Machine Intelligence

9. Supervised Learning Part III: Other Algorithms and Evaluation

Álvaro Torralba

AALBORG UNIVERSITET

Fall 2022

Thanks to Thomas D. Nielsen and Jörg Hoffmann for slide sources

Our Agenda for this Chapter

- **Supervised Learning using probabilistic models**.
  → **How to use Bayesian Networks for learning.**

- **Case-based Reasoning**: is there any alternative to Neural Networks?
  → **Yet another alternative for supervised learning**

- **Overfitting**: What happens when learning too much?
  → **A common problem in supervised learning.**

- **Training ML Models**: How to avoid overfitting?
  → **How to split data into train and test set.**

- **Evaluating the Quality of Classification Models**:
  → **Can we trust the learned models?**

Naive Bayes Classifier: An example

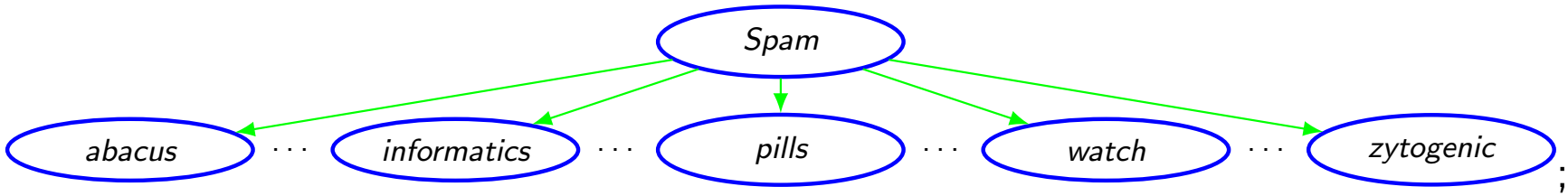
How to detect spam e-mails? Input features: Word occurrence in emails (thousands of input features, one per possible word!):

Mail	abacus	...	informatics	...	pills	...	watch	...	zytogenic	Spam
m_1	n	...	y	...	n	...	n	...	n	no
m_2	n	...	n	...	n	...	n	...	n	yes
m_3	n	...	n	...	n	...	n	...	y	no
m_4	n	...	n	...	n	...	n	...	n	yes
m_5	n	...	n	...	n	...	y	...	n	yes
m_6	n	...	n	...	n	...	n	...	n	yes
m_7	n	...	n	...	n	...	n	...	n	no
m_8	n	...	n	...	n	...	n	...	n	yes
m_9	n	...	y	...	n	...	y	...	n	yes

Observation: We can use **Bayesian Networks as ML model**
In this example: Classify email as *spam* if

$$P(Spam = yes \mid \mathbf{X} = \mathbf{x}) > threshold$$

($\mathbf{X} = (abacus, \ldots, zytogenic)$, $\mathbf{x}$ a corresponding set of y/n values)
We will assume the simplest structural assumption:



;

$$P(a_1, \ldots, a_n, Spam) = P(a_1 \mid Spam) \cdot P(a_2 \mid Spam) \cdots P(a_n \mid Spam) \cdot P(Spam)$$

Making Predictions Using Naive Bayes

Building Naive Bayes as a ML model:

- Training corresponds to setting up the probability tables
- Then we can use inference to perform classification
- **Structural assumption**: Use the simplest possible structure (The target attribute is the cause and the input attributes are all independent symptoms)
  - Simplifies inference
  - We do not need to learn dependencies between variables

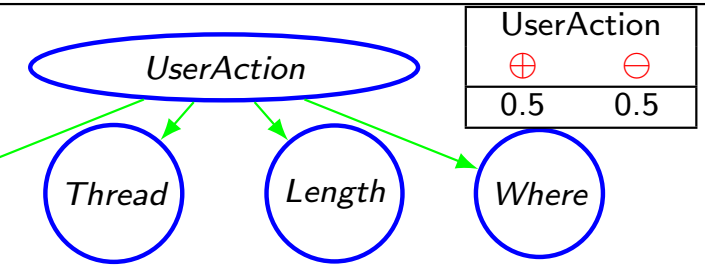
Predictions are given by:

$P(C=\text{true} \mid a_1 \ldots a_n)$

Learning a Naive Bayes

- Need to learn entries in conditional probability tables. **Use empirical frequencies!**

Ex	Author	Thread	Length	WhereRead	UserAction
e_1	known	new	long	home	skips
e_2	unknown	new	short	work	reads
e_3	unknown	follow Up	long	work	skips
e_4	known	follow Up	long	home	skips
e_5	known	new	short	home	reads
e_6	known	follow Up	long	work	skips
e_7	unknown	follow Up	short	work	skips
e_8	unknown	new	short	work	reads
e_9	known	follow Up	long	home	skips
e_{10}	known	new	long	work	skips
e_{11}	unknown	follow Up	short	home	skips
e_{12}	known	new	long	work	skips
e_{13}	known	follow Up	short	home	reads
e_{14}	known	new	short	work	reads
e_{15}	known	new	short	home	reads
e_{16}	known	follow Up	short	work	reads
e_{17}	known	new	short	home	reads
e_{18}	unknown	new	short	work	reads



Act	Author kn	Author unk
⊕	0.7	0.3
⊖	0.7	0.3

Act	Thread new	Thread fol
⊕	0.8	0.2
⊖	0.3	0.7

Act	Length lng	Length sho
⊕	0	1
⊖	0.8	0.2

Act	Where hom	Where wrk
⊕	0.4	0.6
⊖	0.4	0.6

Probabilities to estimate:

Using the Classifier in New Instances

In order to classify a new instance

[Author=unknown, Thread=followUp, Length=short, Where=home]

we calculate

$$P(\text{skips} \mid \text{unknown,followUp,short,home})$$

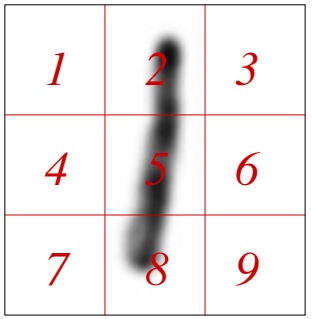
Using the method from **Chapter 6**, we have

However, is this probability reliable?

