

Introduction to Data Science Lecture 9 Machine Learning Part 2

SEAS 6401 Fall 2019

Benjamin Harvey

Outline



- Supervised Learning
 - Classification/Regression
 - Bias/Variance tradeoff
- K-NN
- Naïve Bayes
- Training Issues
 - Measuring model quality
 - Over-fitting
 - Cross-validation

Machine Learning



- Supervised: We are given input/output samples (X, y) which we relate with a function y = f(X). We would like to "learn" f, and evaluate it on new data. Types:
 - Classification: y is discrete (class labels).
 - Regression: y is continuous, e.g. linear regression.

- Unsupervised: Given only samples X of the data, we compute a function f such that y = f(X) is "simpler".
 - Clustering: y is discrete
 - Y is continuous: Matrix factorization, Kalman filtering, unsupervised neural networks.

Machine Learning



Supervised:

- Is this image a cat, dog, car, house?
- How would this user score that restaurant?
- Is this email spam?
- Is this blob a supernova?

Unsupervised:

- Cluster some hand-written digit data into 10 classes.
- What are the top 20 topics in Twitter right now?
- Find and cluster distinct accents of people at GWU.

Techniques



Supervised Learning:

- kNN (k Nearest Neighbors)
- Naïve Bayes
- Linear + Logistic Regression
- Support Vector Machines
- Random Forests
- Neural Networks

Unsupervised Learning:

- Clustering
- Topic Models
- HMMs (Hidden Markov Models)
- Neural Networks

Outline



- Supervised Learning
 - Classification/Regression
 - Bias/Variance tradeoff
- K-NN
- Naïve Bayes
- Training Issues
 - Measuring model quality
 - Over-fitting
 - Cross-validation

Predicting from Samples

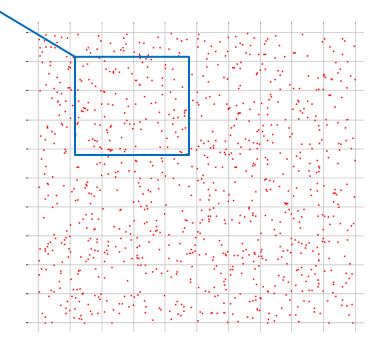


- Most datasets are samples from an infinite population.
- We are most interested in models of the population, but we have access only to a sample of it.

For datasets consisting of (X,y)

features X + label ya model is a prediction y = f(X)

We train on a training sample D and we denote the model as $f_D(X)$



Bias and Variance



Our data-generated model $f_D(X)$ is a **statistical estimate** of the true function f(X).

Because of this, its subject to bias and variance:

Bias: if we train models $f_D(X)$ on many training sets D, bias is the expected difference between their predictions and the true y's.

i.e.
$$Bias = E[f_D(X) - y]$$

E[] is taken over points X and datasets D

Variance: if we train models $f_D(X)$ on many training sets D, variance is the variance of the estimates:

$$Variance = E \left[\left(f_D(X) - f(X) \right)^2 \right]$$

Where $f(X) = E[f_D(X)]$ is the average prediction on X.

Bias and Variance Tradeoff GW

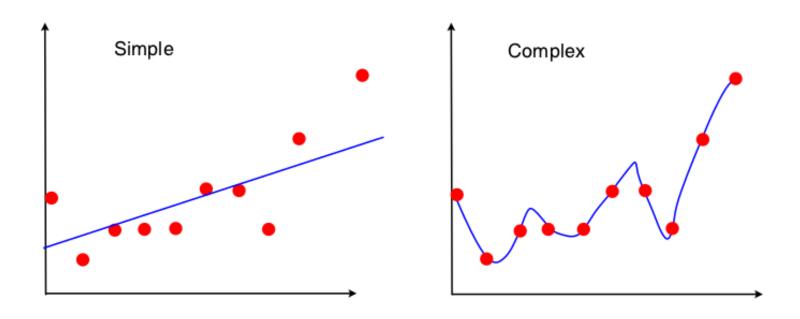
There is usually a bias-variance tradeoff caused by model complexity.

