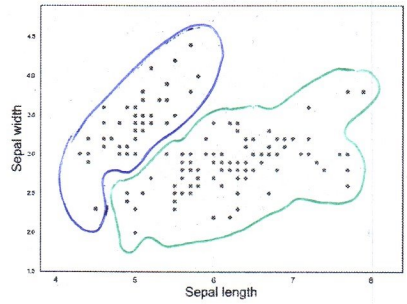
POLITECNICO DI MILANO

Classification

Data Mining and Text Mining

Prof. Pierluca Lanzi



Suppose you receive an unlabeled data point

How would you decide, its class?

Simplest way would be the majority class

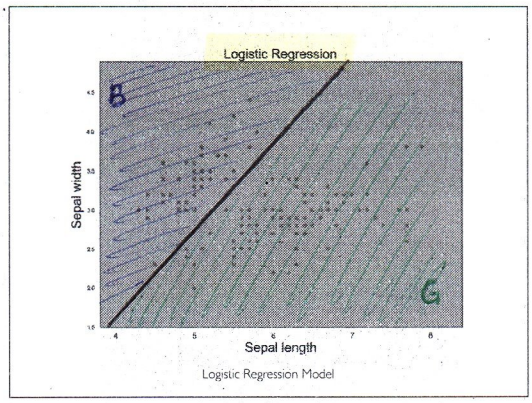
In the example, the number of green data points is larger than the blue ones …

So I might decide to assign a green label to unlabel data points

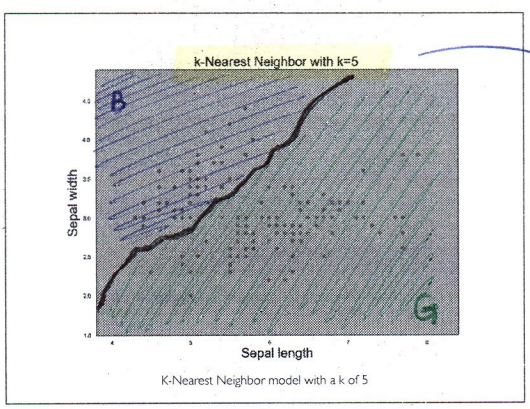
This gives us a baseline performance (majority voting)

— baselines are important because they give a reference to work on

But we can do better than this …

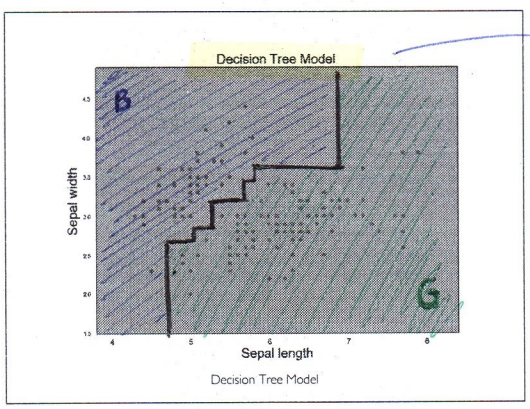


Logistic Regression Model



K-Nearest Neighbor model with a k of 5

we decide a label basing on the majority of the neighbour



Decision Tree Model

this splits the space with hyperplanes that are aligned to the axis of the problem space

As for regression, we must compute a model using known data

And the model should perform well on unknown data.

Contact Lenses Data

10

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	No	Reduced	None
Young	Myope	No	Normal	Soft
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	No	Reduced	None
Young	Hypermetrope	No	Normal	Soft
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	No	Reduced	None
Pre-presbyopic	Myope	No	Normal	Soft
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	No	Reduced	None
Pre-presbyopic	Hypermetrope	No	Normal	Soft
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	No	Reduced	None
Presbyopic	Myope	No	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	No	Reduced	None
Presbyopic	Hypermetrope	No	Normal	Soft
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

TARGET (class / label)

In classification problems …

Rows are typically called examples or data points

Columns are typically called attributes, variables or features

The target variable to be predicted is usually called "class"

A Model for the Contact Lenses Data

12

```
If tear production rate = reduced then recommendation = none
If age = young and astigmatic = no
   and tear production rate = normal then recommendation = soft
If age = pre-presbyopic and astigmatic = no
   and tear production rate = normal then recommendation = soft
If age = presbyopic and spectacle prescription = myope
   and astigmatic = no  then recommendation = none
If spectacle prescription = hypermetrope and astigmatic = no
   and tear production rate = normal then recommendation = soft
If spectacle prescription = myope and astigmatic = yes
   and tear production rate = normal then recommendation = hard
If age young and astigmatic = yes
   and tear production rate = normal then recommendation = hard
If age = pre-presbyopic
   and spectacle prescription = hypermetrope
   and astigmatic = yes then recommendation = none
If age = presbyopic and spectacle prescription = hypermetrope
   and astigmatic = yes then recommendation = none
```


Classification

The target to predict is a label (the class variable)
(good/bad, none/soft/hard, etc.)

the label can be a number but **without** the numerical meaning

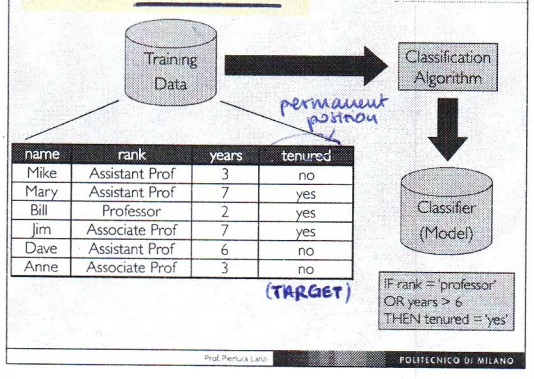
Prediction/Regression

The target to predict is numerical

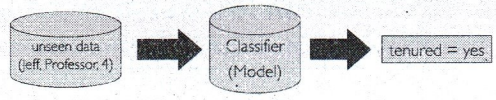
classification = model building + model usage

Classification: Model Construction

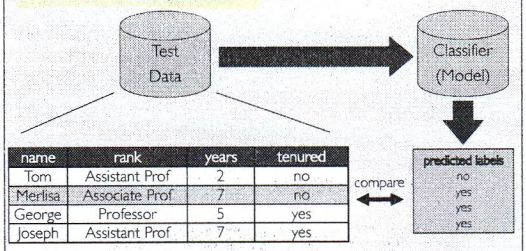
15



Classification: Model Usage



Classification: Model Testing



name	rank	years	tenured
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

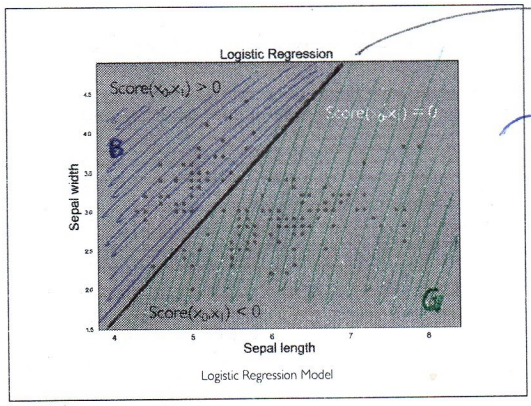
compare

predicted labels
no
yes
yes
yes

Evaluating Classification Methods

- **Accuracy**
  - Classifier accuracy in predicting the correct the class labels
- **Speed**
  - Time to construct the model (training time)
  - Time to use the model to label unseen data
- **Other Criteria**
  - Robustness in handling noise
  - Scalability
  - Interpretability
  - ...

