Computer Science

CSC380: Principles of Data Science

Basics of Predictive Modeling and Classification

Prof. Jason Pacheco

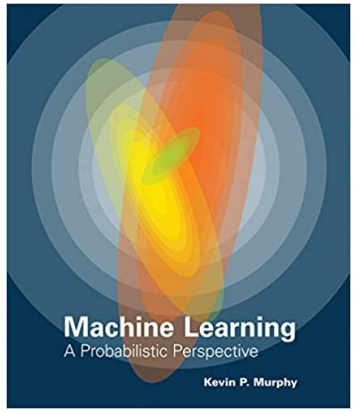
TA: Enfa Rose George

TA: Saiful Islam Salim

Administrative Items

- **No homework assignments this week!**
- Some updates to course webpage (more this weekend)
- Grade updates next week
- Using 2012 edition of Kevin Murphy
  - I have seen a 2021 preprint but sections won't match up with readings

Book for Rest of Course



Murphy, K. "Machine Learning: A Probabilistic Perspective." MIT press, 2012

(UA Library)

Review From Last Week

- **Supervised Learning** - Training data consist of inputs and outputs
  - Classification, regression, translation,

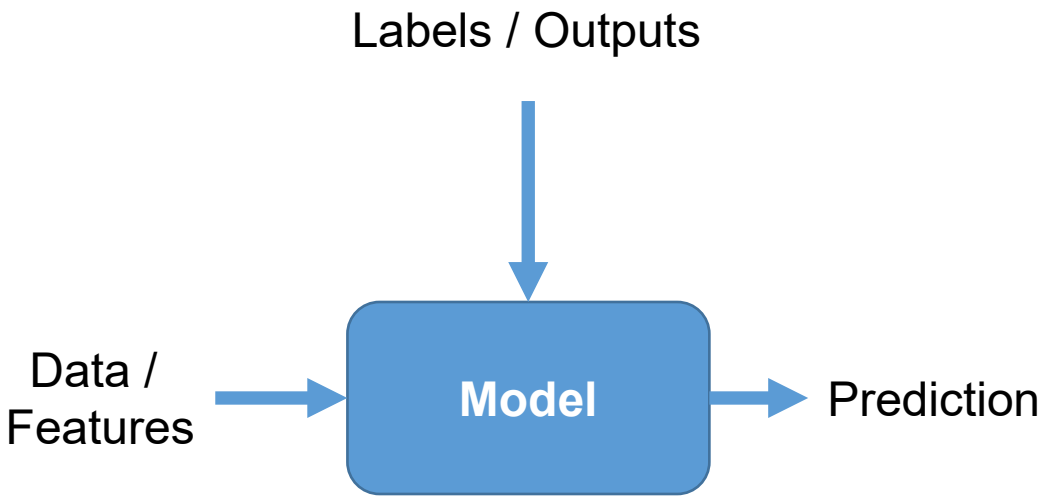
- **Unsupervised Learning** – Training data only contain inputs
  - Clustering, dimensionality reduction, segmentation, …

- **Linear** models generate output as a *linear combination* of inputs,
  - E.g. $y = w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$
  - PCA, linear regression, etc.

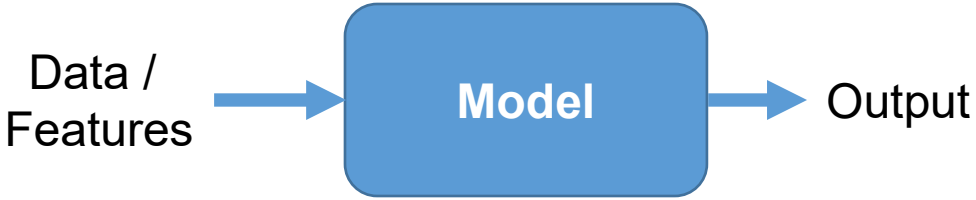
- **Nonlinear** models fit an arbitrary nonlinear function to map inputs-outputs
  - Neural networks, support vector machine, nonlinear dimensionality reduction

Training Machine Learning Models

Supervised Learning



Unsupervised Learning



ML models distinguished by a number of factors

- Number of parameters needed (parametric / nonparametric)
- Whether they model uncertainty (probabilistic / nonprababilistci)
- Do they model the data generation process? (generative / discriminative)

Parametric vs Nonparametric

A **parametric** model has a *fixed* number of parameters, regardless of the amount of data

Example We model data $x \sim \mathcal{N}(\mu, \sigma^2)$ as being Normally distributed with parameters (μ, σ).

Example Data are i.i.d. from some probability model,

$$x \sim p(x; \theta)$$

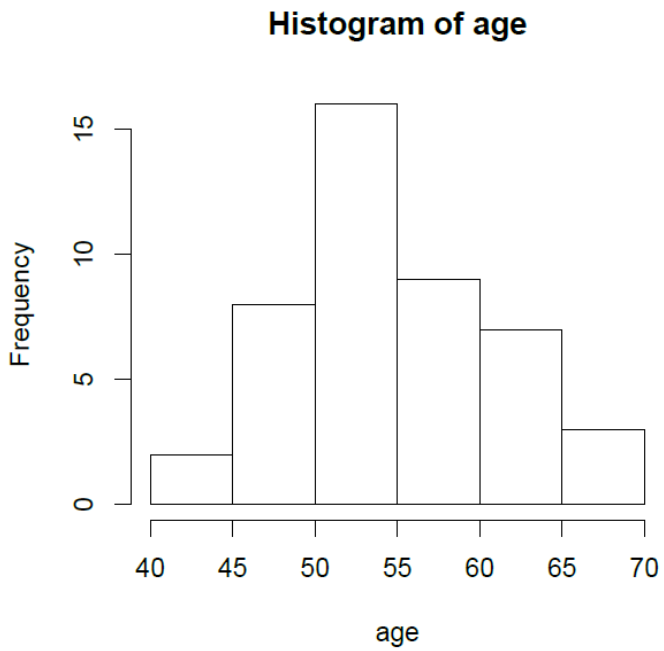
With parameters θ that can be fit (e.g. by MLE)

Advantage *Parametric models are general easy to use, easy to specify, and easy to fit to data…*

Parametric vs Nonparametric

A **nonparametric** model either has an infinite number of parameters, or parameters grow with the amount of data

Example A histogram of iid data $x_1, \ldots, x_n$ estimates the empirical distribution of the data.

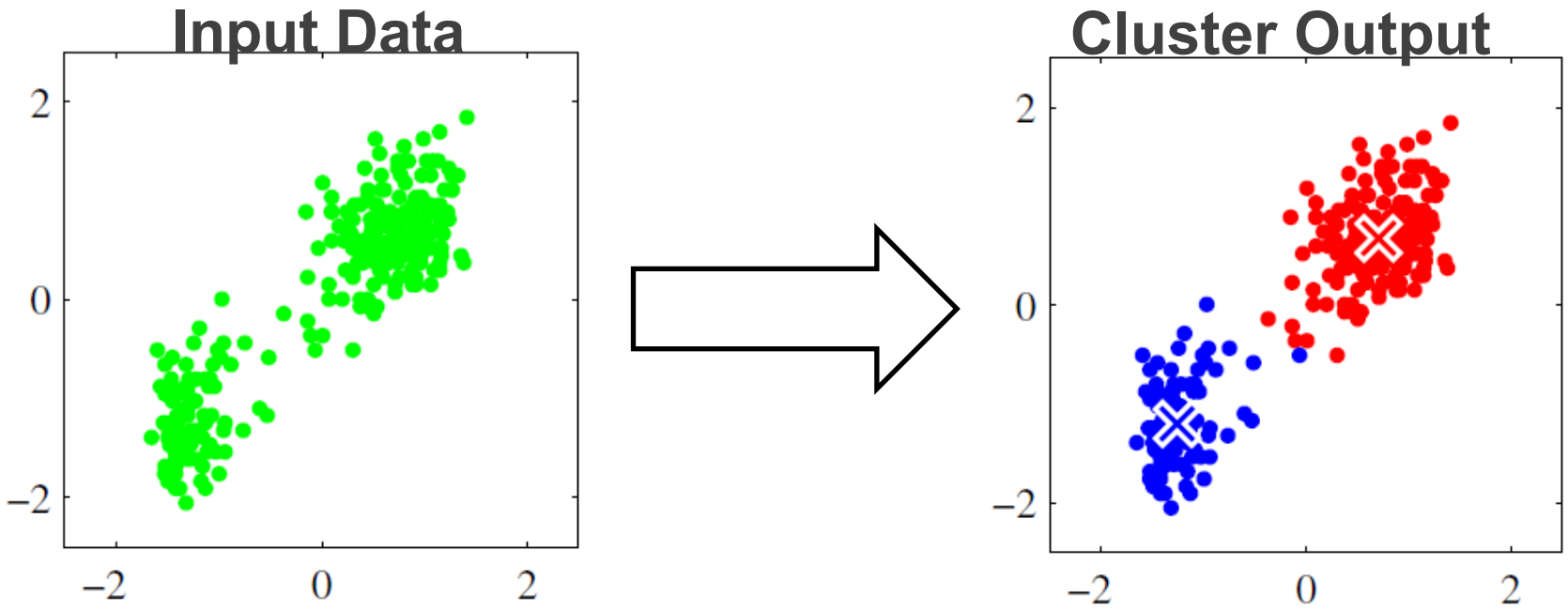


Nonparametric does not mean no parameters!

Advantage Nonparametric models are flexible—can often fit distributions with arbitrary precision (given enough data)

Probabilistic vs Non-Probabilistic

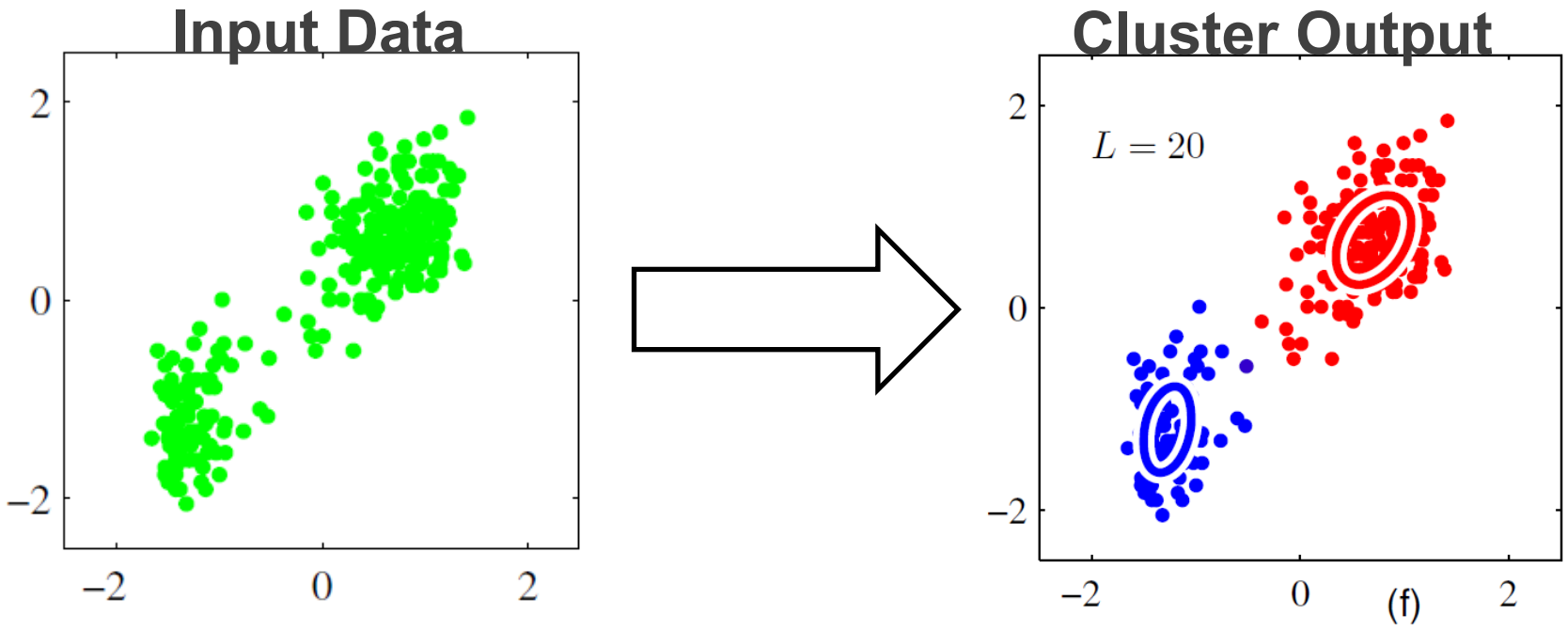
A **non-probabilistic** generates deterministic outputs / predictions from data



Example K-means clustering learns a *hard assignment* of data points to clusters. Assignments are not random.

Probabilistic vs Non-Probabilistic

A **probabilistic** model represents predictions as random variables, with a distribution that is fit to training data



Example A *Gaussian mixture model* models clusters as Normal distributions. Assignments of data to clusters are random variables.

