

AI 6102: Machine Learning Methodologies & Applications

L9: Evaluation & Density Estimation

Sinno Jialin Pan

Nanyang Technological University, Singapore

Homepage: <http://www.ntu.edu.sg/home/sinnopan>

Outline

- Other evaluation metrics for classification
 - Imbalance classification problems
- Density estimation
 - Estimate unknown probability density function
 - Parametric & Non-parametric



Metrics for Classification

Confusion Matrix

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

f_{11} : TP (true positive)

f_{10} : FN (false negative)

f_{01} : FP (false positive)

f_{00} : TN (true negative)

Most widely-used metric:

$$\text{Accuracy} = \frac{\# \text{ correct predictions}}{\# \text{ predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

$$\text{Error rate} = \frac{\# \text{ incorrect predictions}}{\# \text{ predictions}} = 1 - \text{accuracy}$$

Initialization:

TP = 0, FN = 0, FP = 0, TN = 0

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11} (TP)	f_{10} (FN)
	Class = 0	f_{01} (FP)	f_{00} (TN)

Actual Prediction

1	← - - →	0	✗	FN++ ➔ FN=1
0	← - - →	0	✓	TN++ ➔ TN=1
1	← - - →	1	✓	TP++ ➔ TP=1
0	← - - →	1	✗	FP++ ➔ FP=1
0	← - - →	0	✓	TN++ ➔ TN=2
1	← - - →	1	✓	TP++ ➔ TP=2
0	← - - →	0	✓	TN++ ➔ TN=3
1	← - - →	1	✓	TP++ ➔ TP=3

		Predicted Class	
		1	0
Actual Class	1	3	1
	0	1	3

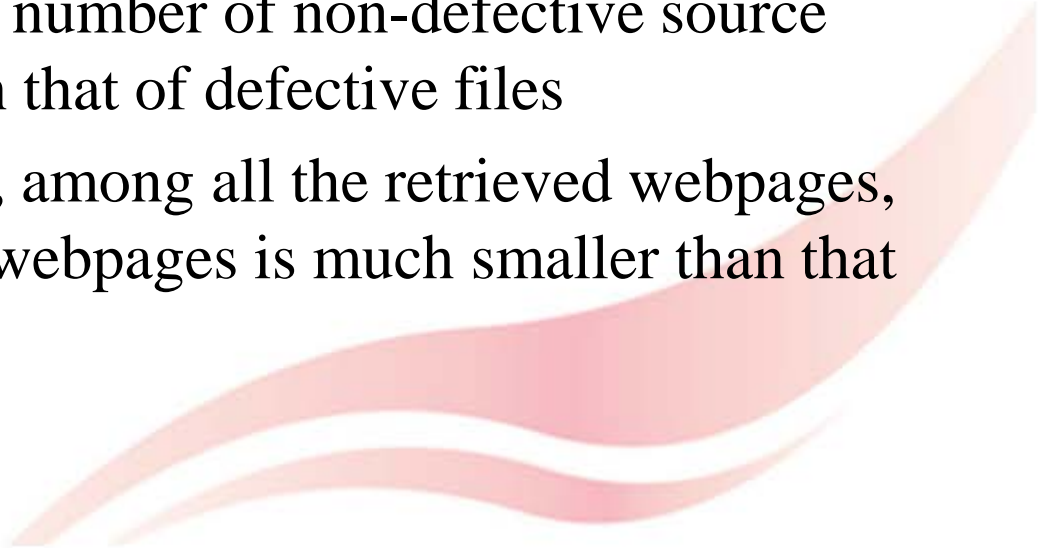
$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\#} = \frac{3 + 3}{8} = 0.75$$

Potential Issue of Accuracy

- Consider a 2-class classification problem
 - Number of Class 0 data instances = 99
 - Number of Class 1 data instances = 1
- If model predicts every data instance to be class 0, accuracy is $99/100 = 99\%$
 - Accuracy is misleading because model does not detect any class 1 data instance



Class Imbalance Problem

- In many real-world applications, data sets may have imbalanced class distributions
 - In medical domain, to diagnose whether a patient has cancer or not, the number of patients without cancer is much larger than that with cancer
 - In software engineering, to predict whether a source file has potential defect, the number of non-defective source files is much larger than that of defective files
 - In information retrieval, among all the retrieved webpages, the number of relevant webpages is much smaller than that of irrelevant webpages
- 

Alternative Metrics

- Accuracy measure treats every class as equally important
- It is not suitable for analyzing imbalance data sets, where the rare class is considered more interesting than the majority class
 - e.g., the patients who really have cancer
- For binary classification, the rare class is often denoted as the positive class, while the majority class is denoted as the negative class
 - i.e., f_{11} (TP) is more important

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11} (TP)	f_{10} (FN)
	Class = 0	f_{01} (FP)	f_{00} (TN)

Recall and Precision

- Recall and precision are two widely used metrics employed in applications where successful detection of one of the classes is considered more significant than detection of the other classes.

- Precision:

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

→ # instances predicted as positive

Precision is to measure among all the predicted positive instances, how many are true positive.

- Recall:

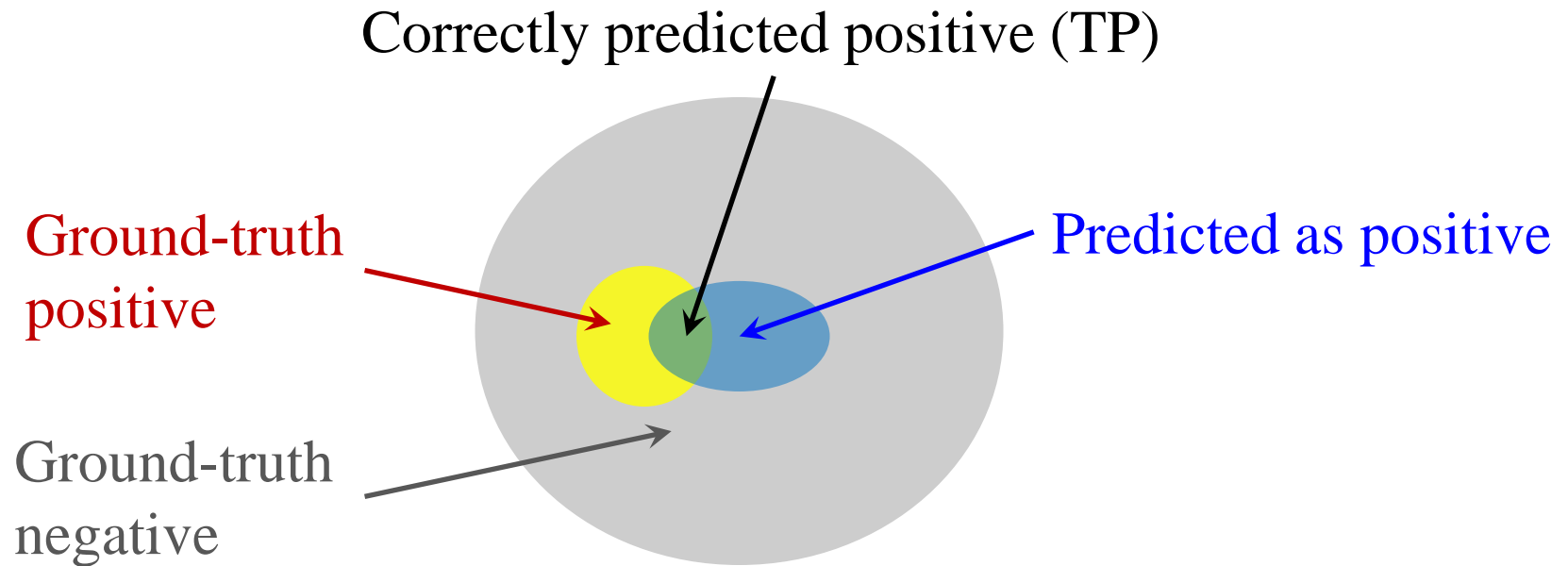
$$r = \frac{TP}{TP + FN}$$

true positive instances

Recall is to measure among all true positive instances, how many are predicted correctly by the classifier

- A good model should have both high precision and recall

Recall and Precision (cont.)