The Naive Bayes Assumption

Naive Bayes Assumption: All Input attributes are independent given the Target

Example: Given Input Attributes *Cell-1*,. . . ,*Cell-9* $\in \{b, w\}$, predict whether *Symbol*=1.



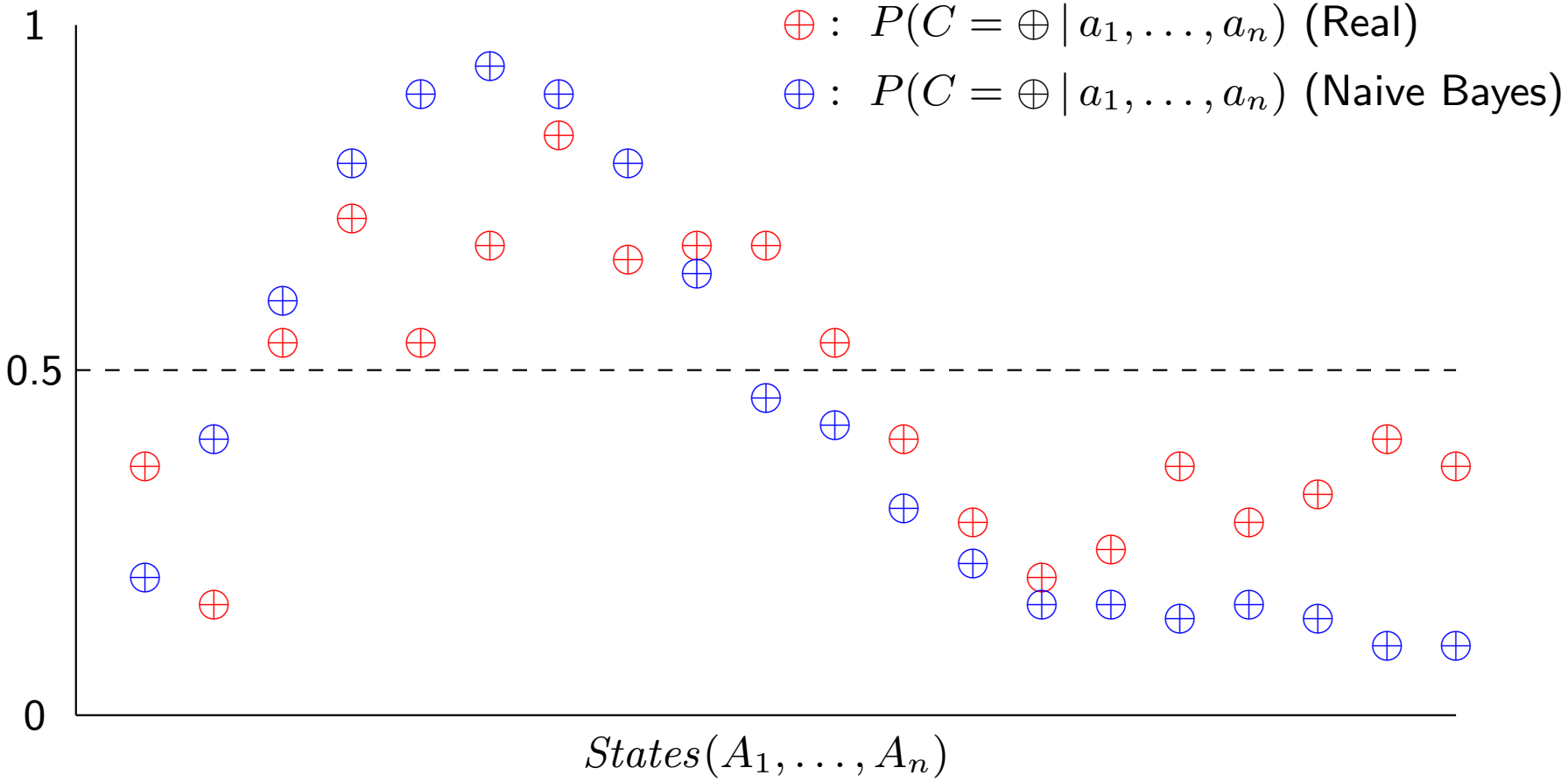
Question!

Is the Naive Bayes Assumption Realistic?

(A): Never
(B): Almost Never
(C): Almost Always
(D): Always

The Paradoxical Success of Naive Bayes

One explanation for the surprisingly good performance of Naive Bayes in many domains: do not require exact distribution for classification, only the right decision boundaries [Domingos, Pazzani 97]



Limitations of Naive Bayes

As happened with Linear Classifiers, no Naive Bayes Classifier can learn a XOR classification:

A	B	Class
yes	yes	$\oplus$
yes	no	$\ominus$
no	yes	$\ominus$
no	no	$\oplus$

Proof (for reference): Assume it did, then:

1. $P(A = y \mid \oplus)P(B = y \mid \oplus)P(\oplus) > P(A = y \mid \ominus)P(B = y \mid \ominus)P(\ominus)$
2. $P(A = y \mid \ominus)P(B = n \mid \ominus)P(\ominus) > P(A = y \mid \oplus)P(B = n \mid \oplus)P(\oplus)$
3. $P(A = n \mid \ominus)P(B = y \mid \ominus)P(\ominus) > P(A = n \mid \oplus)P(B = y \mid \oplus)P(\oplus)$
4. $P(A = n \mid \oplus)P(B = n \mid \oplus)P(\oplus) > P(A = n \mid \ominus)P(B = n \mid \ominus)P(\ominus)$

