GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 15: REGULARIZATION & ML REVIEW

LAST TIME:

- REGRESSION DIAGNOSTICS

QUESTIONS?

I. ML REVIEW
II. OVERFITTING AND REGULARIZATION
III. HANDLING OVERFITTING

EXERCISES:
IV. IMPLEMENTING REGULARIZED MODELS IN SCIKIT-LEARN

I. ML REVIEW

supervised	making predictions
unsupervised	extracting structure

	continuous	*categorical*
supervised	regression	classification
unsupervised	dimension reduction	clustering

Regression: Linear regression

Regression: Linear regression

Classification: k-NN, Naïve Bayes, Decision Trees, Logistic Regression

Regression: Linear regression

Classification: k-NN, Naïve Bayes, Decision Trees, Logistic Regression

Dimensionality Reduction: PCA

Regression: Linear regression

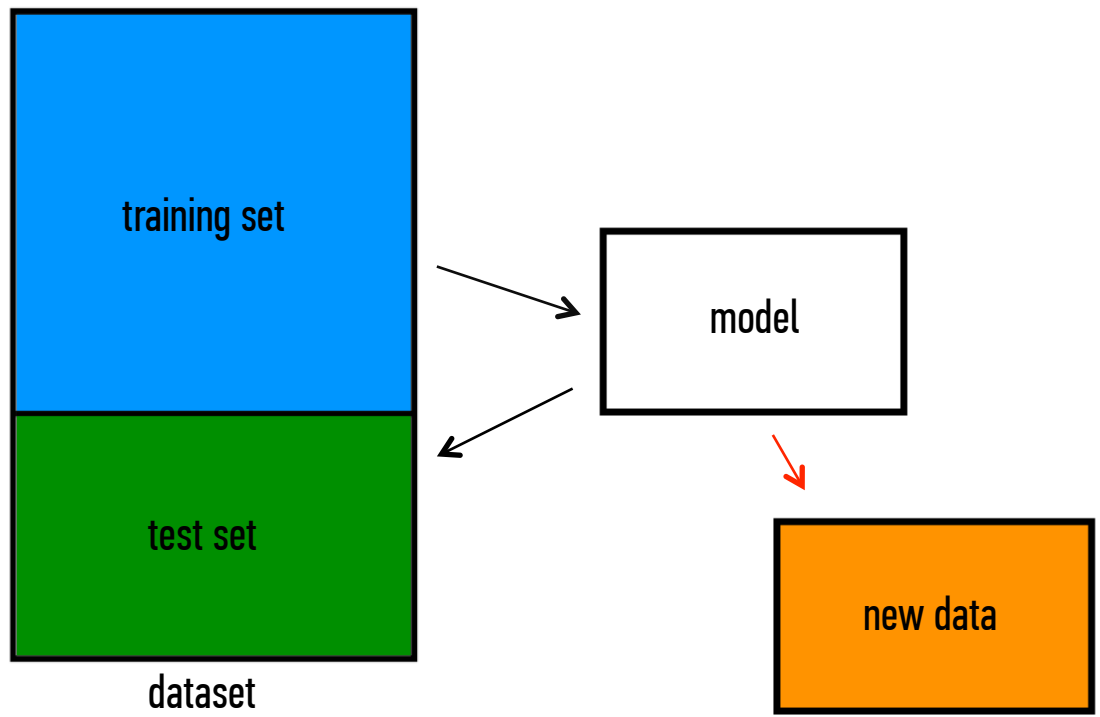
Classification: k-NN, Naïve Bayes, Decision Trees, Logistic Regression

Dimensionality Reduction: PCA

Clustering: k-Means

Regression: Linear regression

r^2, MAE, etc. This was the topic of the last lecture.



Classification: k-NN, Naïve Bayes, Decision Trees, Logistic Regression

Accuracy, Recall, F1

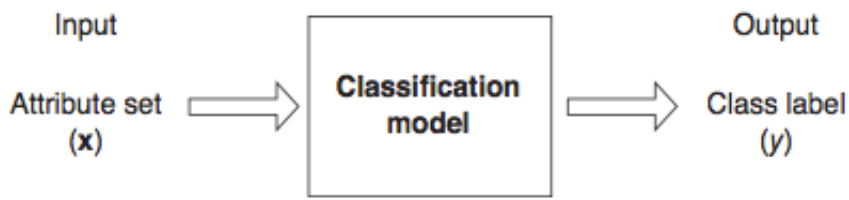


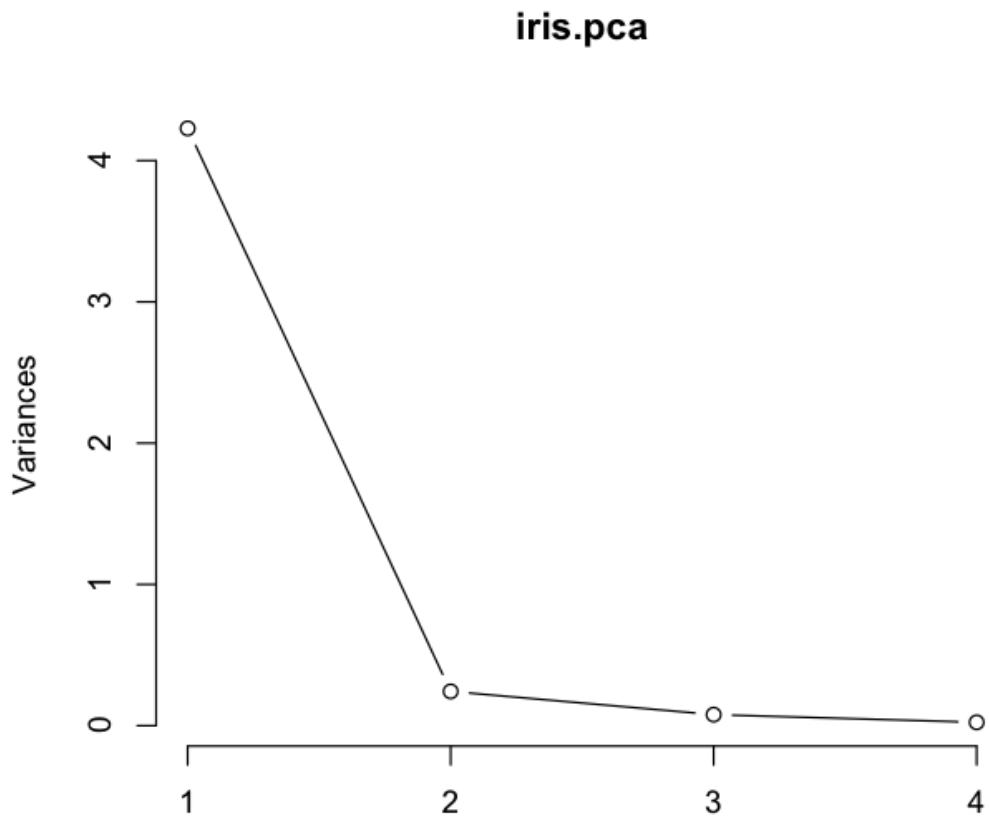
Figure 4.2. Classification as the task of mapping an input attribute set $\mathbf{x}$ into its class label y.