$$\text{Precision: } p = \frac{TP}{TP + FP}$$

The area of the overlapping region between yellow and blue divided by the area of the blue region


$$\text{Recall: } r = \frac{TP}{TP + FN}$$

The area of the overlapping region between yellow and blue divided by the area of the yellow region

F_1 Measure

- Recall and precision can be summarized into another metric known as F_1 measure

$$F_1 = \frac{2rp}{r + p}$$

- A high value of F_1 -measure ensures that precision and recall are reasonably high
- 

ROC


- ROC (Receiver Operating Characteristic) is a graphical approach to displaying the trade-off between true positive rate and false positive rate of a classifier

		Predicted Class		
		Class = 1	Class = 0	
Actual Class	Class = 1	f_{11}	f_{10}	f_{11} : TP (true positive) f_{10} : FN (false negative)
	Class = 0	f_{01}	f_{00}	f_{01} : FP (false positive) f_{00} : TN (true negative)

True positive rate: $\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{\#POS}}$ **Recall**

False positive rate: $\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} = \frac{\text{FP}}{\text{\#NEG}}$

Some Special Points

- (FPR, TPR):
 - $\left(\text{FPR} = \frac{\text{FP}}{\# \text{NEG}}, \text{TPR} = \frac{\text{TP}}{\# \text{POS}} \right)$
 - (0,0): to predict every data instance to be negative class
 - (1,1): to predict every data instance to be positive class
 - (0,1): ideal
- 

Varying Decision Threshold

- For balanced classification problem, the decision threshold is usually 0.5
- For imbalanced problem, as there are only a few positive instances, the threshold is set to be a larger values, e.g., 0.8
 - The best threshold can be tuned via cross-validation
 - For each threshold, generate a 2D point (FPR, TPR)
 - By connecting this points, we can construct a curve

			0.5	0.8
ID	$P(+ \mathbf{X})$	Truth	Pred	Pred
1	0.95	+	+	+
2	0.93	+	+	+
3	0.87	–	+	+
4	0.85	–	+	+
5	0.85	–	+	+
6	0.85	+	+	+
7	0.76	–	+	–
8	0.53	+	+	–
9	0.43	–	–	–
10	0.25	+	–	–

How to Draw ROC Curve?

- Sort the instances according to $P(+|\mathbf{X})$ in decreasing order
- Apply threshold at each unique value of $P(+|\mathbf{X})$
- Count the number of TP, FP, TN, FN based on each threshold

– TP Rate, TPR = $\frac{TP}{TP+FN}$

– FP Rate, FPR = $\frac{FP}{TN+FP}$

Not efficient

random

ID	$P(+ \mathbf{X})$	Truth
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

An Efficient Approach

[illegible][illegible]

An Efficient Approach (cont.)

Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4								
FP	5	5	4								
TN	0	0	1								
FN	0	1	1								
TPR	1	0.8	0.8								
FPR	1	1	0.8								

Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3							
FP	5	5	4	4							
TN	0	0	1	1							
FN	0	1	1	2							
TPR	1	0.8	0.8	0.6							
FPR	1	1	0.8	0.8							

An Efficient Approach (cont.)

Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3						
FP	5	5	4	4	3						
TN	0	0	1	1	2						
FN	0	1	1	2	2						
TPR	1	0.8	0.8	0.6	0.6						
FPR	1	1	0.8	0.8	0.6						

Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3					
FP	5	5	4	4	3	2					
TN	0	0	1	1	2	3					
FN	0	1	1	2	2	2					
TPR	1	0.8	0.8	0.6	0.6	0.6					
FPR	1	1	0.8	0.8	0.6	0.4					

An Efficient Approach (cont.)

Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3				
FP	5	5	4	4	3	2	1				
TN	0	0	1	1	2	3	4				
FN	0	1	1	2	2	2	2				
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6				
FPR	1	1	0.8	0.8	0.6	0.4	0.2				

Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2			
FP	5	5	4	4	3	2	1	1			
TN	0	0	1	1	2	3	4	4			
FN	0	1	1	2	2	2	2	3			
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4			
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2			

An Efficient Approach (cont.)

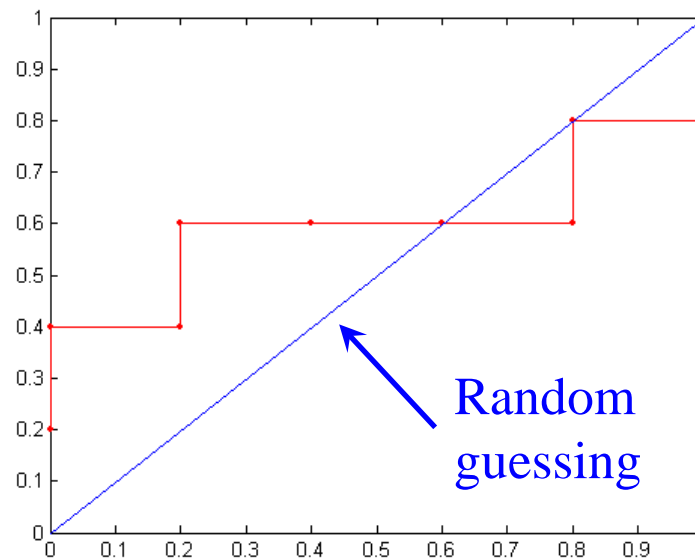
Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2		
FP	5	5	4	4	3	2	1	1	0		
TN	0	0	1	1	2	3	4	4	5		
FN	0	1	1	2	2	2	2	3	3		
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4		
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0		

Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	
FP	5	5	4	4	3	2	1	1	0	0	
TN	0	0	1	1	2	3	4	4	5	5	
FN	0	1	1	2	2	2	2	3	3	4	
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	

An Efficient Approach (cont.)

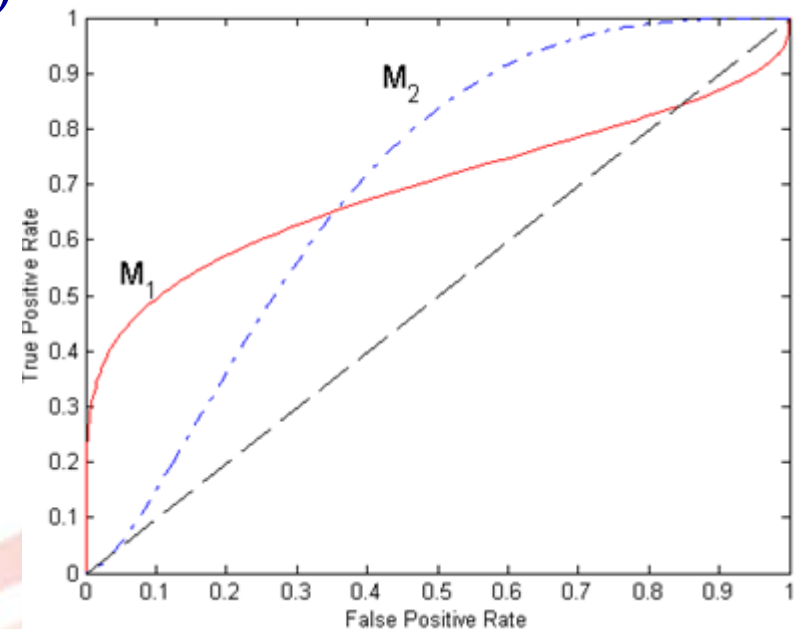
Class	+	-	+	-	-	-	+	-	+	+	
Threshold \geq	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

ROC Curve:



AUC for Classifiers Comparison

- In practice, on a specific dataset, a classifier may not consistently outperform the other with all the thresholds
 - M_1 is better for small FPR
 - M_2 is better for large FPR
- Area Under the ROC Curve (AUC)
 - Ideal: $AUC = 1$
 - Random guess: $AUC = 0.5$



Implementation using scikit-learn

- API: `sklearn.metrics`

<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>

Classification metrics

See the Classification metrics section of the user guide for further details.