Logistic Regression



Logistic Regression Model

→ hyperplane: $score(x_1, x_2) = 0$

we build an hyperplane to separate the labels

To predict the labels we check whether Score(x_0, x_1) is positive or negative

Then, we label the examples accordingly

Intuition Behind Linear Classifiers (Two Classes)

22

- We define a score function similar to the one used in linear regression,

$$Score(\vec{x_i}) = \sum_{j=0}^{D} w_j h_j(\vec{x_i})$$

- The label is determined by the sign of the score value,

$$\hat{y}_i = sign(Score(\vec{x_i}))$$

$$= \begin{cases} +1 & \text{if } Score(\vec{x_i}) \geq 0 \\ -1 & \text{if } Score(\vec{x_i}) < 0 \end{cases}$$

$\vec{x_i}$ = original input
$h_j(\vec{x_i})$ = preprocessed input

Logistic Regression

23

- Well-known and widely used statistical classification method
- Instead of computing the label, it computes the probability of assigning a class to an example, that is,

$$P(y_i|\vec{x_i})$$

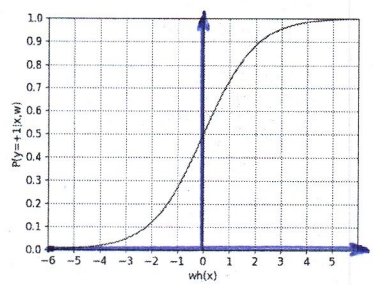
- For this purpose, logistic regression assumes that,

$$P(\hat{y}_i = +1|\vec{x_i}) = \frac{1}{1 + e^{-Score(\vec{x_i})}}$$

- By making the score computation explicit and using h to identify all the feature transformation h, we get,

$$P(\hat{y}_i = +1|\vec{x_i}, \vec{w}) = \frac{1}{1 + e^{-\vec{w}h(\vec{x_i})}}$$

24



Logistic Regression

25

- Logistic Regression search for the weight vector that corresponds to the highest likelihood

$$\ell(\vec{w}) = \prod_{i=0}^{N} P(y_i|\vec{x_i}, \vec{w})$$

- For this purpose, it performs a gradient ascent on the log likelihood function

$$\ell\ell(\vec{w}) = \ln \ell(\vec{w})$$

- Which updates weight j using,

$$\frac{\partial \ell\ell}{\partial w_j} = \sum_{I=1}^{N} h_j(\vec{x_i})(1[y_i = +1] - P(y = +1|\vec{x_i}, \vec{w}))$$

we aim to maximize the product of all the probabilities

Classification using Logistic Regression

- To classify an example x:
  - select the class with the highest probability P(Y=y|x)
  - or check if ratio of probabilities is greater than one:

$$\frac{P(y=+1)|\vec{x})}{P(y=-1)|\vec{x})} = e^{\vec{w}h(\vec{x})} > 1$$

  - choosing the positive class if it is, otherwise, the negative class
- Or equivalently check if natural log of ratio is greater than zero

$$ln(\frac{P(y=+1)|\vec{x})}{P(y=-1)|\vec{x})}) = \vec{w}h(\vec{x}) = \sum_{j=0}^{D} w_j h_j(\vec{x}) > 0$$

  - Note that we're back to a linear classification rule meaning that logistic regression is a linear classifier

Classification using Logistic Regression

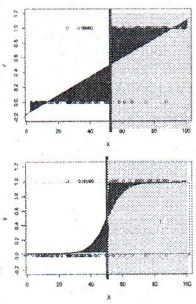
- By taking the natural logarithm of both sides, we obtain a linear classification rule that assign the label Y=-1 if

$$\sum_{i=0}^{D} w_i h_i(x) < 0$$

- Y=+1 otherwise

Linear vs Logistic Regression

- Logistic curve fits better 0/1 data and results in better decision boundary
- Error is monotonic in distance from boundary
  - Red areas indicative of size of error for correctly classified examples
  - Incorrectly classified instances have larger error values
- Note that decision boundary for linear regression is highly sensitive to outliers lying far from boundary
- Logistic regression not sensitive to these value



Overfitting & Regularization

L_1 & L_2 Regularization

- Logistic regression can use L_1 and L_2 regularization to limit overfitting and produce sparse solution
- Like it happened with regression, overfitting is often associated to large weights so to limit overfitting we penalize large weights

Overall Objective= Measure of Fit - Magnitude of Coefficients

we have a "-" because this time we're maximizing (we try to have the highest measure of fit). That's why, to penalize we substract.

- Regularization
  - L_1 uses the sum of absolute values, which penalizes large weights and at the same time promotes sparse solutions
  - L_2 uses the sum of squares, which penalizes large weights

L_1 Regularization

$$\ell(\vec{w}) - \alpha||\vec{w}||_1$$

L_2 Regularization

$$\ell(\vec{w}) - \alpha||\vec{w}||_2^2$$

If α is zero, then we have no regularization

If α tends to infinity,
the solution is a zero weight vector

α must balance fit and the weight magnitude

Example

From now on we are going to use k-fold cross-validation a lot to score models

k-fold cross-validation generates k separate models

Which one should be deployed? ?

K-fold cross-validation provides an evaluation of model performance on unknown data

Its output it's the evaluation, not the model!

The model should be obtained by running the algorithm on the entire dataset!

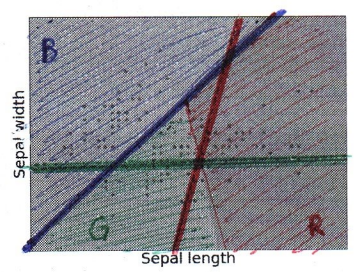
Multiclass Classification

Logistic regression assumes that
there are only two class values (e.g., +1/-1)

What if we have more?
(e.g., none, soft, hard or Setosa/Versicolour/Virginica)

One Versus the Rest Evaluation

- For each class, it creates one classifier that predicts the target class against all the others
- Given three classes A, B, C, it computes three models
  - One that predicts A against B and C
  - One that predicts B against A and C, and
  - One that predicts C against A and B
- Then, given an example, all the three classifiers are applied and the label with the highest probability is returned
- Alternative approaches include the minimization of loss based on the multinomial loss fit across the entire probability distribution



One Versus Rest multiclass model using the Iris dataset

Multiclass Model and loss function

- Alternative approach is to change model itself to be multi-class
- For example, use model that directly predicts 3 (or more) probabilities and predicts multinomial distribution instead of binomial distribution
- Model will have more parameters, 2 vectors for 3 classes, 3 vectors for 4, etc.
- Optimisation routine minimises loss or maximises likelihood over the multiple classes at once

Multiclass Logistic Regression

- Multinomial Logistic Regression model uses the softmax function:

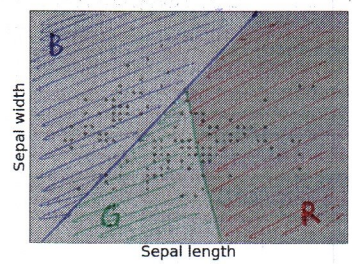
$$P(Y_i = j) = \frac{e^{\vec{w}_j h(\vec{x}_i)}}{\sum_j e^{\vec{w}_j h(\vec{x}_i)}}$$