Complex models (many parameters) usually have lower bias, but higher variance.

Simple models (few parameters) have higher bias, but lower variance.

Bias and Variance Tradeoff GW

e.g. a linear model can only fit a straight line. A high-degree polynomial can fit a complex curve. But the polynomial can fit the individual sample, rather than the population. Its shape can vary from sample to sample, so it has high variance.



Bias and Variance Tradeoff GW

The total expected error is

 $Bias^2 + Variance$

Because of the bias-variance trade-off, we want to **balance** these two contributions.

If *Variance* strongly dominates, it means there is too much variation between models. This is called **over-fitting**.

If *Bias* strongly dominates, then the models are not fitting the data well enough. This is called **under-fitting**.

Outline



- Supervised Learning
 - Classification/Regression
 - Bias/Variance tradeoff
- K-NN
- Naïve Bayes
- Training Issues
 - Measuring model quality
 - Over-fitting
 - Cross-validation

k-Nearest Neighbors



Given a query item: Find k closest matches in a labeled dataset ↓





Benjamin Harvey, Ph.D., EMSE 6992 – Introduction to Data Analytics

k-Nearest Neighbors



Given a query item: Find k closest matches



Return the most Frequent label



Benjamin Harvey, Ph.D., EMSE 6992 – Introduction to Data Analytics



k = 3 votes for "cat"





Benjamin Harvey, Ph.D., EMSE 6992 - Introduction to Data Analytics

k-Nearest Neighbors



2 votes for cat,1 each for Buffalo,Deer, Lion



Cat wins...



Benjamin Harvey, Ph.D., EMSE 6992 – Introduction to Data Analytics

k-NN issues



The Data is the Model

- No training needed.
- Accuracy generally improves with more data.
- Matching is simple and fairly fast if data fits in memory.
- Usually need data in memory, but can be run off disk.

Minimal Configuration:

- Only parameter is k (number of neighbors)
- But two other choices are important:
 - Weighting of neighbors (e.g. inverse distance)
 - Similarity metric

k-NN Flavors



Classification:

- Model is y = f(X), y is from a discrete set (labels).
- Given X, compute y = majority vote of the k nearest neighbors.
- Can also use a weighted vote* of the neighbors.

Regression:

- Model is y = f(X), y is a real value.
- Given X, compute y = average value of the k nearest neighbors.
- Can also use a weighted average* of the neighbors.
- * Weight function is usually the inverse distance.

K-NN distance measures



• Euclidean Distance: Simplest, fast to compute

$$d(x,y) = ||x - y||$$

Cosine Distance: Good for documents, images, etc.

$$d(x,y) = 1 - \frac{x \cdot y}{\|x\| \|y\|}$$

Jaccard Distance: For set data:

$$d(X,Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

Hamming Distance: For string data:

$$d(x,y) = \sum_{i=1}^{n} (x_i \neq y_i)$$

K-NN metrics



Manhattan Distance: Coordinate-wise distance

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

Edit Distance: for strings, especially genetic data.

 Mahalanobis Distance: Normalized by the sample covariance matrix – unaffected by coordinate transformations.

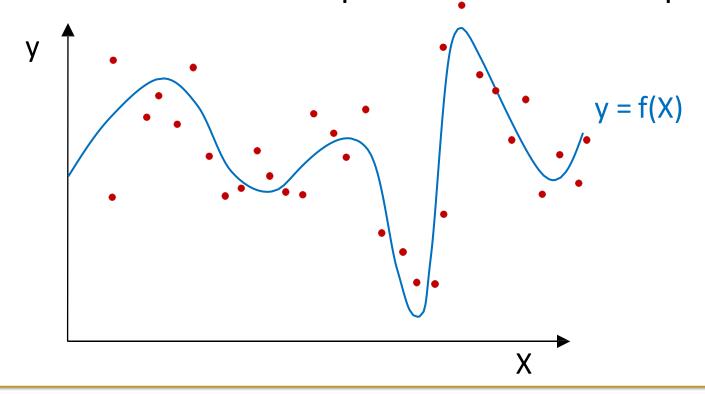


We have a bias/variance tradeoff:

- Small $k \rightarrow ?$
- Large k → ?