source: http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf

Dimensionality Reduction: PCA

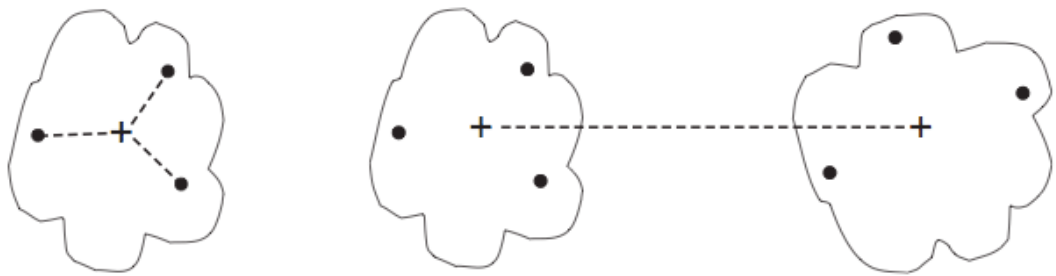
Scree Plot

Though PCA isn't actually a model whose performance you evaluate.



Clustering: k-Means

Silhouette Coefficient, SSE



(a) Cohesion. (b) Separation.

Figure 8.28. Prototype-based view of cluster cohesion and separation.

source: http://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf

Try the ones that match your data and the task you want to achieve.

Then get more data.

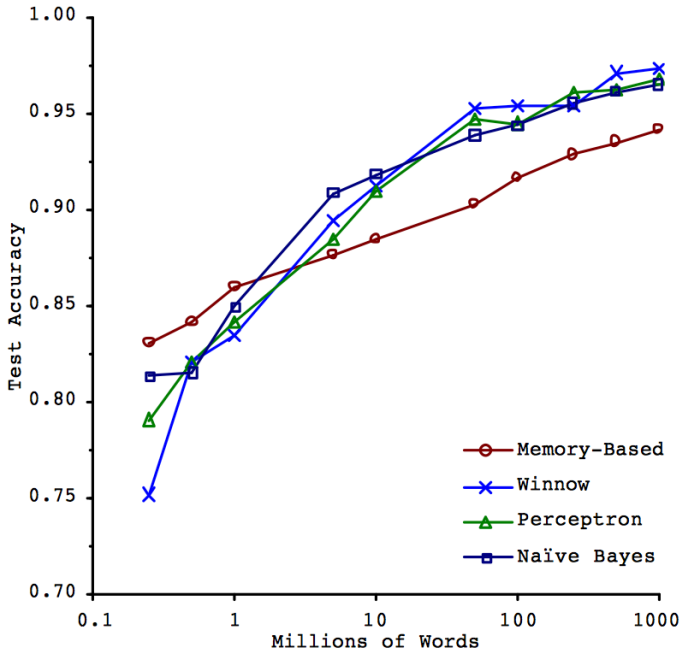


Figure 1. Learning Curves for Confusion Set Disambiguation

source: Banko, Brill, Microsoft Research (2001) http://research.microsoft.com/pubs/66840/acl2001.pdf

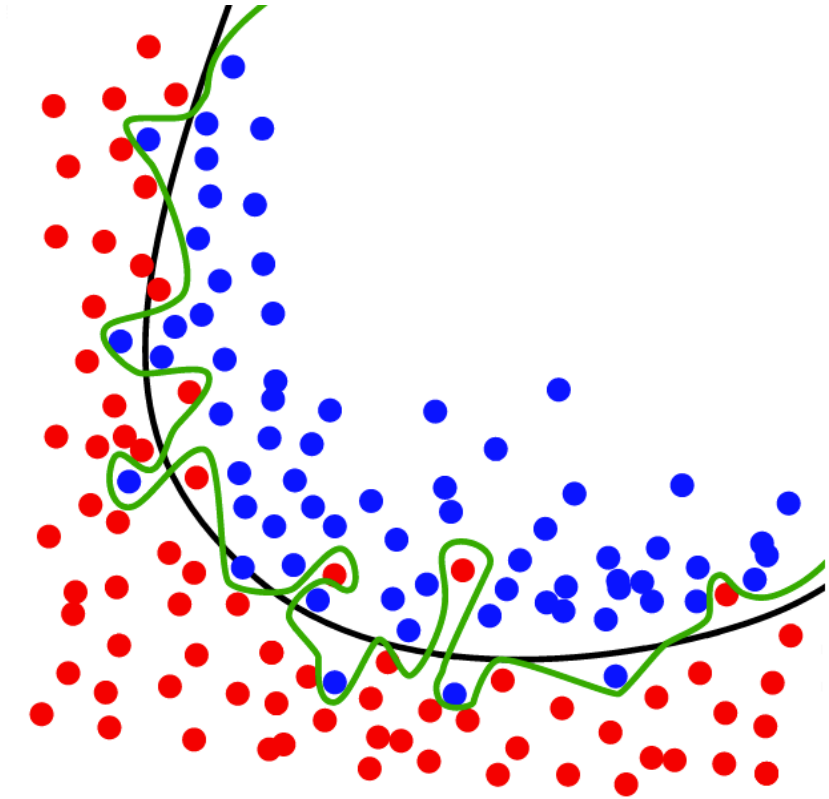
II. OVERFITTING AND REGULARIZATION

We've mentioned overfitting several times.