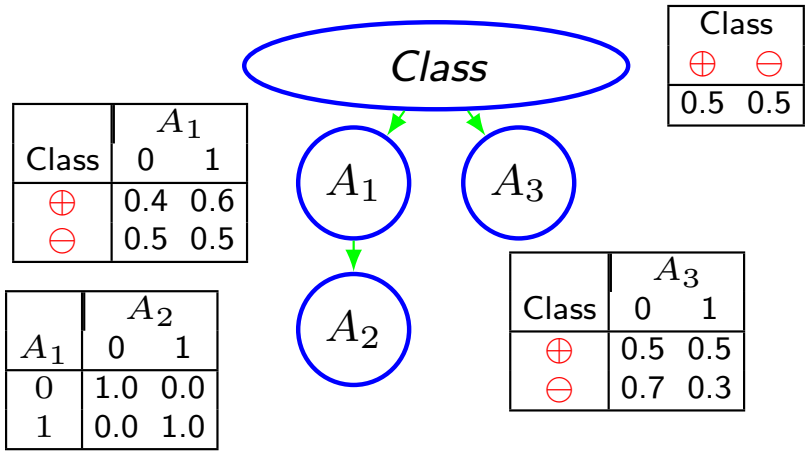
Multiplying the four left sides and the four right sides of these inequalities:

$$\prod_{i=1}^{4}(\textit{left side of } i.) > \prod_{i=1}^{4}(\textit{right side of } i.)$$

But this is false, because both products are actually equal.

Redundant Input Attributes May Be Harmful

Real representation



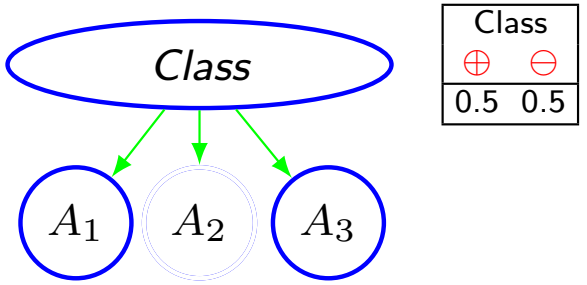
Class	A_1 0	1
⊕	0.4	0.6
⊖	0.5	0.5

A_1	A_2 0	1
0	1.0	0.0
1	0.0	1.0

Class	A_3 0	1
⊕	0.5	0.5
⊖	0.7	0.3

Class ⊕	⊖
0.5	0.5

Naive Bayes representation



Class	A_1 0	1
⊕	0.4	0.6
⊖	0.5	0.5

Class	A_2 0	1
⊕	0.4	0.6
⊖	0.5	0.5

Class	A_3 0	1
⊕	0.5	0.5
⊖	0.7	0.3

Class ⊕	⊖
0.5	0.5

Attribute A_2 is just a duplicate of A_1.

$$P(\oplus \mid A_1 = 1, A_2 = 1, A_3 = 0) = 0.461$$

The Naive Bayes model learned from data:

$$P(\oplus \mid A_1 = 1, A_2 = 1, A_3 = 0) = 0.507$$

Intuitively: the NB model double counts the information provided by A_1, A_2. Recall our previous discussion on the use of intermediate variables (c.f., **Chapter 5**)!

The Naive Bayes model with selected features A_1 and A_3:

$$P(\oplus \mid A_1 = 1, A_3 = 0) = 0.461$$

(and all other posterior class probabilities also are the same as in the true model).

Case-based Reasoning: An intuition

To predict the output feature of a new example e:

- find among the training examples the one (several) most similar to e
- predict the output for e from the known output values of the similar cases

Several names for this approach:

- Instance based learning
- Lazy learning
- Case-based reasoning

Required: **distance measure** on values of input features.

Distance Measures

Distances for numeric features

If all features $\mathbf{X}$ are numeric:

- Euclidean distance: $d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_i (x_i - x'_i)^2}$
- Manhatten distance: $d(\mathbf{x}, \mathbf{x}') = \sum_i |x_i - x'_i|$

Distances for discrete features

For a single feature X with domain $\{x_1, \ldots, x_k\}$:

- Zero-One distance: $d(x_i, x_j) = 0$ if $j = i$, $d(x_i, x_j) = 1$ if $j \neq i$
- Distance matrix: specify for each pair x_i, x_j the distance $d(x_i, x_j)$ in a $k \times k$-matrix. Example:

	low	*medium*	*high*
low	0	2	5
medium	2	0	1
high	5	1	0

For a set of discrete features $\mathbf{X}$:

- Define distance d_i and weight w_i for each $X_i \in \mathbf{X}$
- Define $d(\mathbf{x}, \mathbf{x}') = \sum_i w_i d_i(x_i, x'_i)$

K-Nearest-Neighbor Classifier

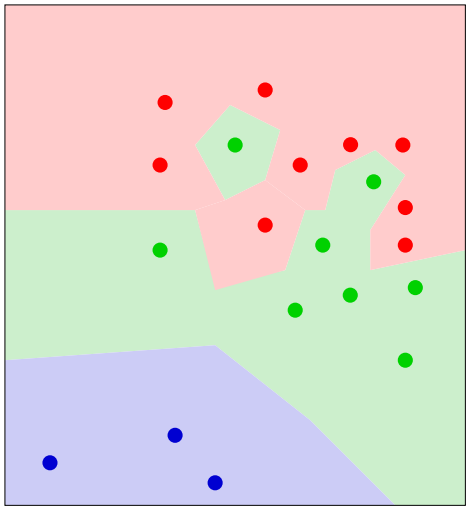
Given

- training examples $(\mathbf{x}_i, y_i)$ $(i = 1, \ldots, N)$
- a new case $\mathbf{x}$ to be classified
- a distance measure $d(\mathbf{x}, \mathbf{x}')$

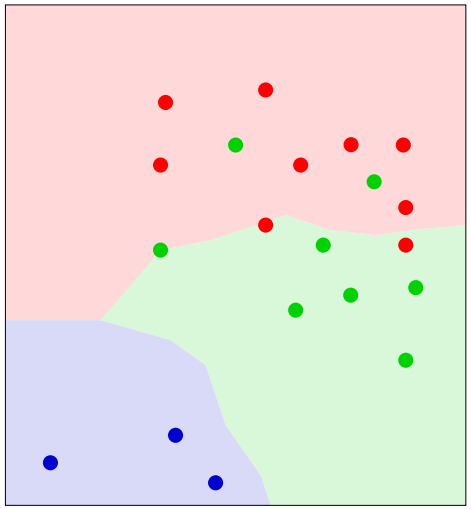
classify $\mathbf{x}$ as follows:

- find the K training examples $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_K}$ with minimial distance to $\mathbf{x}$
- predict for $\mathbf{x}$ the most frequent value in $y_{i_1}, \ldots, y_{i_K}$.