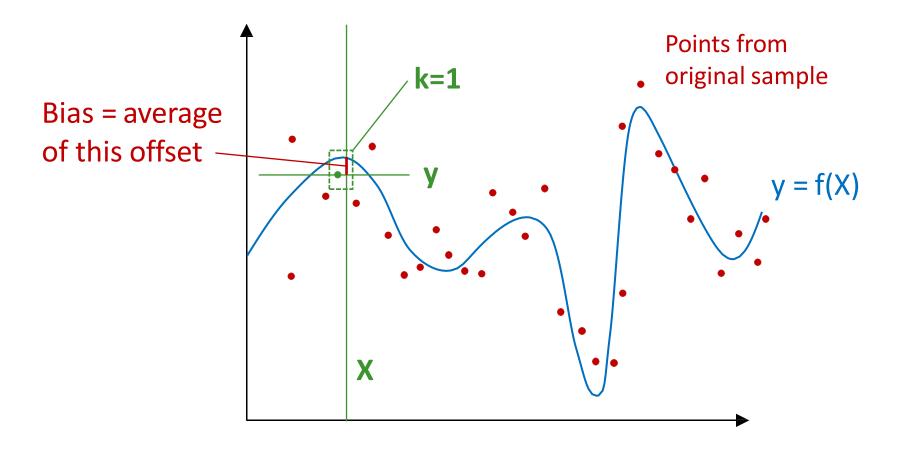


- Small k → low bias, high variance
- Large k → high bias, low variance
- Assume the real data follows the blue curve, with some mean-zero additive noise. Red points are a data sample.



