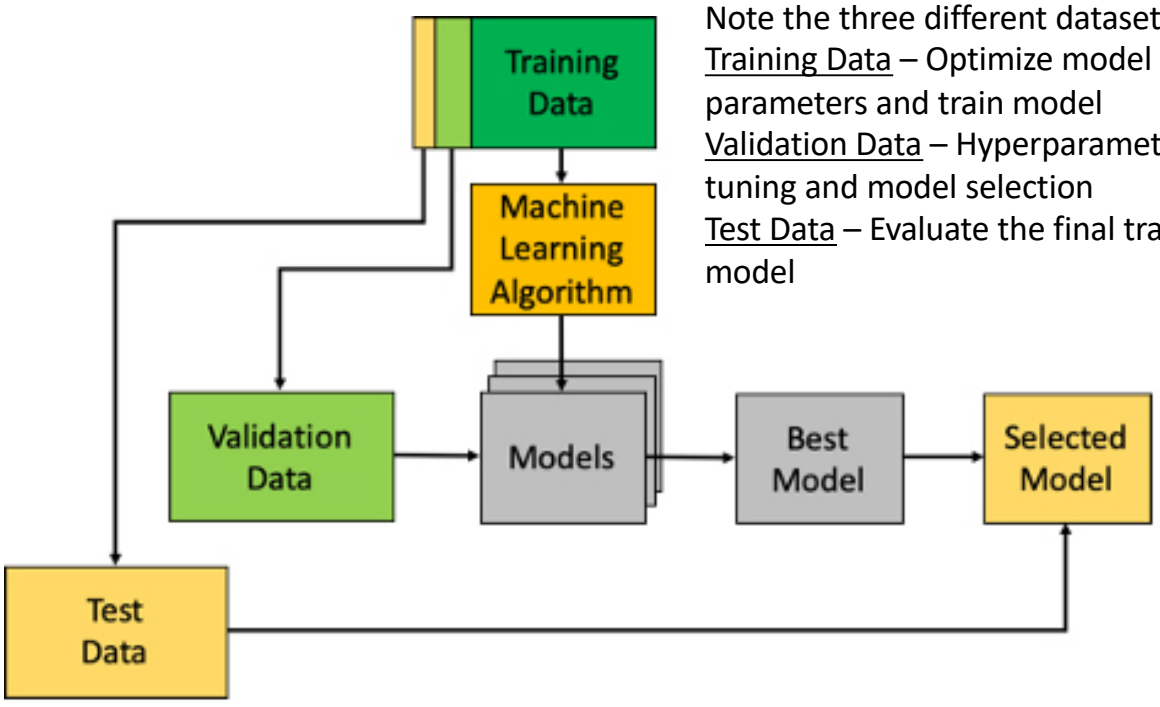
Evaluating Performance of Machine Learning Models

Mehul Motani
Electrical & Computer Engineering
National University of Singapore
Email: motani@nus.edu.sg

Supervised learning paradigm



Note the three different datasets:
Training Data – Optimize model parameters and train model
Validation Data – Hyperparameter tuning and model selection
Test Data – Evaluate the final trained model

