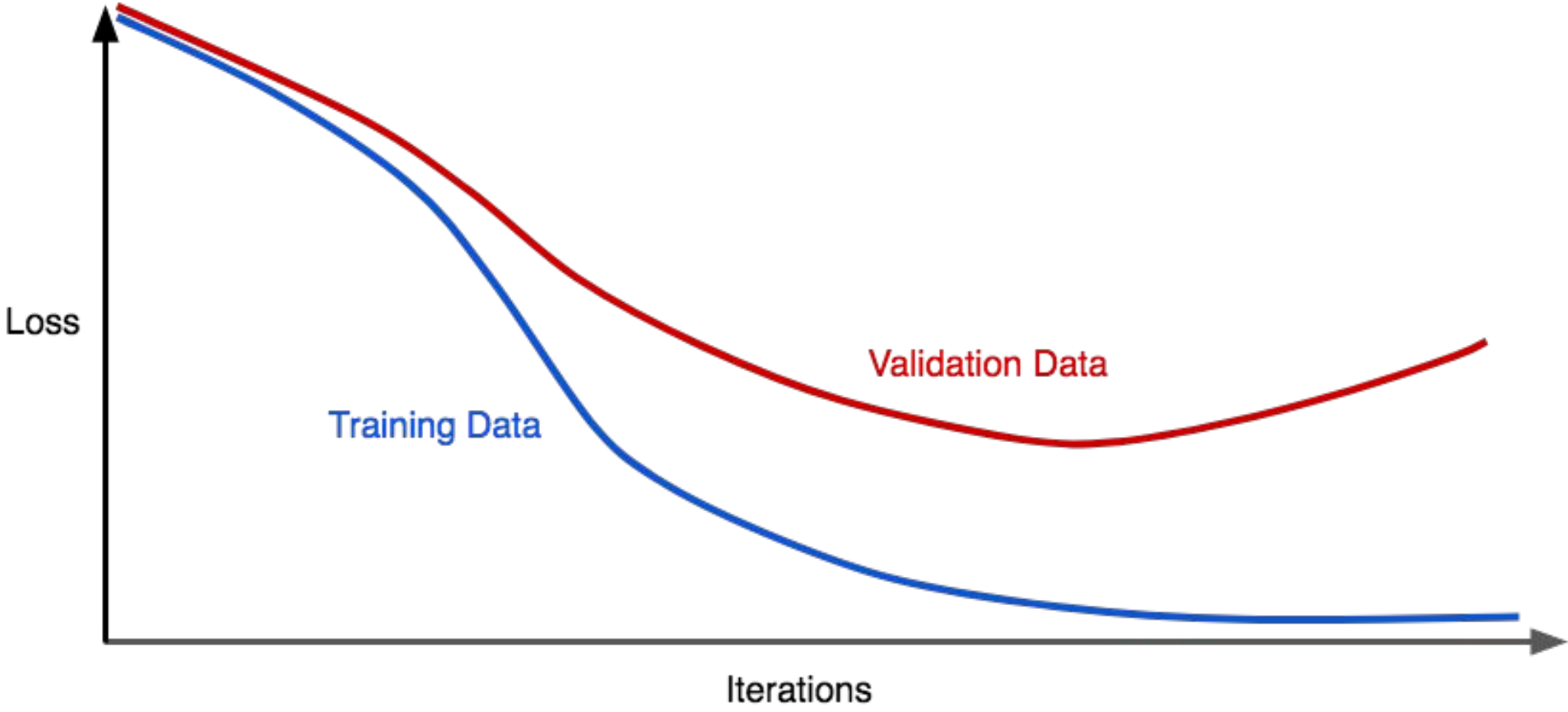
Regularization

Generalization Curve



Risk Minimization

- Loss
  - How “bad” a model is
- Empirical risk minimization
  - Minimizes loss
- Structural risk minimization (SRM)
  - Minimizes loss and **complexity**
    ≃
    $$\text{minimize}(\text{loss function} + \lambda(\text{regularization function}))$$
  - λ = regularization rate (more to follow)
  - An algorithm that balances two goals:
    - The desire to build the most predictive model (for example, lowest loss).
    - The desire to keep the model as simple as possible (for example, strong regularization).

Regularization

- Regularization = The **penalty** on a model's complexity.
  - Remember Ockham's principle for ML, the simpler the model, the better
  - Helps prevent overfitting
- How can we measure model complexity?
  - as a function of the *weights* of all the features in the model.
  - as a function of the *total number of features* with nonzero weights (Regularization for Sparsity).