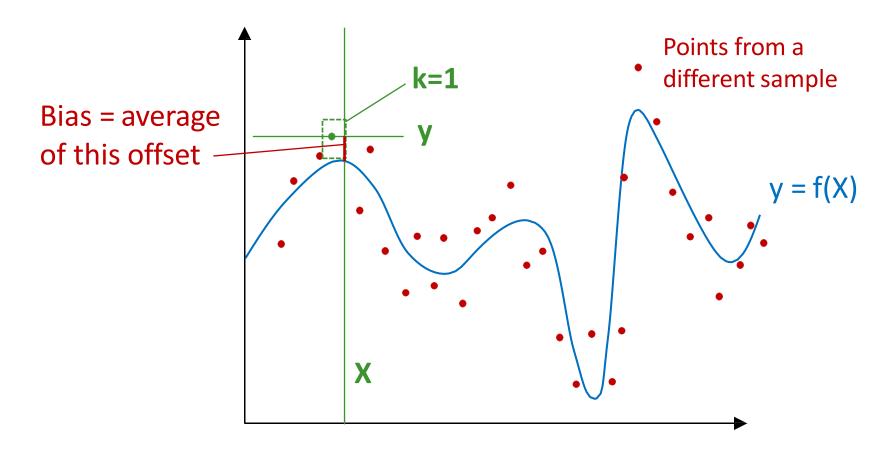


- Small k → low bias, high variance
- Large k → high bias, low variance



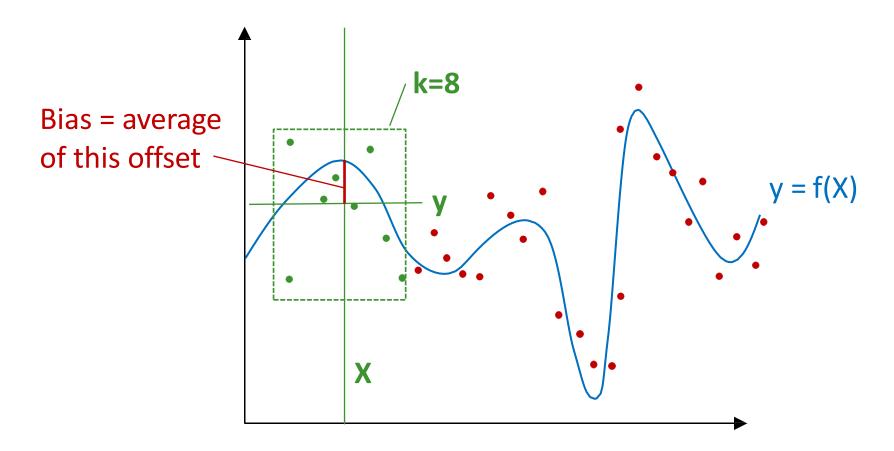


- Small k → low bias, high variance
- Large k → high bias, low variance



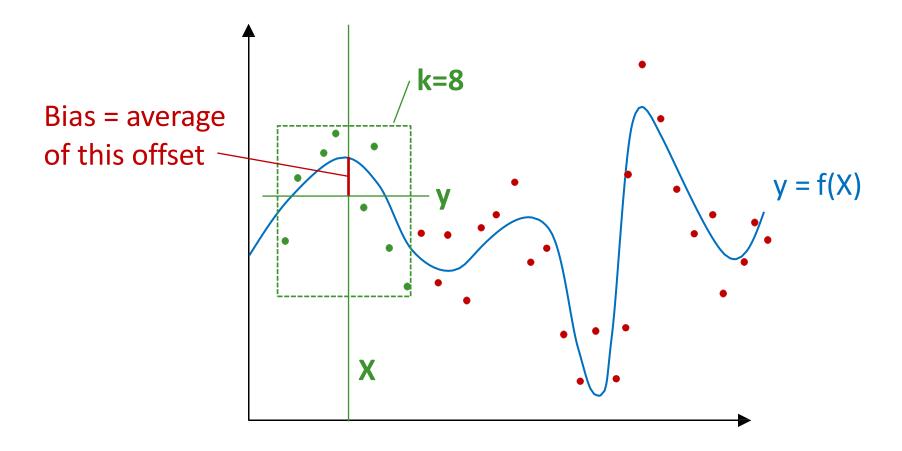


- Small k → low bias, high variance
- Large k → high bias, low variance





- Small k → low bias, high variance
- Large k → high bias, low variance



Choosing k in practice



Use cross-validation! Break data into train, validation and test subsets, e.g. 60-20-20 % random split.

Predict: For each point in the validation set, predict using the k-Nearest neighbors from the training set. Measure the error rate (classification) or squared error (regression).

Tune: try different values of k, and use the one that gives minimum error on the validation set.

Evaluate: test on the test set to measure performance.

kNN and the curse of dimensionality **GW**

The curse of dimensionality refers to phenomena that occur in high dimensions (100s to millions) that do not occur in low-dimensional (e.g. 3-dimensional) space.

In particular data in high dimensions are much sparser (less dense) than data in low dimensions.

For kNN, that means there are less points that are very close in feature space (very similar), to the point we want to predict.

kNN and the curse of dimensionality **GW**

Example: Consider a collection of uniformly random points in the unit cube. In one dimension, the average squared Euclidean distance between any two points is:

$$\int_0^1 \int_0^1 (x - y)^2 dx \, dy = \frac{1}{6}$$

In N dimensions, we add up the squared differences for all N coordinates (because the coordinates are independent in a uniform random cube), giving:

$$d^2 = E[||x - y||^2] = \frac{N}{6}$$

So the euclidean distance scales as \sqrt{N}

kNN and the curse of dimensionality **GW**

From this perspective, its surprising that kNN works at all in high dimensions.