Training and evaluating the learning algorithm

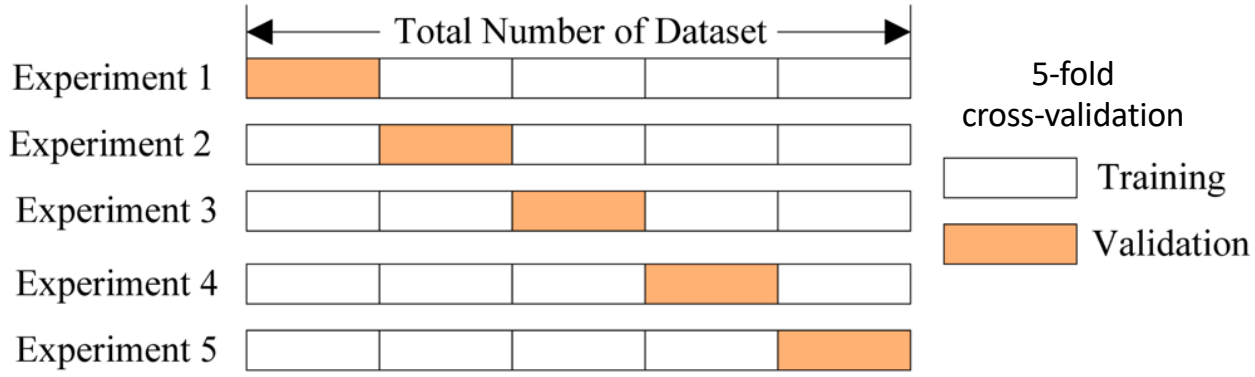
- Divide available labeled data into three sets:
- Training set:
  - Used for model parameter optimization
- Validation set
  - Used for hyperparameter tuning and model selection
- Test set:
  - Used only for final evaluation of the trained model
  - Done after training and validation are completely finished
- Avoid data leakage
  - The test data should not influence the choice of model structure or optimization of parameters.
  - If after evaluating on the test set, you don't like the results, you must set aside a new test set before training a new model.

More on the validation set

- Validation set is used when you have enough labeled data available
- Validation set
  - Used to gauge status of generalization error
  - Used to optimize small number of high-level meta parameters
    - regularization constants; number of gradient descent iterations
    - model structure: number of nodes and connections
    - types and numbers of parameters: coefficients, weights, etc.
  - Used to perform model-selection
    - For example, linear vs polynomial regression

More at: https://en.wikipedia.org/wiki/Training,_test,_and_validation_sets

K-fold cross-validation



- K-fold cross validation is used when we have little data
  - Typically, we use block folds (shown above) as this allows every sample to be in validation set.
  - We can also use random folds if samples are independent.
- Report average performance over different experiments
- Or use cross-validation for hyperparameter tuning and then report results on held-out test set.

Important – Avoid Data leakage

- Data leakage is when the test set (or validation set) leaks information to the model. This gives you an optimistic performance prediction and invalidates your entire experiment.
- If you pre-process your data (e.g., normalization), you must do this on the training set only, not on the entire dataset.
  - For example, if you include the test set in normalization, then information about the test set will leak in to the training set and the model.
  - This also applies to K-fold CV with the training and validation sets.
- In K-fold cross validation, you must discard the model and restart after every experiment.
- If after testing on the test set, you want to train a new model, you must restart with a new test set. Otherwise, information about the test set can leak into your model tweaking.

How good is a classifier?

- Accuracy
  - *a* = No. of test samples with label correctly predicted
  - *b* = No. of test samples with label incorrectly predicted
  - Example: 75 samples in test set
  - correct class label predicted for 62 samples
  - wrong class label predicted for 13 samples

$$\text{accuracy} = \frac{a}{a+b}$$

$$\text{accuracy} = \frac{62}{75} = 82.67\%$$

- Limitations of accuracy
  - Consider a two-class problem
    - number of class 1 test samples = 9990
    - number of class 2 test samples = 10
  - What if model predicts everything to be class 1?
    - accuracy is extremely high: 9990 / 10000 = 99.9 %
    - but model will never correctly predict any sample in class 2
    - in this case accuracy is misleading and does not give a good picture of model quality

Metrics for classifier performance

Confusion matrix for binary classification

		actual class: class 1 *negative*	actual class: class 2 *positive*
predicted class	**class 1** *negative*	**21** (*TN*)	**6** (*FN*)
	class 2 *positive*	**7** (*FP*)	**41** (*TP*)

