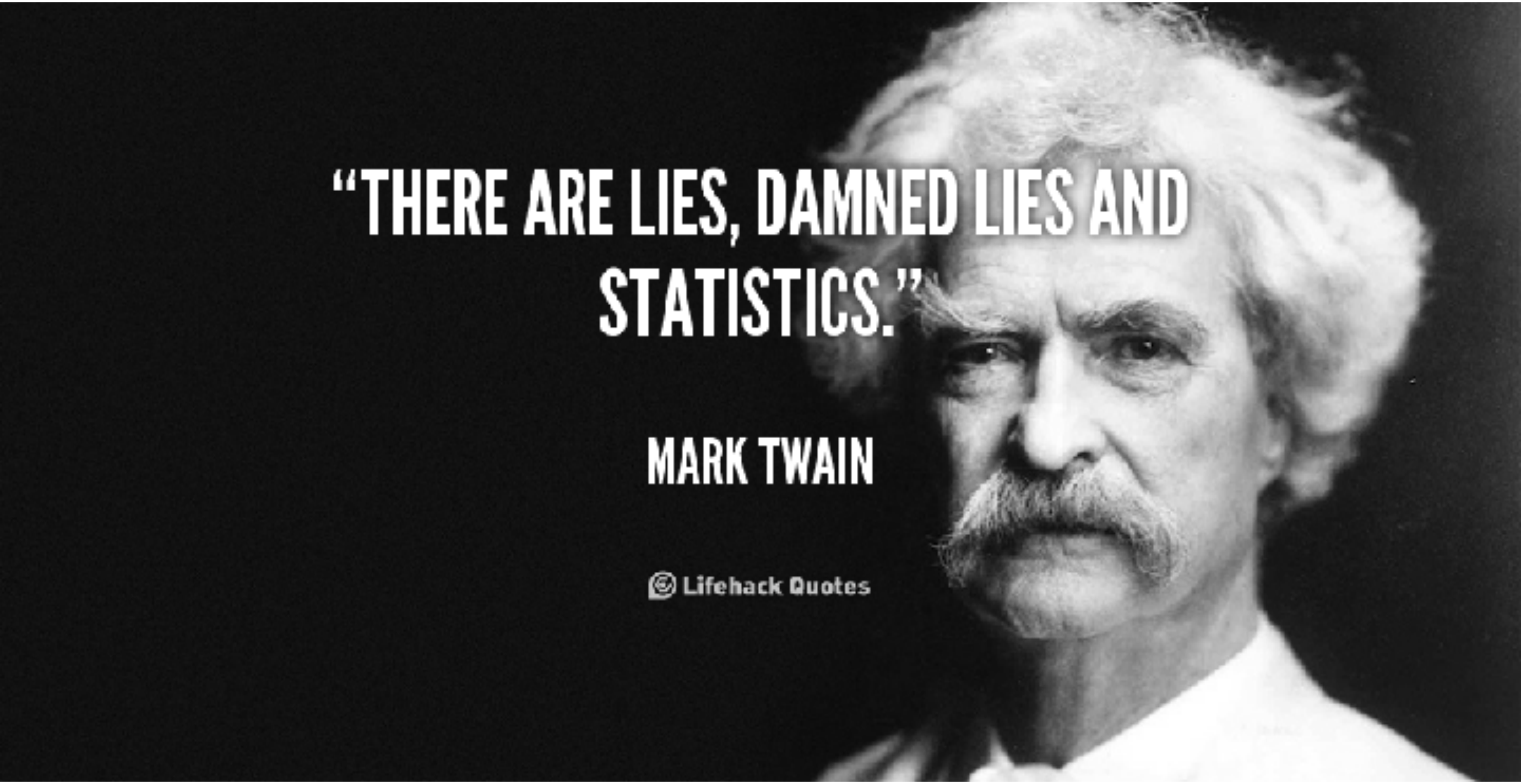
$$\hat{\sigma}_d^2 = \frac{0.15(1-0.15)}{30} + \frac{0.25(1-0.25)}{5000} = 0.0043$$

- At 95% confidence level, $Z_{\alpha/2}=1.96$

$$d_t = 0.100 \pm 1.96 \times \sqrt{0.0043} = 0.100 \pm 0.128$$

- => Interval contains 0 => difference may not be statistically significant

Another illustrative example



M_1: $n_1 = 30$, $e_1 = 0.15$
M_2: $n_2 = 5000$, $e_2 = 0.25$

Use common sense and choose the classifier with 5000 test examples

Folded statistical comparisons

- Each learning algorithm may produce k models:
  - L_1 may produce M_{11} , M_{12}, …, M_{1k}
  - L_2 may produce M_{21} , M_{22}, …, M_{2k}
- If models are generated on the same test sets D_1,D_2, …, D_k (e.g., via cross-validation)
  - For each set: compute $d_j = e_{1j} - e_{2j}$, the j^{th} difference
  - d_j has mean d and variance σ_t

$$\sigma_t^2 = \frac{1}{k-1}\sum_j^k(\bar{d} - d_j)^2$$

now we can bound to get a better idea about how the criterion varies

$$d_t = \bar{d} \pm \frac{1}{\sqrt{k}} t_{1-\alpha,k-1}\sigma_t$$

t(95%,k=10) = 2.26

mean of folds

confidence multiplier

standard deviation of folds

An illustrative Example

```
acc1 = cross_val_score(clf1, X, y=y, cv=cv)
acc2 = cross_val_score(clf2, X, y=y, cv=cv)


#==================================

t = 2.26 / np.sqrt(10)

e = (1-acc1)-(1-acc2)
# std1 = np.std(acc1)
# std2 = np.std(acc2)
stdtot = np.std(e)

dbar = np.mean(e)
print 'Range of:', dbar-t*stdtot,dbar+t*stdtot
```

$$\sigma_t^2 = \frac{1}{k-1}\sum_j^k (\bar{d} - d_j)^2$$

$$d_t = \bar{d} \pm \frac{1}{\sqrt{k}} t_{1-\alpha, k-1} \sigma_t$$

For in class assignment:

Using Cross Validation, Folding, ROC, and Statistical Significance to Evaluate Algorithm Performance