L_1 vs L_2 Regularization

- Penalties
  - L_1 penalizes $|weight|$
  - L_2 penalizes $weight^2$
- Derivatives
  - The derivative of L_2 is 2 * *weight*
    - It removes x% of the weight every time
  - The derivative of L_1 is *k* (a constant)
    - It subtracts some constant from the weight every time
- Given this, which regularization could drive some weights to 0?

L_1 Regularization

- Penalizes **weights** in proportion to the sum of the **absolute values** of the weights.
- Aka **LASSO** regression
- In models relying on **sparse features**, L_1 regularization helps drive the weights of irrelevant or barely relevant features **to exactly 0**, which removes those features from the model.

L_2 Regularization

- Penalizes **weights** in proportion to the sum of the ***squares*** of the weights.
- Aka **Ridge** Regression
- L_2 regularization helps drive **outlier weights** (those with high positive or low negative values) **closer to 0 but not quite to 0**.
- L_2 regularization always improves generalization in linear models.

New Error Functions

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

L1 Regularization Gradient & Update

$$E = \frac{1}{2} * \sum (t_j - o_j)^2 + \lambda * \sum |w_i|$$

plain error

weight penalty

elegant math

oops, non-differentiable

$$\frac{\partial E}{\partial w_{ij}}$$

$$\Delta w_{ij} = \eta * [x_i * (o_j - t_j) * o_j * (1 - o_j)] + [\lambda * sgn(w_{ij})]$$

learning rate

signal

L2 Regularization Gradient & Update

$$E = \frac{1}{2} * \sum (t_k - o_k)^2 + \frac{\lambda}{2} * \sum {w_i}^2$$

plain error

weight penalty

elegant math

simple math

$$\frac{\partial E}{\partial w_{jk}}$$

$$\Delta w_{jk} = \eta * [\, x_j * (o_k - t_k) * o_k * (1 - o_k)\,] + [\, \lambda * w_{jk}\,]$$

learning rate

signal

Why are they called L*?

- From Linear Algebra, a norm is the total length of all vectors in space.
- L0 norm = total number of non-zero vector elements
- L1 norm = manhattan taxicab distance = sum of the magnitudes of vectors in a space

$$||X||_1 = |3| + |4| = 7$$

- L2 norm = euclidean norm = sum of shortest distances

$$||X||2 = \sqrt{(|3|^2 + |4|^2)} = \sqrt{9+16} = \sqrt{25} = 5$$

Regularization for Sparsity

- Large models require more RAM
  - Sparse vectors often contain many dimensions
  - Feature crosses create even more dimensions
- Zeroing out features will save RAM and may reduce noise in the model.
  - How do we encourage weights to drop to exactly 0 where possible, effectively removing the corresponding features from the model?

L_1 Regularization for Sparsity

- L_2 regularization encourages weights to be small, but doesn't force them to exactly 0.0.
- L_1 has a discontinuity at 0 (thanks to absolute values), which causes subtraction results that cross 0 to become zeroed out.
- L_1 regularization—penalizing the absolute value of all the weights—turns out to be quite efficient for wide models.
- Simulation

When to use L1 vs Lv2?

- Ridge Regression (L2) is good if want to keep all the features and avoid the weights to blow up it is a good default.
- But if you think that only few features are useful it would be best to go with Lasso Regression as it sets the weights of features that are less useful to 0.
- Further Reading
  - https://medium.com/analytics-vidhya/types-of-regularization-and-when-to-use-them-f0350ca651a7
  - https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization

Dropout Regularization

- A form of **regularization** useful in training **neural networks**.
- Dropout regularization works by removing a random selection of a fixed number of the units in a network layer for a single gradient step.
- The more units dropped out, the stronger the regularization.
- This is analogous to training the network to emulate an exponentially large ensemble of smaller networks.
- For full details, see Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

Early Stopping Regularization

- A method for **regularization** that involves ending model training *before* training loss finishes decreasing.
- In early stopping, you end model training when the loss on a **validation dataset** starts to increase, that is, when **generalization** performance worsens.
- Not a formal regularization method, but can effectively limit overfitting.