<code>metrics.accuracy_score(y_true, y_pred, *[, ...])</code>	Accuracy classification score.
<code>metrics.auc(x, y)</code>	Compute Area Under the Curve (AUC) using the trapezoidal rule
<code>metrics.average_precision_score(y_true, ...)</code>	Compute average precision (AP) from prediction scores
<code>metrics.balanced_accuracy_score(y_true, ...)</code>	Compute the balanced accuracy
<code>metrics.brier_score_loss(y_true, y_prob, *)</code>	Compute the Brier score.
<code>metrics.classification_report(y_true, y_pred, *)</code>	Build a text report showing the main classification metrics.
<code>metrics.cohen_kappa_score(y1, y2, *[, ...])</code>	Cohen's kappa: a statistic that measures inter-annotator agreement.
<code>metrics.confusion_matrix(y_true, y_pred, *)</code>	Compute confusion matrix to evaluate the accuracy of a classification.
<code>metrics.dcg_score(y_true, y_score, *[, k, ...])</code>	Compute Discounted Cumulative Gain.
<code>metrics.f1_score(y_true, y_pred, *[, ...])</code>	Compute the F1 score, also known as balanced F-score or F-measure
<code>metrics.fbeta_score(y_true, y_pred, *, beta)</code>	Compute the F-beta score
<code>metrics.hamming_loss(y_true, y_pred, *[, ...])</code>	Compute the average Hamming loss.
<code>metrics.hinge_loss(y_true, pred_decision, *)</code>	Average hinge loss (non-regularized)
<code>metrics.jaccard_score(y_true, y_pred, *[, ...])</code>	Jaccard similarity coefficient score
<code>metrics.log_loss(y_true, y_pred, *[, eps, ...])</code>	Log loss, aka logistic loss or cross-entropy loss.
<code>metrics.matthews_corrcoef(y_true, y_pred, *)</code>	Compute the Matthews correlation coefficient (MCC)
<code>metrics.multilabel_confusion_matrix(y_true, ...)</code>	Compute a confusion matrix for each class or sample
<code>metrics.ndcg_score(y_true, y_score, *[, k, ...])</code>	Compute Normalized Discounted Cumulative Gain.
<code>metrics.precision_recall_curve(y_true, ...)</code>	Compute precision-recall pairs for different probability thresholds
<code>metrics.precision_recall_fscore_support(...)</code>	Compute precision, recall, F-measure and support for each class
<code>metrics.precision_score(y_true, y_pred, *[, ...])</code>	Compute the precision
<code>metrics.recall_score(y_true, y_pred, *[, ...])</code>	Compute the recall
<code>metrics.roc_auc_score(y_true, y_score, *[, ...])</code>	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
<code>metrics.roc_curve(y_true, y_score, *[, ...])</code>	Compute Receiver operating characteristic (ROC)
<code>metrics.zero_one_loss(y_true, y_pred, *[, ...])</code>	Zero-one classification loss.


Outline

- Other evaluation metrics for classification
 - Imbalance classification problems
- Density estimation
 - Estimate unknown probability density function
 - Parametric & Non-parametric

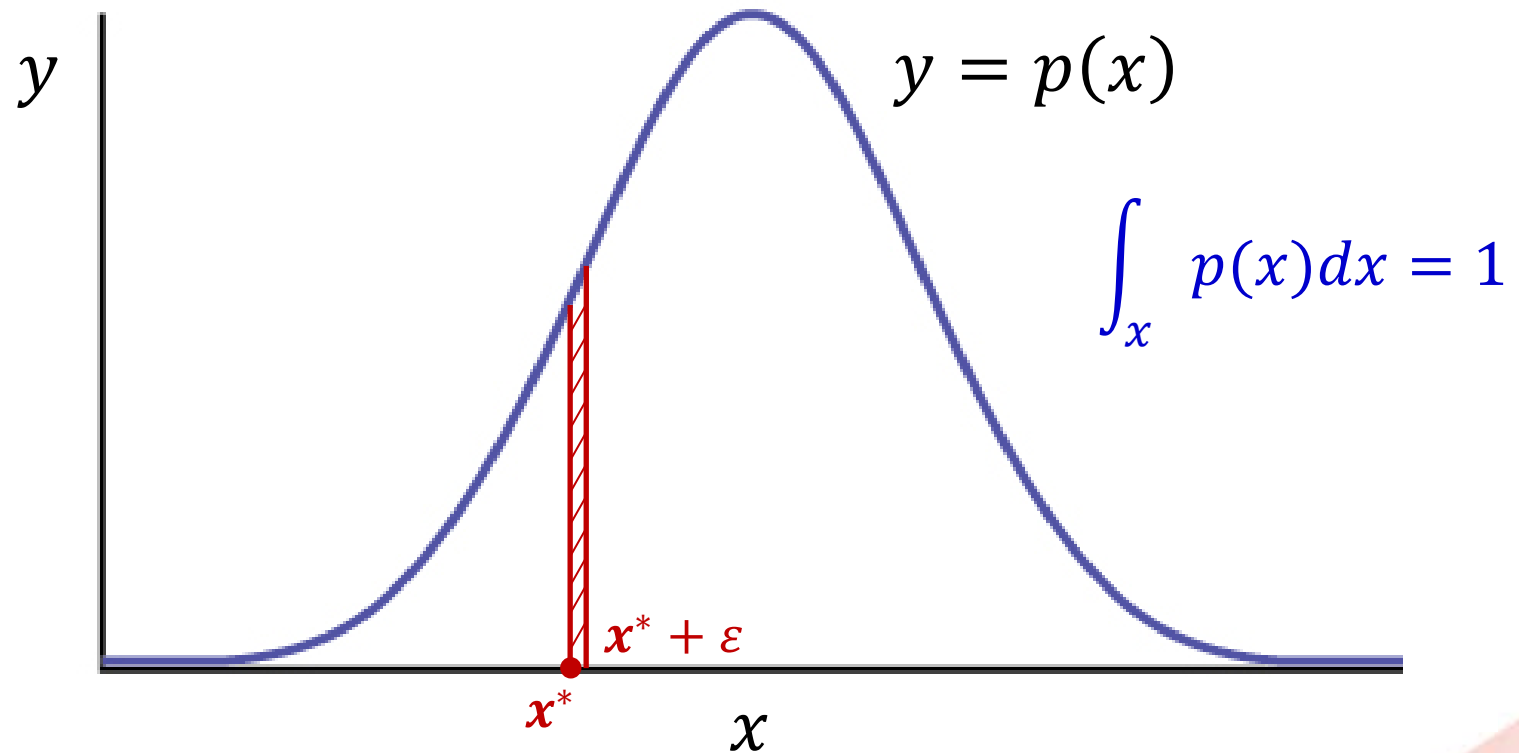


Density Estimation

- Density estimation aims to estimate an unobservable underlying probability density function based on observed data
- Denote by $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ the set of observed data points, drawn from an unknown $p(\mathbf{x})$,
$$\mathbf{x}_i \sim p(\mathbf{x}), \text{ for } i = 1, 2, \dots, N$$
- The goal is to estimate the probability density function $p(\mathbf{x})$


$$\int_{\mathbf{x}} p(\mathbf{x}) d\mathbf{x} = 1$$


Probability Density Function



$$P(\mathbf{x}^*) = \int_{x^*}^{x^* + \epsilon} p(x) dx \approx p(\mathbf{x}^*) \times \epsilon$$

Approaches

- Parametric density estimation
 - Assume a form for $p(\mathbf{x}; \boldsymbol{\theta})$, defined up to parameters, $\boldsymbol{\theta}$
 - E.g., Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, $\boldsymbol{\theta} = \{\mu, \sigma^2\}$
 - Estimate $\boldsymbol{\theta}$ from the observed data points
 - Maximum Likelihood Estimation
 - To find $\boldsymbol{\theta}$ that makes sampling \mathbf{x}_i from $p(\mathbf{x}; \boldsymbol{\theta})$ as likely as possible
 - Nonparametric density estimation
- 

Maximum Likelihood Estimation

- Likelihood of parameter $\boldsymbol{\theta}$ given sample \mathcal{D} :

$$l(\boldsymbol{\theta}|\mathcal{D}) \triangleq p(\mathcal{D}; \boldsymbol{\theta})$$

- Suppose $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ are i.i.d., the above likelihood is the product of the likelihoods of the individual data points

$$l(\boldsymbol{\theta}|\mathcal{D}) \triangleq p(\mathcal{D}; \boldsymbol{\theta}) = \prod_{i=1}^N p(\mathbf{x}_i; \boldsymbol{\theta})$$

- In MLE, we aim to find $\boldsymbol{\theta}$ that makes \mathcal{D} the most likely to be drawn. Mathematically, we aim to search for $\hat{\boldsymbol{\theta}}$ such that

$$\hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} l(\boldsymbol{\theta}|\mathcal{D})$$

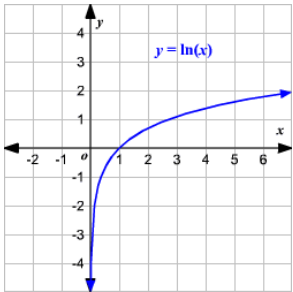
MLE (cont.)