Overfitting occurs when a model is complex enough to start fitting noise in the training data.

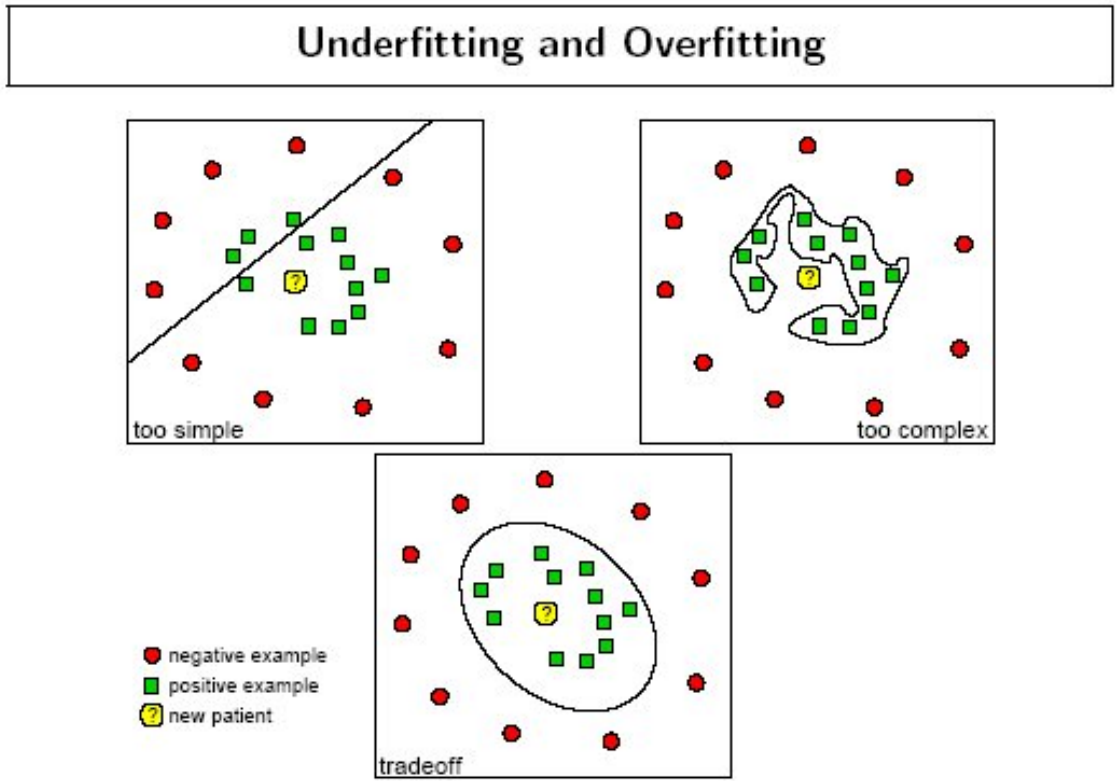
This renders the model increasingly irrelevant to other data.