Generative vs Disciminative

Let **X** represent the observed data / features and **Y** the model output / prediction

A **generative** model is a probabilistic model of *both* the data and the output via a *joint probability distribution* P(X,Y)

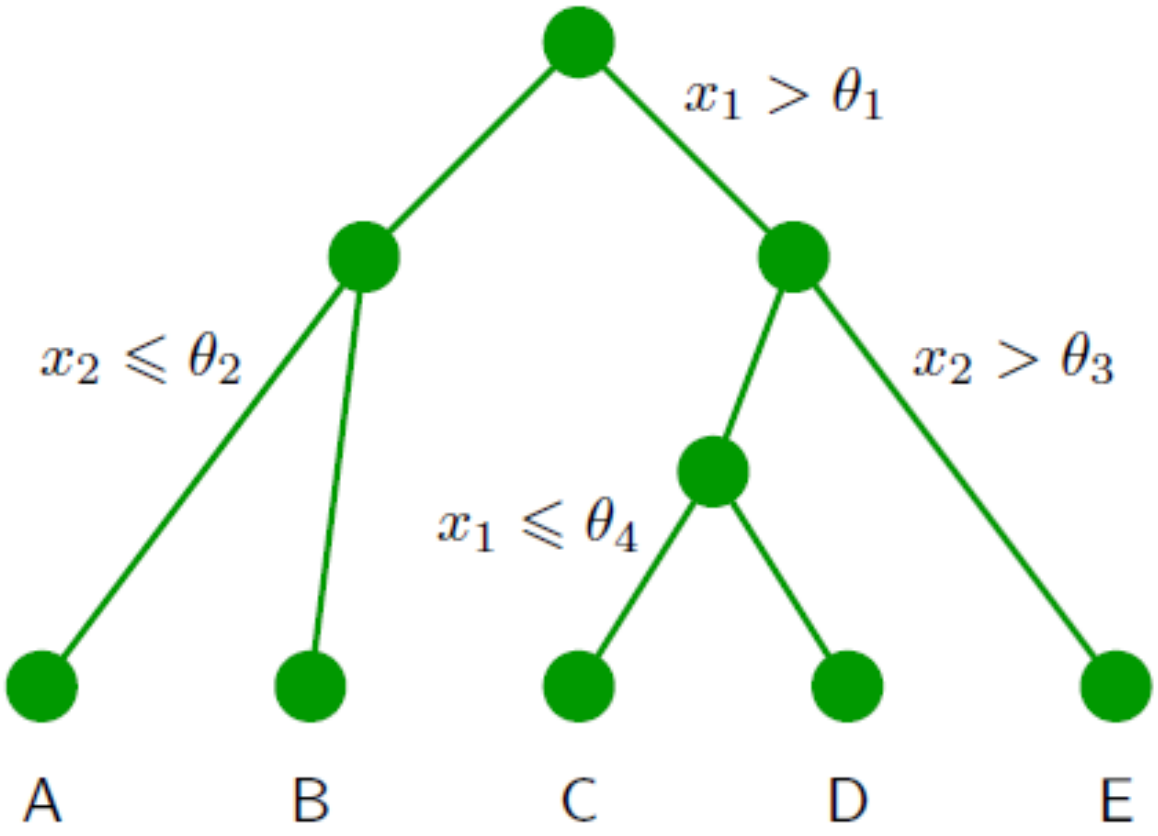
A **discriminative** model only models the output *conditioned* on observed data, e.g. the *conditional probability* P(Y|X)

Non-probabilistic models are generally all considered <u>discriminative</u> (e.g. a Neural Network) since they map inputs-to-outputs via deterministic function,

$$Y = f(X)$$

Decision Tree Classifier

Learn a set of decision boundaries to classify input



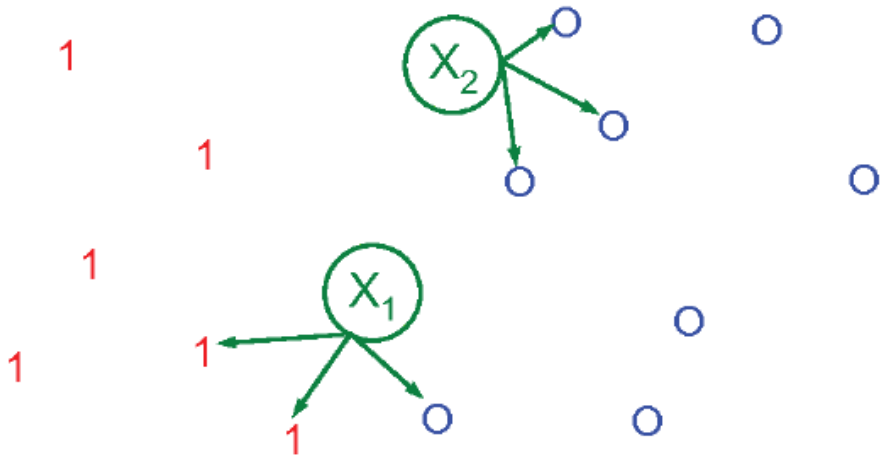
Example Learn thresholds $\theta_1, \theta_2, \ldots$ to classify **x** into categories A, B, C, D, E

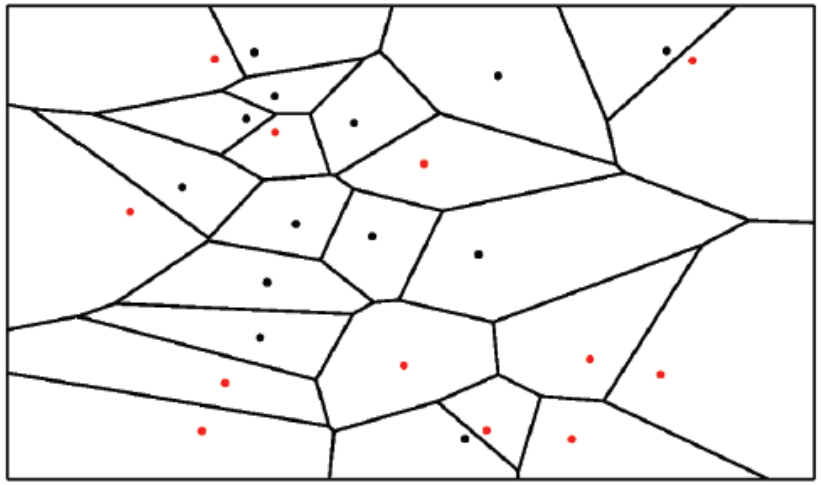
Is a decision tree classifier…

- Probabilistic?
- Generative / Discriminative?
- Parametric / Nonparametric?

K-Nearest Neighbor Classifier

For a test point **x** the KNN classifier “looks at” the classes of the K closest neighbors and predicts based on commonality of classes in neighboring points





Partitions the data space according to a Voronoi diagram…

Source: Duda et al. 2001

K-Nearest Neighbor Classifier

Given Training data consisting of n pairs $(x_1, y_1), \ldots, (x_N, y_N)$ of data x and class labels $y = \{\mathcal{C}_1, \ldots, \mathcal{C}_L\}$.

Model Suppose N_ℓ is number of points in class $\mathcal{C}_\ell$ and $\sum_\ell N_\ell = N$ and the *volume* of K neighbors is V with K_l points from class l then for a new test point x,

Likelihood

$$p(x \mid \mathcal{C}_\ell) = \frac{K_\ell}{N_\ell V}$$

Prior

$$p(\mathcal{C}_\ell) = \frac{N_\ell}{N}$$

Intuition Probability of class Ck proportional to number of neighbors in that class

Classify based on the *posterior* distribution,

$$p(\mathcal{C}_\ell \mid x) = \frac{p(x|\mathcal{C}_\ell)p(\mathcal{C}_\ell)}{p(x)} = \frac{K_\ell}{K}$$

where

$$p(\mathbf{x}) = \frac{K}{NV}$$

K-Nearest Neighbor Classifier

To minimize misclassification error, classify point based on maximum posterior probability…

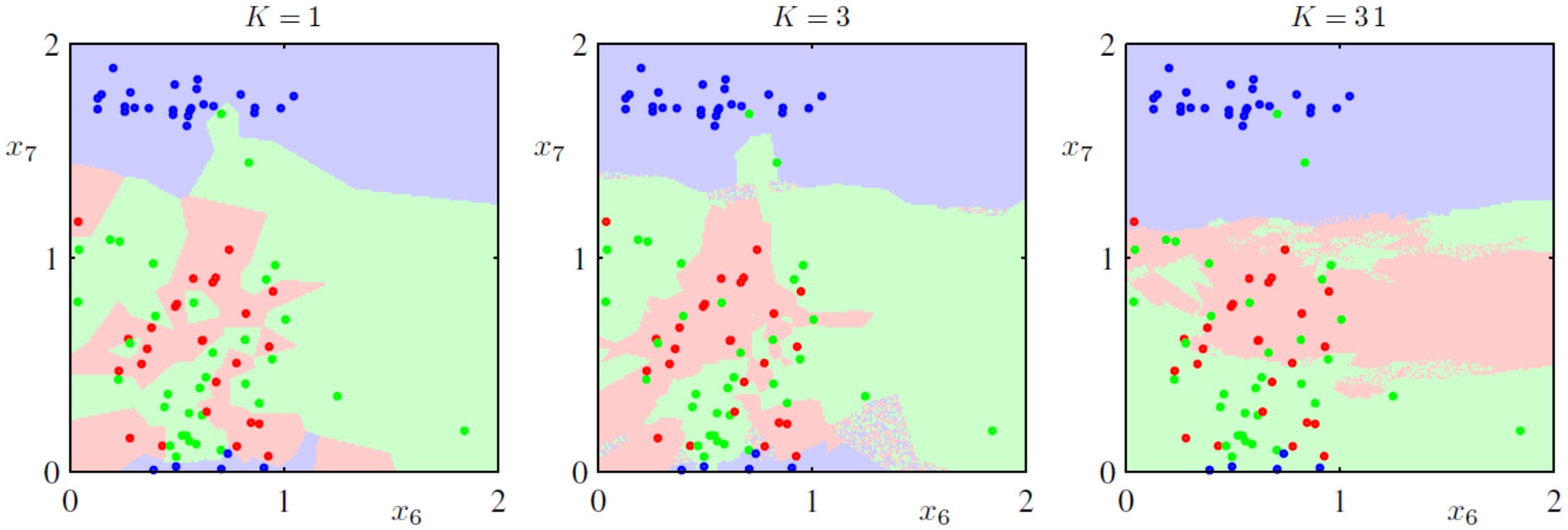
$$y = \arg\max_\ell \, p(\mathcal{C}_\ell \mid x) = \frac{K_\ell}{K}$$

How should we handle ties?

- Is KNN Discriminative / Generative?
- Is KNN Parametric / Non-parametric?
- Linear / Nonlinear?
- What is the likelihood distribution $p(x \mid C)$?
- What parameter(s) control the likelihood?
- What are any other design parameters that determine the KNN?
- What are storage requirements for this model?

K-Nearest Neighbor Classifier

The number of neighbors K controls how the space of data are partitioned into class distributions…

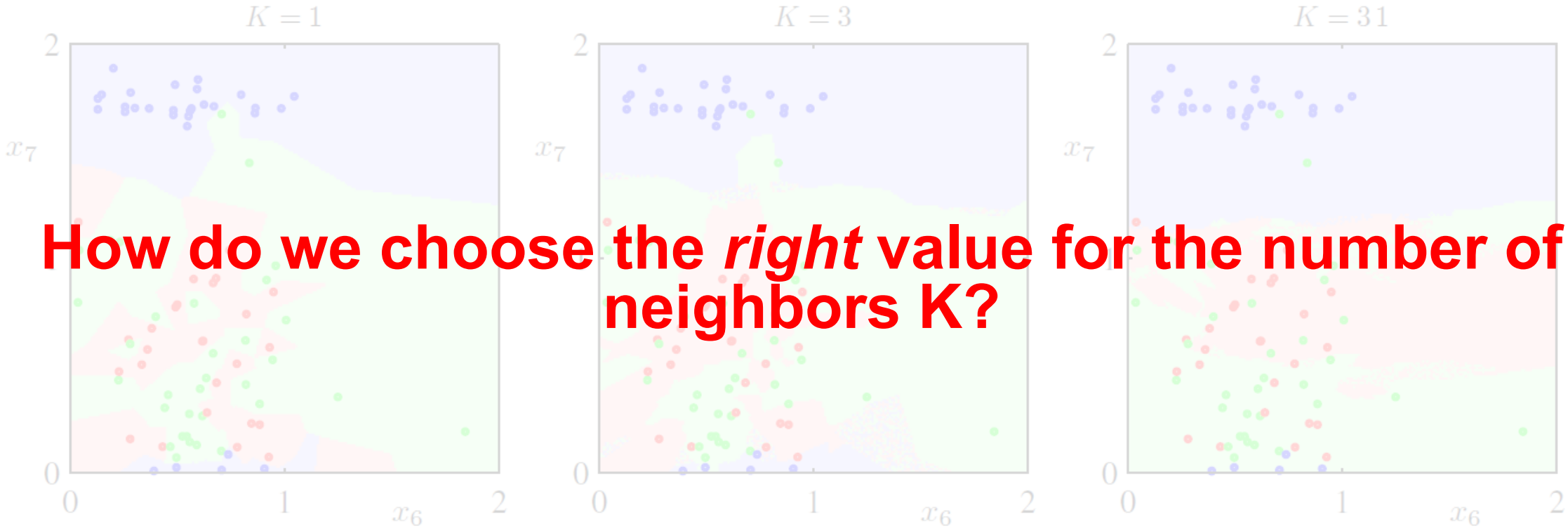


Small K break the space into many small regions (overfitting) large K tend to over-smooth the region (underfitting)

Source: Bishop, C.