- Recall that we maximize the log-likelihood:

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) \triangleq \ln l(\boldsymbol{\theta}|\mathcal{D})$$

- Why (refer to logistic regression)?

- The $\ln(\cdot)$ function is a strictly increasing function, one can maximize the likelihood without changing the value where it takes its maximum



$$\hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} l(\boldsymbol{\theta}|\mathcal{D}) \Leftrightarrow \hat{\boldsymbol{\theta}} = \arg \max_{\boldsymbol{\theta}} \ln l(\boldsymbol{\theta}|\mathcal{D})$$

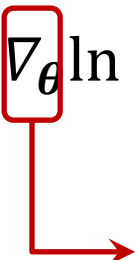
- The $\ln(\cdot)$ function converts the product into a sum

$$\ln l(\boldsymbol{\theta}|\mathcal{D}) = \ln p(\mathcal{D}; \boldsymbol{\theta}) = \ln \left(\prod_{i=1}^N p(\mathbf{x}_i; \boldsymbol{\theta}) \right) = \sum_{i=1}^N \ln p(\mathbf{x}_i; \boldsymbol{\theta})$$

Solution to MLE

- As $\max_{\theta} \ln l(\theta|\mathcal{D})$ is an unconstrained optimization problem, to solve it, we first set to the derivative of $\ln l(\theta|\mathcal{D})$ w.r.t. θ to be zero

$$\nabla_{\theta} \ln l(\theta|\mathcal{D}) = \nabla_{\theta} \left(\sum_{i=1}^N \ln p(\mathbf{x}_i; \theta) \right) = \sum_{i=1}^n \nabla_{\theta} \ln p(\mathbf{x}_i; \theta) = \mathbf{0},$$



$$\nabla_{\theta} = \left[\frac{\partial}{\partial \theta_1} \quad \frac{\partial}{\partial \theta_2} \quad \cdots \quad \frac{\partial}{\partial \theta_p} \right]^T$$

- If we can obtain a closed form solution $\hat{\theta}$ by solving the p equations, then it is done
- Otherwise, we may use gradient descent to find $\hat{\theta}$

Univariate Gaussian

- Suppose $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$. Each data instance \mathbf{x}_i is 1 dimensional, and drawn from Gaussian distribution with unknown μ and σ^2 :

$$\mathbf{x}_i \sim p(\mathbf{x}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

- The log-likelihood is

$$\begin{aligned}\ln l(\boldsymbol{\theta}|\mathcal{D}) &= \sum_{i=1}^N \ln p(\mathbf{x}_i; \mu, \sigma^2) = \sum_{i=1}^N \ln \left(\exp \left(-\frac{(\mathbf{x}_i - \mu)^2}{2\sigma^2} \right) \right) - \sum_{i=1}^N \ln(\sqrt{2\pi\sigma^2}) \\ &= -\frac{\sum_{i=1}^N (\mathbf{x}_i - \mu)^2}{2\sigma^2} - \sum_{i=1}^N \frac{1}{2} \ln(2\pi\sigma^2) \\ &= -\frac{\sum_{i=1}^N (\mathbf{x}_i - \mu)^2}{2\sigma^2} - \frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln \sigma^2\end{aligned}$$

Univariate Gaussian (cont.)

- The log-likelihood:

$$\ln l(\boldsymbol{\theta}|\mathcal{D}) = -\frac{\sum_{i=1}^N (\mathbf{x}_i - \mu)^2}{2\sigma^2} - \frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln \sigma^2$$

- The derivative of the log-likelihood:

$$\nabla_{\boldsymbol{\theta}} \ln l(\boldsymbol{\theta}|\mathcal{D}) = \begin{bmatrix} \nabla_{\mu} \ln l(\boldsymbol{\theta}|\mathcal{D}) \\ \nabla_{\sigma^2} \ln l(\boldsymbol{\theta}|\mathcal{D}) \end{bmatrix} = \begin{bmatrix} \frac{1}{\sigma^2} \sum_{i=1}^N (\mathbf{x}_i - \mu) \\ \frac{\sum_{i=1}^N (\mathbf{x}_i - \mu)^2}{2(\sigma^2)^2} - \frac{N}{2\sigma^2} \end{bmatrix}$$

Univariate Gaussian (cont.)

- By setting the derivative to be zero:

$$\begin{cases} \frac{1}{\sigma^2} \sum_{i=1}^N (x_i - \mu) = 0 \\ \frac{\sum_{i=1}^N (x_i - \mu)^2}{2(\sigma^2)^2} - \frac{N}{2\sigma^2} = 0 \end{cases}$$

- We have

$$\begin{cases} \hat{\mu} = \frac{1}{N} \sum_{i=1}^N x_i \\ \hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu})^2 \end{cases}$$

Unbiased estimation $\mathbb{E}[\hat{\mu}] = \mu$ ← True

$\mathbb{E}[\hat{\sigma}^2] = \frac{N-1}{N} \sigma^2$ ← Biased estimation

To correct bias

$$\begin{aligned} \tilde{\sigma}^2 &= \frac{N}{N-1} \hat{\sigma}^2 \\ &= \frac{1}{N-1} \sum_{i=1}^N (x_i - \hat{\mu})^2 \end{aligned}$$

Multivariate Gaussian

- Suppose $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and each data instance \mathbf{x}_i is m -dimensional, and drawn from Gaussian distribution with unknown $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:
 - $\boldsymbol{\mu}$: m -dimensional mean vector
 - $\boldsymbol{\Sigma}$: $m \times m$ covariance matrix

$$\mathbf{x}_i \sim p(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{m/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

The determinant of $\boldsymbol{\Sigma}$

- By using MLE

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$$

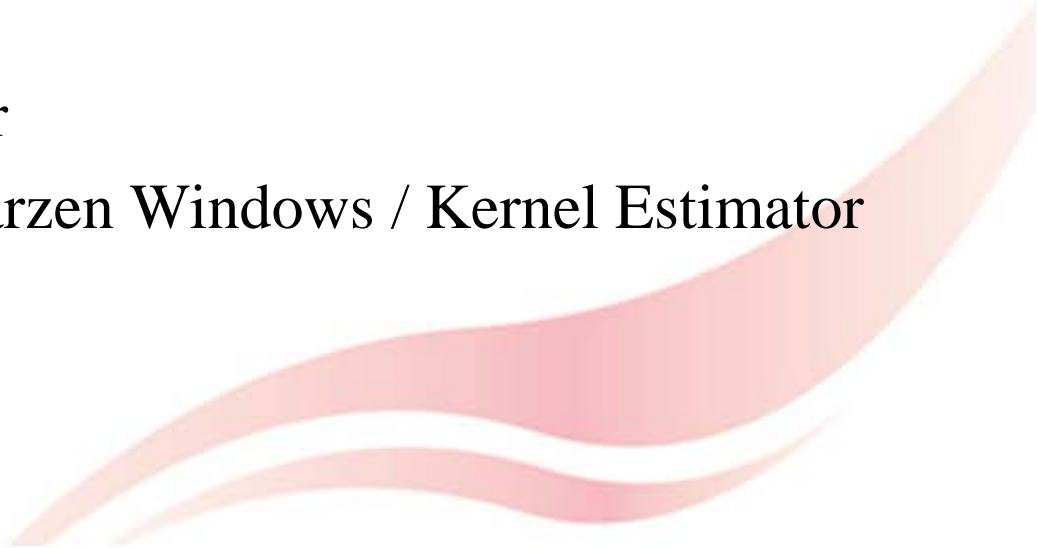
Unbiased estimation:

$$\mathbb{E}[\hat{\boldsymbol{\mu}}] = \boldsymbol{\mu}$$

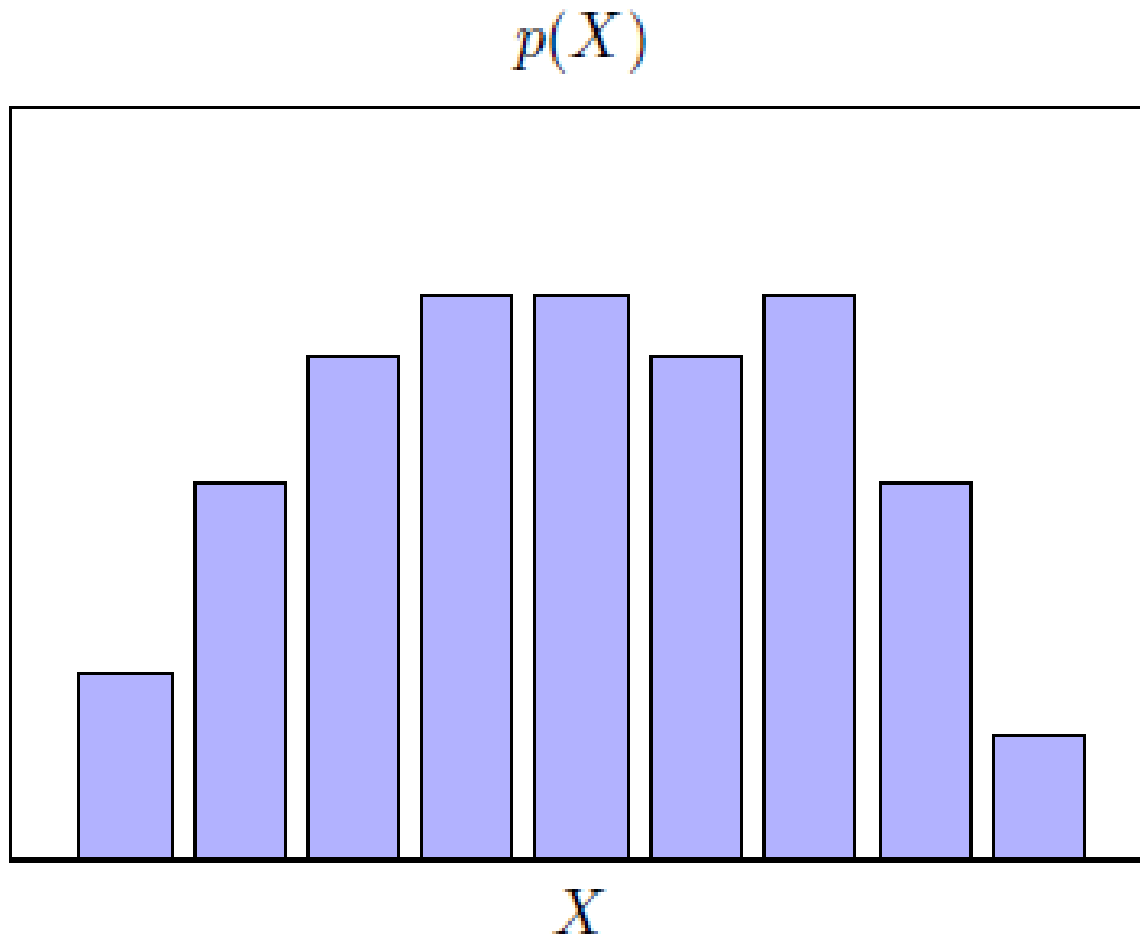
$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T \quad \mathbb{E}[\hat{\boldsymbol{\Sigma}}] = \frac{N-1}{N} \boldsymbol{\Sigma}$$

$$\tilde{\boldsymbol{\Sigma}} = \frac{1}{N-1} \sum_{i=1}^N (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T \quad \text{Unbiased estimation}$$

Non-Parametric Density Estimation

- Assume that $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ are drawn from some unknown probability density $p(\mathbf{x})$
 - Without assuming any forms for the underlying density
 - To learn the estimator $\hat{p}(\mathbf{x})$ for $p(\mathbf{x})$
 - Assume that similar inputs have similar outputs: if \mathbf{x}_i and \mathbf{x}_j are similar, then $p(\mathbf{x}_i)$ and $p(\mathbf{x}_j)$ are similar
 - Approaches
 - Histogram Estimator
 - Naïve Estimator / Parzen Windows / Kernel Estimator
- 

Histogram Estimator



Histogram Estimator (cont.)

- Simply partition \mathbf{x} (suppose 1-dimensional for simplicity) into distinct bins of width Δ_t
- Count the number N_t of data instances falling into bin t
- Turn this count into a normalized probability density via dividing by the total number of observed data points N and by the width Δ_t of the bins:

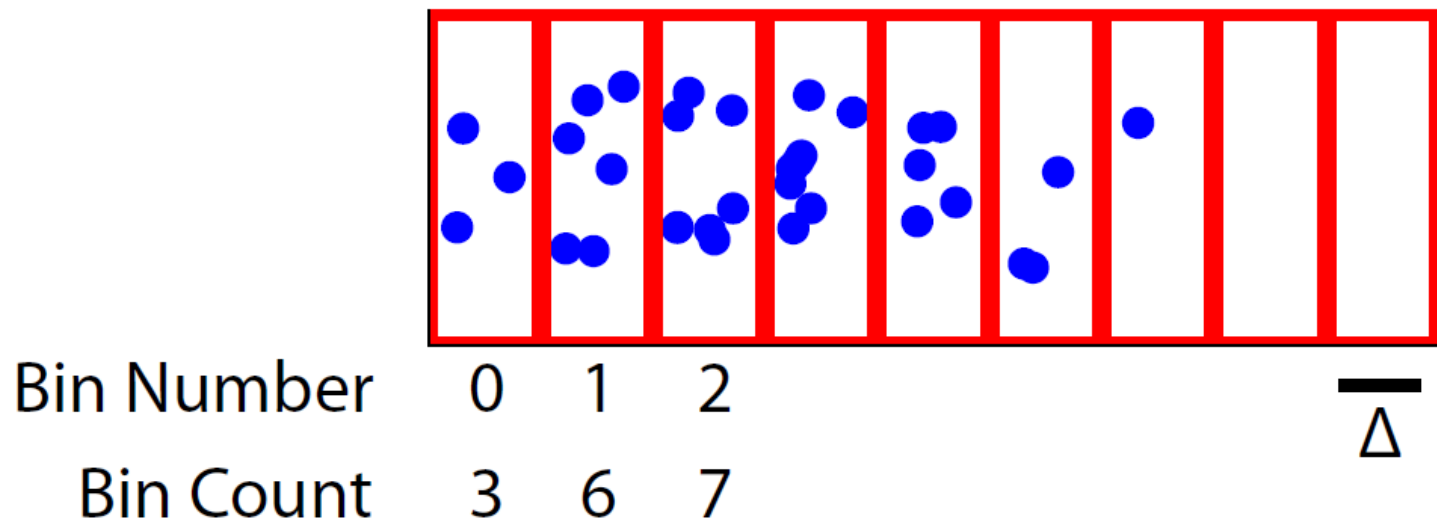
$$p_t = \frac{N_t}{N\Delta_t} \quad \begin{array}{c} \text{Suppose all the bins have} \\ \text{the same width } \Delta_t = \Delta \end{array} \longrightarrow p_t = \frac{N_t}{N\Delta}$$

- The model for the density $p(\mathbf{x})$ is constant over the width of each bin: find the bin where \mathbf{x} is in (e.g., bin t), then

$$\hat{p}(\mathbf{x}) = \frac{\#\{\mathbf{x}_i \text{ in the same bin as } \mathbf{x}\}}{N\Delta} = p_t$$

Histogram Estimator (cont.)

- Typically, the bins are chosen to have the same width $\Delta_t = \Delta$



$$p_t = \frac{N_t}{N\Delta}$$

Why divided by Δ ?



Histogram Estimator (cont.)