OVERFITTING – EXAMPLE



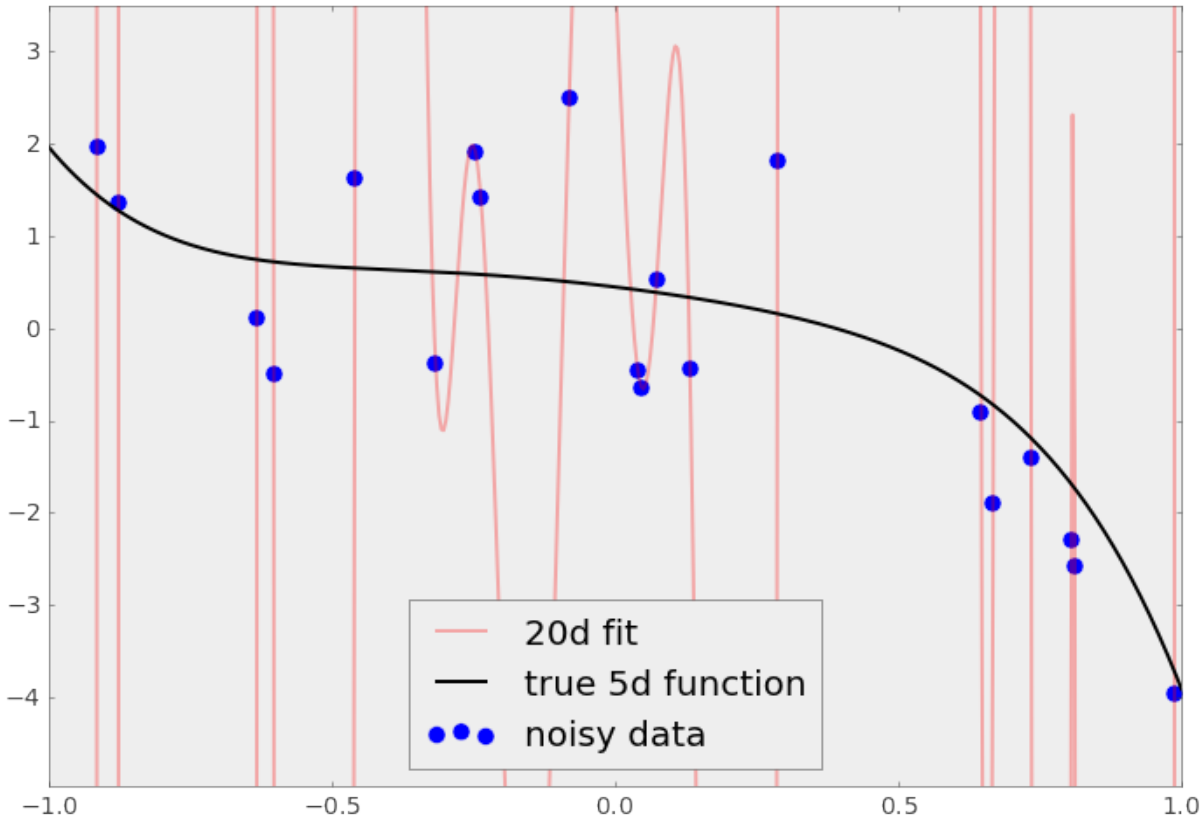
source: http://www.dtreg.com

OVERFITTING – EXAMPLE



source: http://www.dtreg.com

OVERFITTING



You can tell that your model is overfitting if its performance on training data is much better than its performance on test data.

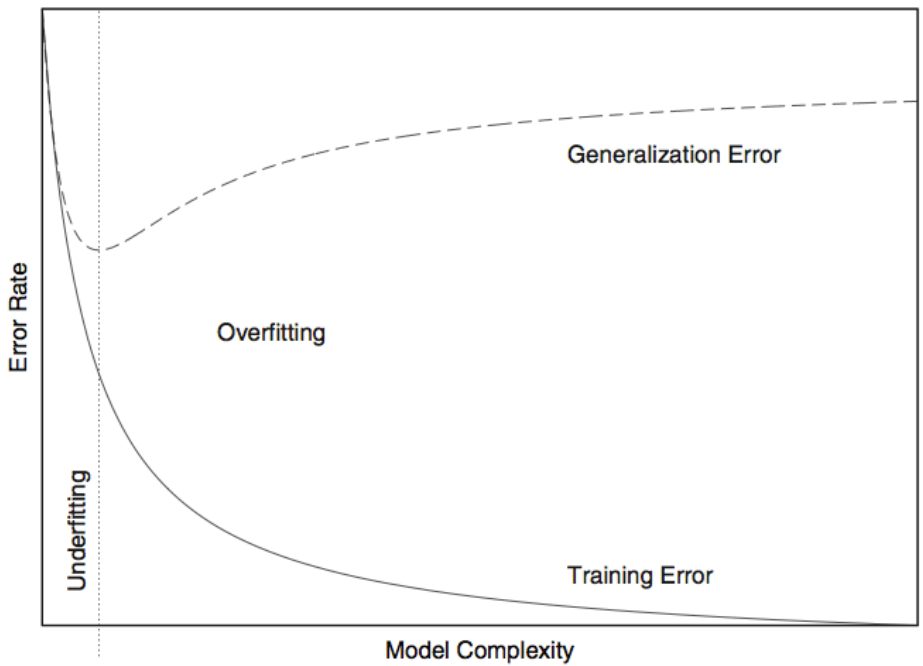
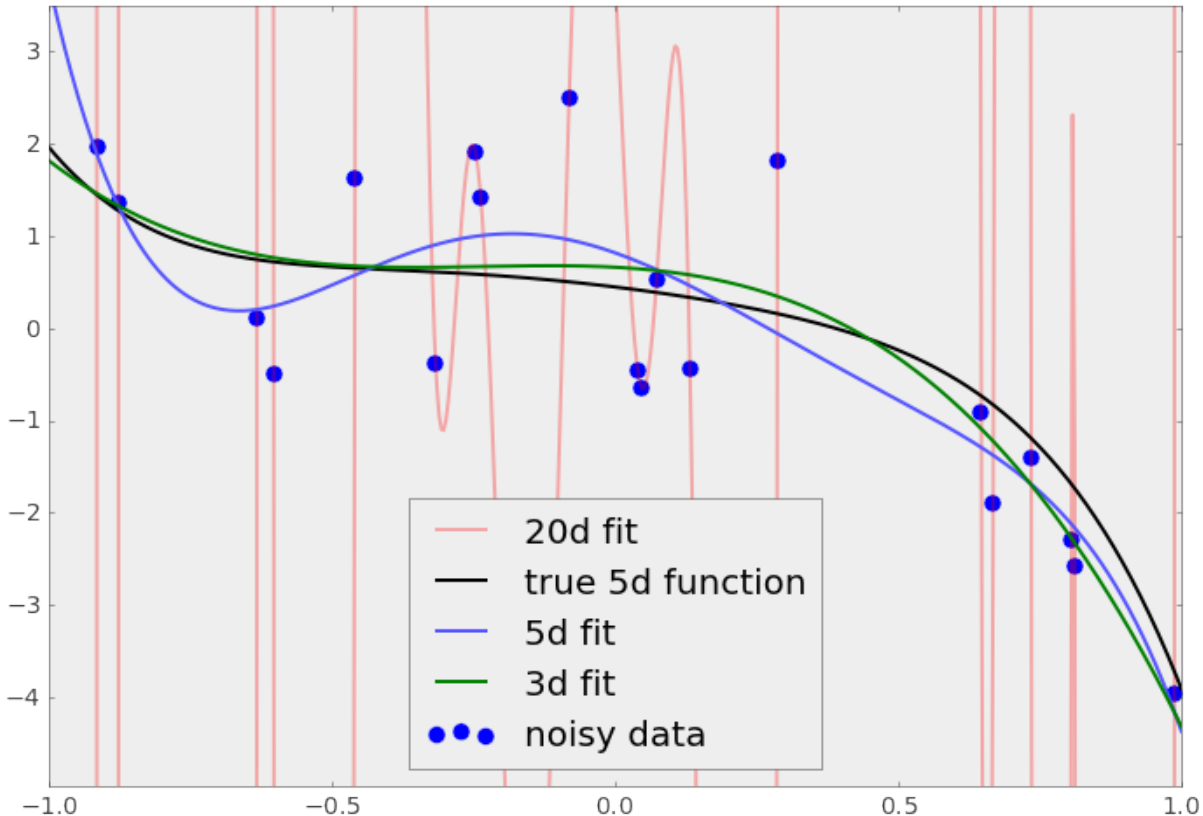


FIGURE 18-1. Overfitting: as a model becomes more complex, it becomes increasingly able to represent the training *data. However, such a model is overfitted and will not generalize well to data that was not used during training.*

source: Data Analysis with Open Source Tools, by Philipp K. Janert. O'Reilly Media, 2011.