K-Nearest Neighbor Classifier

The number of neighbors K controls how the space of data are partitioned into class distributions…



How do we choose the *right* value for the number of neighbors K?

Small K break the space into many small regions (overfitting) large K tend to over-smooth the region (underfitting)

Source: Bishop, C.

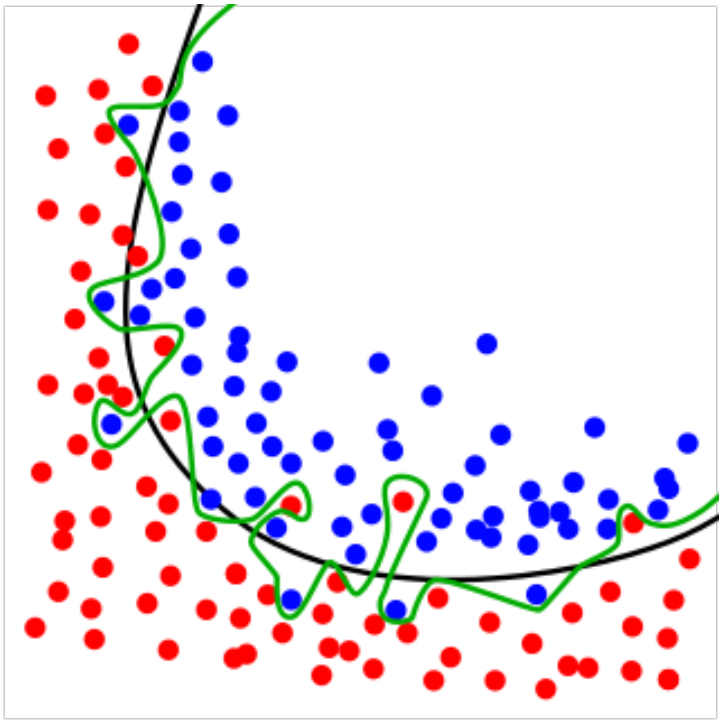
The Challenge: How to Learn a Function

Okay, we have a training data. Why not learn the most complex function that can work flawlessly for the training data and be done with it? (i.e., classifies every data point correctly)

Extreme Let's memorize the data. To predict an unseen data, just follow the label of the closest memorized data.

Doesn't generalize to unseen data – called *overfitting* the training data.

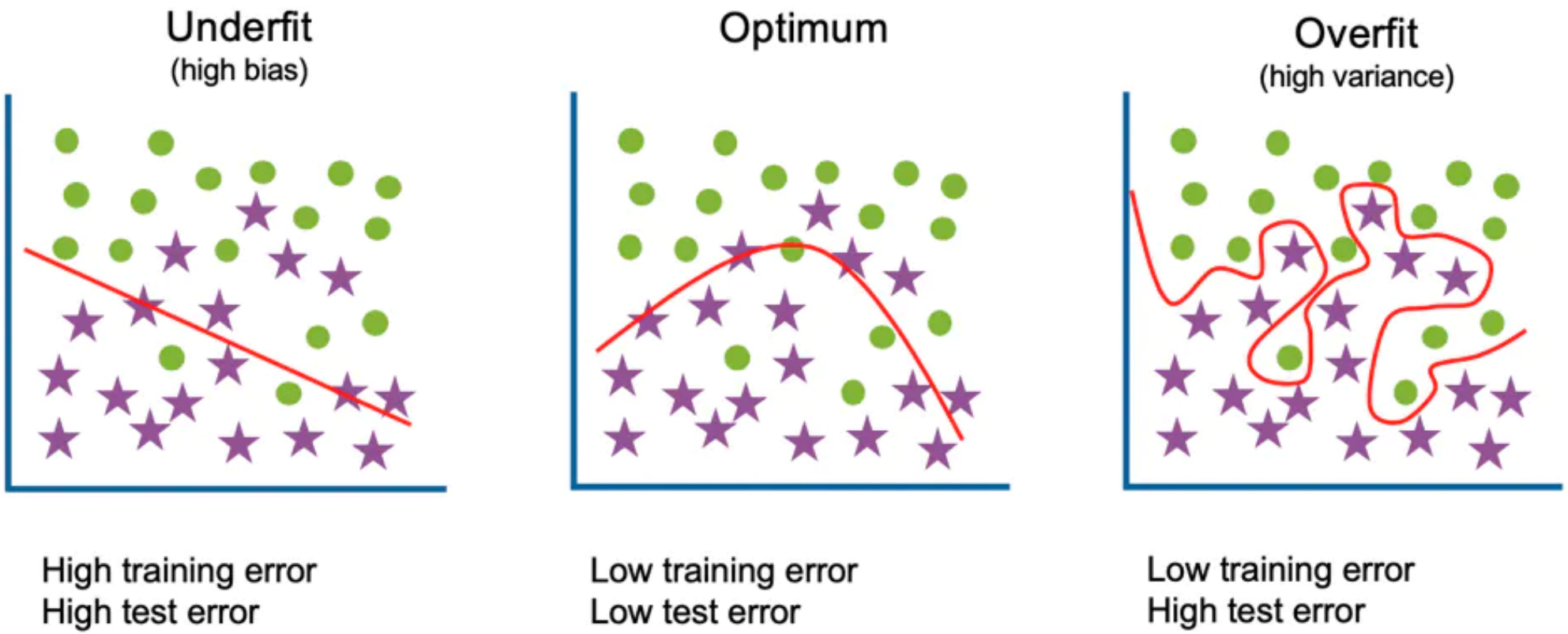
Solution Learn training dataset but don't "over-do" it. This is called "regularization".



green: memorization
black: true decision boundary

Source: Kwang Sung-Jun

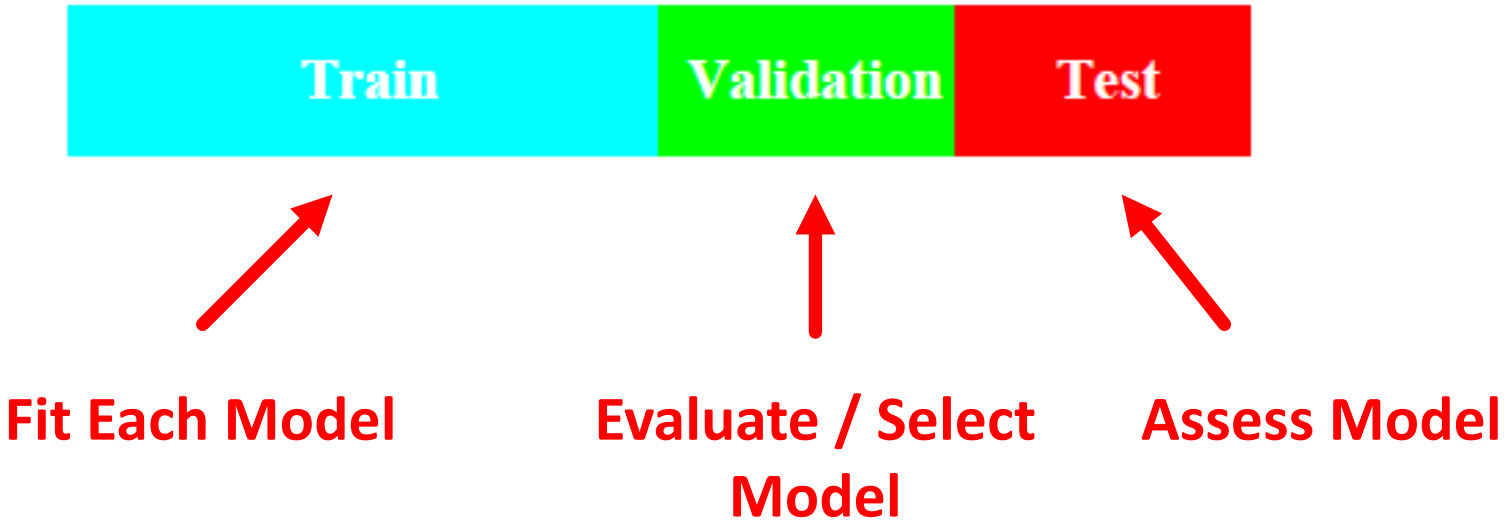
Overfitting vs Underfitting



Source: ibm.com

Model Selection / Assessment

Partition your data into Train-Validation-Test sets

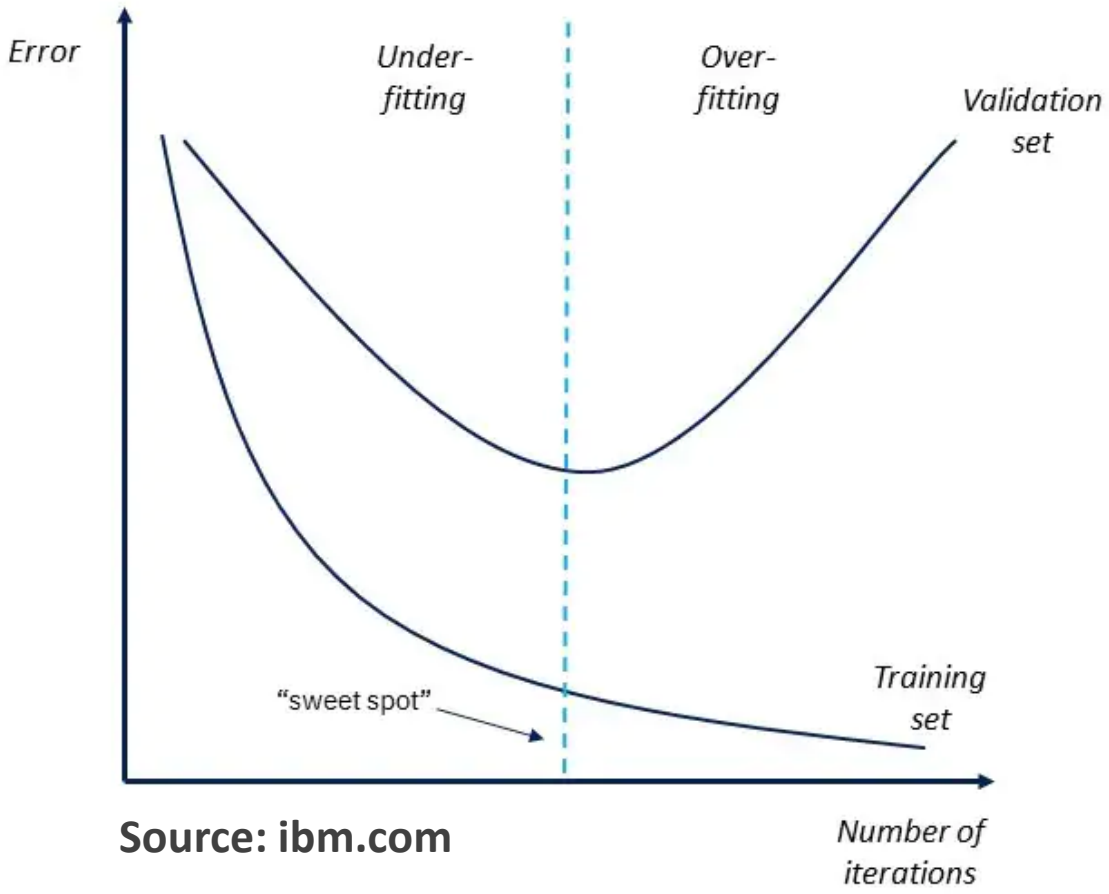


- Ideally, Test set is kept in a “vault” and only peek at it once model is selected
- Training-Validation-Test splits work if you have enough data (“data rich”)
- As a general rule 50% Training, 25% Validation, 25% Test (very loose rule)

Source: Hastie, Tibishrani, Freidman

Overfitting vs Underfitting

Underfitting performs poorly on *both* training and validation…



Source: ibm.com

…overfitting performs well on training but not on validation

KNN Model Selection / Assessment

1. Train a set of models K=1,…,K^{max} on training data:

$$\text{model}_{K=1}(\mathcal{D}^{\text{train}}),\ \ldots,\ \text{model}_{K=K^{\max}}(\mathcal{D}^{\text{train}})$$

2. Evaluate model accuracy on validation data:

$$\text{Error}(\text{model}_{K=1}, \mathcal{D}^{\text{val}}),\ \ldots,\ \text{Error}(\text{model}_{K=K^{\max}}, \mathcal{D}^{\text{val}})$$

3. Select model with lowest validation error:

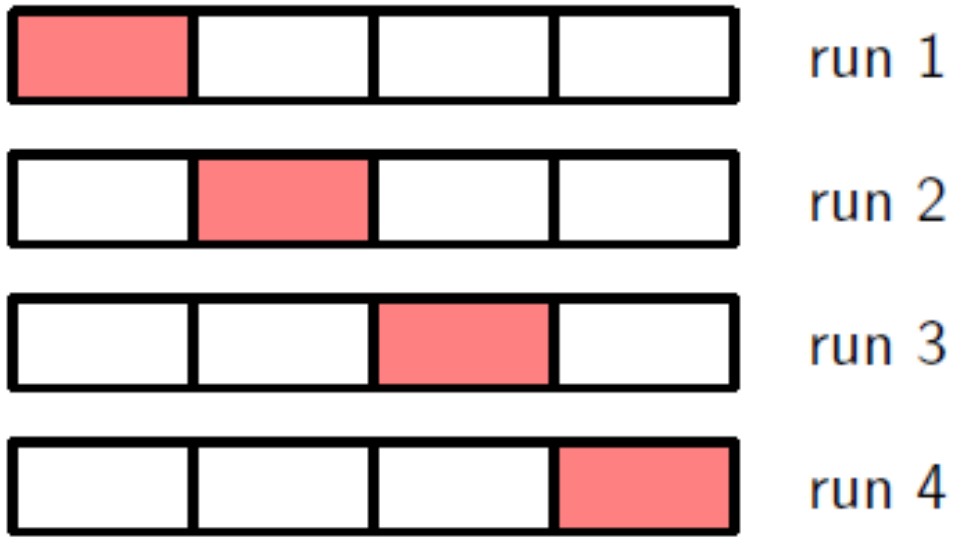
$$K^* = \arg\min_K \text{Error}(\text{model}_K, \mathcal{D}^{\text{val}})$$

3. Evaluate model error on test:

$$\text{Error}(\text{model}_{K^*}, \mathcal{D}^{\text{test}})$$

What are some drawbacks of this approach?