Regularization Rate λ

- A scalar value, represented as lambda, specifying the relative importance of the regularization function.
- The following simplified **loss** equation shows the regularization rate's influence:

$$\text{minimize}(\text{loss function} + \lambda(\text{regularization function}))$$

- Raising the regularization rate reduces **overfitting** but may make the model less **accurate**.

Regularization Rate λ

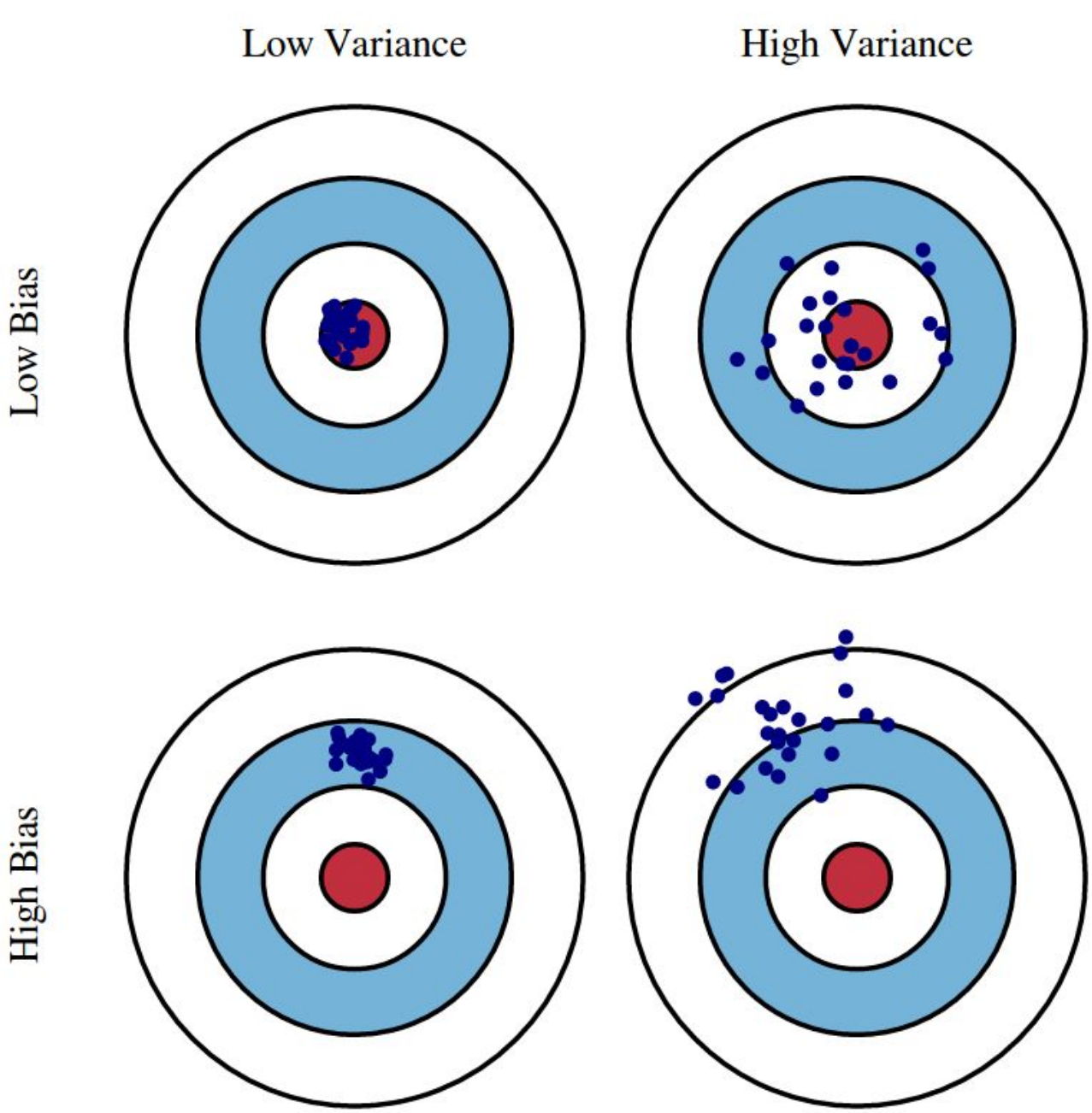
- When choosing a lambda value, the goal is to strike the right balance between simplicity and training-data fit:
  - If your lambda value is too high, your model will be simple, but you run the risk of *underfitting* your data. Your model won't learn enough about the training data to make useful predictions.
  - If your lambda value is too low, your model will be more complex, and you run the risk of *overfitting* your data. Your model will learn too much about the particularities of the training data, and won't be able to generalize to new data.
- Setting lambda to zero removes regularization completely. In this case, training focuses exclusively on minimizing loss, which poses the highest possible overfitting risk.
- The ideal value of lambda produces a model that generalizes well to new, previously unseen data. Unfortunately, that ideal value of lambda is data-dependent, so you'll need to do some tuning.