Example



1-Nearest Neighbor

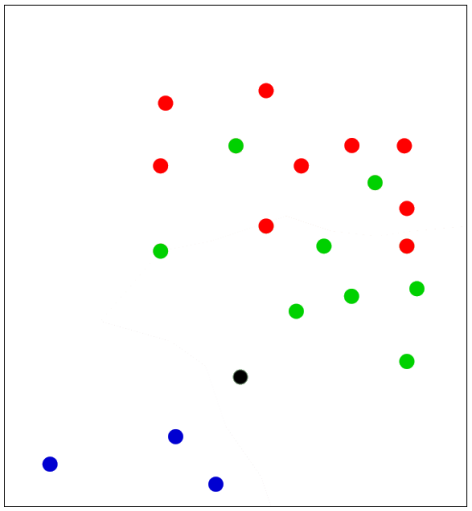


5-Nearest Neighbor

- Output feature: *red*,*blue*,*green*
- Two numeric input features
- Euclidean distance
- Colored dots: training examples
- Colored regions: regions of input values that will be classified as that color

Questionnaire!

If we had a new input, represented as the black point.



Question!

What will be the prediction for the black point when applying 3-Nearest Neighbor?

(A): Red

(B): Blue

(C): Green

(D): Undefined

What is to Learn?

Reminder: Learning Task (Chapter 7)

Find the model in the hypothesis space that minimizes the error on the training data.

This is not true!

Technically, we already know the answer on the training set. What we care about is that the model gives correct predictions on new examples!

Is that even possible? We do not know what the future examples will be!

Question!

What is the next number in the sequence? $1, 3, 5, 7, ?$

(A): 9 **(B):** 11

(C): 13 **(D):** 217341

Generalization and The Big Assumption Behind Machine Learning

Assumption

The Training data and the Future data are taken from the same probability distribution

In other words: **training examples are representative of future examples**.

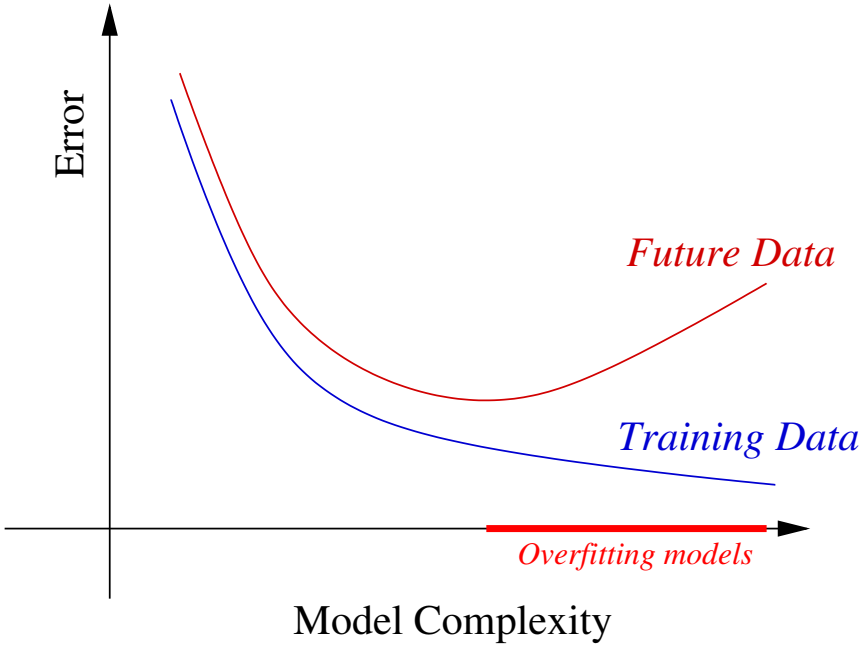
Overfitting

Noise in data may lead to a bad classifier. In particularly, if the decision tree fits the data perfectly. This is called overfitting.

Definition

A hypothesis h is said to overfit the training data if there exists some alternative hypotheses h', such that:

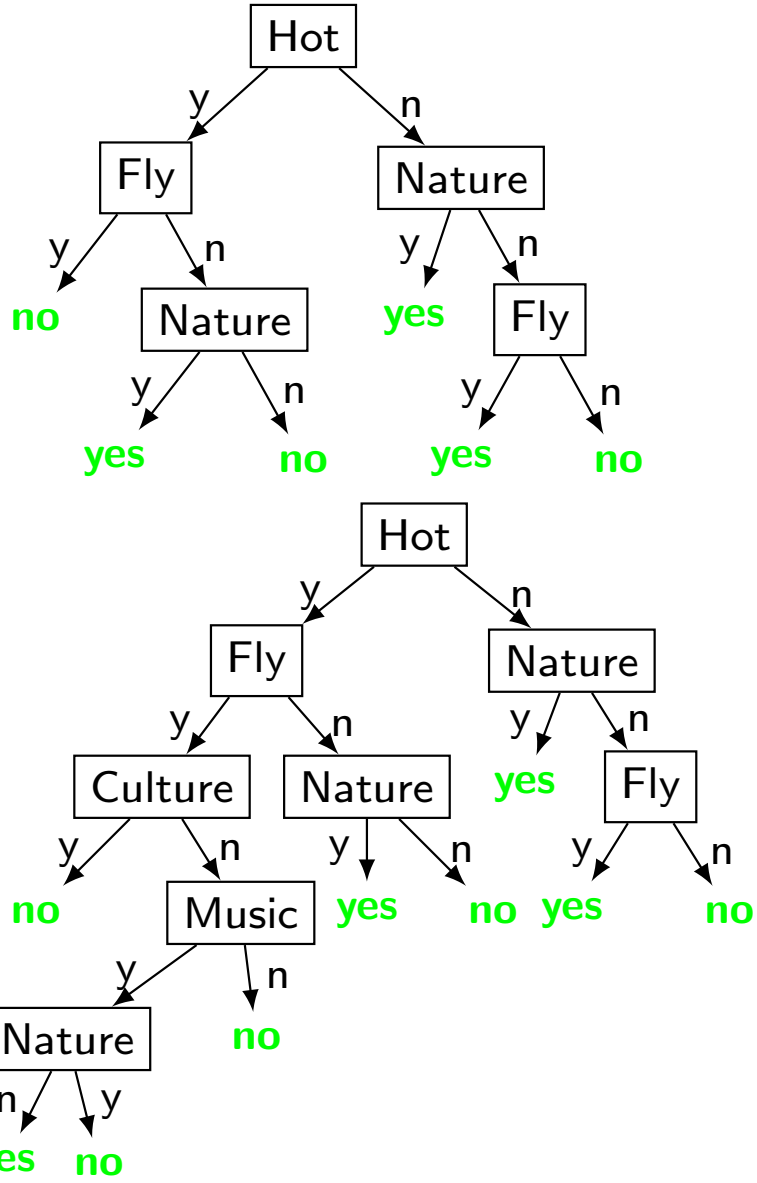
- h has smaller error than h' over the training data, but
- h' has a smaller error than h over the entire distribution of instances.