Cross-Validation



N-fold Cross Validation Partition training data into N "chunks" and for each run select one chunk to be validation data

For each run, fit to training data (N-1 chunks) and measure accuracy on validation set. Average model error across all runs.

Drawback Need a lot of training data to partition.

Source: Bishop, C. PRML

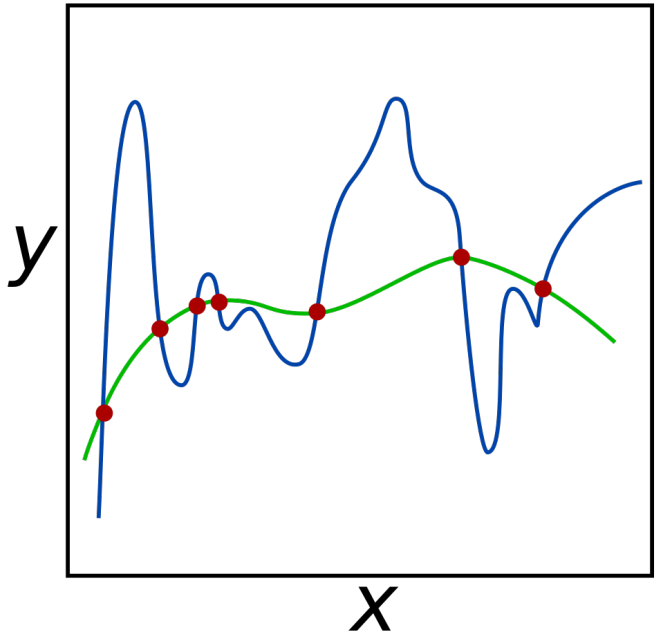
Regularization

Model fitting typically minimizes some loss function (or maximizes log-probability):

$$\text{Model} = \min_{\text{model}} \text{Loss}(\text{Model}, \text{Data}) + \lambda \cdot \text{Regularizer}(\text{Model})$$

Regularization Strength

Regularization Penalty



Both models (blue/green) have zero error on training data…Green will better generalize to unseen data

Akaike Information Criterion (AIC)

Penalizes models with many parameters,

$$\text{AIC} = \min_{\text{model}} \text{Num. Parameters} - \text{Log-Likelihood}(\text{Model}, \text{Data})$$

To apply in practice

- Suppose there are R candidate models
- Compute AIC scores: AIC_1, AIC_2, …, AIC_R
- Select model with minimum AIC score AIC_{min}

Quantity $exp(AIC_{min} - AIC_i)$ proportional to probability that i^{th} model minimizes *information loss*

Bayesian Information Criterion (BIC)

$$\mathrm{BIC} = \min_{\text{model}} \frac{1}{2} M \log(N) - \text{Log-Likelihood}(\text{Model}, \text{Data})$$

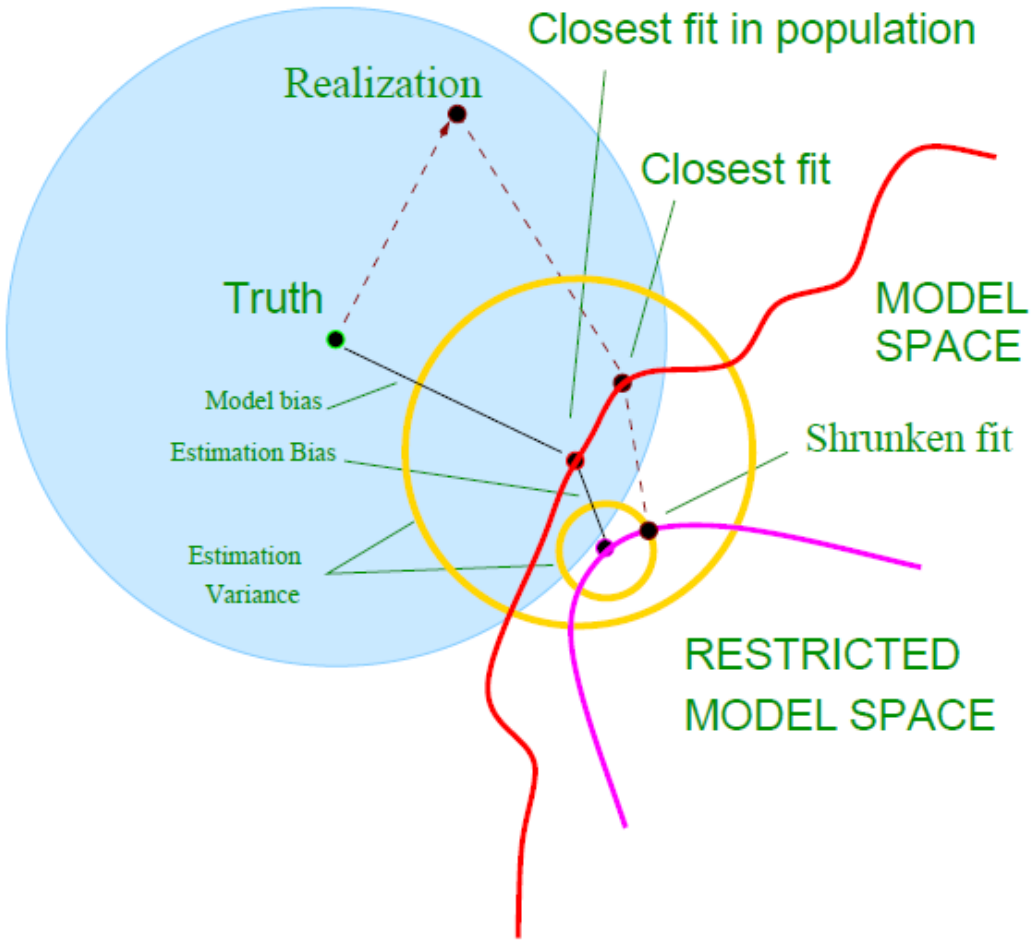
Where:

M : Number of model parameters

N : Number of training data

- Penalizes model complexity more heavily due to log(N) factor
- Both AIC and BIC are approximations and can be rough
- Equivalent to Minimum Description Length (with a negative sign)

Bias-Variance Tradeoff



Regularizatoin introduces bias, but reduces variance…

We need to account for noise in the data, expressivity of the chosen model family, and bias of restricted model families…

Error = Irreducible Error + Variance + Bias^2

Source: Hastie, Tibishrani, Freidman

Scikit-Learn

Python library for machine learning. Install using Anaconda:

```
$ conda install -c conda-forge scikit-learn
```

Check your Anaconda configuration:

```
$ conda list scikit-learn  # to see which scikit-learn version is installed
$ conda list  # to see all packages installed in the active conda environment
$ python -c "import sklearn; sklearn.show_versions()"
```

Or using PyPi:

```
$ pip install -U scikit-learn
```

Dependency	Minimum Version	Purpose
numpy	1.14.6	build, install
scipy	1.1.0	build, install
joblib	0.11	install
threadpoolctl	2.0.0	install
cython	0.28.5	build
matplotlib	2.2.2	benchmark, docs, examples, tests
scikit-image	0.14.5	docs, examples, tests
pandas	0.25.0	benchmark, docs, examples, tests
seaborn	0.9.0	docs, examples
memory_profiler	0.57.0	benchmark, docs
pytest	5.0.1	tests
pytest-cov	2.9.0	tests
flake8	3.8.2	tests
black	21.6b0	tests
mypy	0.770	tests
pyamg	4.0.0	tests
sphinx	4.0.1	docs
sphinx-gallery	0.7.0	docs
numpydoc	1.0.0	docs
Pillow	7.1.2	docs
sphinx-prompt	1.3.0	docs
sphinxext-opengraph	0.4.2	docs

Scikit-Learn

Models called “estimators”, can be fit using the `fit()` function. E.g. Random Forest Classifier,

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> clf = RandomForestClassifier(random_state=0)
>>> X = [[ 1,  2,  3],  # 2 samples, 3 features
...      [11, 12, 13]]
>>> y = [0, 1]  # classes of each sample
>>> clf.fit(X, y)
RandomForestClassifier(random_state=0)
```

Question Is this model *supervised* or *unsupervised*? How do you know?

`fit()` Generally accepts 2 inputs

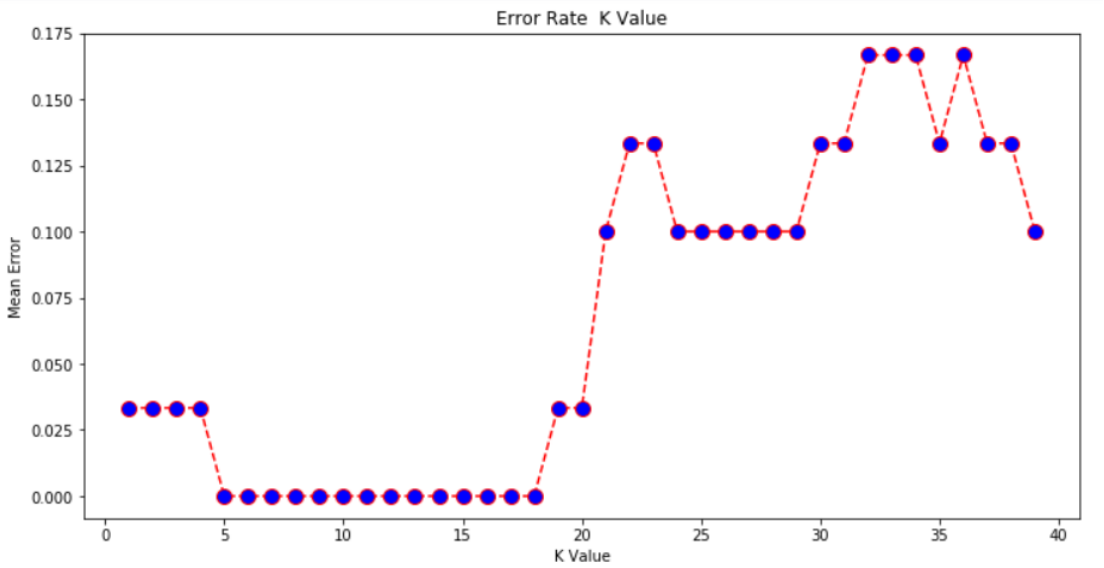
- Sample matrix X—typically (n_samples, n_features)
- Target values Y—real numbers for regression, integer for classification (not necessary for unsupervised models)

K-Nearest Neighbors

Train / evaluate the KNN classifier for each value K,