Learning Rate and λ

- There's a close connection between learning rate and lambda.
- Strong L_2 regularization values tend to drive feature weights closer to 0.
- Lower learning rates (with early stopping) often produce the same effect because the steps away from 0 aren't as large.
- Consequently, **tweaking learning rate and lambda simultaneously may have confounding effects**.

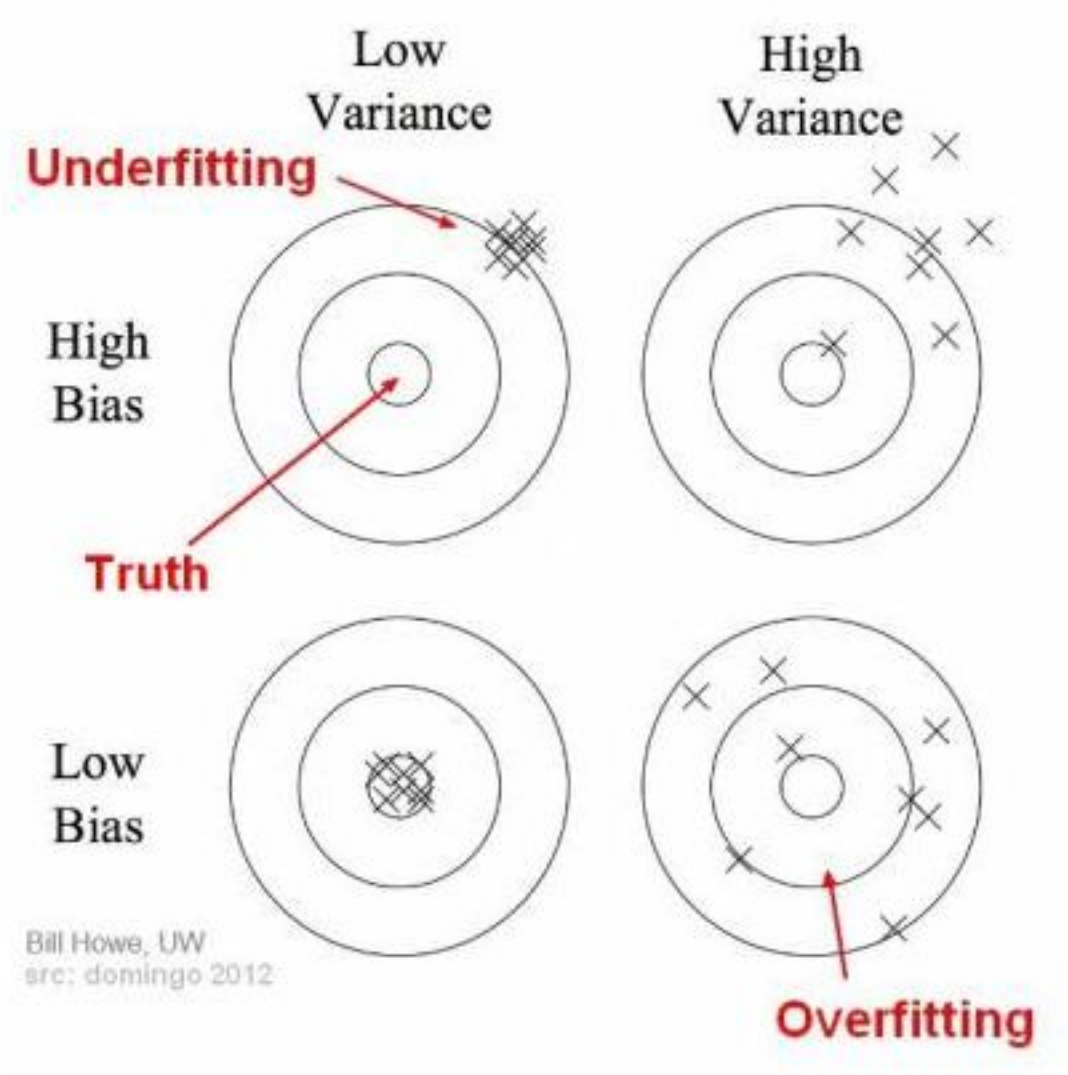
Bias-Variance Tradeoff

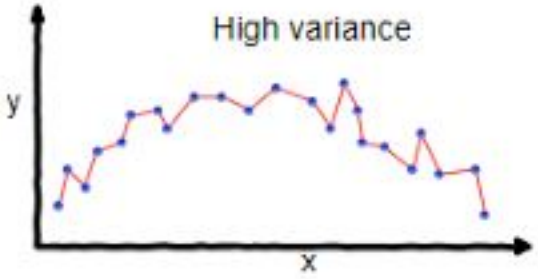
- Bias (error)
  - The bias is known as the difference between the prediction of the values by the ML model and the correct value.
  - The simplifying assumptions made by a model to make the target function easier to learn.
  - High bias gives a large error in training as well as testing data.
- Variance (error)