TN: true negatives *FN*: false negatives

FP: false positives *TP*: true positives

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

$$Recall\ /\ Sensitivity = \frac{TP}{TP+FN}$$

$$Specificity = \frac{TN}{TN+FP}$$

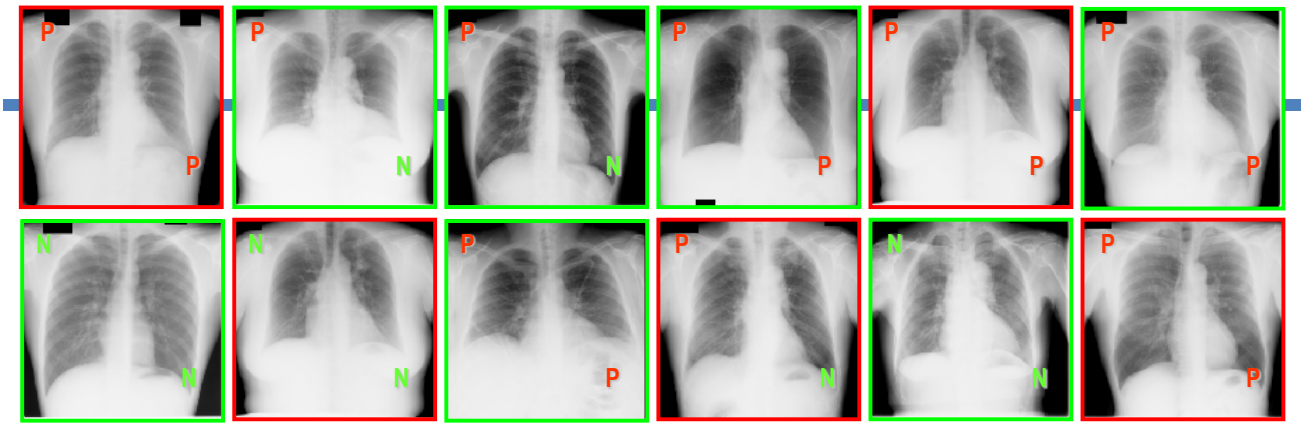
$$Precision = \frac{TP}{TP+FP}$$

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

https://en.wikipedia.org/wiki/Confusion_matrix

Recall, Specificity, and Precision

Recall and Specificity	Recall and Precision
• Recall → True positive rate • Specificity → True negative rate • Are useful when false positives and false negatives have different consequences • 'stupid' methods can achieve large recall at the expense of low specificity (and vice versa) • Which one is more important depends upon application • Recall is important when false negatives are catastrophic (e.g., missed cancer detection) • Specificity is important when false positives are bad (e.g., identifying the wrong person in a DNA test)	• Recall → True positive rate • Precision → Positive predictive value • 'stupid' methods can achieve large recall at the expense of low precision (and vice versa) • Which one is more important depends upon application • Recall is important when false negatives are catastrophic and you want detect all positive cases. • Precision in important when being right (positive prediction is correct) outweighs detecting all positives. • F1-Score is the harmonic mean of precision and recall (used when both are important).



Algorithm 1

P true positives (TP) = 3
P false positives (FP) = 3
N false negatives (FN) = 2
N true negatives (TN) = 4

Accuracy = (TP+TN) /(TP+TN+FP+FN) = 7 / 12 = **0.58**
Recall = TP / (TP + FN) = 3 / 5 = **0.6**
Specificity = TN / (TN + FP) = 4 / 7 = **0.57**
Precision = TP / (TP + FP) = 3 / 6 = **0.5**

Algorithm 2

P = 4 N = 1
P = 5 N = 2

Accuracy = (TP+TN) /(TP+TN+FP+FN) = 6 / 12 = **0.5**
Recall = TP / (TP + FN) = 4 / 5 = **0.8**
Specificity = TN / (TN + FP) = 2 / 7 = **0.29**
Precision = TP / (TP + FP) = 4 / 9 = **0.44**

Which algorithm is better?