Luckily real data are not like random points in a high-dimensional cube. Instead they live in dense clusters and near much lower-dimensional surfaces.

Finally, points can be very "similar" even if their euclidean distance is large. E.g. documents with the same few dominant words (with tfidf weighing) are likely to be on the same topic.

Outline



- Supervised Learning
 - Classification/Regression
 - Bias/Variance tradeoff
- K-NN
- Naïve Bayes
- Training Issues
 - Measuring model quality
 - Over-fitting
 - Cross-validation

Bayes' Theorem



P(A | B) = probability of A given that B is true.

•
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

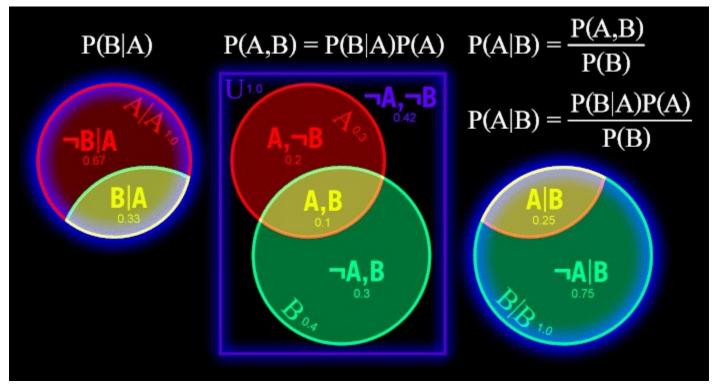
- In practice we are most interested in dealing with events e and data D
- e = "I have a cold"
- D = "runny nose," "watery eyes," "coughing"

•
$$P(e|D) = \frac{P(D|e)P(e)}{P(D)}$$

So Bayes' theorem is "diagnostic".

Bayes' Theorem





Assumes probability of event is proportional to area of set.

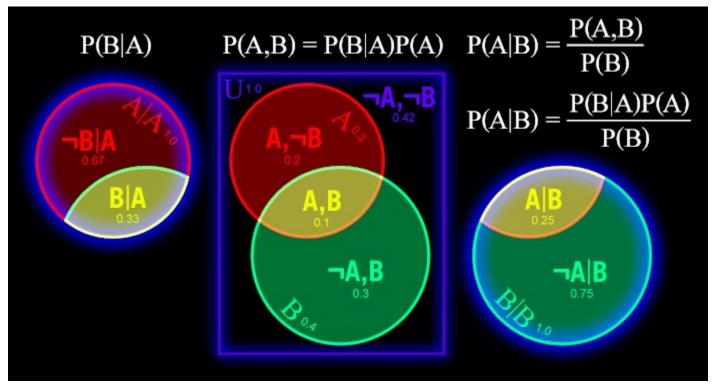
P(A) = area of A.

P(A,B) = area of A intersect B.

P(A|B) = P(A,B) / P(B)

Bayes' Theorem





The theorem follows by writing:

$$P(A|B)P(B) = P(A,B) = P(B|A)P(A)$$

From which we get:

$$P(A|B) = P(B|A) P(A) / P(B)$$

Bayes' Terminology



$$P(e|D) = \frac{P(D|e)P(e)}{P(D)}$$

- P(e) is called the **prior probability** of e. Its what we know (or think we know) about e with no other evidence.
- P(D|e) is the **conditional probability** of D given that e happened, or just the likelihood of D. This can often be measured or computed precisely it follows from your model assumptions.
- P(e|D) is the **posterior probability** of e given D. It's the answer we want, or the way we choose a best answer.
- You can see that the posterior is heavily colored by the prior, so Bayes' has a GIGO liability. e.g. its not used to test hypotheses

Naïve Bayes Classifier



Let's assume we have an instance (e.g. a document d) with a set of features $(X_1, ..., X_k)$ and a set of classes $\{c_j\}$ to which the document might belong.

We want to find the **most likely class** that the document belongs to, given its features.