```
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_val)
    error.append(np.mean(pred_i != y_val))
```

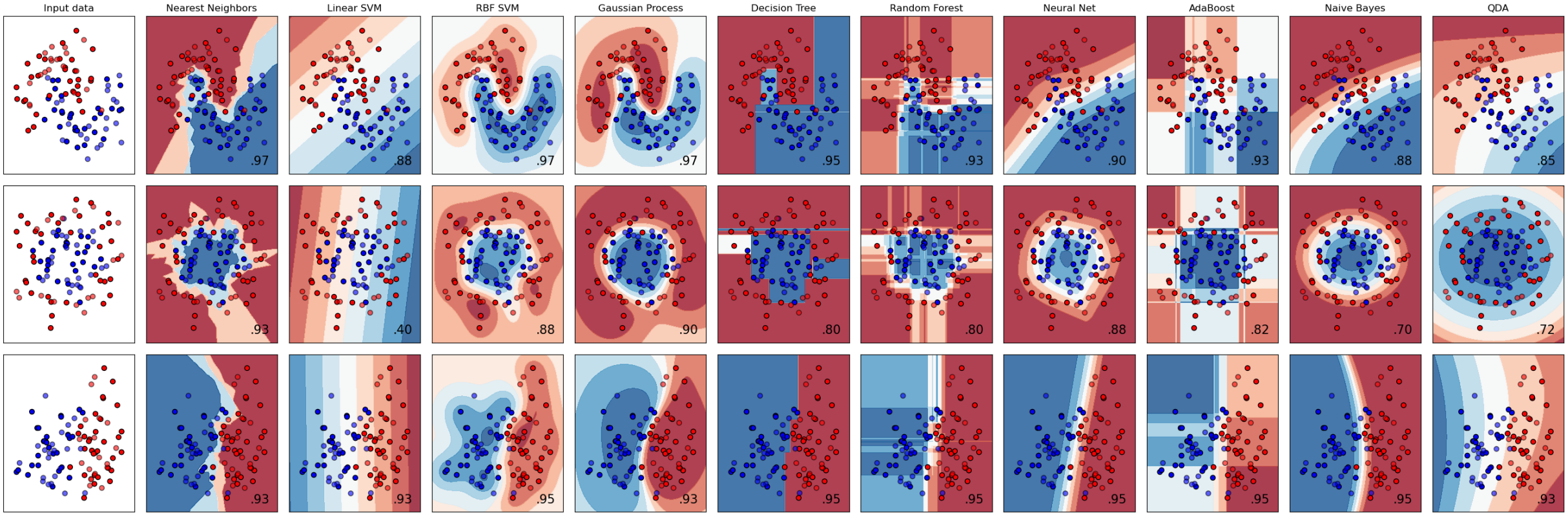


Print error:

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Scikit-Learn

Easily try out *all* the classifiers…

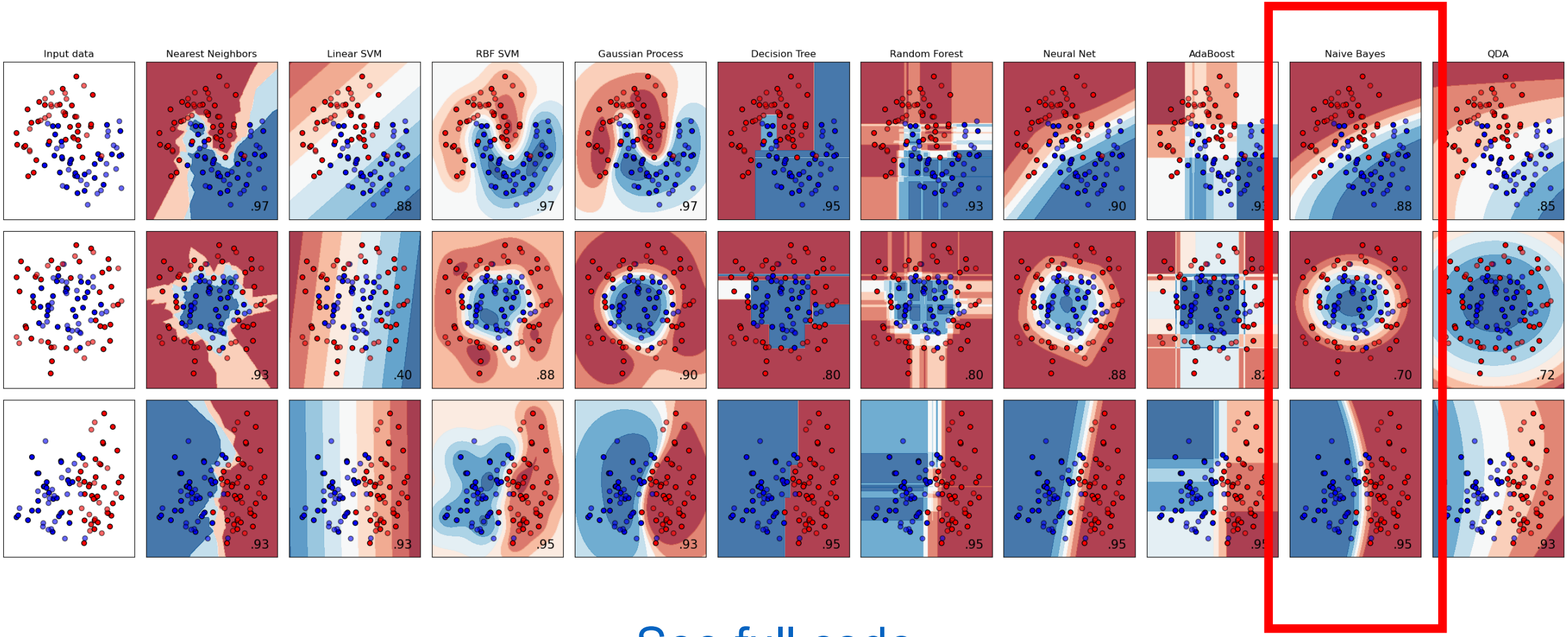


See full code.

Scikit-Learn

Easily try out *all* the classifiers…

Naïve Bayes



[See full code.]

Naïve Bayes Overview

Heads Up This section will return to some math as we go in depth. But, much of it is review of MLE/MAP that you already know with a new application (Naïve Bayes Classification) – **ask questions if you are lost**

- Introduction to Naïve Bayes Classifier
- Maximum Likelihood Estimation
- Maximum a Posteriori Estimation

We will have some review of MLE / MAP

Example: Naïve Bayes Classifier

Training Data:

Person	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Features

Task: Observe features $x_1, \ldots, x_D$ and predict class label C_k

Model: Treat features as *conditionally independent*, given class label,

$$p(x, C) = p(C) \prod_{d=1}^{D} p(x_d \mid C)$$

Doesn't capture correlation among features, but is easier to learn.

Classification: Bayesian model so classify by posterior,

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

Naïve Bayes Classifier

Simplifying Assumption *Class conditional* distribution,

$$p(x \mid C_\ell) = \prod_{d=1}^{D} p(x_d \mid C_\ell)$$

Assumes features are conditionally independent given class

- "Naïve" as we do not expect features to be conditionally independent
- **Easy to learn** For L classes and D features only $\mathcal{O}(LD)$ parameters
- Every feature can have a different class-conditional distribution
- Compare to KNN class conditional, which is *uniform*

$$p(x \mid \mathcal{C}_\ell) = \frac{K_\ell}{N_\ell V}$$

#-Neighbors-in-class-l

#-Volume-of-K-neighbors

#-class-l-in-training

Naïve Bayes Classifier

Features are typically not independent!

Example 1 If a recent news article contains word “Donald” it is much more likely to contain the word “Trump”.

Example 2 If flower petal width is very large then petal length is also likely to be high.

Source: [Matt Gormley]

Naïve Bayes Classifier

For real-valued features we can use Normal distribution:

$$p(x \mid C_\ell) = \prod_{d=1}^{D} \mathcal{N}(x_d \mid \underbrace{\mu_{d\ell}, \sigma^2_{d\ell}}_{\text{Parameters of feature d for class } l})$$

Recall Product of Normals is a Normal distribution

For binary features $x_d \in \{0, 1\}$ can use Bernoulli distributions:

$$p(x \mid C_\ell) = \prod_{d=1}^{D} \text{Bernoulli}(x_d \mid \theta_{d\ell})$$

"Coin bias" for dth feature and class l

- K-valued discrete features use Categorical, etc.
- Can mix-and-match, e.g. some discrete, some continuous features

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x, C_\ell) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{d\ell})$$

Prior Parameters **Likelihood Parameters**

Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ maximize the likelihood function,

$$\theta^{\mathrm{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Substitute general form of Naïve Bayes distribution and simplify…

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x \mid C_\ell) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{d\ell})$$

Prior Parameters

Likelihood Parameters

Let's review maximum likelihood estimation…

Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ maximize the likelihood function,

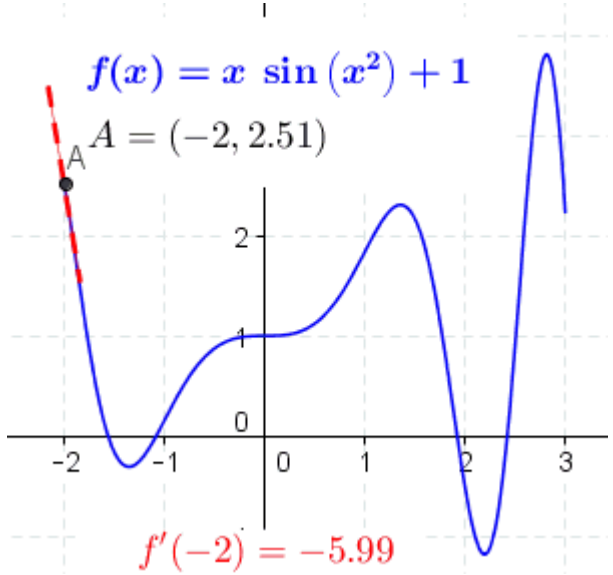
$$\theta^{\text{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Substitute general form of Naïve Bayes distribution and simplify…

REVIEW Maximum Likelihood

Maximum Likelihood Estimator (MLE) as the name suggests, maximizes the likelihood function.

$$\hat{\theta}^{\mathrm{MLE}} = \arg\max_{\theta} \mathcal{L}_N(\theta) = \prod_{i=1}^{N} p(x_i; \theta)$$



Question How do we find the MLE?

Answer Remember calculus…

Is $\mathcal{L}_N(\theta)$ convex?

Yes → Unique, closed-form solution → **Approach**

- Compute derivative $\frac{d\mathcal{L}_N}{d\theta}$
- Set to zero and solve

$$\frac{d\mathcal{L}_N}{d\theta} = 0 \Rightarrow \hat{\theta}^{\mathrm{MLE}}$$

No → Gradient-based optimization → Still have to compute derivative…

REVIEW Maximum Likelihood

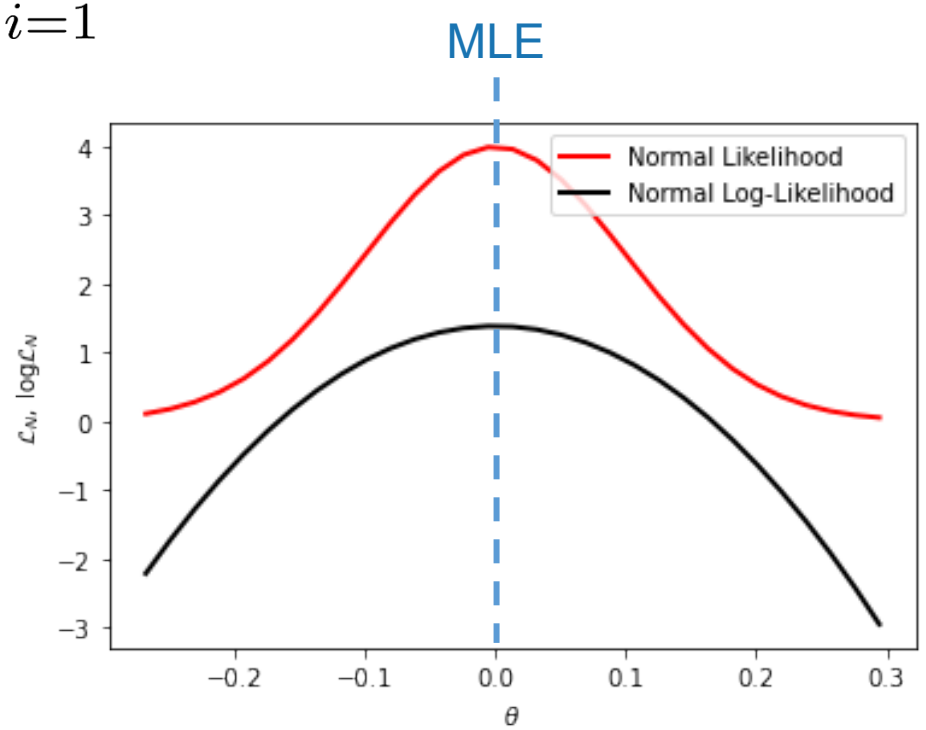
Maximizing log-likelihood makes the math easier (as we will see) and doesn't change the answer (logarithm is an increasing function)

$$\hat{\theta}^{\mathrm{MLE}} = \arg\max_{\theta} \; \log \mathcal{L}_N(\theta) = \sum_{i=1}^{N} \log p(x_i; \theta)$$

Derivative is a linear operator so,

$$\frac{d}{d\theta} \log \mathcal{L}_N(\theta) = \underbrace{\sum_{i=1}^{N} \frac{d}{d\theta} \log p(x_i; \theta)}_{\text{One term per data point}}$$

One term per data point
Can be computed in parallel
(big data)

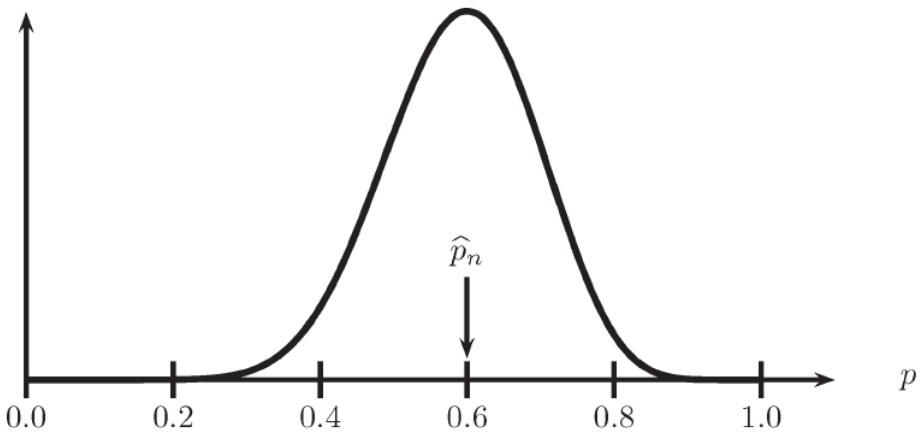


REVIEW Maximum Likelihood

[Source: Wasserman, L. 2004]

Example Suppose we have N coin tosses with $X_1, \ldots, X_n \sim \text{Bernoulli}(p)$ but we don't know the coin bias p. The likelihood function is,

$$\mathcal{L}_n(p) = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i} = p^S(1-p)^{n-S}$$



Likelihood function for Bernoulli with n=20 and $\sum_i x_i = 12$ heads

where $S = \sum_i x_i$. The log-likelihood is,

$$\log \mathcal{L}_n(p) = S \log p + (n - S)\log(1-p)$$

Set the derivative of $\log \mathcal{L}_n(p)$ to zero and solve,

$$\hat{p}^{\text{MLE}} = S/n = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Maximum likelihood is equivalent to sample mean in Bernoulli

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x \mid C_\ell) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{d\ell})$$

Prior Parameters

Likelihood Parameters

…OK, back to Naïve Bayes

Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ maximize the likelihood function,

$$\theta^{\mathrm{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Substitute general form of Naïve Bayes distribution and simplify…

Naïve Bayes Model : Maximum Likelihood

Fitting the model requires learning all parameters…

$$p(x, C_\ell; \pi, \theta) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d; \theta_{d\ell})$$

Prior Parameters **Likelihood Parameters**

Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ maximize the likelihood function,

$$\theta^{\mathrm{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Substitute general form of Naïve Bayes distribution and simplify…

Naïve Bayes Model : Maximum Likelihood

$$\theta^{\mathrm{MLE}} = \arg\max_{\pi,\theta} \log p(\mathcal{D}; \pi, \theta)$$

Since data are iid

$$= \arg\max_{\pi,\theta} \log \prod_{i=1}^{D} p(x_i, y_i; \pi, \theta)$$

log(ab) = log a + log b

$$= \arg\max_{\pi,\theta} \sum_{i=1}^{N} \log p(x_i, y_i; \pi, \theta)$$

Naïve Bayes

$$= \arg\max_{\pi,\theta} \sum_{i=1}^{N} \log p(y_i; \pi) + \sum_{i=1}^{N} \sum_{d=1}^{D} \log p(x_{id} \mid \theta_{dy_i})$$

Find zero-gradient if concave, or gradient-based optimization otherwise

Example: Naïve Bayes with Bernoulli Features

Roll a weighted K-sided die

Y =

Flip D biased coins

X | Y=1

X | Y=2

...

X | Y=K

y	x_1	x_2	...			x_D
5	0	1	1	0	1	0
2	1	0	0	0	0	0
3	1	1	1	1	0	0
...			...			
4	0	0	1	1	0	1

While we *can* generate data since Naïve Bayes is a generative model, we typically don't, since data are given.

Adapted from: Matt Gormley

Example: Naïve Bayes with Bernoulli Features