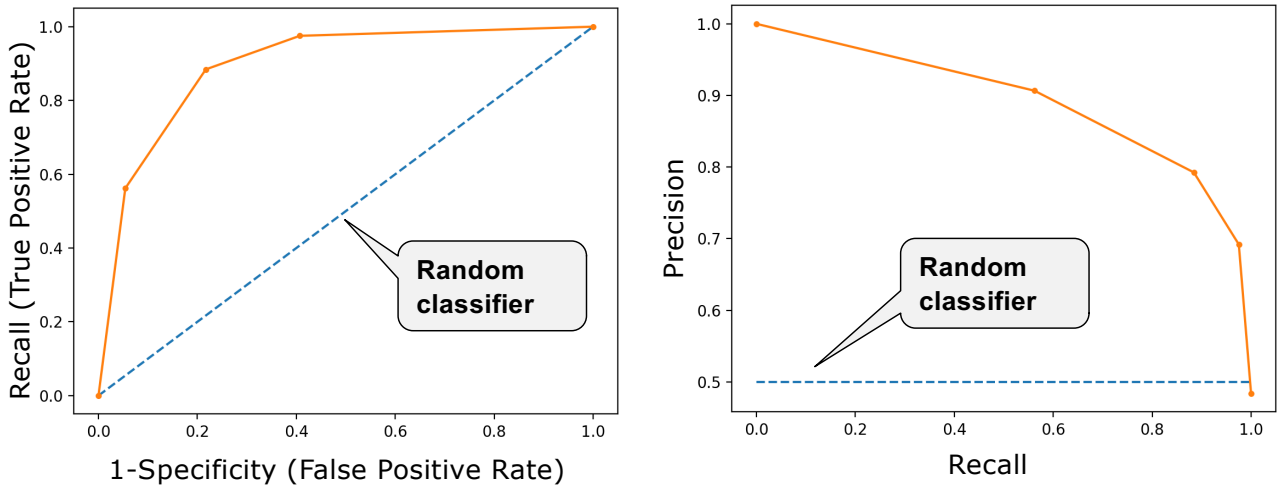
Exploring the performance tradeoffs

- In a classification problem, we may decide to predict the class values directly or predict the probabilities for each class instead.
- Computing probabilities allows us to tradeoff false positives and false negatives using a threshold.
- Two diagnostic tools that help in the interpretation of probabilistic forecast for binary classification problems are Receiver Operating Characteristic (ROC) curves and Precision-Recall curves.
- ROC Curves summarize the trade-off between the true positive rate (Recall) and false positive rate (1-Specificity) for a predictive model using different probability thresholds.
- Precision-Recall curves summarize the trade-off between the true positive rate (Recall) and the positive predictive value (Precision) for a predictive model using different probability thresholds.
- ROC curves are appropriate when the observations are balanced between each class, whereas Precision-Recall curves are appropriate for imbalanced datasets.
- For both tradeoffs, the area under the curve (AUC) can be used as a summary of the tradeoff.

More on the ROC Curve and AUC-ROC

- Many algorithms (e.g., logistic regression) return a probability which can then be mapped to two or more classes.
- Other algorithms (e.g., SVM and Random Forest) can be configured to return probabilities instead of class decisions.
- Mapping from probabilities is done by comparing to a threshold. For example, a value below the threshold can be class 0 (negative) and a value above the threshold can be class 1 (positive).
- You might think that a threshold of 0.5 is right but thresholds are problem dependent and must be tuned based on the impact of false positives and missed detections.
- Varying the threshold allows us to explore tradeoffs.
  - For example, lowering the threshold classifies more items as positive, thus increasing both False Positives and True Positives.
- The ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at different classification thresholds. This curve plots true positives vs false positives.
  - The area under the ROC (AUC-ROC) is a single metric to evaluate a classifier. An AUC-ROC value closer to 1 indicates a good classification algorithm.
  - A random classifier has an AUC-ROC of 0.5.