The joint probability of the class and features is:

$$\Pr(X_1,\ldots,X_k,c_j)$$

Summary

- Supervised Learning
 - Classification/Regression
 - Bias/Variance tradeoff
- K-NN
- Naïve Bayes
- Training Issues
 - Measuring model quality
 - Over-fitting
 - Cross-validation

Model Quality

Almost every model optimizes some quality criterion:

- For linear regression it was the Residual Sum-of-Squares
- For k-Means it is the "Inertia" the mean squared distance from each sample to its cluster center.

•

The quality criterion is chosen often because of its good properties:

- Convexity: so that there is a unique, best solution
- Closed form for the optimum (linear regression) or at least for the gradient (for SGD).
- An algorithm that provably converges.

Model Quality

There are typically other criteria used to measure the quality of models. e.g. for clustering models:

- Silhouette score
- Inter-cluster similarity (e.g. mutual information)
- Intra-cluster entropy

For regression models:

- Stability of the model (sensitivity to small changes)
- Compactness (sparseness or many zero coefficients)

Evaluating Clusterings: Silhouette

The silhouette score is

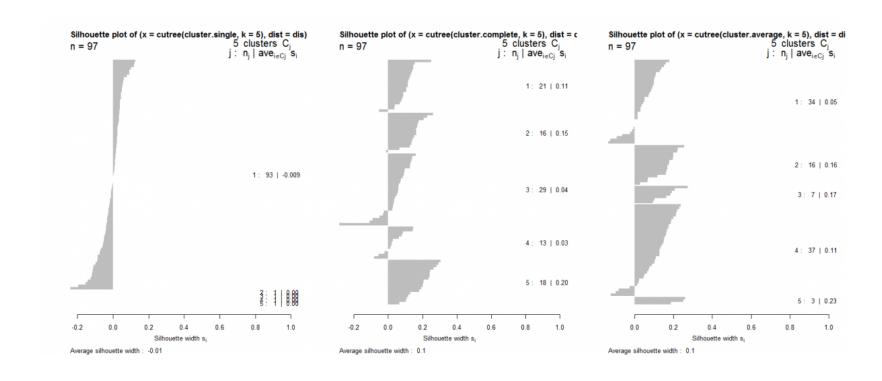
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where a(i) is the mean distance from sample i to its own cluster, b(i) the mean distance from i to the second-closest cluster.

 Perhaps surprisingly, silhouette scores can be, and often are, negative.

Evaluating Clusterings: Silhouette

Silhouette plot: horizontal bars with cluster score. Sort (vertically) first by cluster, then by score.



Regularization with Secondary Criteria

While secondary criteria can be measured after the model is built, its too late then to affect the model.

Using secondary criteria during the optimization process is called "regularization".

Examples:

- L1 regularization adds a term to the measure being optimized which is the sum of absolute value of model coefficients.
- L2 regularization adds a term to the measure being optimized which is the sum of squares of model coefficients.

Regularization with Secondary Criteria

L1 regularization in particular is very widely used. It has the following impacts:

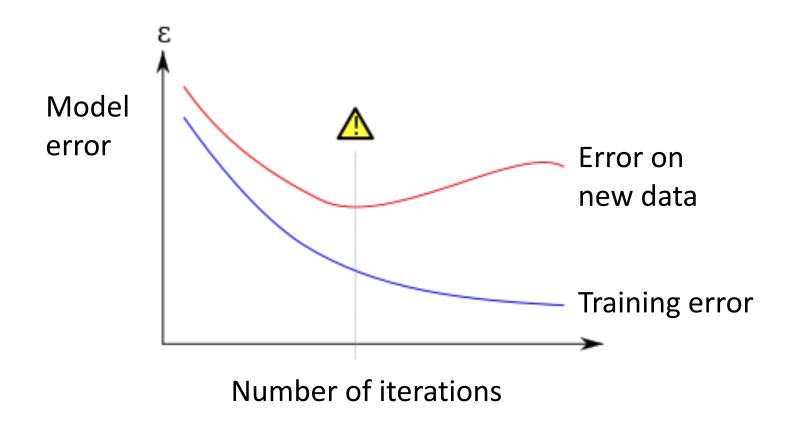
- Yields a convex optimization problem in many cases, so there is a unique solution.
- The solution is usually stable to small input changes.
- The solution is quite sparse (many zero coefficients) and requires less disk and memory to run.
- L1 regularization on factorization models tends to decrease the correlation between model factors.