  - using a weight vector now for each class j

- Don't actually need that many parameters
  - (model is overparameterized)
  - so can set the vector for one class to zero: $\vec{w}_C = \vec{0}$

$$P(Y_i = j) = \frac{e^{\vec{w}_j h(\vec{x}_i)}}{1 + \sum_j e^{\vec{w}_j h(\vec{x}_i)}} \quad \text{if } j \neq C$$

$$P(Y_i = C) = \frac{1}{1 + \sum_j e^{\vec{w}_j h(\vec{x}_i)}}$$



Multinomial multiclass model using the Iris dataset

Classification Metrics

Metrics for Performance Evaluation: Confusion Matrix

- Focus on the predictive capability of a model
- Confusion Matrix:

		PREDICTED CLASS: Yes	PREDICTED CLASS: No
TRUE CLASS	Yes	TP true positives	FN false negatives
	No	FP false positives	TN true negatives

Note: tipically the "positive" is the class we're focusing on

Metrics for Performance Evaluation: Accuracy

		PREDICTED CLASS: Yes	PREDICTED CLASS: No
ACTUAL CLASS	Yes	#TP	#FN
	No	#FP	#TN

- Most widely-used metric:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Sometimes Accuracy is not Enough

46

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10
- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
- Accuracy is misleading because model does not detect any class 1 example

since accuracy alone is not enough

Metrics for Performance Evaluation: Cost

47

		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	Cost(TP)	Cost(FN)
	No	Cost(FP)	Cost(TN)

Cost(x): Cost of misclassifying examples of type x

Computing Cost of Classification

48

cost matrix

Cost Matrix	PREDICTED CLASS		
	C(i,j)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

confusion matrix of model 1

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Accuracy = 80%
Cost = 3910

confusion matrix of model 2

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 90%
Cost = 4255

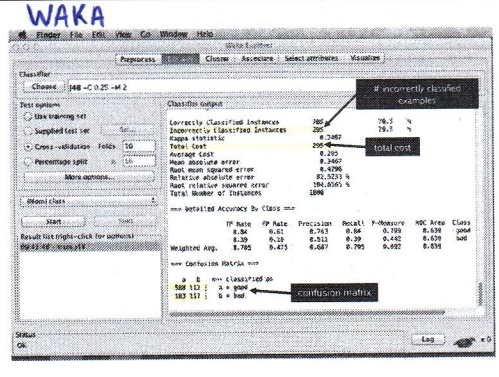
In terms of accuracy the model 2 is better, however its cost is higher than the model 1's.

Costs can be used to evaluate an existing classification models

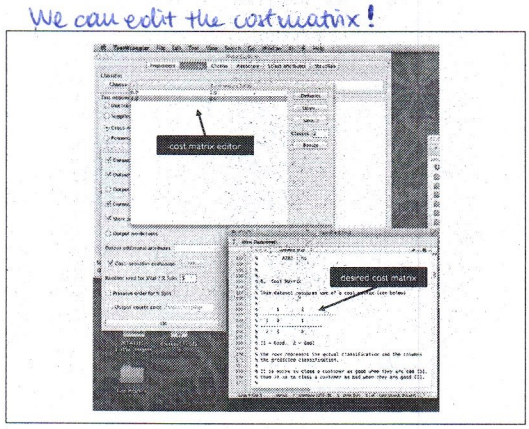
Or can be used by the algorithm to guide the search for the model

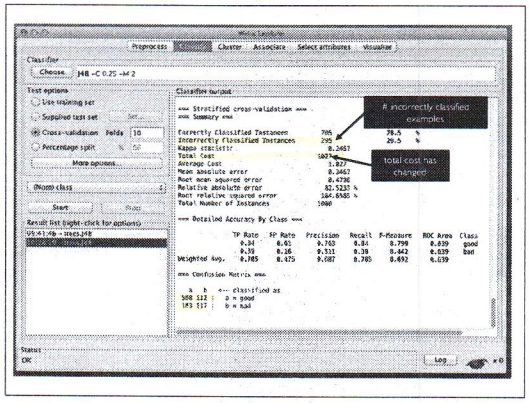
Some algorithms can use the cost matrix to build the model (e.g., decision trees)

WAKA



We can edit the costmatrix!





Precision & Recall

53

- Alternatives to accuracy, introduced in the area of information retrieval and search engine
- **Precision**
  - Percentage of items classified as positive that are actually positive
  - In the information retrieval context represents the percentage of actually good documents that have been shown as a result.
- **Recall**
  - Percentage of positive examples that are classified as positive
  - In the information retrieval context, recall represents the percentage of good documents shown with respect to the existing ones.

Prof. Pierluca Lanzi POLITECNICO DI MILANO

true positive classified positive / all classified positive

true positive classified positive / all true positive

Cost-Sensitive Measures

54

$$Precision(p) = \frac{TP}{TP+FP} = \frac{a}{a+c}$$

The higher the precision, the lower the FPs

$$Recall(r) = \frac{TP}{TP+FN} = \frac{a}{a+b}$$

The higher the recall, the lower the FNs

$$F1-measure = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

The higher the F1, the lower the FPs & FNs

- Precision is biased towards TP & FP
- Recall is biased towards TP & FN
- F1-measure is biased towards all except TN
  it is high when both precision and recall are reasonably high

Prof. Pierluca Lanzi POLITECNICO DI MILANO

p ↑ false positive ↓

r ↑ false negative ↓

F1 ↑ false positive ↓ false negative ↓

Sensitivity & Specificity

55

- Sensitivity evaluates the ability to correctly identify the elements of the positive class and it is computed as the true positive rate (TPR)

  TPR = TP/(TP+FN)

- Specificity estimates the probability to correctly identify the elements of the negative class and it is computed as the true negative rate

  TNR = TN/(TN+FP)

Prof. Pierluca Lanzi POLITECNICO DI MILANO

How to compare the relative performance among competing models?

How to Compare the Performance of Two Models?

- Suppose we have two models
  - Model M_A with an accuracy = 82% computed using 10-fold crossvalidation
  - Model M_B with an accuracy = 80% computed using 10-fold crossvalidation
- How much confidence can we place on accuracy of M_A and M_B?
- Can we say M_A is better than M_B?
- Can the difference in performance measure be explained as a result of random fluctuations in the test set?

How do we know that the difference in performance is not just due to chance?

We computes the odds of it!

Apply the t-test and compute the p-value !

The p-value represents the probability that the reported difference is due to chance

Paired t-Test Evaluation using k-fold Crossvalidation

- Generate the k folds and for each configuration compute the performance of model A and B,

$$\theta_1^A \dots \theta_k^A \qquad \theta_1^B \dots \theta_k^B$$

- compute mean and standard deviation of the differences:

$$\delta_i = \theta_i^A - \theta_i^B \quad \mu_\delta = \frac{1}{k}\sum_i \delta_i \quad \sigma_\delta = \sqrt{\frac{1}{k}\sum_i (\delta_i - \mu_\delta)^2}$$

- And two hypotheses, the null hypothesis and the alternative hypothesis

$$H_0 : \mu_\delta = 0 \qquad H_a : \mu_\delta \neq 0$$

- We apply the t-test to check whether we can reject the null hypothesis H_0 with a target confidence