ROC Curves and PR Curves

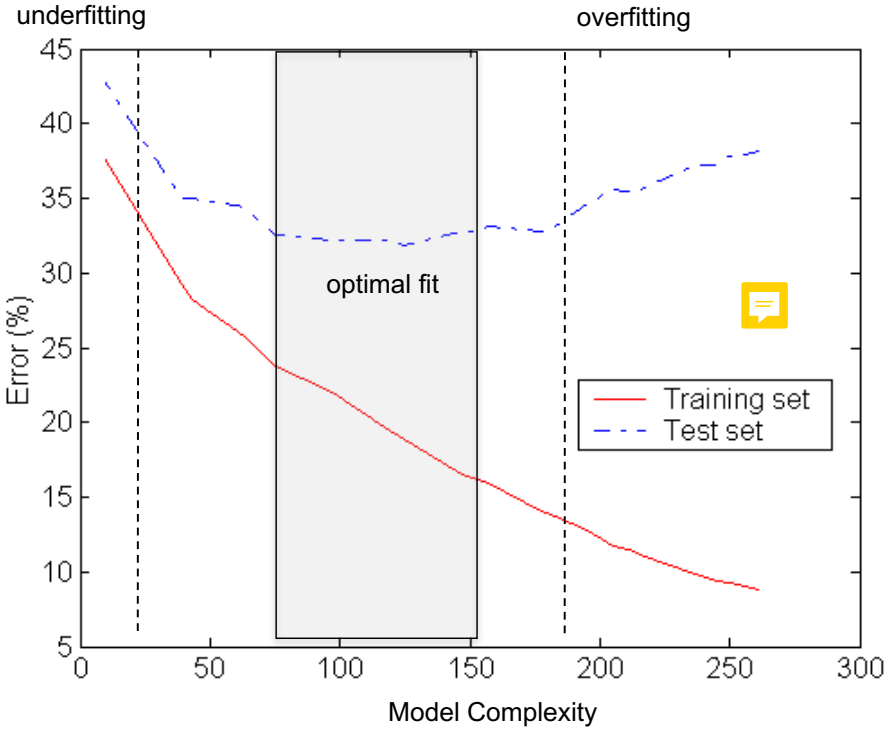


- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_curve.html
- How to Use ROC Curves and Precision-Recall Curves for Classification in Python: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

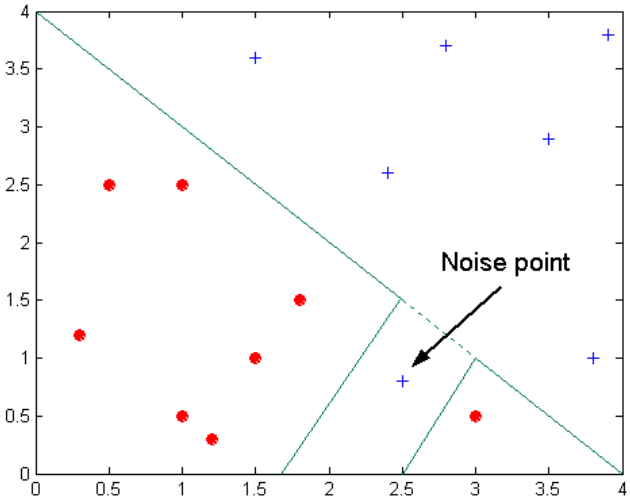
Underfitting and overfitting

- Fit of model to training and test sets is controlled by:
  - model capacity/complexity (≈ number of parameters)
    - Example: number of nodes/levels in decision tree
    - Example: polynomial degree for regression
    - Example: Number of nodes/layers in neural network
  - stage of optimization
    - example: number of iterations in a gradient descent optimization
- Underfitting leads to poor performance
  - On both training and test sets
- Overfitting leads to poor generalization
  - Good on training set, bad on test set