Over-fitting

- Your model should ideally fit an infinite sample of the type of data you're interested in.
- In reality, you only have a finite set to train on. A good model for this subset is a good model for the infinite set, up to a point.
- Beyond that point, the model quality (measured on new data) starts to decrease.
- Beyond that point, the model is over-fitting the data.

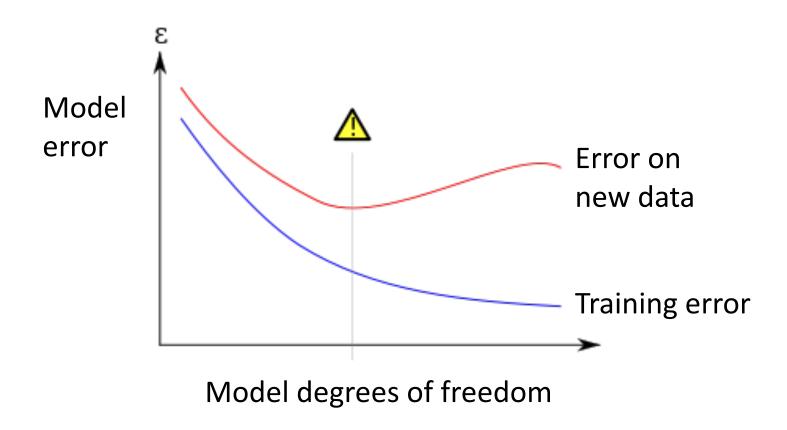
Over-fitting

Over-fitting during training



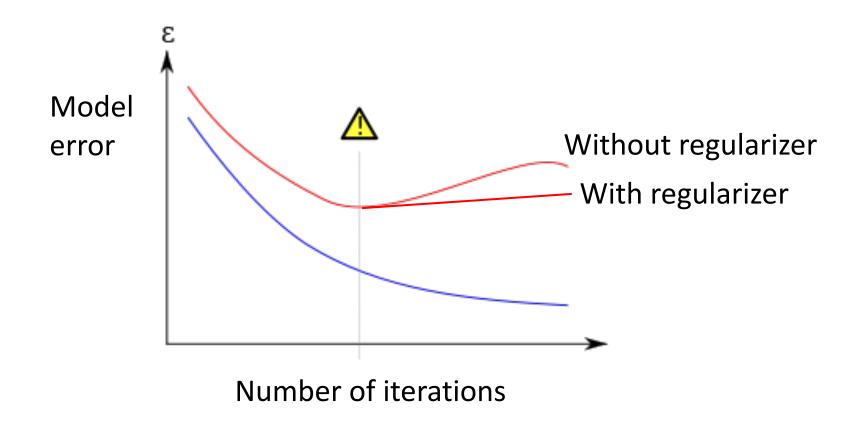
Over-fitting

Another kind of over-fitting



Regularization and Over-fitting

Adding a regularizer:



Cross-Validation

 Cross-validation involves partitioning your data into distinct training and test subsets.

The test set should never be used to train the model.

 The test set is then used to evaluate the model after training.

K-fold Cross-Validation

- To get more accurate estimates of performance you can do this k times.
- Break the data into k equal-sized subsets A_i
- For each i in 1,...,k do:
 - Train a model on all the other folds A₁,..., A_{i-1}, A_{i+1},..., A_k
 - Test the model on A_i
- Compute the average performance of the k runs

5-fold Cross-Validation