Let each feature follow a Bernoulli distribution then the model is…

$$y = c \sim \mathrm{Cat}(\pi) \qquad x_j | y = c \sim \mathrm{Ber}(\theta_{jc})$$

The Naïve Bayes joint distribution is then:

$$p(\mathcal{D} \mid \pi, \theta) = p(y_i | \boldsymbol{\pi}) \prod_j p(x_{ij} | \boldsymbol{\theta}_j)$$

$$= \prod_c \pi_c^{\mathbb{I}(y_i = c)} \prod_j \prod_c p(x_{ij} | \boldsymbol{\theta}_{jc})^{\mathbb{I}(y_i = c)}$$

Write down log-likelihood and optimize…

Bernoulli Naïve Bayes MLE

Let $N_c \triangleq \sum_i \mathbb{I}(y_i = c)$ be number of training examples in class c then,

$$\log p(\mathcal{D} \mid \pi, \theta) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:y_i=c} \log p(x_{ij} | \boldsymbol{\theta}_{jc})$$

Log-likelihood function is concave in all parameters so…

1. Take derivatives with respect to π and θ
2. Set derivatives to zero and solve

$$\hat{\pi}_c = \frac{N_c}{N}$$

Fraction of training examples from class c

$$\hat{\theta}_{jc} = \frac{N_{jc}}{N_c}$$

Number of "heads" in training set from class c

$$N_{jc} = \sum_i \mathbb{I}(y_i = c)\mathbb{I}(x_{ji} = 1)$$

Bernoulli Naïve Bayes MLE

Let's just review how easy it is to fit Bernoulli Naïve Bayes with MLE…

Given Training pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$

1. For each class c set prior probability to equal proportion of training data assigned to that class,

$$\hat{\pi}_c = \frac{N_c}{N} \qquad N_c \triangleq \sum_i \mathbb{I}(y_i = c)$$

2. For each feature j in each class c set Bernoulli parameter (coin bias),

$$\hat{\theta}_{jc} = \frac{N_{jc}}{N_c} \qquad N_{jc} = \sum_i \mathbb{I}(y_i = c)\mathbb{I}(x_{ji} = 1)$$

Bernoulli Naïve Bayes MLE

$$\hat{\pi}_c = \frac{N_c}{N}$$

Fraction of training examples from class c

$$\hat{\theta}_{jc} = \frac{N_{jc}}{N_c}$$

Number of "heads" in training set from class c

What if there are *no* examples of class c in the training set?

$$\hat{\pi}_c = 0$$

Model will never learn to guess class c

What if for class c all features x_{ij} = 0 in the training set?

$$\theta_{jc} = 0$$

Model will not know how to handle x_{ij}=1 heads for class c

Under MLE estimate training data needs to see <u>*every possible outcome*</u>

Any ideas how we can fix this problem?

Fixing Bernoulli MLE

We could add a small constant to prevent zero probabilities…

$$\hat{\pi}_c \propto N_c + \alpha \qquad\qquad \hat{\theta}_{jc} \propto N_{jc} + \beta$$

Fudge factors
Pseudocounts
Smoothing terms
….

…this is precisely the *Maximum a Posteriori (MAP)* estimate…

To see this let's convert to a Bayesian model

Bayesian Naïve Bayes

$$p(x, C_\ell; \pi, \theta) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d; \theta_{d\ell})$$

Prior Parameters **Likelihood Parameters**

How do we convert this into a Bayesian model?

$$p(x, C_\ell, \pi, \theta) = \underbrace{p(\pi)p(\theta)}_{\text{Prior distribution over parameters}} p(C_\ell \mid \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{d\ell})$$

Bayesian Naïve Bayes

$$p(x, C_\ell; \pi, \theta) = p(C_\ell; \pi) \prod_{d=1}^{D} p(x_d; \theta_{d\ell})$$

Prior Parameters

Likelihood Parameters

Let's review MAP estimation…

How do we convert this into a Bayesian model?

$$p(x, C_\ell, \pi, \theta) = \underbrace{p(\pi)p(\theta)}_{\text{Prior distribution over parameters}} p(C_\ell \mid \pi) \prod_{d=1}^{D} p(x_d \mid \theta_{d\ell})$$

REVIEW MLE vs MAP Estimation

Suppose we have data $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$

Maximum Likelihood Estimate (MLE)

$$\boldsymbol{\theta}^{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(\mathbf{x}^{(i)} | \boldsymbol{\theta})$$

Maximum *a posteriori* (MAP) estimate

$$\boldsymbol{\theta}^{\text{MAP}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(\mathbf{x}^{(i)} | \boldsymbol{\theta}) \underbrace{p(\boldsymbol{\theta})}_{\text{Prior}}$$

Source: Matt Gormley

REVIEW Maximum a Posteriori (MAP)

Equivalent to maximizing joint probability,

Constant

$$\arg\max_{\theta} p(\theta \mid \mathcal{D}) = \arg\max_{\theta} \frac{p(\theta, \mathcal{D})}{p(\mathcal{D})} = \arg\max_{\theta} p(\theta, \mathcal{D})$$

For iid $\mathcal{D} = x_1, \ldots, x_N$ solve in log-domain,

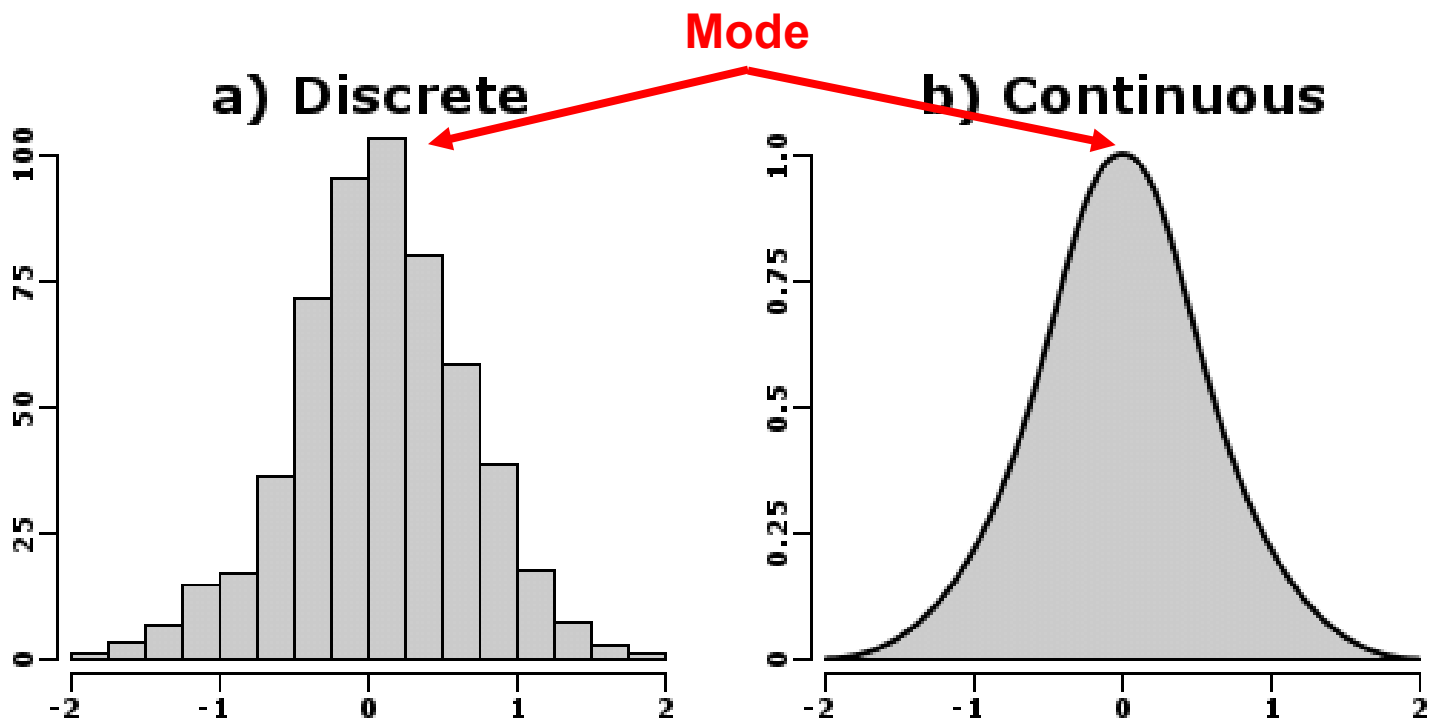
$$\hat{\theta}^{\mathrm{MAP}} = \arg\max_{\theta} \log p(\theta, \mathcal{D}) = \underbrace{\log p(\theta)}_{\substack{\text{Log-Prior} \\ \text{(how well it} \\ \text{agrees with prior)}}} + \underbrace{\sum_i \log p(x_i \mid \theta)}_{\substack{\text{Log-Likelihood} \\ \text{(how well it fits data)}}}$$

Intuition *MAP is like MLE but with a "penalty" term (log-prior)*

REVIEW Maximum a Posteriori (MAP)

$$\hat{\theta}^{\mathrm{MAP}} = \arg\max_{\theta}\ p(\theta \mid \mathcal{D})$$

*MAP is the **mode** (highest probability outcome) of the posterior*



REVIEW Bayes' Rule : Reminder

prior probability

likelihood function for the parameters

$$p(\theta \mid D) = \frac{p(\theta)p(D \mid \theta)}{p(D)}$$

posterior probability

normalizer, often is not of interest

REVIEW Bayes' Rule : Reminder

prior probability

likelihood function for the parameters

$$p(\theta \mid D) \propto p(\theta)p(D \mid \theta)$$

posterior probability

Posterior is **proportional to** the joint

REVIEW Bayes' Rule : Reminder

prior probability

likelihood function for the parameters

$$q(\theta \mid D) \propto p(\theta)\omega(D \mid \theta)$$

posterior probability

In general, distributions are different functions

REVIEW Bayes' Rule : Reminder

Prior and **likelihood** chosen for model

$$q(\theta \mid D) \propto p(\theta)\omega(D \mid \theta)$$

Posterior determined by algebra

In general, distributions are different functions

REVIEW Conjugate Pairs

For some special models the posterior takes a simple form

$$p(\theta \mid D) \propto p(\theta)\omega(D \mid \theta)$$

Prior and posterior are the same distribution (with different parameters)

We have already seen one example, the Beta-Bernoulli conjugate pair:

$$\mathrm{Beta}(\theta \mid \alpha + \textbf{num.-heads}, \beta + \textbf{num.-tails}) \propto \mathrm{Beta}(\theta \mid \alpha, \beta) \prod_i \mathrm{Bernoulli}(x_i \mid \theta)$$

Same PDF

REVIEW Example: Beta-Bernoulli MAP

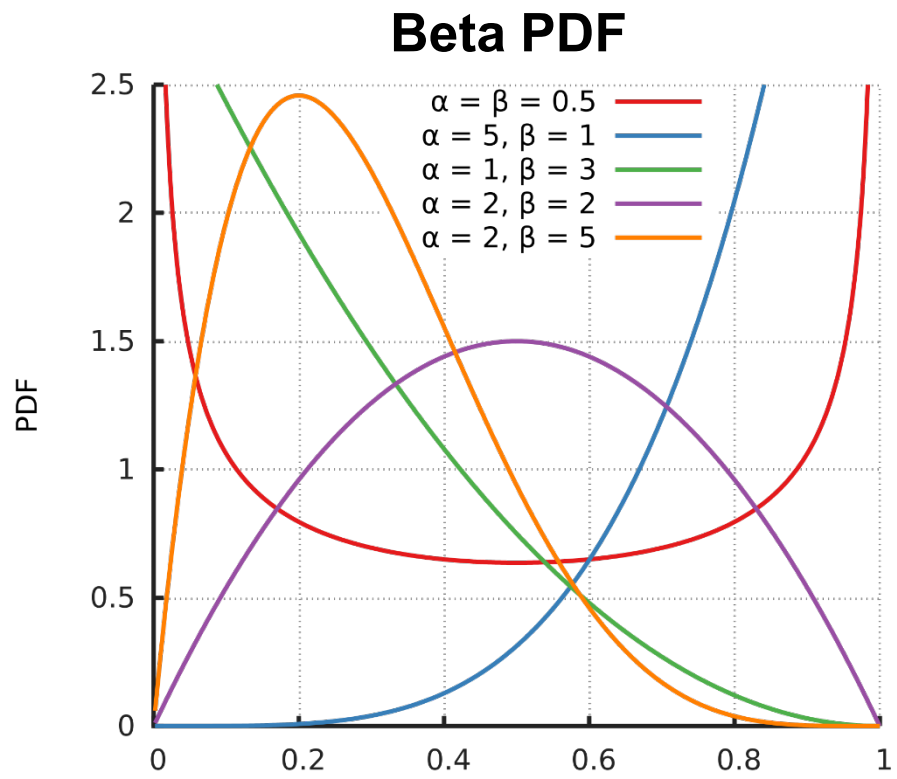
Let $X_1, \ldots, X_N \sim \text{Bernoulli}(\pi)$ and $\pi \sim \text{Beta}(\alpha, \beta)$ then posterior is,

$$p(\pi \mid X_1^N) = \text{Beta}(\alpha + \underbrace{\textbf{number of heads}}_{N_H},\ \beta + \textbf{number of tails})$$

Highest probability (mode) of Beta given by,

Take derivative, set to zero, solve.

$$\hat{\pi}^{\text{MAP}} = \frac{\alpha + N_H - 1}{\alpha + \beta + N - 2}$$



Beta distribution is not always convex!

- MAP is any value for $\alpha = \beta = 1$
- Two modes (bimodal) for $\alpha, \beta < 1$

REVIEW Example: Beta-Bernoulli

$$\text{Beta}(\theta \mid \alpha, \beta) \prod_{i=1}^{N} \text{Bernoulli}(x_i \mid \theta) \propto$$

$$\propto \theta^{\alpha-1}(1-\theta)^{\beta-1} \prod_{i} \theta^{x_i}(1-\theta)^{1-x_i}$$

$$= \theta^{\alpha-1}(1-\theta)^{\beta-1}\theta^{\sum_i x_i}(1-\theta)^{\sum_i (1-x_i)}$$

$$= \theta^{\alpha-1}(1-\theta)^{\beta-1}\theta^{\sum_i x_i}(1-\theta)^{\left(N-\sum_i x_i\right)}$$

$$= \theta^{\alpha-1+\sum_i x_i}(1-\theta)^{\beta-1+N-\sum_i x_i}$$

$$\propto \text{Beta}(\theta \mid \alpha + \sum_i x_i, \beta + N - \sum_i x_i)$$

Bernoulli Naïve Bayes MAP

Recall our original model…

$$y = c \sim \mathrm{Cat}(\pi) \qquad x_j \mid y = c \sim \mathrm{Bernoulli}(\theta_{jc})$$

…now add conjugate priors on the parameters…