You can tell that your model is overfitting if its performance on training data is much better than its performance on test data.

Even if the dimension of your model matches that of the underlying system, you may still overfit.


3
2
1
0
−1
−2
−3
−4
−1.0
−0.5
0.0
0.5
1.0
20d fit
true 5d function
5d fit
3d fit
noisy data

The solution to overfitting may simply be to adjust the complexity of the model until you maximize out-of-sample performance.

In certain models, you can still use lots of features, but add a parameter to the cost function that penalizes complexity.

This is called *regularization*.

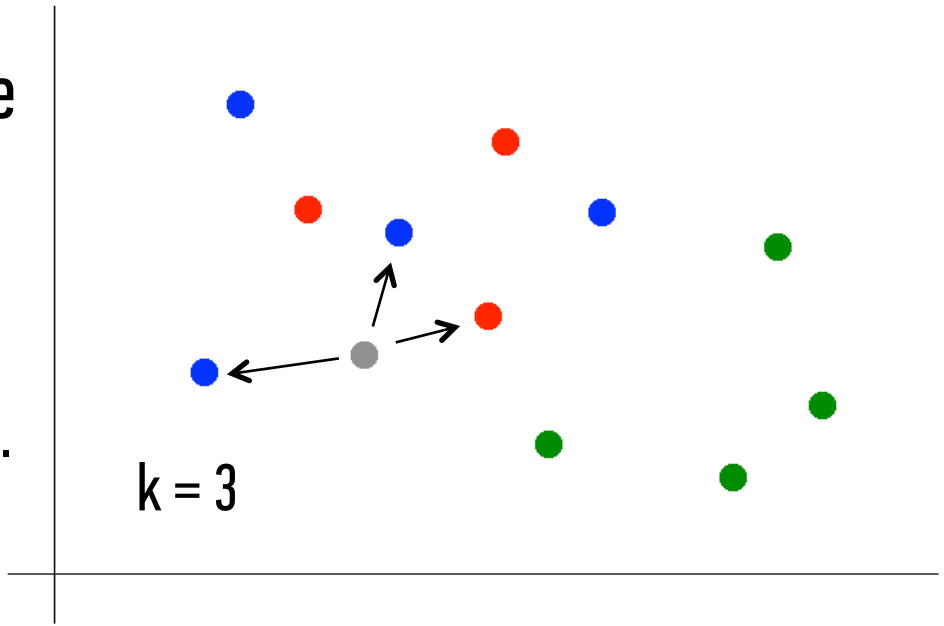
III. HANDLING OVERFITTING

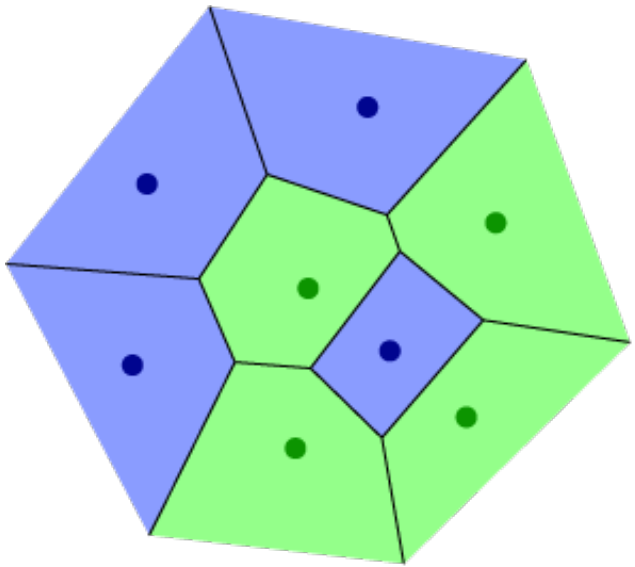
A. K-NEAREST NEIGHBORS

The Algorithm:

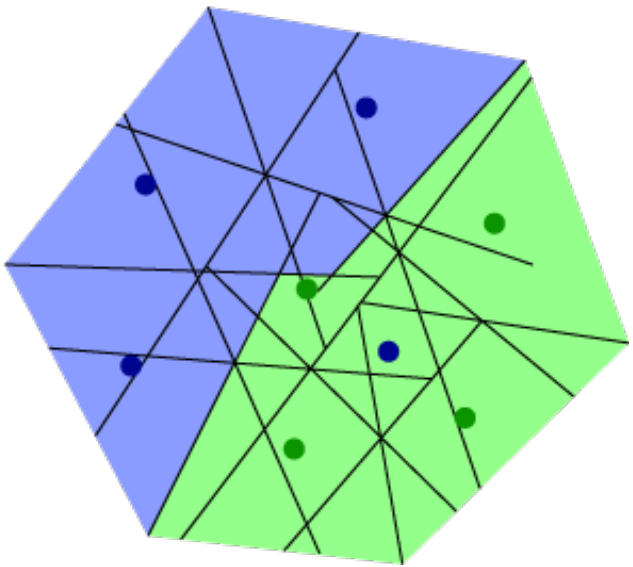
Suppose we want to predict the color of the grey dot.