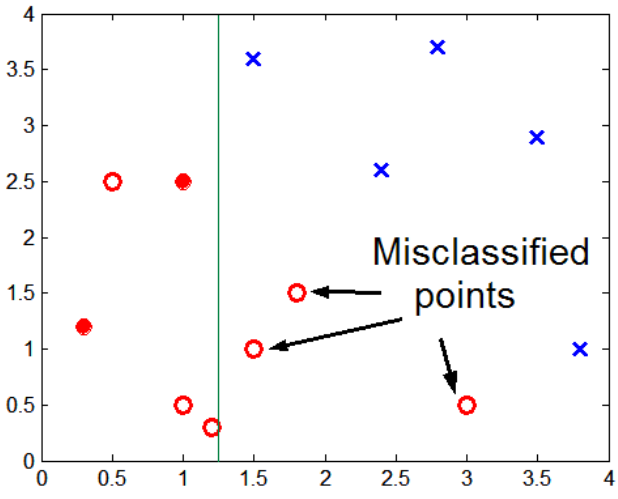
Underfitting and overfitting



Causes of Overfitting



Decision boundary distorted by noise point

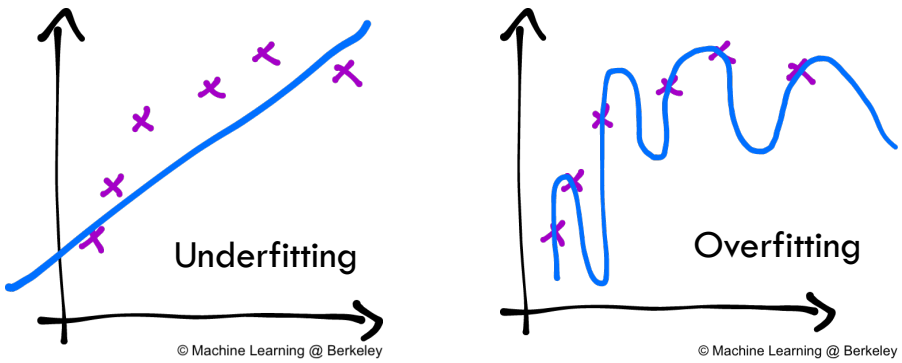


Lack of data points in lower half of diagram makes it difficult to correctly predict class labels in that region.

Occam's Razor

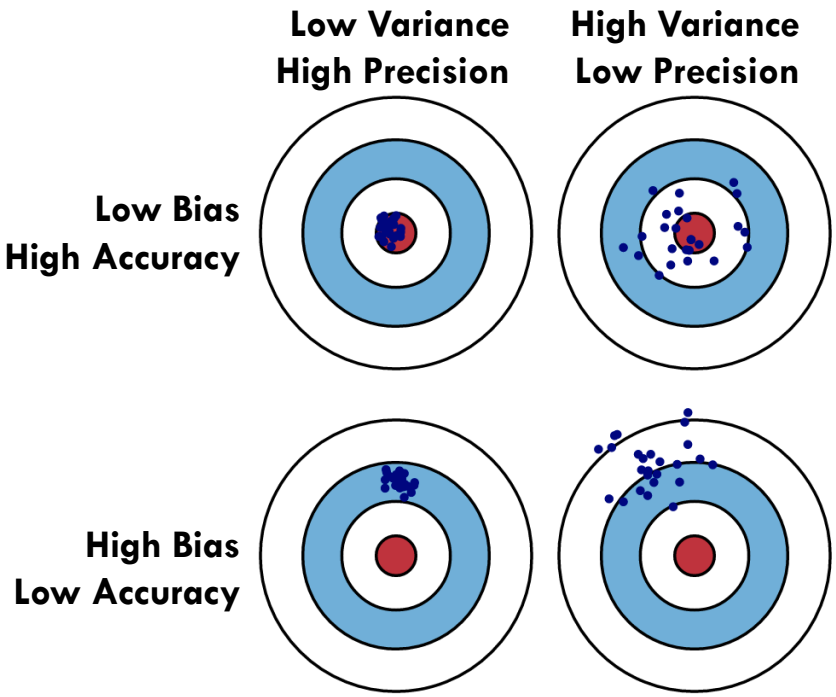
- Given two models with similar generalization errors, one should prefer the simpler model over the more complex model.
- For complex models, there is a greater chance it was fitted accidentally by errors in data.
- Model complexity should therefore be considered when evaluating a model.
  - More "complex" models tend to overfit the training data, and thus have higher variance, but have lower bias.
    - For example: Full depth decision tree or Large C soft margin SVM
  - Less "complex" models tend to underfit the training data, and thus have lower variance, but have higher bias.
    - For example: Limited depth decision tree or Small C soft margin SVM

Model Fit and Bias / Variance



- Underfitting leads to high training error and high test error.
- Underfitting is bad as it means we have not learned enough from our data. This error is known as bias.
- Overfitting leads to low training error and high test error.
- Overfitting is bad as it means we are too sensitive to our data. This error is known as variance.
- We want both low bias (no underfitting) and low variance (no overfitting)!