…back to Bayesian Naïve Bayes…

$$\pi \sim \mathrm{Dirichlet}(\alpha) \qquad \theta_{jc} \sim \mathrm{Beta}(\beta_0, \beta_1)$$

The full joint PDF is now:

$$\mathrm{Dirichlet}(\pi \mid \alpha)\mathrm{Cat}(y \mid \pi) \prod_j \mathrm{Beta}(\theta_{jy} \mid \beta_0, \beta_1)\mathrm{Bernoulli}(x_{jy} \mid \theta_{jy})$$

Bernoulli Naïve Bayes MAP

Recall our original model…

$$y = c \sim \mathrm{Cat}(\pi) \qquad x_j \mid y = c \sim \mathrm{Bernoulli}(\theta_{jc})$$

…now add conjugate priors on the parameters…

$$\pi \sim \mathrm{Dirichlet}(\alpha) \qquad \theta_{jc} \sim \mathrm{Beta}(\beta_0, \beta_1)$$

The full joint PDF is now:

$$\mathrm{Dirichlet}(\pi \mid \alpha)\mathrm{Cat}(y \mid \pi) \prod_j \mathrm{Beta}(\theta_{jy} \mid \beta_0, \beta_1)\mathrm{Bernoulli}(x_{jy} \mid \theta_{jy})$$

Beta and Dirichlet Distributions

Beta (binary case)

- Conjugate prior to Bernoulli and Binomial distributed data (e.g. binary “coinflip” outcomes)
- Draws from Beta are binary PMFs

Dirichlet (multi-outcome case)

- Conjugate prior to multi-outcome Categorical and Multinomial distributed data (e.g. K-valued outcomes)
- Draws from Dirichlet are K-valued PMFs

Bernoulli Naïve Bayes MAP Estimation

1. Write down full joint probability distribution,

$$p(\pi, \theta, \mathcal{D} \mid \alpha, \beta) = \text{From previous slide}$$

2. Maximize log-joint probability (it is concave),

$$\pi^{\text{MAP}}, \theta^{\text{MAP}} = \arg\max_{\pi,\theta} \log p(\pi, \theta, \mathcal{D} \mid \alpha, \beta)$$

3. MAP estimates are the same as what we guessed,

$$\pi_c^{\text{MAP}} \propto N_c + \alpha \qquad \theta_{jc}^{\text{MAP}} \propto N_{jc} + \beta$$

Prior "hyperparameters"

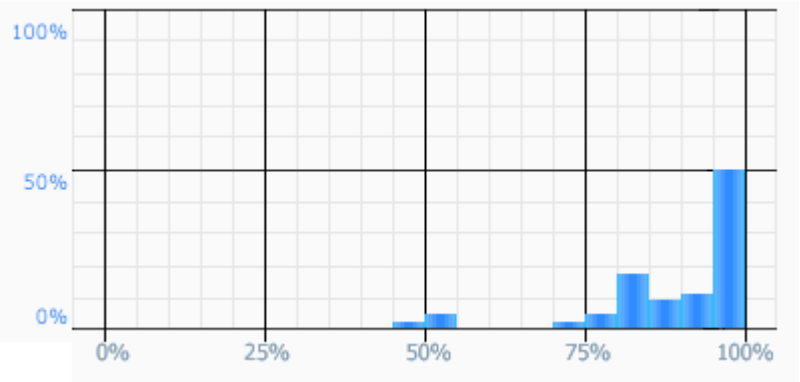
Administrative Items

- Homework 6
  - Out tonight
  - Due next Tuesday (11/2)
  - 7 Points
- Homework 5 grading complete
- Midterm grading mostly complete
- Last day to drop with a “W” (10/31)

Assignment Statistics

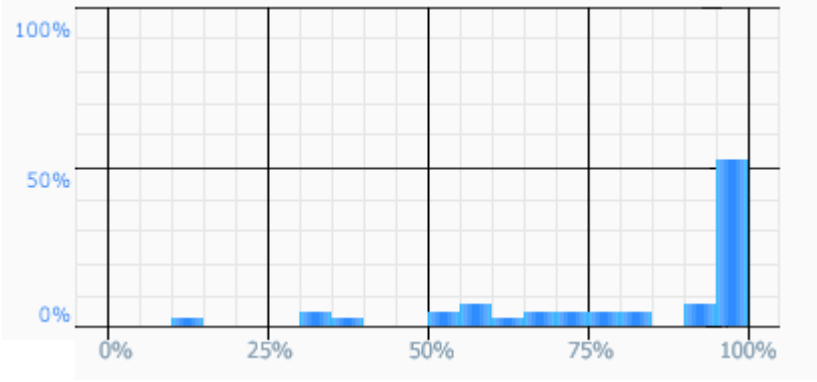
Homework 1

Median: 95% STDEV: 14%



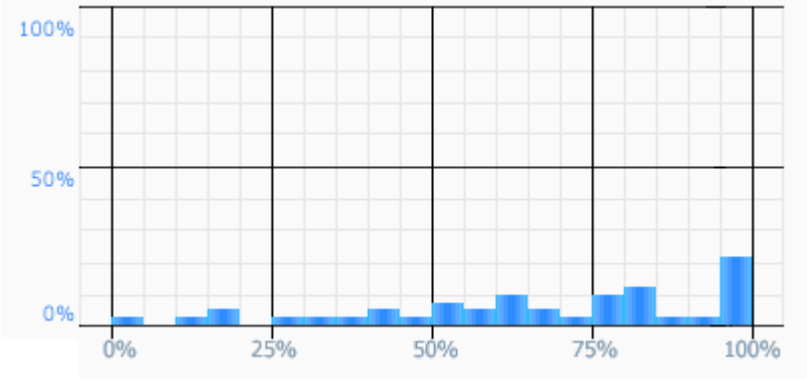
Homework 2

Median: 97% STDEV: 23%



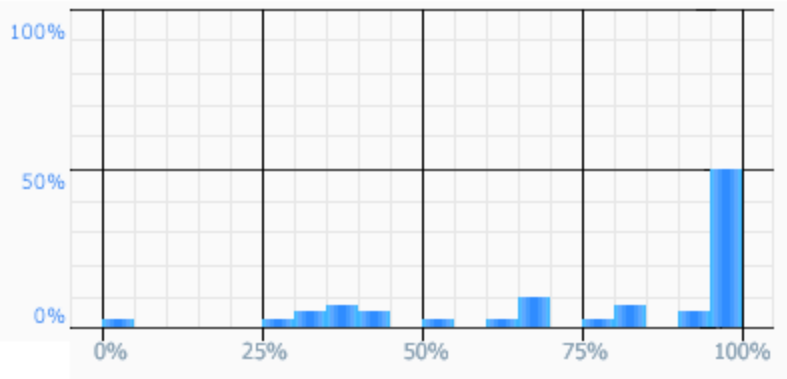
Homework 3

Median: 68% STDEV: 27%



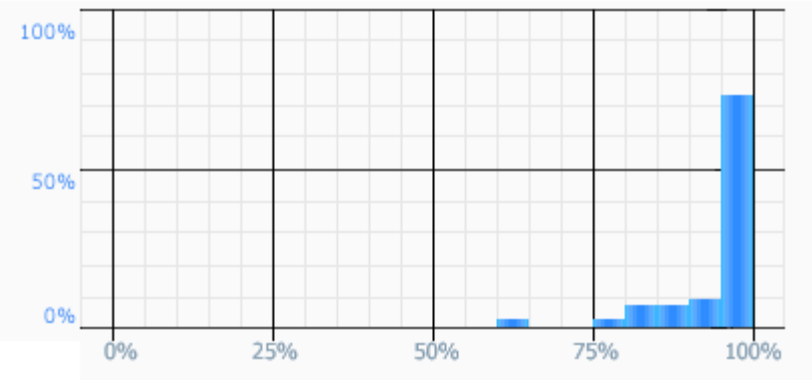
Homework 4

Median: 94% STDEV: 28%



Homework 5

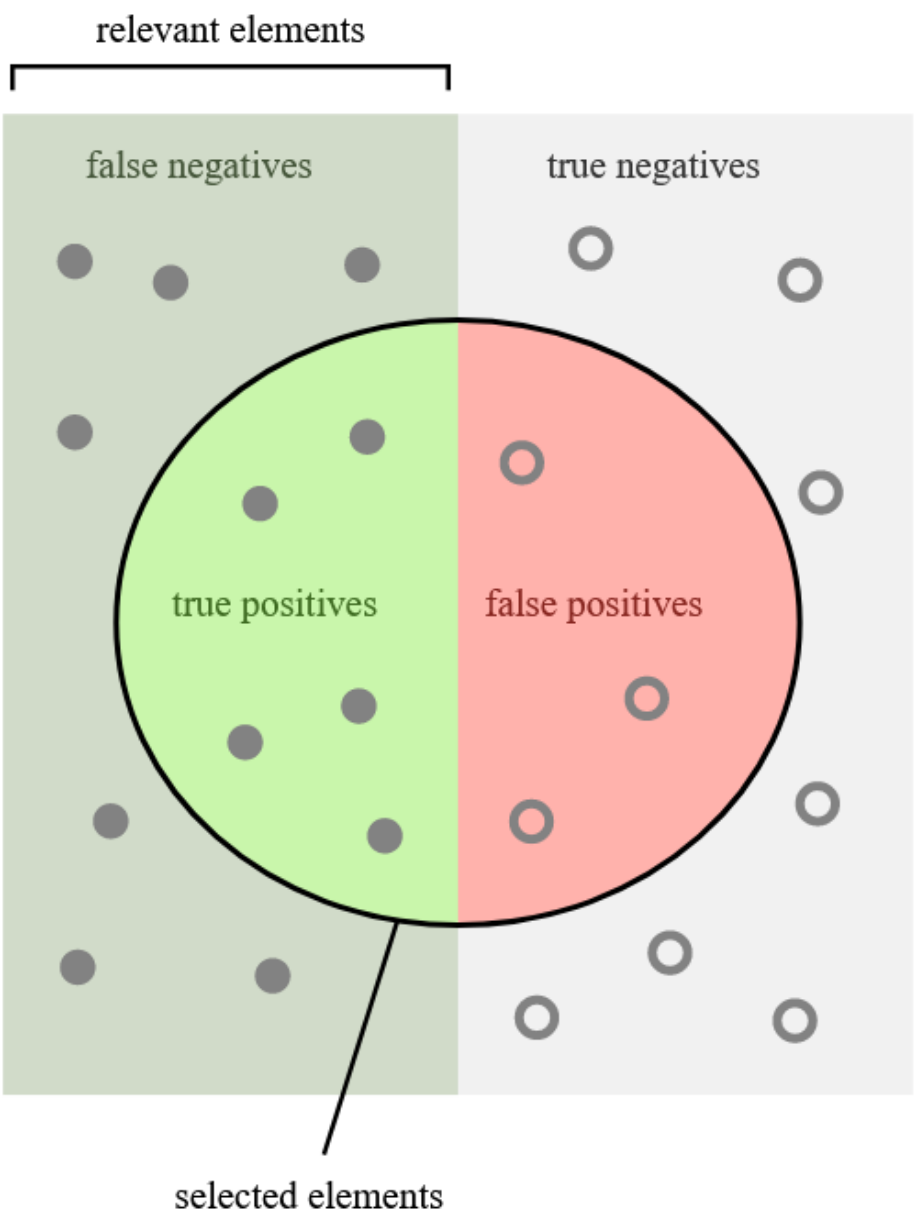
Median: 100% STDEV: 8%



Midterm Grading

- Choose 3 out of 4
- Each worth 6 and 2/3 points for a total of 20
- We are grading out of 10 points to make it easier, then converting (so you may see both)
- If you answer all 4, we take the lowest grade and convert it to out of 3 points (max 3/20 extra credit)

Evaluating Classifiers



For binary classifiers we evaluate a couple standard metrics,

How many selected items are relevant?

Precision = 

$$\frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

How many relevant items are selected?

Recall = 

$$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Evaluating Classifiers

Comparing precision vs. recall can be tricky, so we use F1 score,

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}$$

- This is the *harmonic mean* of precision and recall
- Can be very sensitive to *class imbalance* (num. positives vs negative)
- Use caution comparing F1 score on different data with different class imbalance
- Gives equal importance to precision and recall--might care about one more than the other (e.g. in medical tests we care about recall)

Confusion Matrix

Suppose our classifier distinguishes between cats and non-cats

Predicted class / Actual class	Cat	Non-cat
Cat	**6 true positives**	2 false negatives
Non-cat	1 false positive	**3 true negatives**

Confusion matrix lets us decide if classifier is biased towards certain mistakes (False Positives, False Neg.)

Perceived vowel / Vowel produced	i	e	a	o	u
i	**15**		1		
e	1		1		
a			**79**	5	
o			4	**15**	3
u				2	**2**

Extends to multi-class classification (e.g. classifying vowels)

Evaluation in Scikit-Learn

Evaluation functions live in `metrics`

`metrics.confusion_matrix`(y_true, y_pred, *)	Compute confusion matrix to evaluate the accuracy of a classification.
`metrics.dcg_score`(y_true, y_score, *[, k, ...])	Compute Discounted Cumulative Gain.
`metrics.det_curve`(y_true, y_score[, ...])	Compute error rates for different probability thresholds.
`metrics.f1_score`(y_true, y_pred, *[, ...])	Compute the F1 score, also known as balanced F-score or F-measure.
`metrics.fbeta_score`(y_true, y_pred, *, beta)	Compute the F-beta score.
`metrics.hamming_loss`(y_true, y_pred, *[, ...])	Compute the average Hamming loss.
`metrics.hinge_loss`(y_true, pred_decision, *)	Average hinge loss (non-regularized).
`metrics.jaccard_score`(y_true, y_pred, *[, ...])	Jaccard similarity coefficient score.
`metrics.log_loss`(y_true, y_pred, *[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
`metrics.matthews_corrcoef`(y_true, y_pred, *)	Compute the Matthews correlation coefficient (MCC).
`metrics.multilabel_confusion_matrix`(y_true, ...)	Compute a confusion matrix for each class or sample.
`metrics.ndcg_score`(y_true, y_score, *[, k, ...])	Compute Normalized Discounted Cumulative Gain.
`metrics.precision_recall_curve`(y_true, ...)	Compute precision-recall pairs for different probability thresholds.
`metrics.precision_recall_fscore_support`(...)	Compute precision, recall, F-measure and support for each class.
`metrics.precision_score`(y_true, y_pred, *[, ...])	Compute the precision.
`metrics.recall_score`(y_true, y_pred, *[, ...])	Compute the recall.

Naïve Bayes in Scikit-learn

Scikit-learn has separate classes each feature type

`sklearn.naive_bayes.GaussianNB`

Real-valued features

`sklearn.naive_bayes.MultinomialNB`

Discrete K-valued feature counts (e.g. multiple die rolls)

`sklearn.naive_bayes.BernoulliNB`

Binary features (e.g. coinflip)

`sklearn.naive_bayes.CategoricalNB`

Discrete K-valued features (e.g. single die roll)

For large training data that don't fit in memory use Scikit-learn's out-of-core learning

https://scikit-learn.org/stable/modules/naive_bayes.html

Bernoulli Naïve Bayes in Scikit-learn

sklearn.naive_bayes.BernoulliNB

class sklearn.naive_bayes.BernoulliNB(*, *alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None*) [source]

alpha : *float, default=1.0*

Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing).

Beta prior hyperparameter set to 0 for MLE

binarize : *float or None, default=0.0*

Threshold for binarizing (mapping to booleans) of sample features. If None, input is presumed to already consist of binary vectors.

fit_prior : *bool, default=True*

Whether to learn class prior probabilities or not. If false, a uniform prior will be used.

class_prior : *array-like of shape (n_classes,), default=None*

Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

Gaussian Naïve Bayes in Scikit-learn

sklearn.naive_bayes.GaussianNB

class sklearn.naive_bayes.GaussianNB(*, *priors=None*, *var_smoothing=1e-09*) [source]

Parameters:

priors : *array-like of shape (n_classes,)*

Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

var_smoothing : *float, default=1e-9*

Portion of the largest variance of all features that is added to variances for calculation stability.

New in version 0.20.

Bayesian prior on class-conditional variances
MLE if set to 0

Preprocessing : Z-Score

Typical ML workflow starts with *pre-processing* or *transforming* data into some useful form, which Scikit-Learn calls *transformers*:

```
>>> from sklearn.preprocessing import StandardScaler
>>> X = [[0, 15],
...      [1, -10]]
>>> # scale data according to computed scaling values
>>> StandardScaler().fit(X).transform(X)
array([[-1.,  1.],
       [ 1., -1.]])
```

Example StandardScaler used to compute *Z-score*,

$$Z = \frac{X - \mu}{\sigma}$$

Used to make data fit to standard Normal $\mathcal{N}(0, 1)$

- Features are standardized independently (columns of X)
- Other transformers live in `sklearn.preprocessing`

Preprocessing : Encoding Labels

Oftentimes, categorical labels come as strings, which aren't easily modeled (e.g. with Naïve Bayes),

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```

`LabelEncoder` transforms these into integer values, e.g. for categorical distributions

Can *undo* using `inverse_transform` so we don't have to store two copies of the data

Pipeline

```
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
...
>>> # create a pipeline object
>>> pipe = make_pipeline(
...     StandardScaler(),
...     LogisticRegression()
... )
...
>>> # load the iris dataset and split it into train and test sets
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
...
>>> # fit the whole pipeline
>>> pipe.fit(X_train, y_train)
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
>>> # we can now use it like any other estimator
>>> accuracy_score(pipe.predict(X_test), y_test)
0.97...
```

ML workflows can be complicated. Chain tasks into a *pipeline…*

Example Standardizes data and fits logistic regression classifier

Nice `train_test_split` helper function

Cross-Validation

Easily do cross validation for model selection / evaluation…

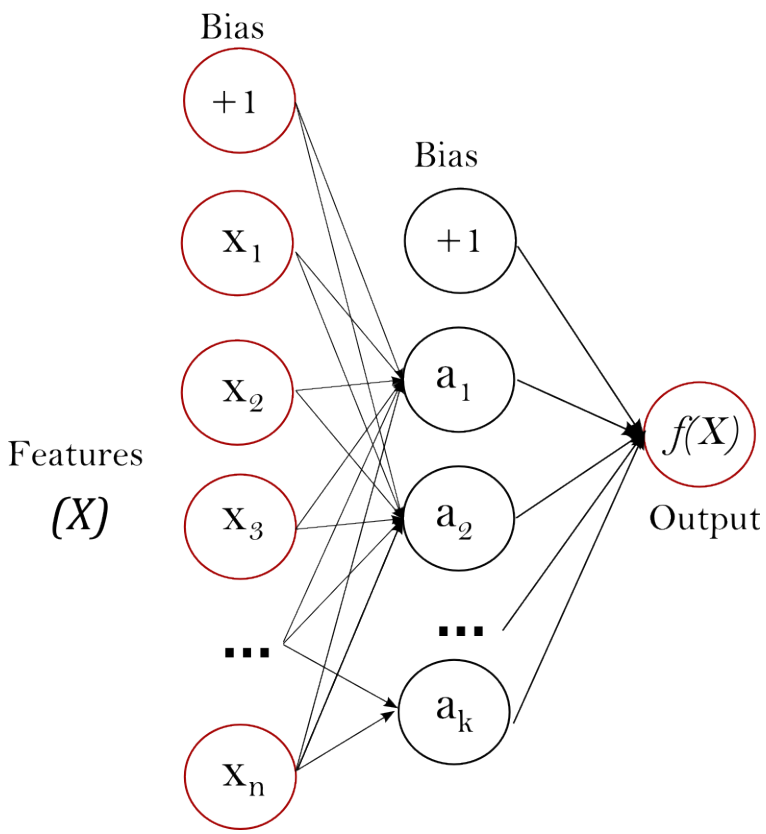
```
>>> from sklearn.datasets import make_regression
>>> from sklearn.linear_model import LinearRegression
>>> from sklearn.model_selection import cross_validate
...
>>> X, y = make_regression(n_samples=1000, random_state=0)
>>> lr = LinearRegression()
...
>>> result = cross_validate(lr, X, y)  # defaults to 5-fold CV
>>> result['test_score']  # r_squared score is high because dataset is easy
array([1., 1., 1., 1., 1.])
```

- `sklearn.model_selection`
- Many split functions: K-fold, leave-one-out, leave-P-out, etc.
- `cross_val_score` just computes the CV score (more common)

Scikit-Learn

Can fit Neural Networks as well, for example a *multilayer perceptron* (MLP) for classification,

```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
```



Now do some prediction on new data…

```
>>> clf.predict([[2., 2.], [-1., -2.]])
array([1, 0])
```

Neural nets for regression too:
`sklearn.neural_network.MLPRegressor`