ALGORITHM 22.4. Paired *t*-Test via Cross-Validation

```
PAIRED t-TEST(α, K, D):
D ← randomly shuffle D
{D_1, D_2, ..., D_K} ← partition D in K equal parts
foreach i ∈ [1, K] do
    M_i^A, M_i^B ← train the two different classifiers on D \ D_i
    θ_i^A, θ_i^B ← assess M_i^A and M_i^B on D_i
    δ_i = θ_i^A − θ_i^B
μ̂_δ = (1/K) Σ_{i=1}^{K} δ_i
σ̂_δ^2 = (1/K) Σ_{i=1}^{K} (δ_i − μ̂_δ)^2
Z_δ^* = √K μ̂_δ / σ̂_δ
if Z_δ^* ∈ (−t_{α/2,K−1}, t_{α/2,K−1}) then
    Accept H_0; both classifiers have similar performance
else
    Reject H_0; classifiers have significantly different performance
```

Comparing the Performance of 2 Models using k-fold Cross-validation

61

- First decide on a confidence level, e.g. 95%
  - Corresponds to false discovery (false positive) rate: $\alpha = 5\%$
  - How frequently you are willing to declare difference when there is none
- Apply k-fold cross-validation to each model
  - Obtaining k evaluations for each algorithm over same folds
- Apply Student's t-test and compute p-value to determine whether reported difference is statistically significant
  - If p-value>α then difference is not significant (can claim nothing)
  - If p-value<α then difference is significant (claim one better than the other)
  - Note that the t-test can be paired or unpaired

"paired" when the estimates are from the same datasets

"unpaired" when the estimates are from different datasets

$\theta_1^A, .., \theta_k^A$ and $\theta_1^B, .., \theta_k^B$ are obtained from the same dataset (we're trying two methods (A, B) and we're deciding which one is better using one dataset)

Multiple Hypothesis Testing

Multiple Testing

46

- Say that you perform a statistical test with a 0.05 threshold, but you repeat the test on twenty different observations.
- For example, you want to compare the performance of several classification algorithms
- Assume that all of the observations are explainable by the null hypothesis
- What is the chance that at least one of the observations will receive a p-value less than 0.05?

! saying 10 statements and after each saying "im sure with 95% probability" is different from saying all the 10 things together and then adding once "i'm sure at 95% probability"

Multiple Testing - Example

47

- Say that you perform a statistical test with a 0.05 threshold (95% confidence level), but you repeat the test on 20 different observations. What is the chance that at least one of the observations will receive a p-value less than 0.05?
- P(making a mistake) = 0.05
- P(not making a mistake) = 0.95
- P(not making any mistake) = $0.95^{20} = 0.358$
- P(making at least one mistake) = 1 - 0.358 = 0.642
- There is a 64.2% chance of making at least one mistake.

Bonferroni Correction

48

- Assume that individual tests are independent.
- Divide the desired p-value threshold by the number of tests performed.
- Example
  - We now have, the threshold set to 0.05/20 = 0.0025.
  - P(making a mistake) = 0.0025
  - P(not making a mistake) = 0.9975
  - P(not making any mistake) = 0.9975^{20} = 0.9512
  - P(making at least one mistake) = 1 - 0.9512 = 0.0488

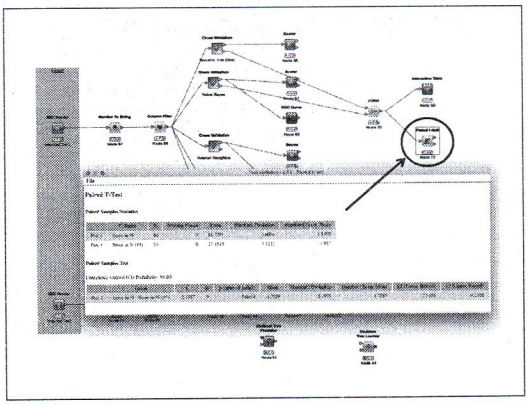
instead of α we use $\frac{\alpha}{\#tests}$

Non Parametric Tests

67

- They do not make any assumption about the distribution of the variable in the population
- Mann-Whitney U Test
  - Nonparametric equivalent of the independent t-test
- Wilcoxon matched-pairs signed rank test
  - Used to compare two related groups

	Nonparametric tests		Parametric tests
	Nominal data	Ordinal data	Ordinal, interval, ratio data
One group	Chi square goodness of fit	Wilcoxon signed rank test	One group t-test
Two unrelated groups	Chi square	Wilcoxon rank sum test, Mann-Whitney test	Student's t-test
Two related groups	McNemar's test	Wilcoxon signed rank test	Paired Student's t-test
K-unrelated groups	Chi square test	Kruskal -Wallis one-way analysis of variance	ANOVA
K-related groups		Friedman matched samples	ANOVA with repeated measurements



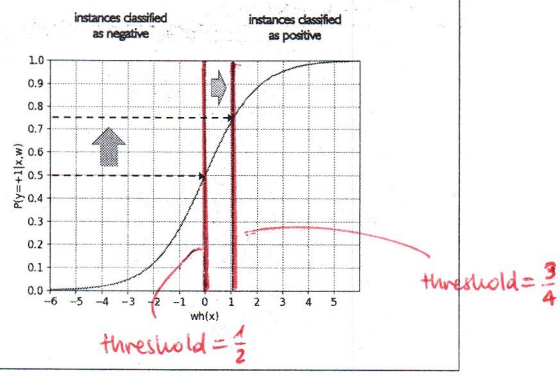
Probabilistic Classifiers

Probabilistic Classifiers & Classification Thresholds

- Up to now we used logistic regression to predict classifier labels, however, logistic regression returns a probability

$$P(y_i|\vec{x}_i)$$

- Given an example x, its predicted class is the label with the largest probability so it is equivalent to using a threshold of 0.5 to decide which class to assign to an example
- However, we can use a different threshold and for instance label as positive only examples we return a 1 only when $P(+1|x)>0.75$
- This would label as positive only cases for which we are more confident that should be labeled as positive.



threshold = $\frac{1}{2}$

threshold = $\frac{3}{4}$

How the Classification Threshold Influence Precision and Recall?

- Suppose we use a near one threshold to classify positive examples
- Then, we will classify as positives only examples for which we are very confident (this is a pessimistic classifier)
- Precision will be high
  - In fact, we are not likely to produce few false positives
- Recall will be low
  - In fact, we are likely to produce many false negatives

How the Classification Threshold Influence Precision and Recall?

- Suppose we use a near zero threshold to classify positive examples
- Then, we will classify everything as positives (this is an optimistic classifier)
- Precision will be low as we are going to generate the maximum number of false positives (everything is positive!)
- Recall will be high since by classifying everything as positive we are going to generate the minimum number of false negatives

We can use the threshold to optimize our precision and recall

a higher the threshold, increases precision and lower recall

a lower threshold, decreases precision and increase recall

Precision-Recall Tradeoff Example

- In the notebook, a simple logistic regression model applied to the loans data returns the confusion matrix below, corresponding to a precision of 0.82 and recall of 0.99

	Classified -1	Classified +1
Labeled -1	1212	21910
Labeled +1	1057	98283

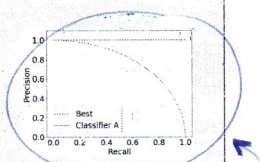
- By increasing the classification threshold for positive (+1) examples to 0.75 we obtained a new confusion matrix with precision of 0.86 and a recall of 0.80

	Classified -1	Classified +1
Labeled -1	10632	12490
Labeled +1	20106	79234

- Overall, we reduced the number of false positives (we did not accept risky loans and we were better at identifying risky loans)