Overfitting: Decision Trees

Decision tree learned from the holiday data on the left

Culture	Fly	Hot	Music	Nature	Likes
no	no	yes	no	no	no
no	yes	yes	no	no	no
yes	yes	yes	yes	yes	no
no	yes	yes	yes	yes	no
no	yes	yes	no	yes	no
yes	no	no	yes	yes	yes
no	no	no	no	no	no
no	no	no	yes	yes	yes
yes	yes	yes	no	no	no
yes	yes	no	yes	yes	yes
yes	yes	no	no	no	yes
yes	no	yes	no	yes	yes
no	no	no	yes	no	no
yes	no	yes	yes	no	no
yes	yes	yes	yes	no	no
yes	no	no	yes	no	no
yes	yes	yes	no	yes	no
no	no	no	no	yes	yes
no	yes	no	no	no	yes

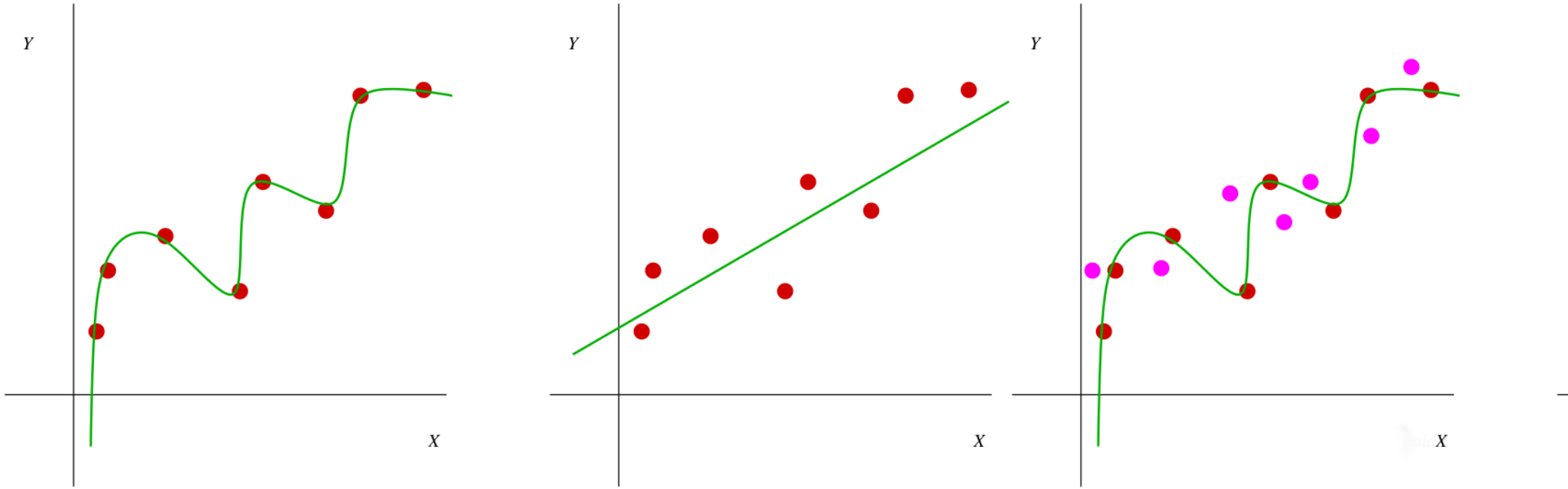


What happen if we add one more example?

Culture	Fly	Hot	Music	Nature	Likes
no	yes	yes	yes	no	yes

Trees provide very accurate representation of training examples, but are they the best trees to predict the preferences of *future* customers?

Overfitting: Neural Networks



Left model: minimizes SSE on training examples

Right model: may have smaller SSE on future observations

Overfitting: Naive Bayes

When learning the naive Bayes model we estimated $P(\text{long}|\text{reads})$ as

$$P(\text{long}|\text{reads}) = \frac{0}{9},$$

based on 9 cases only $\rightsquigarrow$ unreliable parameter estimates and risk of zero probabilities.