- For a specific bin denoted by t , the probability that a data instance \mathbf{x} falls into the bin (e.g., $(x^{(t)}, x^{(t)} + \Delta]$) is

The start value
of the bin t

$$P = \int_{x^{(t)}}^{x^{(t)} + \Delta} p(\mathbf{x}') d\mathbf{x}' \approx \boxed{p(\mathbf{x})\Delta}$$

||

- On the other hand,

$$P = \frac{\#\{\mathbf{x}_i \text{ in the bin } i\}}{N} = \boxed{\frac{N_t}{N}}$$

- Therefore:

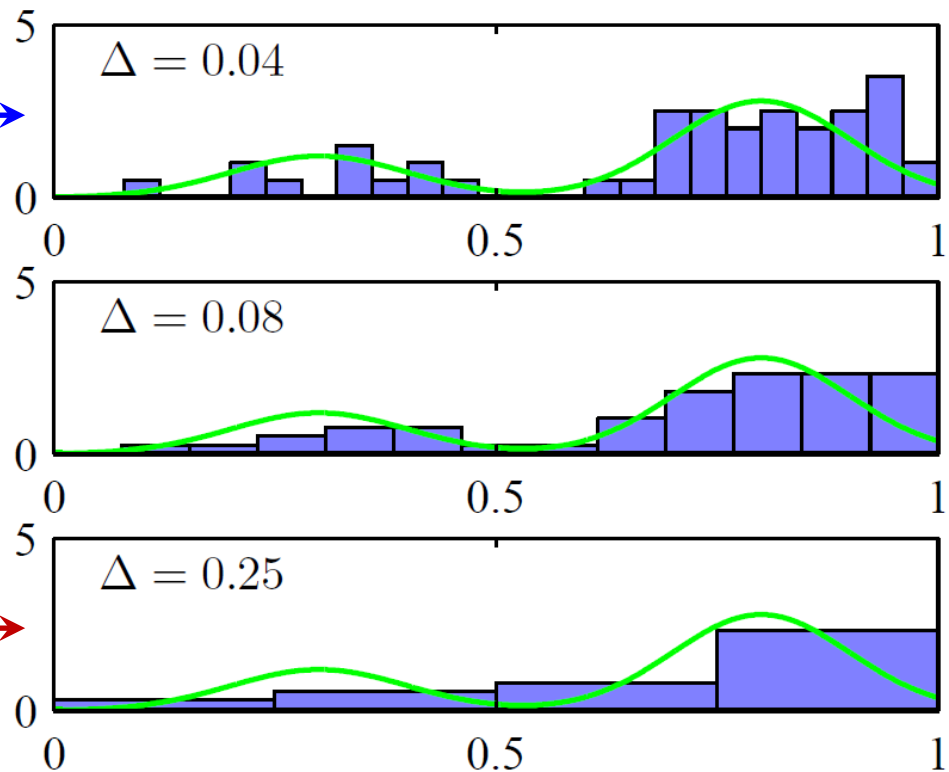
$$p(\mathbf{x}) = \frac{N_t}{N\Delta}$$

Histogram Estimator (cont.)

- Histogram density as a function of bin width Δ

The green curve is the underlying true density from which the data points were drawn

When Δ is very small, the resulting density is quite spiky and hallucinates a lot of structure not present in the true density



When Δ is very big, the resulting density is quite smooth and consequently fails to capture the bimodality of the true density

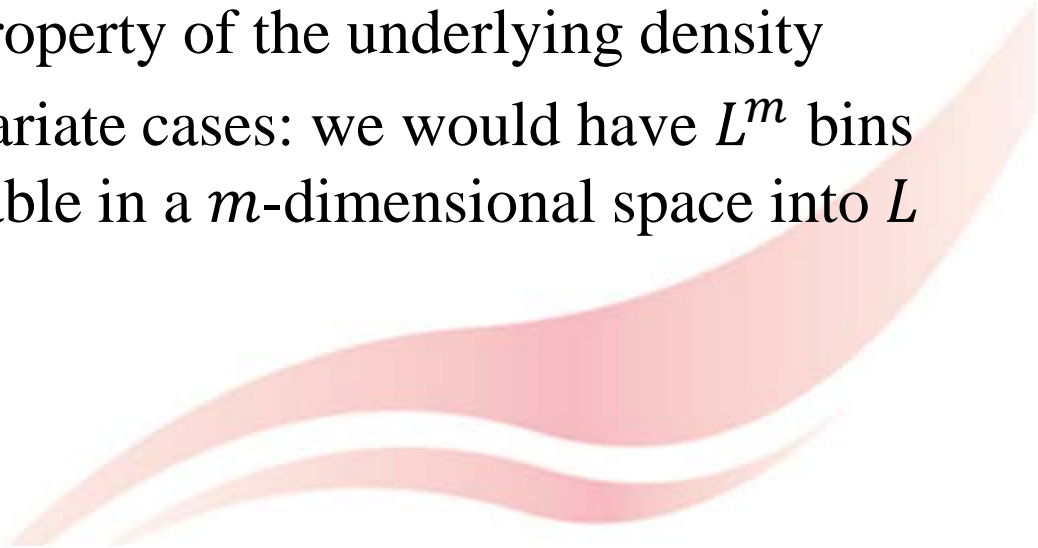


Discussion

- Advantages:

- Simple to evaluate and simple to use
- One can throw away \mathcal{D} once the histogram is computed
- Can be updated incrementally

- Disadvantages:

- The estimated density has discontinuities due to the bin edges rather than any property of the underlying density
 - Scales poorly to multivariate cases: we would have L^m bins if we divided each variable in a m -dimensional space into L bins
- 

Naïve Estimator: An Alternative


- In Histogram Estimator, besides Δ , we have to choose an origin x_0 as well, the bins are the intervals defined as

$$(x_0 + \boxed{t}\Delta, x_0 + (t + 1)\Delta]$$

 0, positive or negative integers

- The Naïve Estimator does not need to set on origin

$$\hat{p}(x) = \frac{\#\{x_i \text{ in the same bin as } x\}}{N\Delta}$$



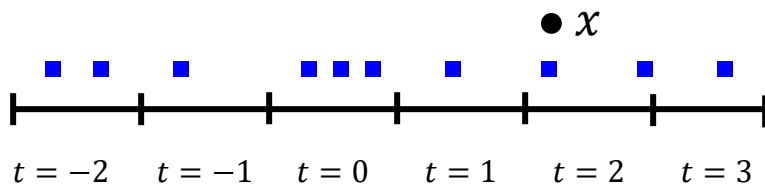
$$\left(-\frac{\Delta}{2}, x, \frac{\Delta}{2}\right]$$

$$\hat{p}(x) = \frac{\#\left\{x - \frac{\Delta}{2} < x_i \leq x + \frac{\Delta}{2}\right\}}{N\Delta}$$