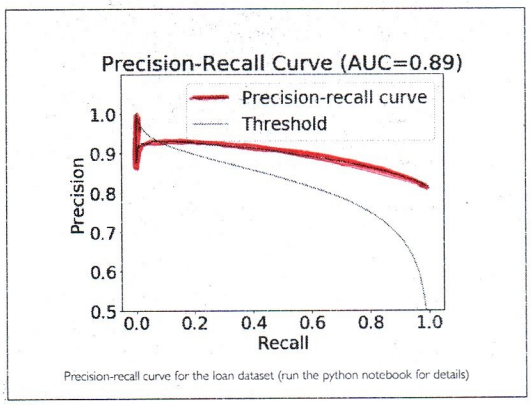
Precision-Recall Curves

- Plot precision as a function of recall for varying threshold values
- The best classifier would be the one that has always a precision equal to one (but never happens)



- More in general classifiers will show of different shapes
- How to decide among more classifiers?
  - Use the area under the curve (the nearer to one, the better)
  - Use F1 measure

we get a function for every classifier



Precision-recall curve for the loan dataset (run the python notebook for details)

Receiver Operating Characteristic (ROC) Curves

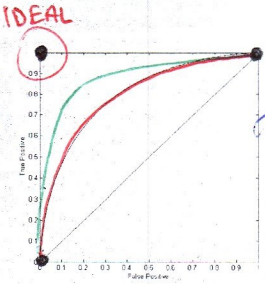
ROC Curve (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze signals
- Plot the True Positive Rate (TPR=TP/(TP+FN)) against the False Positive Rate (FPR=FP/(TN+FP))
- Performance of each classifier represented as a point on the ROC curve
- Changing the classification threshold, sample distribution or cost matrix changes the location of the point

ROC Curve

81

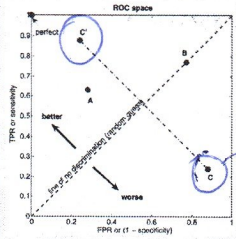
- (FPR, TPR)
  - (0,0): declare everything to be negative class
  - (1,1): declare everything to be positive class
  - (0,1): ideal
- Diagonal line:
  - Random guessing
  - Below diagonal line, prediction is opposite of the true class



The green is better than the red because it's closer to the ideal point

ROC Curve

82



A			
TP=63	FP=28	91	
FN=37	TN=72	109	
100	100	200	

TPR = 0.63
FPR = 0.28
ACC = 0.68

B			
TP=77	FP=77	154	
FN=23	TN=23	46	
100	100	200	

TPR = 0.77
FPR = 0.77
ACC = 0.50

C			
TP=24	FP=88	112	
FN=76	TN=12	88	
100	100	200	

TPR = 0.24
FPR = 0.88
ACC = 0.18

C'			
TP=88	FP=24	112	
FN=12	TN=76	88	
100	100	200	

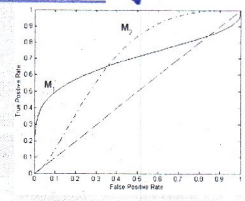
TPR = 0.88
FPR = 0.24
ACC = 0.82

being under the diagonal is **very bad**. By switching the classification we obtain the reflection w.r.t. the diagonal ($C \longrightarrow C'$)

Using ROC for Model Comparison

83

- No model consistently outperform the other
- M_1 is better for small FPR
- M_2 is better for large FPR
- Area Under the ROC curve
- Ideal, area = 1
- Random guess, area = 0.5



There are techniques similar to ROC in other areas

Lift charts are an example

Which model should we prefer?
(Model selection)

Model Selection Criteria

- Model selection criteria attempt to find a good compromise between:
  - The complexity of a model
  - Its prediction accuracy on the training data
- Reasoning: a good model is a simple model that achieves high accuracy on the given data
- Also known as Occam's Razor : the best theory is the smallest one that describes all the facts

William of Ockham, born in the village of Ockham in Surrey (England) about 1285, was the most influential philosopher of the 14th century and a controversial theologian.

Which Classification Algorithm?

- Among the several algorithms, which one is the "best"?
  - Some algorithms have a lower computational complexity
  - Different algorithms provide different representations
  - Some algorithms allow the specification of prior knowledge
- If we are interested in the generalization performance, are there any reasons to prefer one classifier over another?
- Can we expect any classification method to be superior or inferior overall?
- According to the No Free Lunch Theorem, the answer to all these questions is no

No Free Lunch Theorem

- If the goal is to obtain good generalization performance, there are no context-independent or usage-independent reasons to favor one classification method over another
- If one algorithm seems to outperform another in a certain situation, it is a consequence of its fit to the particular problem, not the general superiority of the algorithm
- When confronting a new problem, this theorem suggests that we should focus on the aspects that matter most
  - Prior information
  - Data distribution
  - Amount of training data
  - Cost or reward
- The theorem also justifies skepticism regarding studies that "demonstrate" the overall superiority of a certain algorithm

No Free Lunch Theorem

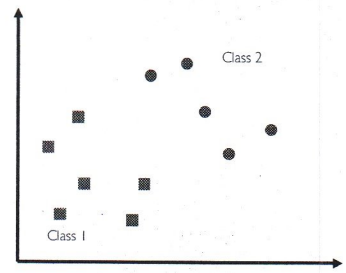
- "[A]ll algorithms that search for an extremum of a cost [objective] function perform exactly the same, when averaged over all possible cost functions." [1]
- "[T]he average performance of any pair of algorithms across all possible problems is identical." [2]
- Wolpert, D.H., Macready, W.G. (1995), No Free Lunch Theorems for Search, Technical Report SFI-TR-95-02-010 (Santa Fe Institute).
- Wolpert, D.H., Macready, W.G. (1997), No Free Lunch Theorems for Optimization, IEEE Transactions on Evolutionary Computation 1, 67.

Additional Material

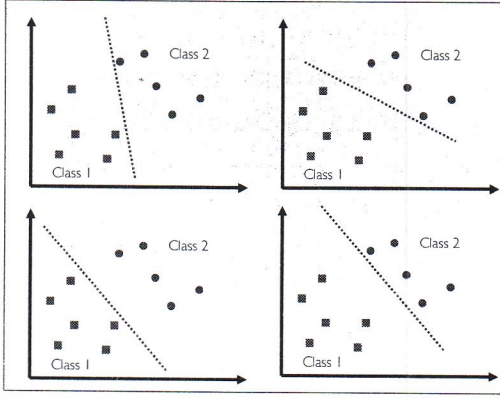
- https://arxiv.org/pdf/1811.12808.pdf
- https://arxiv.org/pdf/1809.09446.pdf

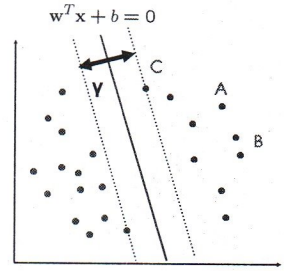
And now some names
you might know ...

Support Vector Machines



Many decision boundaries can separate these two classes
Which one should we choose?



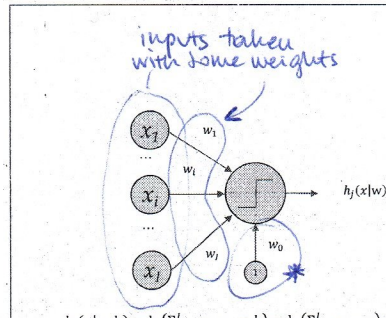


SVMs work by searching for the hyperplane that
maximizes the margin or the largest γ such that

$$\forall i, y_i(wx_i + b) \geq \gamma$$

What is the relation with logistic regression? Logistic regr. has a way of finding the weights of the hyperplane, support vector machines has another. However the goal is always to find the better separating hyperplane.
(Specifically: SVM makes it an optimization problem)

Neural Networks



$$h_j(x|\mathrm{w},\mathrm{b}) = \mathrm{h}_\mathrm{j}\left(\Sigma_{i=1}^{I} w_i \cdot x_i - b\right) = \mathrm{h}_\mathrm{j}\left(\Sigma_{i=0}^{I} w_i \cdot x_i\right) = h_j(w^T x)$$

Computation in an artificial neurons

We take inputs with some weights, we sum them up and we pass them through a function (step function). We obtain $h_j(x|w)$.