1) Pick a value for k.
2) Find colors of k nearest neighbors.
3) Assign the most common color to the grey dot.





k = 1 (Overfit)



k = 3

source: http://shapeofdata.wordpress.com/2013/05/07/k-nearest-neighbors/

To prevent overfitting, just choose the right value of k!

B. LINEAR REGRESSION

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n + \varepsilon$$

y = **response variable** (the one we want to predict)

x_i = **input variables** (the ones we use to train the model)

β_0 = **intercept** (where the line crosses the y-axis)

β_i = **regression coefficients** (the model "parameters")

ε = **residual** (the prediction error)

We minimize the errors by minimizing the error (cost function) for all of our training data:

$$min\ \varepsilon = min(y - \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)$$

Or in matrix form:

$$min(||\mathbf{y} - \mathbf{x}\boldsymbol{\beta}||^2)$$

The values of the coefficients, β, are solved using the Normal Equations:

$$\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

In order to prevent overfitting, we can penalize large coefficients in our cost function.

$$min(||\mathbf{y} - \boldsymbol{x}\boldsymbol{\beta}||^2 + \lambda||\boldsymbol{\beta}||^2)$$

In order to prevent overfitting, we can penalize large coefficients in our cost function.

$$min(||\mathbf{y} - \mathbf{x}\boldsymbol{\beta}||^2 + \lambda||\boldsymbol{\beta}||^2)$$

NOTE

The new term is proportional to the sum of the squared values of the coefficients.

In order to prevent overfitting, we can penalize large coefficients in our cost function.

$$min(||\mathbf{y} - \mathbf{x}\beta||^2 + \lambda||\beta||^2)$$

This results in the *regularized* Normal Equations:

$$\beta = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

(This is called Ridge Regression and $||\beta||^2$ is called an L2-norm.)

Q: What are bias and variance?

Q: What are bias and variance?

A: Bias refers to predictions that are *systematically inaccurate*.

Q: What are bias and variance?

A: Bias refers to predictions that are *systematically inaccurate*. Variance refers to predictions that are *generally imprecise*.

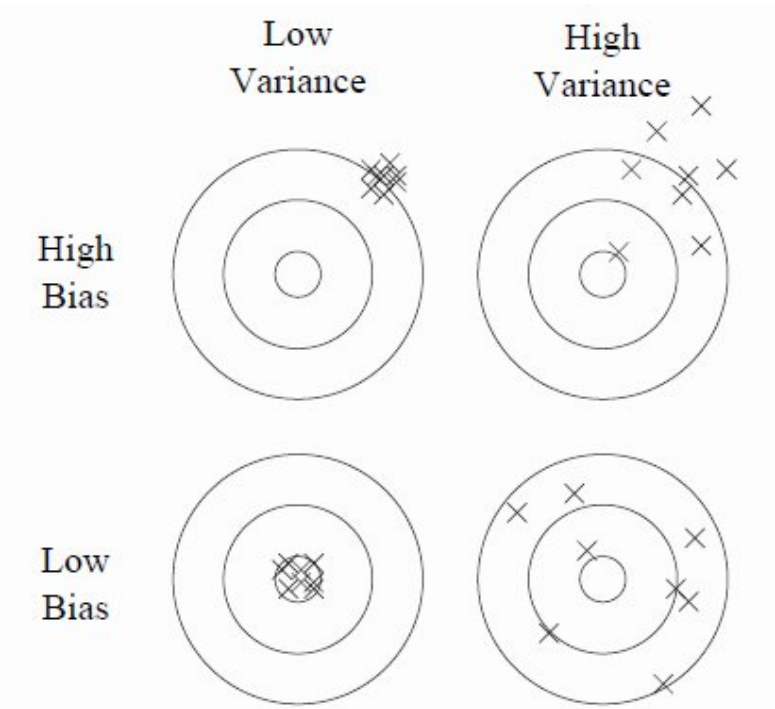


Figure 1: Bias and variance in dart-throwing.

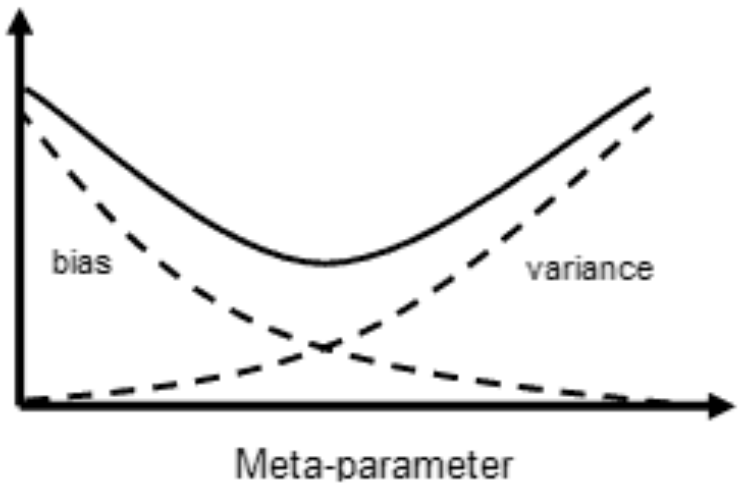
source: http://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

Q: What are bias and variance?

A: Bias refers to predictions that are *systematically inaccurate*. Variance refers to predictions that are *generally imprecise*.