Bias-Variance Tradeoff

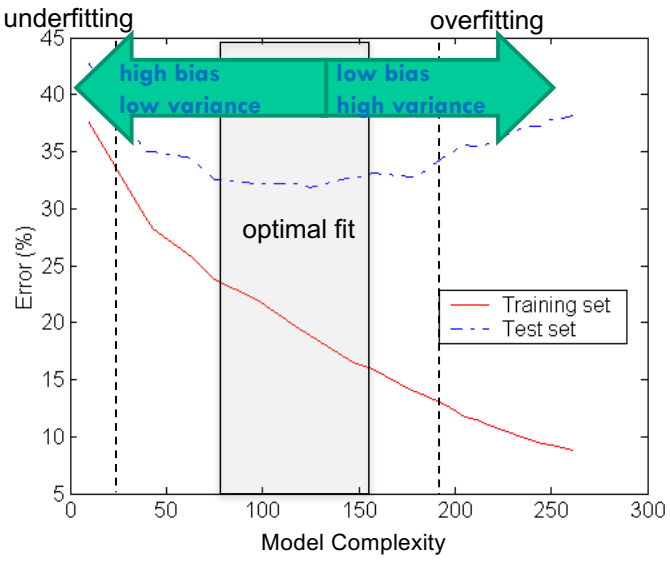


Bias – Measures the accuracy of the model. It is the error due to underfitting.

Variance – Measures how precise the model is. It is the error due to overfitting.

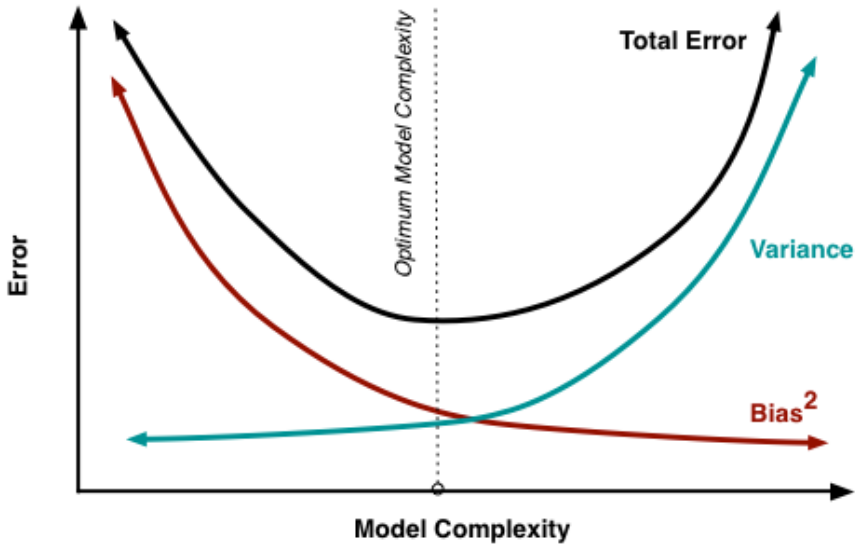
We want to reduce both bias and variance!

Bias-Variance Tradeoff



- Error on the dataset used to *fit* the model doesn't predict future performance.
- Too much complexity can diminish model's accuracy on future data.
- Complex model:
  - Low 'bias': the model fit is good, i.e., the model value is close to the data's expected value.
  - High 'variance': Model more likely to make a wrong prediction.
- Sweet spot: the best complexity lies where the test error reaches a minimum, that is, somewhere in between a very simple and a very complex model.
- Data science is both art and science!
- https://ml.berkeley.edu/blog/2017/07/13/tutorial-4/

The Bias squared-Variance Curve



- A curve of squared bias vs variance showing the inverse correlation that is typical of the relation between the two as the model gets more complex.
- It is not uncommon for the resulting Total Error to follow some variant of the U-shape shown in the figure.

XKCD: Computers vs. Humans