  - The variability of model prediction for a given data point (for multiple builds of the model) which tells us spread of our data is called the variance of the model.
  - The amount that the estimate of the target function will change if different training data was used.
  - The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn't seen before.
- Irreducible (error)
  - Cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.
- If the algorithm is too simple (few parameters) then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex (too many parameters) then it may be on high variance and low bias. In the latter condition, the new entries will not perform well (overfitting).



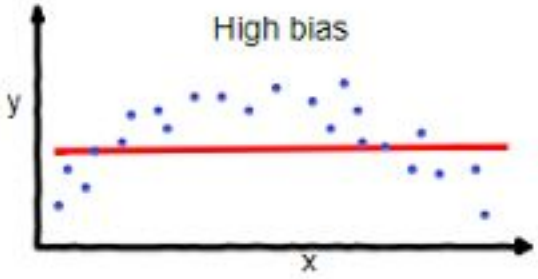
Dot = realization of model building process

Further reading here

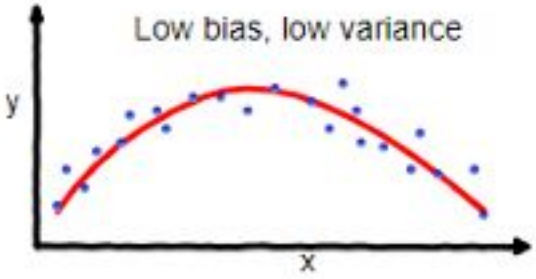

Low
Variance
High
Variance
Underfitting
High
Bias
Truth
Low
Bias
Bill Howe, UW
src: domingo 2012
Overfitting