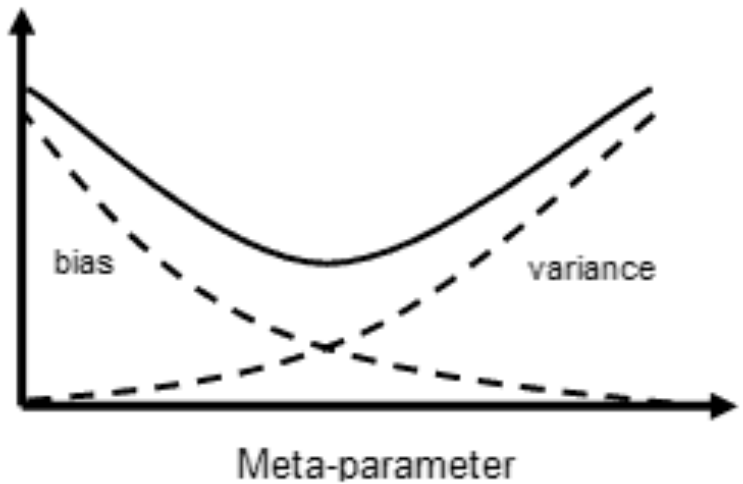
It turns out (after some math) that the generalization error in our model can be decomposed into a bias component and variance component.

This is another example of the **bias-variance tradeoff**.



source: http://www.isu.edu/chem/images/kalivasmeta.gif

This is another example of the **bias-variance tradeoff**.



NOTE

The “meta-parameter” here is the lambda we saw above.

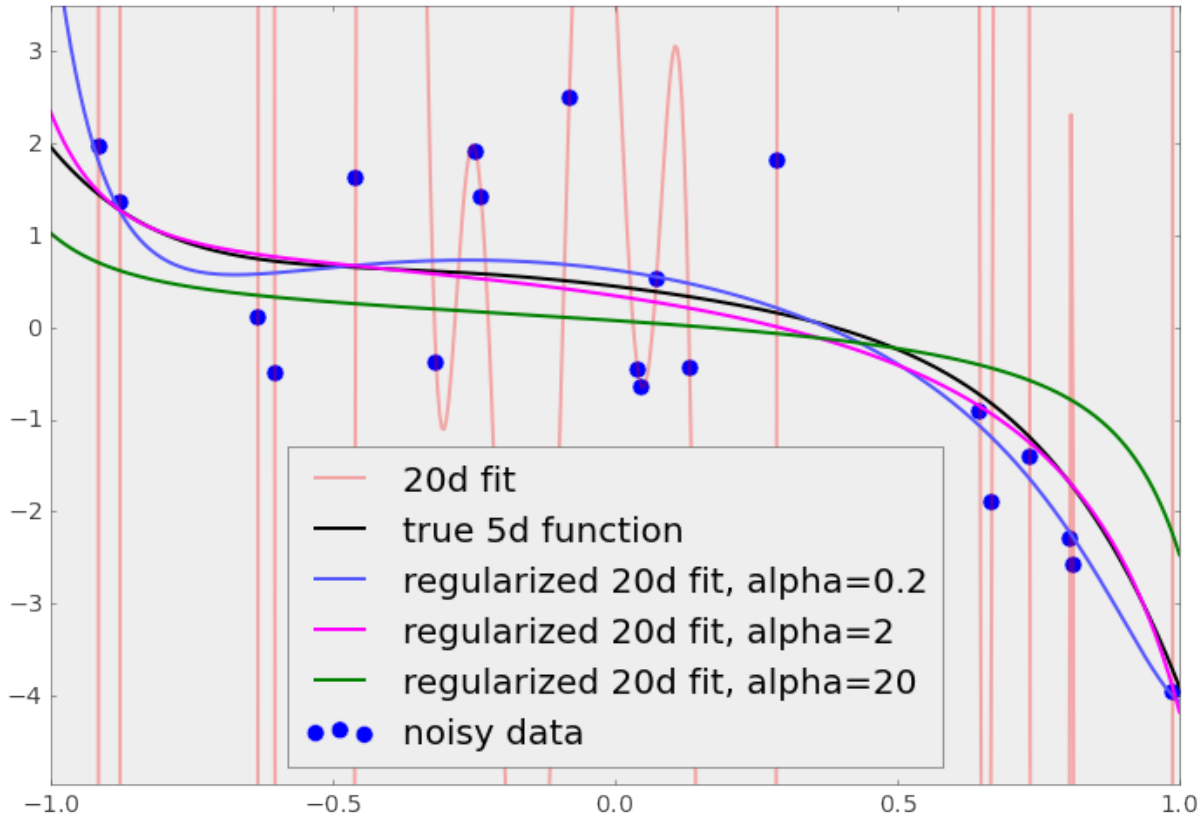
A more typical term is “hyperparameter”.

source: http://www.isu.edu/chem/images/kalivasmeta.gif

This tradeoff is regulated by a **hyperparameter** λ, which we've already seen:

$$\beta = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

So regularization represents a method to trade away some variance for a little bias in our model, thus achieving a better overall fit.


20d fit
true 5d function
regularized 20d fit, alpha=0.2
regularized 20d fit, alpha=2
regularized 20d fit, alpha=20
noisy data
3
2
1
0
−1
−2
−3
−4
−1.0
−0.5
0.0
0.5
1.0

C. NAÏVE BAYES

Suppose we have a dataset with features $x_1, \ldots, x_n$ and a class label C.

$$P(\text{class } C \mid \{x_i\}) = \frac{P(\{x_i\} \mid \text{class } C) \cdot P(\text{class } C)}{P(\{x_i\})}$$

Bayes' theorem can help us to determine the probability of a record belonging to a class, *given* the data we observe.

source: Data Analysis with Open Source Tools, by Philipp K. Janert. O'Reilly Media, 2011.

From Bayes' Theorem and assuming conditional independence, we came to this formula for classifying a record with features, {**x**}:

$$C_{MAP} = \underset{C}{\operatorname{argmax}} P(\{x\})|C) \cdot P(C) = \underset{C}{\operatorname{argmax}} P(x_1|C) \cdot P(x_2|C) \cdot P(x_3|C) \cdot \cdots \cdot P(x_n|C) \cdot P(C)$$

From Bayes' Theorem and assuming conditional independence, we came to this formula for classifying a record with features, {**x**}:

$$C_{MAP} = \underset{C}{\operatorname{argmax}} P(\{x\})|C) \cdot P(C) = \underset{C}{\operatorname{argmax}} P(x_1|C) \cdot P(x_2|C) \cdot P(x_3|C) \cdot \cdots \cdot P(x_n|C) \cdot P(C)$$

What happens if feature x_k was never seen in the training data?