Considering that we have weighted inputs and a bias (*), this is equivalent to the evaluation of the hyperplane.

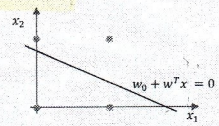
Perceptron Math

98

- A perceptron computes the value of a weighted sum and returns its Sign (Thresholding)

$$h_j(x|\mathrm{w}) = \mathrm{h}_\mathrm{j}\left(\Sigma_{i=0}^{I} w_i \cdot x_i\right) = Sign(w_0 + w_1 \cdot x_1 + \cdots + w_I \cdot x_I)$$

- It is basically a linear classifier for which the decision boundary is the hyperplane $w_0 + w_1 \cdot x_1 + \cdots + w_I \cdot x_I = 0$

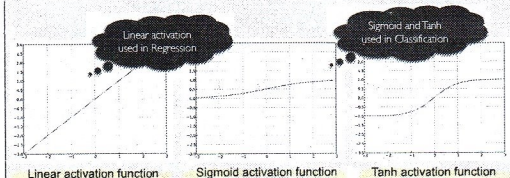


- In 2D, this turns into

$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0$$
$$w_2 \cdot x_2 = -w_0 - w_1 \cdot x_1$$
$$x_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} \cdot x_1$$

Which Activation Function?

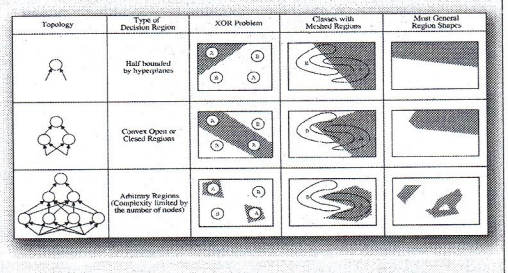
99



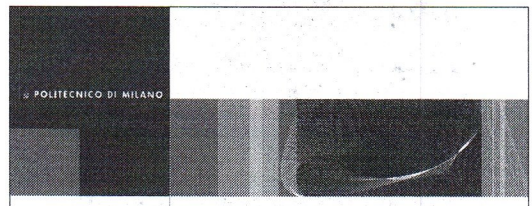
Linear activation function	Sigmoid activation function	Tanh activation function
$g(a) = a$	$g(a) = \frac{1}{1+\exp(-a)}$	$g(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$
$g'(a) = 1$	$g'(a) = g(a)(1-g(a))$	$g'(a) = 1 - g(a)^2$

What Can We Do With it?

100



POLITECNICO DI MILANO



Naïve Bayes & Bayesian Networks

Data Mining and Text Mining

Bayes Probability

- Conditional Probability:

$$P(C|A) = \frac{P(A \wedge C)}{P(A)} \qquad P(A|C) = \frac{P(A \wedge C)}{P(C)}$$

- Bayes theorem,

$$P(C|A) = \frac{P(A|C)P(C)}{P(A)}$$

- A priori probability of C, P(C), is the probability of event before evidence is seen
- A posteriori probability of C, P(C|A), is the probability of event after evidence is seen

Naïve Bayes Classifiers

- What's the probability of the class given an example?
- An example is represented as a tuple of attributes

$$\vec{x} = \langle x_1, \ldots, x_n \rangle$$

- Given the target y (identifying the class value for the instance) we are looking for the class with the highest probability for X

$$class = \arg\max_y P(y|\vec{x})$$

Naïve Bayes Classifiers

- Given the target y and the example x described by n attributes, Bayes Theorem says that

$$P(y|\vec{x}) = \frac{P(\vec{x}|y)P(y)}{P(\vec{x})}$$

- Naïve Bayes classifiers assume that attributes are statistically independent
- Thus, evidence splits into parts that are independent

$$P(y|\vec{x}) = \frac{P(x_1|y) \cdots P(x_n|y)P(y)}{P(\vec{x})}$$

Naïve Bayes for Classification

- **Training**
  - Count the frequency of tuples (x,y) for each attribute value x_i and each class value y in the dataset
  - Use the counts to compute estimates for the class probability P(y) and the conditional probability P(x|y)
- **Testing**
  - Given an example x, computes the most likely class as

$$\begin{aligned} class &= \arg\max_y P(y|\vec{x}) \\ &= \arg\max_y \frac{P(x_1|y) \cdots P(x_n|y)P(y)}{P(\vec{x})} \\ &= \arg\max_y P(x_1|y) \cdots P(x_n|y)P(y) \end{aligned}$$

If we assume that all the cases have more or less the same probability then $P(\vec{x})$ is like a constant and we can get rid of it

Naïve Bayes Classifier

6

- **Two assumptions**
  - Attributes are equally important
  - Attribute are statistically independent
- **Statistically independent means**
  - That knowing the value of one attribute x_i says nothing about the value of another x_j if the class y is known, that is, $P(x_i|x_j,y) = P(x_i|y)$
  - Independence assumption is almost never correct! But the scheme works well in practice

The Weather Dataset

7

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

y

$P(class = yes) = 9/14$
$P(class = no) = 5/14$
we now need $P(x_i|y)$, i.e.
$P(outlook = overcast \mid y = yes)$
$P(outlook = overcast \mid y = no)$
$P(outlook = rainy \mid y = yes)$
⋮

Computing Probabilities

8

Outlook			Temperature			Humidity			Windy			Play	
	Yes	No		Yes	No		Yes	No		Yes	No	Yes	No
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

What is the assigned class?

This, for example, stands for:
$P(outlook = sunny \mid y = yes)$
Notice one thing:
$P(outlook = sunny \mid yes) \neq 1 - P(sunny \mid no)$
(ATTENTION!)

$P(no)$

$P(yes)$

Using the Probabilities for Classification

9

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

$$\text{Likelyhood of yes} = P(Sunny|yes)P(Cool|yes)P(High|yes)P(True|yes)P(yes) = 0.0053$$
$$\text{Likelyhood of no} = P(Sunny|no)P(Cool|no)P(High|no)P(True|no)P(no) = 0.0206$$

- The sum of the two values should be one. Thus, we convert the two values into actual probabilities using normalization:

$$P(yes|Sunny, Cool, High, True) = 0.0053/(0.0053 + 0.0206) = 0.205$$
$$P(no|Sunny, Cool, High, True) = 0.0206/(0.0053 + 0.0206) = 0.795$$

→ we assign "no"
(if we consider as a trash value $= \frac{1}{2}$)

The "Zero-Frequency Problem"

10

- What if an attribute value does not occur with every class value? (for instance, "Humidity = high" for class "yes")
- The corresponding probability will be zero,
$$P(Humidity = high|yes) = 0$$
- A posteriori probability will also be zero!
(No matter how likely the other values are!)
$$P(yes|\langle \ldots, Humidity = high, \ldots \rangle) = 0$$
- Typical remedy is to add 1 to count for every (attribute value, class) pair
  - Process called *smoothing*.
  - Adding 1 is called a *Laplace estimator*
- Resulting probabilities will never be zero! It also stabilizes probability estimates