overfitting

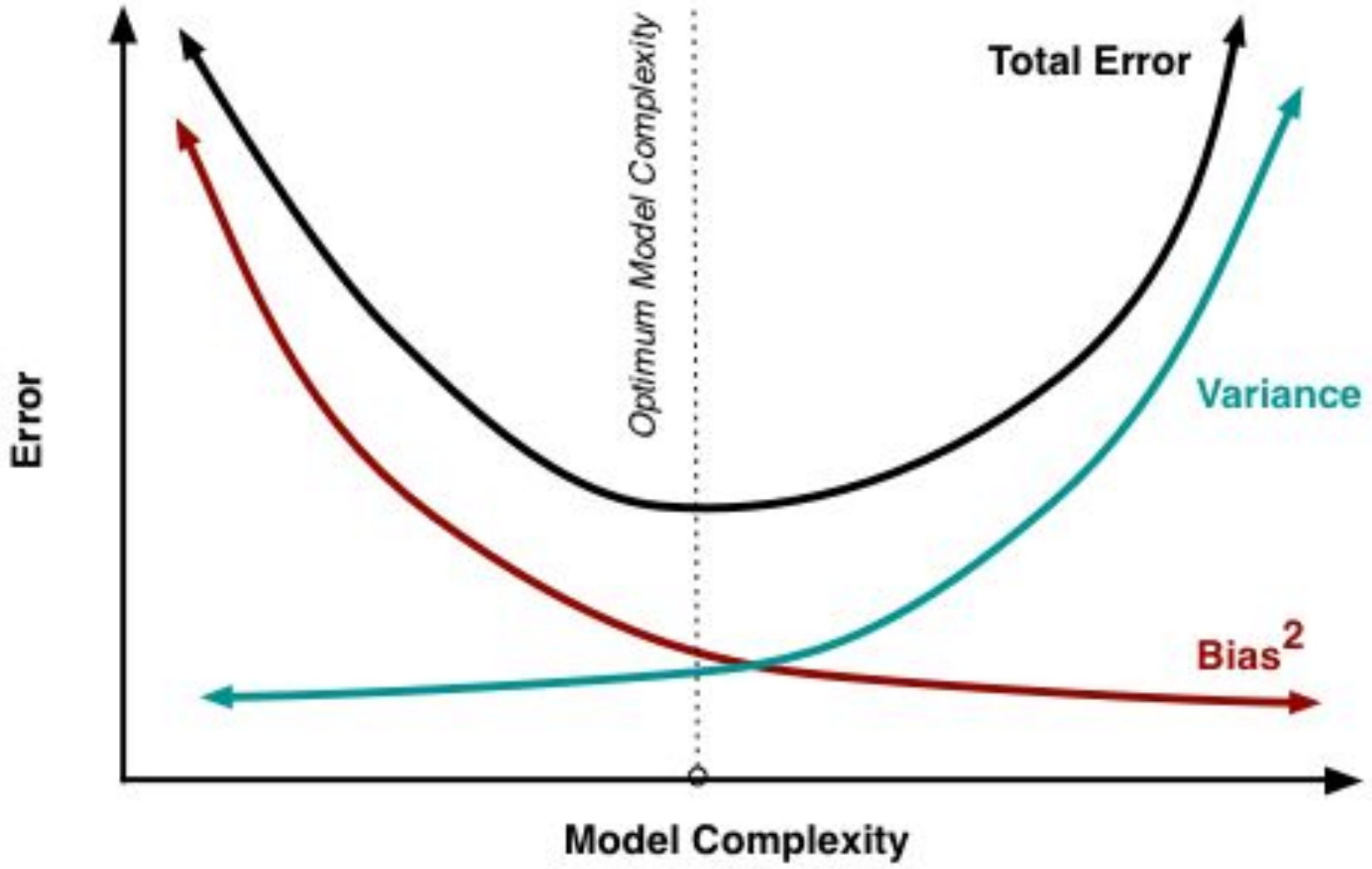


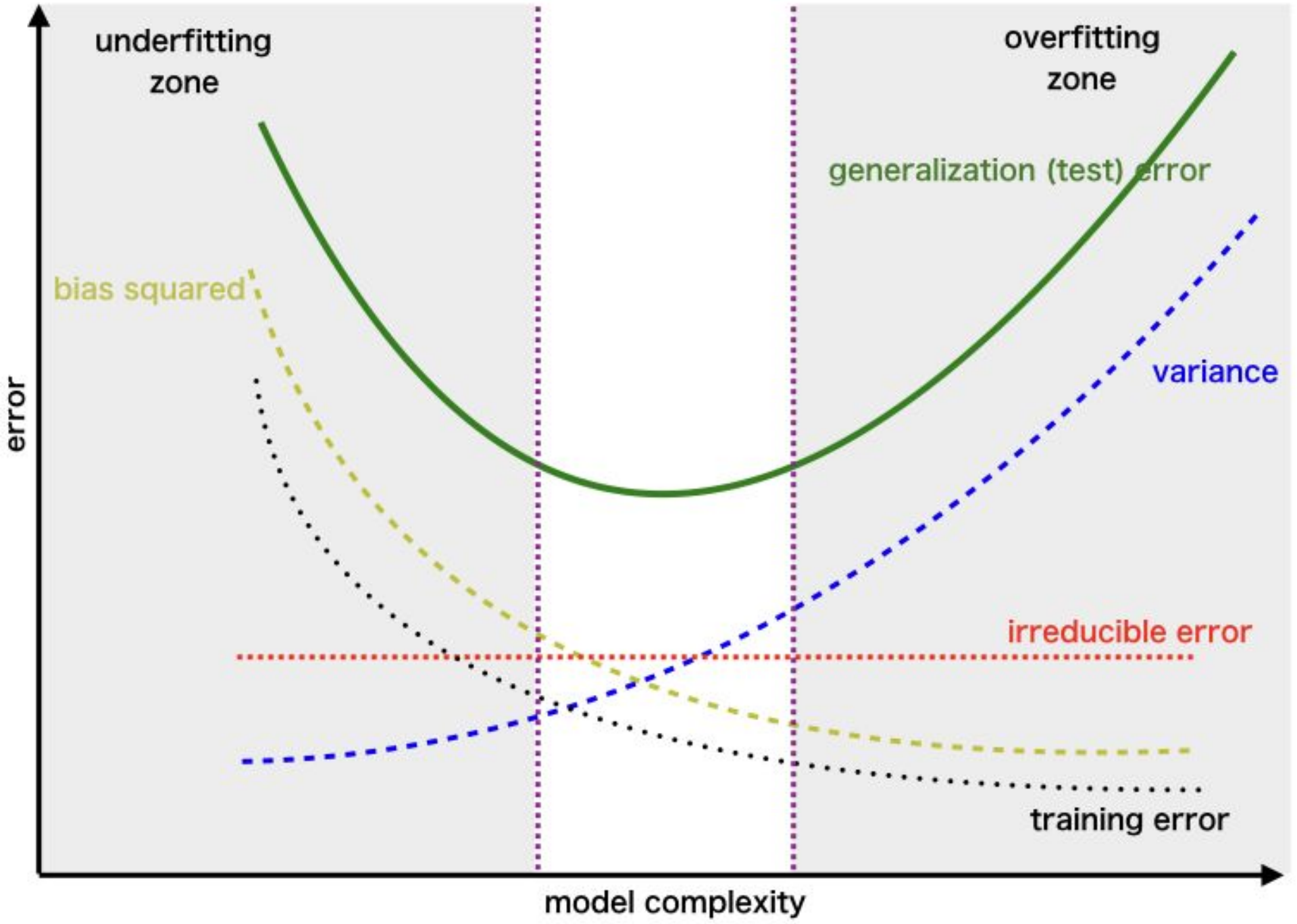
underfitting



Good balance

Bias-Variance Tradeoff




underfitting zone
overfitting zone
generalization (test) error
bias squared
variance
irreducible error
training error
error
model complexity

Bias-Variance Tradeoff

- Examples of low-bias machine learning algorithms include:
  - Decision Trees, k-Nearest Neighbors and Support Vector Machines.
- Examples of high-bias machine learning algorithms include:
  - Linear Regression, Linear Discriminant Analysis and Logistic Regression.
- Examples of low-variance machine learning algorithms include:
  - Linear Regression, Linear Discriminant Analysis and Logistic Regression.
- Examples of high-variance machine learning algorithms include:
  - Decision Trees, k-Nearest Neighbors and Support Vector Machines.
- **The goal of any machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.**
- Further reading
  - https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

Curse of Dimensionality (CoD)

- Refers to a set of problems that arise when working with high-dimensional data.
  - The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse.
  - In order to obtain a reliable result, the amount of data needed often grows exponentially with the dimensionality.
- The dimension of a dataset corresponds to the number of attributes/features that exist in a dataset.
- A dataset with a large number of attributes, generally of the order of a hundred or more, is referred to as high dimensional data.

Ex: Need 10 data points per binary feature for good accuracy