Histogram v.s. Naïve Estimator

Histogram Estimator

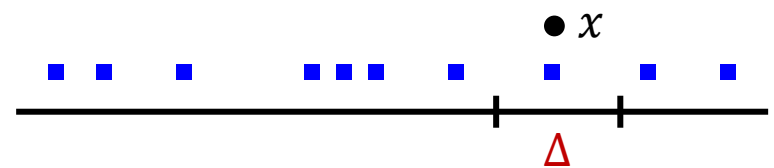
$$p(x) = \frac{2}{10 \times \Delta}$$



Count: 2 1 3 1 2 1

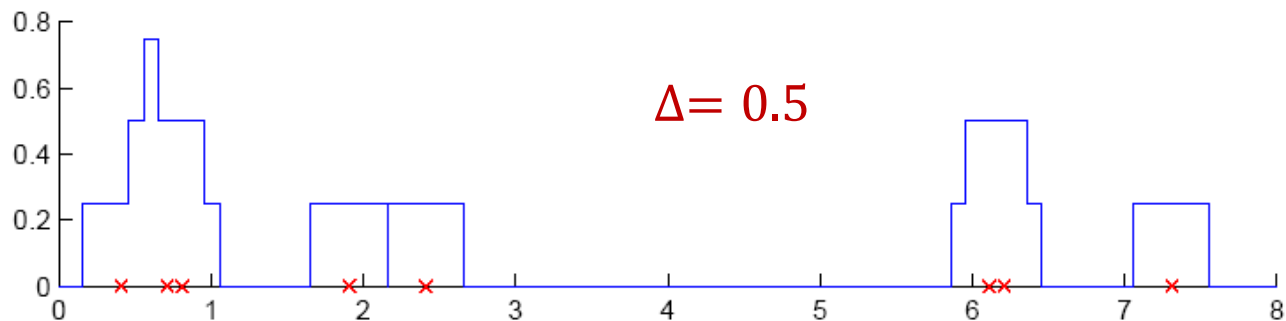
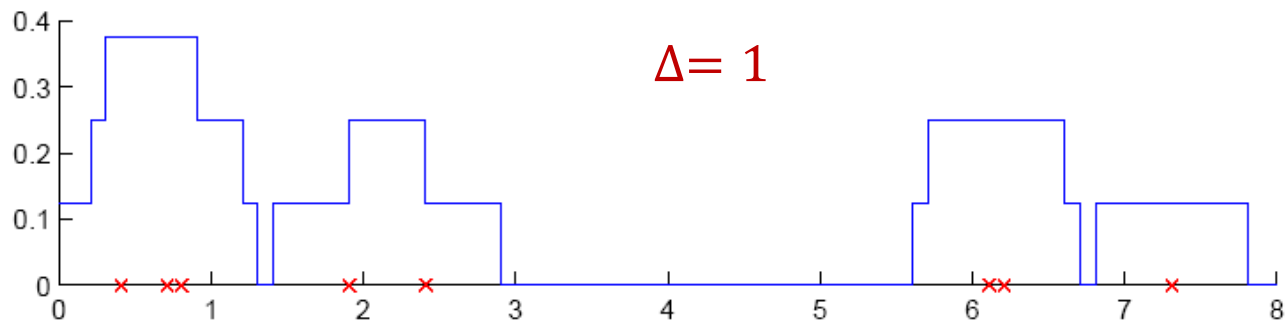
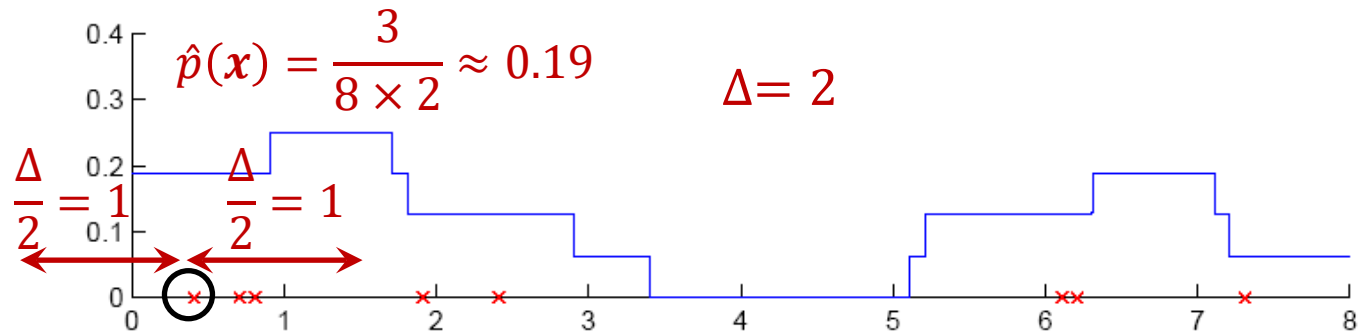
Naïve Estimator

$$p(x) = \frac{1}{10 \times \Delta}$$



1

Naïve Estimator: Example



Parzen Windows

- The Naïve Estimator can also be written as

$$\hat{p}(x) = \frac{\# \left\{ x - \frac{\Delta}{2} < x_i \leq x + \frac{\Delta}{2} \right\}}{N\Delta}$$

x_i is in the bin with width Δ centered at x

$$\hat{p}(x) = \frac{1}{N\Delta} \sum_{i=1}^N w \left(\frac{x - x_i}{\Delta} \right)$$

Windowing function $w(u) = \begin{cases} 1 & \text{if } -\frac{1}{2} \leq u < \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$

Parzen windows

$$-\frac{1}{2} \leq \frac{x - x_i}{\Delta} < \frac{1}{2}$$
$$u = \frac{x - x_i}{\Delta}$$

$$\begin{cases} 1 & \text{if } -\frac{1}{2} \leq u < \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

Multiple Dimensions Case

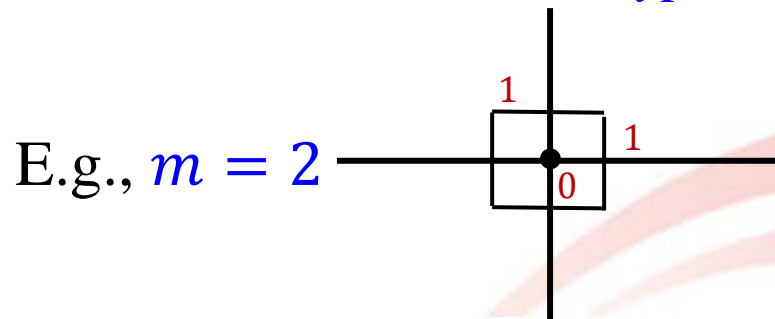
- Given $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, each data instance is m -dimensional
- We define \mathcal{R} is a m -dimensional hypercube with h being the length of each edge. The volume of the hypercube is given by

$$V = h^m$$

- The window function can be defined as

$$w(\mathbf{u}) = \begin{cases} 1 & \text{if } -\frac{1}{2} \leq u_i < \frac{1}{2} \text{ for all } i \in \{1, \dots, m\} \\ 0 & \text{otherwise} \end{cases}$$

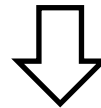
This window function defines a unit hypercube centered at the origin



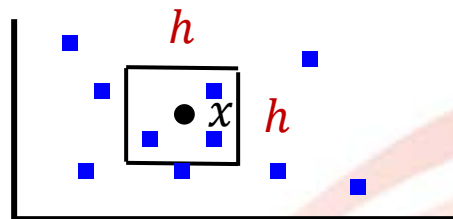
Multiple Dimensions Case (cont.)

- Hence, $w\left(\frac{x-x_i}{h}\right)$ is equal to unity if \mathbf{x}_i falls within the hypercube of volume V centered at \mathbf{x} , and is zero otherwise
- The density estimator can be written as

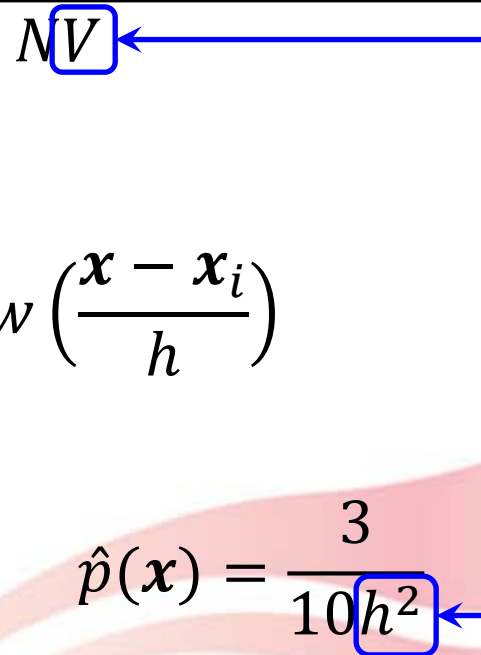
$$\hat{p}(\mathbf{x}) = \frac{\#\{\mathbf{x}_i \text{ in the same hypercube as } \mathbf{x}\}}{NV}$$



$$\hat{p}(\mathbf{x}) = \frac{1}{NV} \sum_{i=1}^N w\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$