Actually in the above example is:
$P(outlook = overcast \mid y = no) = 0$
(**not** $humidity = high \mid yes$)

! It's used almost standard (even when we have no zero). Moreover it's applied everywhere, not in only one variable.
We apply LAPLACE ESTIMATOR ONLY ON VARIABLES, **NOT** IN THE TARGETS (y)

Modified Probability Estimates

11

- Sometimes, it is more appropriate to adda constant different from one; for example, we can add the same $\mu/3$ to each count

$\frac{2+\mu/3}{9+\mu}$	$\frac{4+\mu/3}{9+\mu}$	$\frac{3+\mu/3}{9+\mu}$
Sunny	Overcast	Rainy

- μ here acts as a regularization (hyper)parameter
- If background knowledge is available on the frequency of certain values, we can add different weights to each count

$\frac{2+\mu p_1}{9+\mu}$	$\frac{4+\mu p_2}{9+\mu}$	$\frac{3+\mu p_3}{9+\mu}$

- $(p_1+p_2+p_3=1)$ are prior estimates on probability of each values.

Example (1)

12

- Suppose we want to compute the probablities of a Naïve Bayes classifier using the Laplace estimator for the dataset:

location	size	pets	value
good	small	yes	high
good	big	no	high
good	big	no	high
bad	medium	no	medium
good	medium	only cats	medium
good	small	only cats	medium
bad	medium	yes	medium
bad	small	yes	low
bad	medium	yes	low
bad	small	no	low

$\vec{x}$ | y

We need:
$\mathbb{P}(\text{high})$, $\mathbb{P}(\text{medium})$, $\mathbb{P}(\text{low})$
and all the conditioned:
$\mathbb{P}(x_i|\text{high})$, $\mathbb{P}(x_i|\text{medium})$, $\mathbb{P}(x_i|\text{low})$

Example (2)

13

- As done in the example involving the weather dataset first we count the occurrences and add one due to the Laplace estimator

Location	Class high	Class medium	Class low
Good	3+1	2+1	0+1
Bad	0+1	2+1	3+1

Size	Class high	Class medium	Class low
Small	1+1	1+1	2+1
Medium	0+1	3+1	1+1
Big	2+1	0+1	0+1

Pets	Class high	Class medium	Class low
Yes	1+1	1+1	2+1
No	2+1	1+1	1+1
Only Cats	0+1	2+1	0+1

Example (3)

14

- Next, we can compute the frequencies

Location	Class high	Class medium	Class low
Good	4/5	3/6	1/5
Bad	1/5	3/6	4/5

Size	Class high	Class medium	Class low
Small	2/6	2/7	3/6
Medium	1/6	4/7	2/6
Big	3/6	1/7	1/6

Pets	Class high	Class medium	Class low
Yes	2/6	2/7	3/6
No	3/6	2/7	2/6
Only Cats	1/6	3/7	1/6

- Note that, Laplace estimator is not applied to the class, that is,
  - P(high) = 3/10
  - P(medium) = 4/10
  - P(low) = 3/10

(e.g.)
$\mathbb{P}(\text{location} = \text{bad}|\text{high})$

How to procede with prediction?
Suppose we have:
$\vec{x} = (\text{good}, \text{small}, \text{yes})$
we evaluate:
1. $\mathbb{P}(\vec{x}|\text{high})\,\mathbb{P}(\text{high})$
2. $\mathbb{P}(\vec{x}|\text{medium})\,\mathbb{P}(\text{medium})$
3. $\mathbb{P}(\vec{x}|\text{low})\,\mathbb{P}(\text{low})$

(where $\mathbb{P}(\vec{x}|\cdot) = \prod_{j=1}^{3}\mathbb{P}(x_j|\cdot)$)
then we normalize:
$1. + 2. + 3. = 1$
then we choose the highest.

How does Naïve Bayes deal with missing values?

15

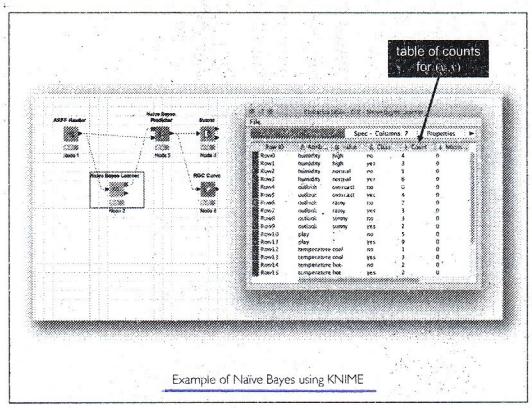
- Training
  - Instance is not included in frequency count for attribute value-class combination
- Testing
  - The attribute will be omitted from calculation

Outlook	Temperature	Humidity	Windy	Play
?	Cool	High	True	?

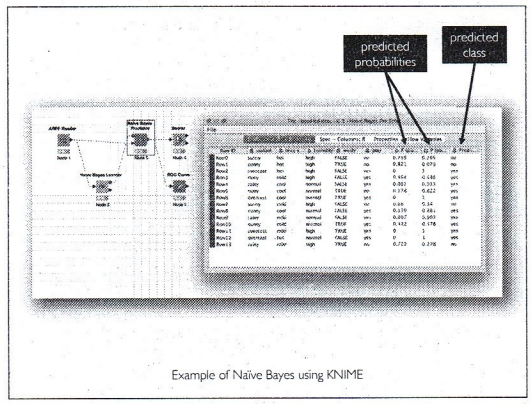
Likelihood of "yes" = 3/9 x 3/9 x 3/9 x 9/14 = 0.0238
Likelihood of "no" = 1/5 x 4/5 x 3/5 x 5/14 = 0.0343
P("yes" | ?, Cool, High, True) = 0.0238 / (0.0238 + 0.0343) = 41%
P("no" | ?, Cool, High, True) = 0.0343 / (0.0238 + 0.0343) = 59%

Notice: in logistic regression we cannot deal with missing values. More precisely: we can work on missing values before the logistic regression, but not during.

$\mathbb{P}(\text{cool}, \text{high}, \text{true}|\text{yes})\,\mathbb{P}(\text{yes})$
$\mathbb{P}(\text{cool}, \text{high}, \text{true}|\text{no})\,\mathbb{P}(\text{no})$
$\Rightarrow$ higher
We act like outlook doesn't exist.