Solution: using pseudo counts

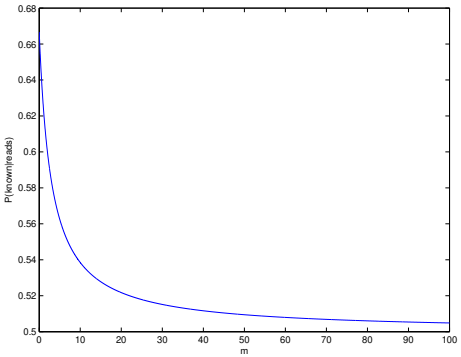
$$P(A = a|C = c) = \frac{N(A = a, C = c) + p_{ac} \cdot m}{N(C = c) + m}$$

where

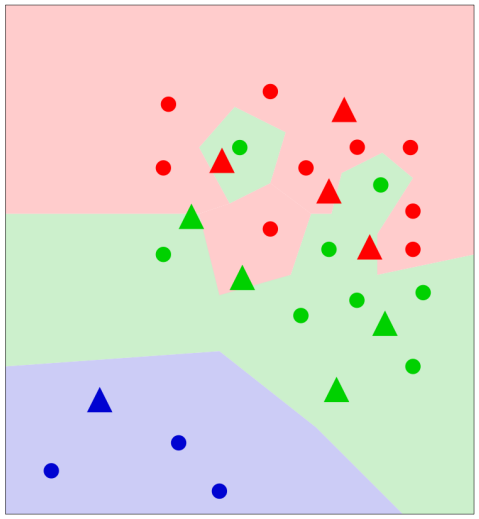
- p_{ac} is our prior estimate of the probability (often chosen as a uniform distribution) and
- m is a virtual sample size (determining the weight of p_{ac} relative to the observed data).

Example

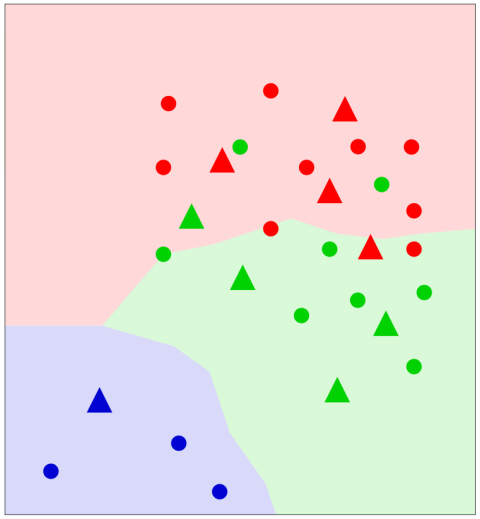
$$P(\text{known}|\text{reads}) = \frac{2 + 0.5 \cdot m}{3 + m}$$



Overfitting: Nearest Neighbor



1-Nearest Neighbor

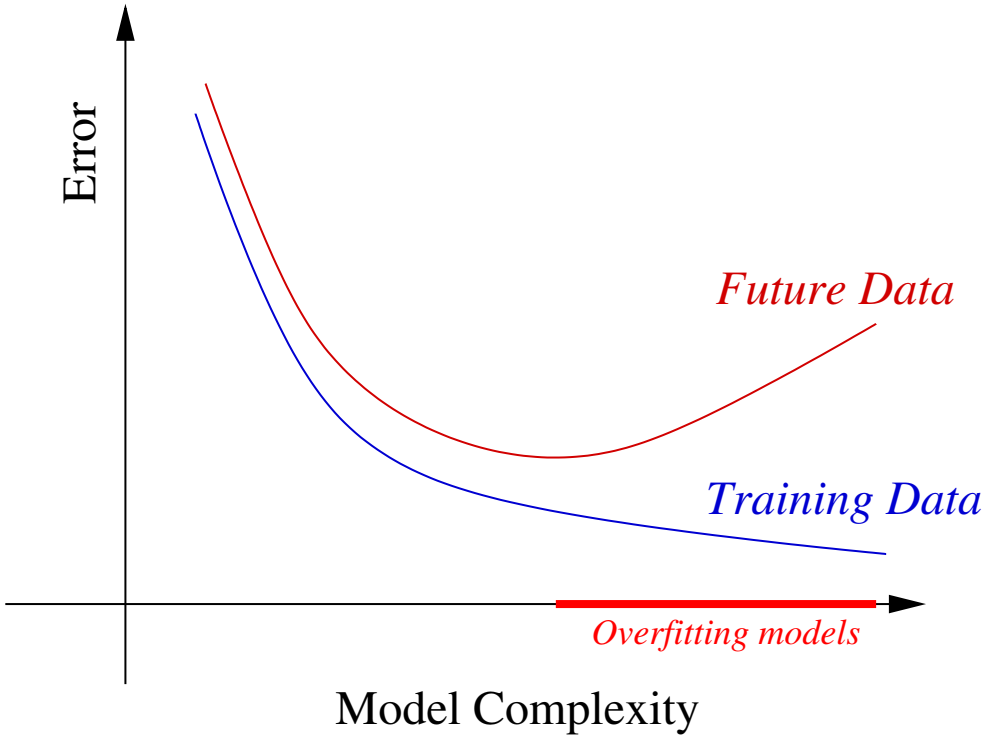


5-Nearest Neighbor

1-Nearest Neighbor correctly classifies all training examples

5-Nearest Neighbor may be better for future observations (triangles)

The General Picture



Model	Complexity	Error
Decision Tree	Depth, # Nodes	Classification error
Neural Network	# hidden nodes/layers	SSE
Nearest Neighbor	Decision regions	Classification error

A model **overfits** the training data, if a smaller error on future data would be obtained by a less complex model.

Avoiding Overfitting

In this section, we see three separate ideas to avoid overfitting:

1. Keep the models simple
2. Evaluate our models to check if they are overfitting
3. Smooth Training Data (Using Pseudo-counts in Naive Bayes, see slide 25)

Avoiding Overfitting by Preferring Simpler Models

Reduced hypothesis space

Do not allow overly complex models by setting up hyperparameters:

- Naive Bayes model (instead of more complex Bayesian models)
- K-NN for "large" K
- Maximum tree size in decision trees

Modified Search/Scoring (Regularization)

Do not allow to learn models that are too complex (relative to the available data):

- Decision Trees: use an early stopping criterion, e.g. stop construction when (sub-) set of training examples contains fewer than k elements (for not too small k).
- Add to evaluation measure a **penalty term** for complexity. This penalty term can have an interpretation as a **prior model probability**, or **model description length**

$\rightarrow$ These techniques will usually lead to more complex models only when the data strongly supports it (especially: large number of examples).