$$\hat{p}(\mathbf{x}) = \frac{3}{10h^2}$$



Kernel Estimator

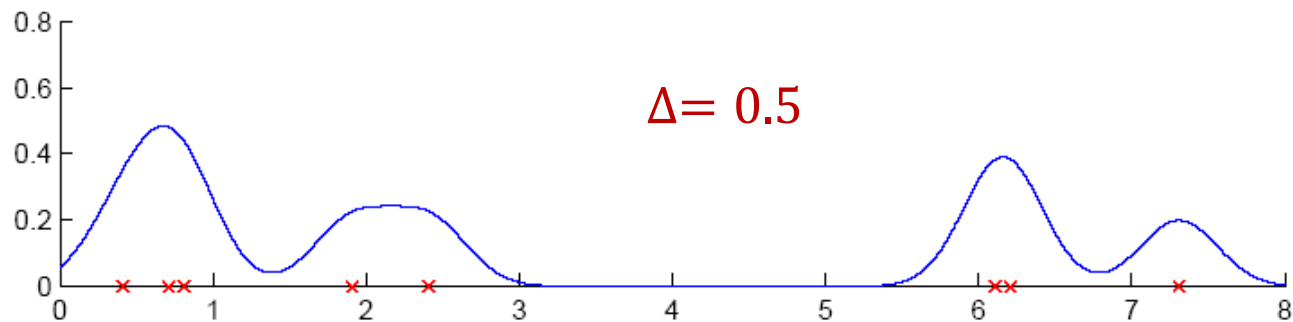
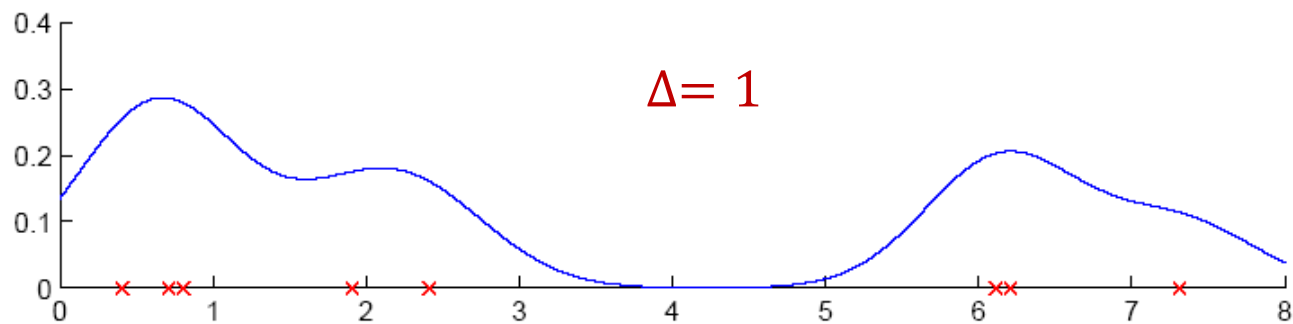
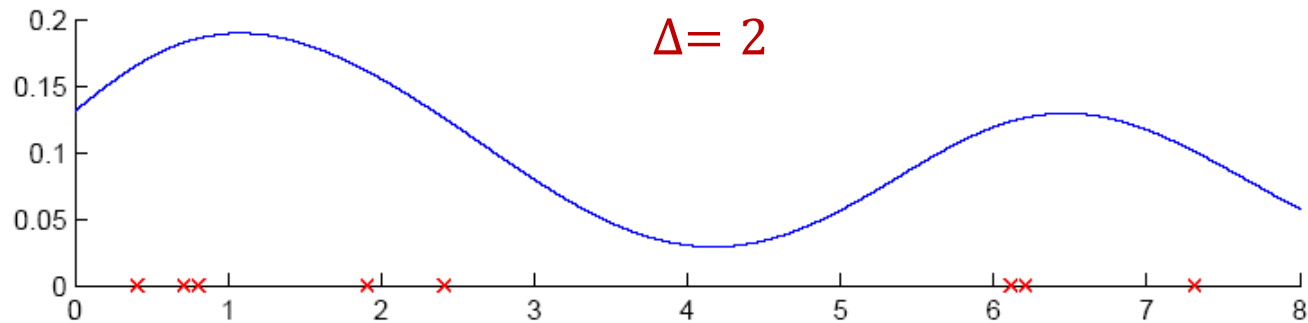
- To get a smooth estimate, we use a smooth weight function, kernel function, e.g., the Gaussian kernel
- For the univariate case, i.e., each data point is 1-dimensional, the Gaussian kernel is defined as

$$k(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$$

- The Kernel Estimator is computed via

$$\hat{p}(x) = \frac{1}{N\Delta} \sum_{i=1}^N k\left(\frac{x - x_i}{\Delta}\right)$$

Kernel Estimator: An Example

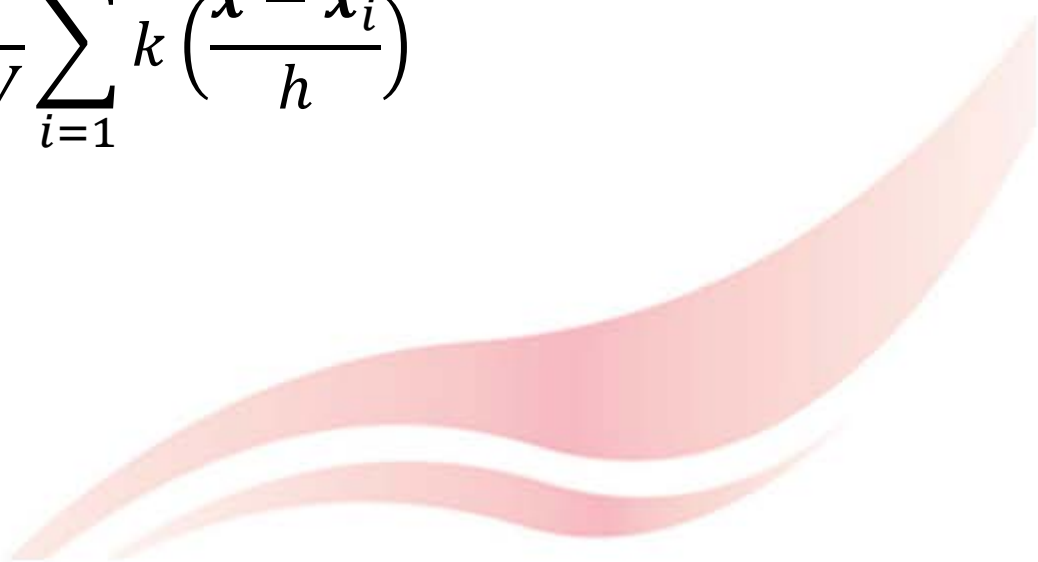


Kernel Estimator (cont.)

- For the multivariate case, i.e., each data instance is m -dimensional, the Gaussian kernel is defined as

$$k(\mathbf{u}) = \frac{1}{(2\pi)^{m/2}} \exp\left(-\frac{\|\mathbf{u}\|_2^2}{2}\right)$$

- The Kernel Estimator is computed via

$$\hat{p}(\mathbf{x}) = \frac{1}{NV} \sum_{i=1}^N k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$


Implementation using scikit-learn

- API: `sklearn.neighbors.KernelDensity`

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html#sklearn.neighbors.KernelDensity>

`sklearn.neighbors.KernelDensity`

```
class sklearn.neighbors.KernelDensity(*, bandwidth=1.0, algorithm='auto', kernel='gaussian', metric='euclidean', atol=0, rtol=0, breadth_first=True, leaf_size=40, metric_params=None)
```

[\[source\]](#)

Kernel Density Estimation.

Read more in the [User Guide](#).

Parameters:	bandwidth : <i>float</i> The bandwidth of the kernel.
	algorithm : <i>str</i> The tree algorithm to use. Valid options are ['kd_tree' 'ball_tree' 'auto']. Default is 'auto'.
	kernel : <i>str</i> The kernel to use. Valid kernels are ['gaussian' 'tophat' 'epanechnikov' 'exponential' 'linear' 'cosine'] Default is 'gaussian'.

Example

```
>>> from sklearn.neighbors import KernelDensity  
>>> import numpy as np
```

```
>>> n_samples, n_features = 10, 5  
>>> rng = np.random.RandomState(0)  
>>> X = rng.randn(n_samples, n_features)
```

```
>>> kde = KernelDensity( kernel='gaussian', bandwidth=0.5 )  
>>> kde.fit(X)
```

```
>>> log_den_X= kde.score_samples(X)
```

Return log density of X



Thank you!