From Bayes' Theorem and assuming conditional independence, we came to this formula for classifying a record with features, {**x**}:

$$C_{MAP} = \underset{C}{\operatorname{argmax}} P(\{x\})|C) \cdot P(C) = \underset{C}{\operatorname{argmax}} P(x_1|C) \cdot P(x_2|C) \cdot P(x_3|C) \cdot \cdots \cdot P(x_n|C) \cdot P(C)$$

What happens if feature x_k was never seen in the training data?

Then the likelihood $P(x_k|C) = 0$, which zeroes everything out. Overfitting!

For Naïve Bayes, one strategy to avoid this kind of overfitting is additive smoothing, a.k.a. Laplace smoothing.

Recall that the feature likelihoods are:

$$P(x_k|C) = N_{k,C} / N_C$$

With additive smoothing:

$$P'(x_k|C) = (N_{k,C} + \alpha) / (N_C + \alpha * n)$$

where n is the number of model features and α is a hyperparameter.

This decreases the individual likelihoods, but allows us to assign a probability to new features outside the training set.

This decreases the individual likelihoods, but allows us to assign a probability to new features outside the training set.

$$P(x_{new} \mid C) = \alpha / (N_C + \alpha * n)$$

D. DECISION TREES

Q: What is a decision tree?

A: A non-parametric hierarchical classification technique.

non-parametric: no parameters, no distribution assumptions

hierarchical: consists of a sequence of questions which yield a class label when applied to any record

Q: How is a decision tree represented?
A: Using a configuration of **nodes** and **edges**.

More concretely, as a *multiway tree*, which is a type of (directed acyclic) **graph**.

In a decision tree, the nodes represent questions (**test conditions**) and the edges are the answers to these questions.

The basic method used to build (or "grow") a decision tree is **Hunt's algorithm**.

This is a **greedy recursive** algorithm that leads to a **local optimum**.

greedy – algorithm makes locally optimal decision at each step
recursive – splits task into subtasks, solves each the same way
local optimum – solution for a given neighborhood of points

We can make splits that maximize the **gain** at each step:

$$\Delta = I(\text{parent}) - \sum_{\text{children } j} \frac{N_j}{N} I(\text{child } j)$$

(Here I is the impurity measure, N_j denotes the number of records at child node j, and N denotes the number of records at the parent node.)

When I is the entropy, this quantity is called the **information gain**.

Generally speaking, a test condition with a high number of outcomes can lead to **overfitting** (ex: a split with one outcome per record).

One way of dealing with this is to restrict the algorithm to binary splits only (CART).

Another way is to use a splitting criterion which explicitly penalizes the number of outcomes (C4.5)

We can use a function of the information gain called the **gain ratio** to explicitly penalize high numbers of outcomes:

$$\text{gain ratio} = \frac{\Delta_{info}}{-\sum p(v_i) log_2 p(v_i)}$$

(Where $p(v_i)$ refers to the probability of label *i* at node *v*)

Another type of **overfitting** involves splitting into too many leaf nodes that are each too specific to generalize well.

One strategy to avoid this is **pre-pruning**, which involves setting a minimum threshold on the gain, and stopping when no split achieves a gain above this threshold.

This prevents overfitting, but is difficult to calibrate in practice (may preserve bias!)

Alternatively we could build the full tree, and then perform **pruning** as a post-processing step.

Complicated subtrees can be replaced either with a single node, or with a simpler (child) subtree.

The first approach is called **subtree replacement**, and the second is **subtree raising**.

Finally, we can train a legion of small, weak decision trees on small subsets of the features and aggregate their classifications.

We lose the wonderful transparency of decision trees regarding how the classifications are made, but gain hugely in performance.

This is what random forests or boosted trees do.

E. LOGISTIC REGRESSION

In linear regression, we used a set of covariates to predict the value of a (continuous) outcome variable.

In logistic regression, we use a set of covariates to predict *probabilities* of (binary) class membership.

These probabilities are then mapped to *class labels*, thus solving the classification problem.

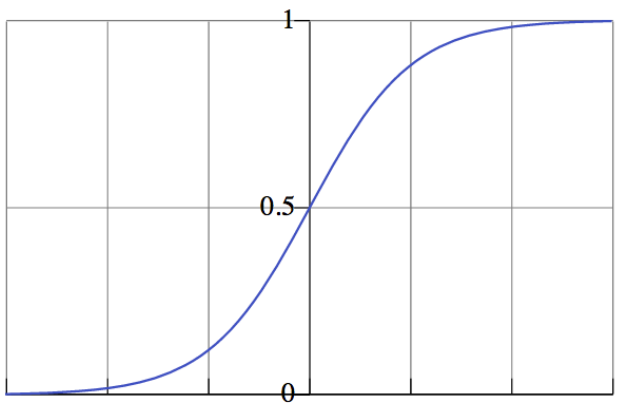
The logistic regression model is an *extension* of the linear regression model, with a couple of important differences.

The first difference is in the outcome variable.

The second difference is in the error term.

logistic function:

$$E(y|x) = \pi(x) = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$



The **logit function** is an important transformation of the logistic function. Notice that it returns the linear model!

$$g(x) = ln(\frac{\pi(x)}{1-\pi(x)}) = \alpha + \beta x$$

The logit function is also called the **log-odds function**.

Regularizing logistic regression is therefore very similar to regularizing linear regression.

Logistic regression is usually solved via iterative methods beyond the scope of the course, so it's difficult to describe the regularization in theoretic detail.

See the lab for practical details.

EX: IMPLEMENTING REGULARIZED MODELS IN SCIKIT-LEARN