Evaluate our Models to Check if they are Overfitting

Reserve part of the training examples as a **validation set**:

- not used during training
- used as proxy for future data in model evaluation

Simplest approach: split the available data randomly into a **training** and a **validation** set. Typically: 50%/50% or 66%/33% split.

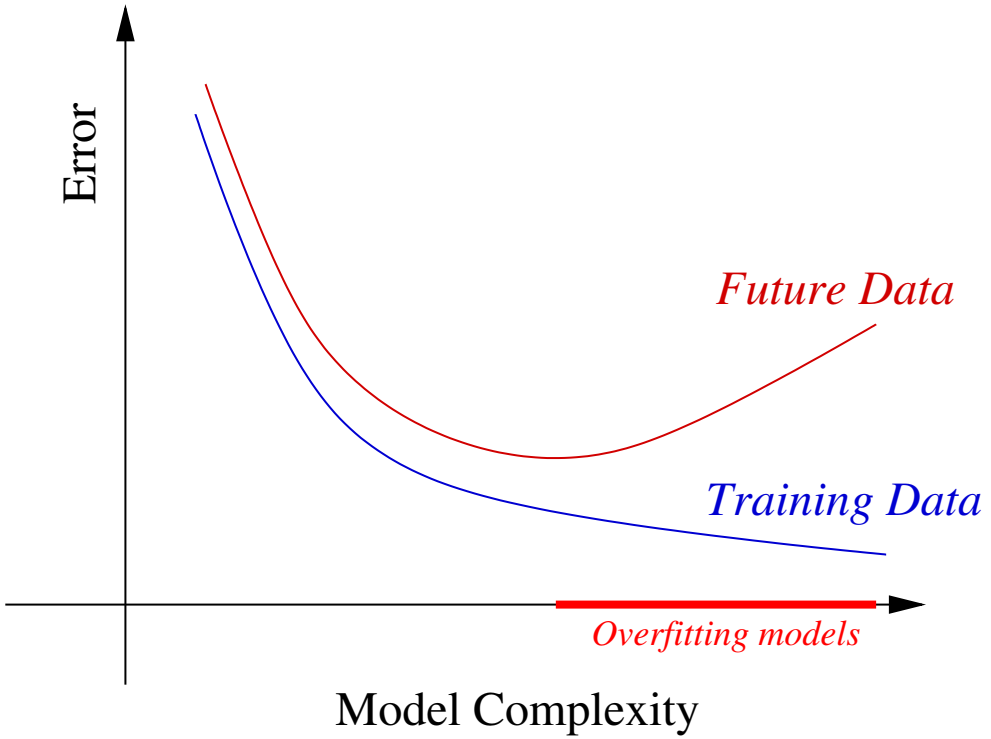
(Note: in some cases a random split is not good. For example, in time-series is better to use the latest data as validation/test set)

Observation: If the validation test is used for tuning the learning algorithm is not really anymore "future" data, so we need yet more data to determine the quality of our final model!

We divide our data into three sub-sets:

- **Training set**: Used for the learning algorithms
- **Validation set**: Used for tuning hyperparameters and/or decide when to stop the training
- **Test set**: Used for evaluating how good the resulting model is (e.g. to report accuracy on the test set)

Using Validation Test as Stopping Condition



- While training we measure error on validation set
- We stop training if error starts increasing

Using Validation Data on Decision Trees

Post pruning

Use of validation set in decision tree learning:

- Build decision tree using *training* set
- For each internal node:
  - test whether accuracy on *validation* set is improved by replacing sub-tree rooted at this node by single leaf (labeled with most frequent target feature value of *training* examples in this sub-tree)
  - if yes: replace sub-tree with leaf (*prune* sub-tree)

Tuning Hyperparameters

Selection of K in K-NN

for $K = 1, 2, 3, \ldots$:

- measure accuracy of K-NN based on *training* examples for *validation* examples

Next:

- use K with best performance on *validation* examples
- *validation* examples can now be merged with *training* examples for predicting future cases

Selection of Neural Network Structure

for $\#hl$ = 1,2,..., max, and $\#nd$ = 1,2,..., max:

- learn Neural Network with $\#hl$ hidden layers and $\#nd$ nodes per hidden layer using *training* examples
- evaluate SSE of learned network on *validation* examples

Next:

- select network structure with minimal SSE
- re-learn weights using merged training and validation examples

Cross-Validation

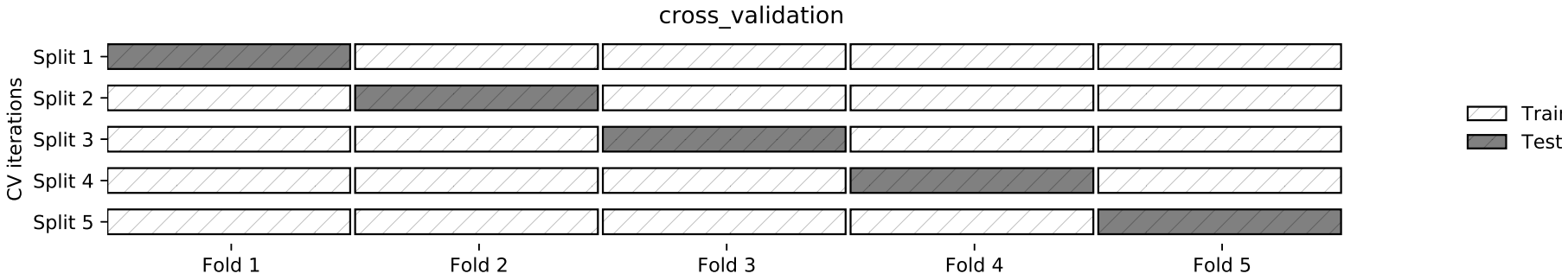
Disadvantage of training/validation splits (for small datasets):

- the examples in the validation set are partly "wasted"
- (unrepresentative) patterns in the validation set can bias the model selection

Cross Validation

The n-**fold cross-validation** approach:

- Divide the examples into n equal sized subsets, or *folds* (e.g. $n = 10$)
- for $i = 1, \ldots, n$:
  - learn model (with given "complexity setting") using folds $1, \ldots, i-1, i+1, \ldots, n$
  - evaluate using fold i
  - average the evaluation scores from the n folds
- Choose "complexity setting" that obtained highest average evaluation score
- Learn final model with chosen "complexity setting" using all available examples



Evaluating the Quality of Classification Models

Accuracy: Measure the percentage of **correct classifications**.