Need for Data Points with Increase in Dimensions

1 Binary feature →	2^1 unique values →	$2^1 \times 10 = 20$ data points
2 Binary features →	2^2 unique values →	$2^2 \times 10 = 40$ data points
3 Binary features →	2^3 unique values →	$2^3 \times 10 = 80$ data points
.	.	.
.	.	.
.	.	.
k Binary features →	2^k unique values →	$2^k \times 10$ data points

CoD – Data Sparsity Issue

- The training samples do not capture all combinations in high dimensional data.
- Training a model with sparse data could lead to high-variance or overfitting condition. This is because while training the model, the model has learnt from the frequently occurring combinations of the attributes and can predict the outcome accurately.
- In real-time when less frequently occurring combinations are fed to the model, it may not predict the outcome accurately.

CoD – Distance Concentration

- Refers to the problem of all the pairwise distances between different samples/points in the space converging to the same value as the dimensionality of the data increases.
- Several machine learning models such as clustering or nearest neighbours' methods use distance-based metrics to identify similar or proximity of the samples.
- Due to distance concentration, the concept of proximity or similarity of the samples may not be qualitatively relevant in higher dimensions.

CoD – Further Reading

- https://www.mygreatlearning.com/blog/understanding-curse-of-dimensionality/
- https://en.wikipedia.org/wiki/Curse_of_dimensionality
- https://towardsdatascience.com/curse-of-dimensionality-a-curse-to-machine-learning-c122ee33bfeb