Learned Model A

	Predicted NO	Predicted YES	
Actual: NO	5	10	15
Actual: YES	8	142	150
	13	152	165

Learned Model B

	Predicted NO	Predicted YES	
Actual: NO	0	15	15
Actual: YES	0	150	150
	0	165	165

Learned Model C

	Predicted NO	Predicted YES	
Actual: NO	15	0	15
Actual: YES	20	130	150
	35	130	165

Question!

Which of these classifiers is better?

Confusion Matrix

Four possible outcomes for each test example:

- **TRUE Positive** (TP): the model correctly predicts the positive class
- **TRUE Negative** (TN): the model correctly predicts the negative class
- **FALSE Positive** (FP): the model incorrectly predicts the positive class
- **FALSE Negative** (FN): the model incorrectly predicts the negative class

	Predicted: NO	**Predicted: YES**	
Actual: NO	TN	FP	
Actual: YES	FN	TP	
			n

Therefore **Accuracy** is given by $\frac{TP+TN}{\textbf{Total}}$

$\rightarrow$ But we can also compute other metrics: precision, recall, specificity

Precision

Measure the **estimated performance** when TN cannot be assessed.

Precision is given by $\frac{TP}{\text{Predicted YES}} = \frac{TP}{TP+FP}$

- An issue with accuracy (and specifity) is that TN cannot always be counted. (Example: TN of a Google search query?)
- In those cases, use precision instead: When the model predicts YES, how often is it correct?
- Precision is the estimated probability that a randomly selected retrieved document is relevant.

Recall (True-Positive Rate or Sensitivity)

Measure the performance on YES instances. It measures the **sensitivity** of the model.

Recall is given by $\frac{TP}{\text{Actual YES}} = \frac{TP}{TP+FN}$

- When the true value is YES, how often does the classifier predict YES?
- It is specially important in some applications (like medical applications), where the objective is to detect all positive cases, even if accuracy or precision is affected.
- Other example, recall is the estimated probability that a ramdomly selected relevant document is retrieved in a search

Specificity

Measure the performance on NO instances.

$$\textbf{Specificity is given by } \frac{TN}{\textbf{Actual NO}} = \frac{TN}{TN+FP}$$

- When the true value is NO, how often does the classifier predict NO?
- True negative rate: proportion of actual negatives that the classifier correctly identifies.

Probabilistic Interpretation of the Evaluation Metrics

- **Precision** is the probability that a random patient from all those with a positive test indeed has cancer:

$$\frac{TP}{\text{Predicted YES}} = \frac{P(\text{Cancer} = \text{YES}, \text{Classifier} = \text{YES})}{P(\text{Classifier} = \text{YES})}$$

$$\textbf{P (Real = YES } | \textbf{Prediction = YES)}$$

- **Recall** is the probability of a positive test given that the patient has cancer:

$$\frac{TP}{\text{ActualYES}} = \frac{P(\text{Classifier} = \text{YES}, \text{Cancer} = \text{YES})}{P(\text{Cancer} = \text{YES})}$$

$$\textbf{P (Prediction = YES } | \textbf{Real = YES)}$$

- **Specificity** is the probability of a negative test given that the patient does not have cancer:

$$\frac{TP}{\text{ActualNO}} = \frac{P(\text{Classifier} = \text{NO}, \text{Cancer} = \text{NO})}{P(\text{Cancer} = \text{NO})}$$

$$\textbf{P (Prediction = NO } | \textbf{Real = NO)}$$

F1 Score

- We must consider pairs of measures such as recall/ specificity or recall/ precision for interpreting the performance of a classifier
- **Observation:** Models that have 0 precision or 0 recall are worthless (they simply classify every instance as positive or negative)
- Precision and Recall are two sides of the same coin, can we summarize them in a single metric?

The **F1 score** provides a single measure that summarizes the overall performance:

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- F1 is the harmonic mean between precision and recall
- Maximum value of F1 is 1 (perfect precision and recall)
- Minimum value of F1 is 0 (when recall or precision are 0)
- F1 by design does not take TN into account!

Confusion Matrix with more classes

Predicted fruit / Actual fruit	**Perceived: Apple**	**Banana**	**Orange**	**Clementine**
Apple	10	1	2	2
Banana	5	18	1	2
Orange	8	3	15	11
Clementine	5	3	8	12

$\rightarrow$ We need to compute precision and recall for each individual class!

Summary

- **Naive Bayes** is a simple way of using probabilistic reasoning to perform classification. Despite making strong assumptions it has good accuracy on many applications.
- **Case-based reasoning** methods perform classification without an expensive training phase
- **Overfitting** is a common problem when the learned model minimizes the error on the training set at expenses of having more error on future examples
- We can address overfitting by keeping our hypothesis space simple, or by using a validation dataset
- There are many metrics to measure the performance of a classifier, such as **accuracy**, **precision**, and **recall**.

Reading

- *Chapter 7.7 Case-Based Reasoning* from the book "Artificial Intelligence:Foundations of Computational Agents (2nd edition)
- *Chapter 7.4 Overfitting* from the book "Artificial Intelligence:Foundations of Computational Agents (2nd edition)
- *Chapter 10.1 Probabilistic learning* (only 10.1.1 and 10.1.2) from the book "Artificial Intelligence:Foundations of Computational Agents (2nd edition)
- Extra Reading: To go much further, you can read the Lecture Notes of the Stanford Course in Machine Learning. In this lecture we cover