Example of Naïve Bayes using KNIME



Example of Naïve Bayes using KNIME

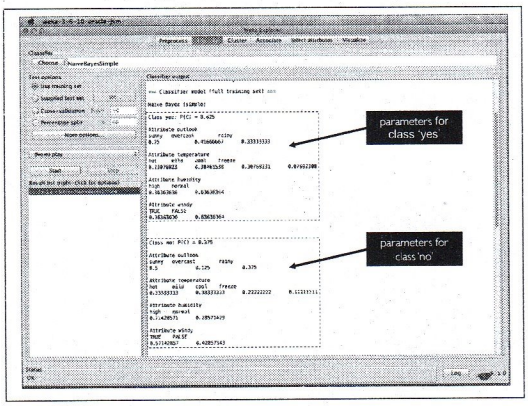
Another Version of the Weather Data

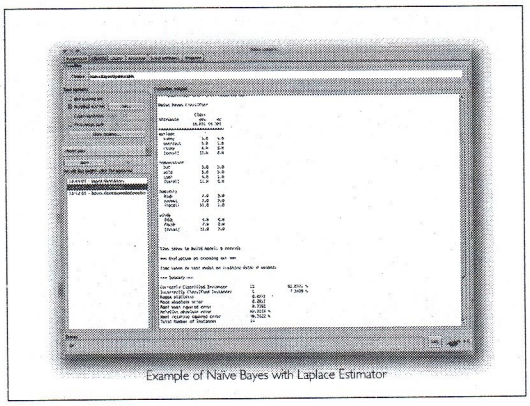

```
@relation weather.symbolic

@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool, freeze}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,hot,high,FALSE,no
sunny,hot,high,TRUE,no
...
```

(what if we get something in X that we never saw ? (somehow, a new value)





Example of Naïve Bayes with Laplace Estimator

Numeric Attributes in Naïve Bayes

21

- So far, we applied Naïve Bayes to categorical data. What if some (or all) of the attributes are numeric? Two options
- Discretize the data to make it binary or discrete
- Compute a probability density for each class
  - Assume parametric form for distribution and estimate its parameters. E.g., assume attribute values for class follow Gaussian distribution
  - Directly estimate probability density from the data. E.g., use kernel smoothing to estimate density of values along axis for class

Numeric Attributes

22

- We assume that the attributes have a normal or Gaussian probability distribution (given the class)
- The probability density function for the normal distribution is defined by two parameters, the mean and the standard deviation
- Sample mean,

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- Standard deviation,

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}$$

- Then the density function f(x) is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Statistics for Weather data with numeric temperature

23

Outlook			Temperature		Humidity		Windy			Play	
	Yes	No	Yes	No	Yes	No		Yes	No	Yes	No
Sunny	2	3	**64, 68,**	65, 71,	65, 70,	70, 85,	False	6	2	9	5
Overcast	4	0	**69, 70,**	72, 80,	70, 75,	90, 91,	True	3	3		
Rainy	3	2	**72, …**	85, …	80, …	95, …					
Sunny	2/9	3/5	μ =73	μ =75	μ=79	μ=86	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	σ =6.2	σ =7.9	σ=10.2	σ=9.7	True	3/9	3/5		
Rainy	3/9	2/5									

$$f(temperature = 66|yes) = \frac{1}{\sqrt{2\pi}6.2} e^{-\frac{(66-73)^2}{2\times 6.2^2}} = 0.0340$$

the categorical still works the same: we may decide to apply laplace estimator on all the categorical

we compute μ and σ for all the cases and then

Classifying a new day

24

- A new day,

Outlook	Temperature	Humidity	Windy	Play
Sunny	66	90	true	?

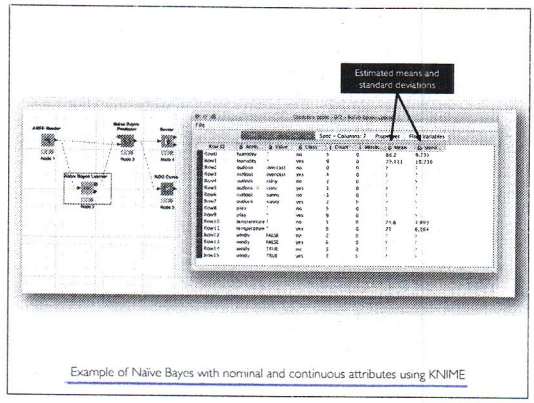
- Missing values during training are not included in calculation of mean and standard deviation

Likelihood of "yes" = 2/9 × 0.0340 × 0.0221 × 3/9 × 9/14 = 0.000036
Likelihood of "no" = 3/5 × 0.0291 × 0.0380 × 3/5 × 5/14 = 0.000136
P("yes" | Sunny, 66, 90, True) = 0.000036 / (0.000036 + 0.000136) = 20.9%
P("no" | Sunny, 66, 90, True) = 0.000136 / (0.000036 + 0.000136) = 79.1%

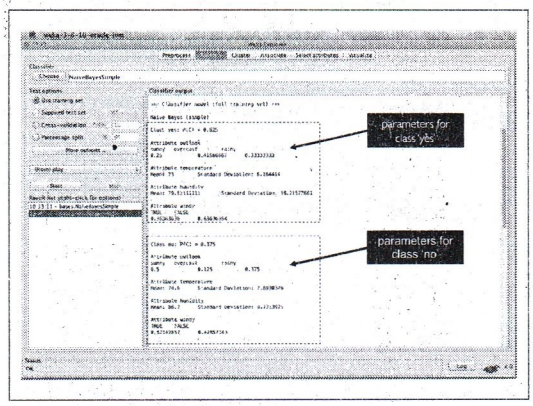
yes:
$P(sunny|yes) \cdot f(66|\mu_{yes}, \sigma_{yes}) \cdot f(90|\mu_{yes}, \sigma_{yes}) \cdot P(true|yes) \cdot P(yes)$

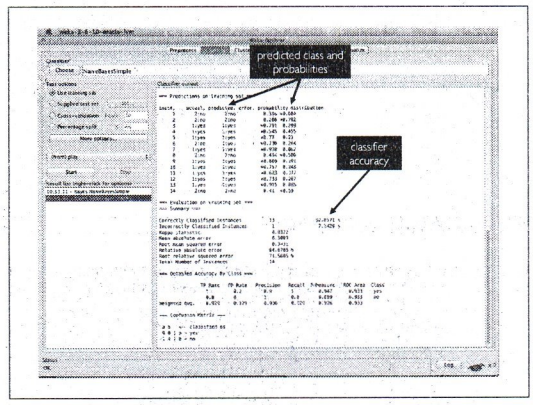
no:
$P(sunny|no)\, f(66|\mu_{no}, \sigma_{no}) f(90|\mu_{no}, \sigma_{no}) \cdot P(true|no)\, P(no)$

→ normalization → higher



Example of Naïve Bayes with nominal and continuous attributes using KNIME





Discussion

28

- Naïve Bayes works surprisingly well, even if independence assumption is clearly violated
- Why? Because classification doesn't require accurate probability estimates as long as maximum probability is assigned to correct class
- However,
  - Adding too many redundant attributes will cause problems e.g. adding identical attributes will make classifier worse and worse
  - Also, many numeric attributes are not normally distributed so that Gaussian assumption may be poor and need substituting

(it's very **flexible**) continuos, categorical and missing data ✓

we just need to know that one class is more probable than another

Bayesian Belief Networks

The assumption of independence among variables is really strong. we may want something based almost on the same principle but allows to specify the relation between variables.

Bayesian Belief Networks

30

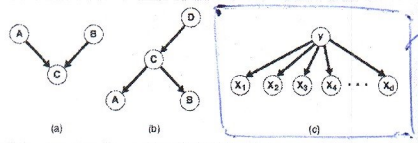
- The conditional independence assumption,
  - makes computation possible
  - yields optimal classifiers when satisfied
  - but is seldom satisfied in practice, as attributes (variables) are often correlated
- Bayesian Belief Networks (BBN) allows us to specify which pair of attributes are conditionally independent
- They provide a graphical representation of probabilistic relationships among a set of random variables

Bayesian Belief Networks

- Describe the probability distribution governing a set of variables by specifying
  - Conditional independence assumptions that apply on subsets of the variables
  - A set of conditional probabilities
- Two key elements
  - A direct acyclic graph, encoding the dependence relationships among variables
  - A probability table associating each node to its immediate parents node

Examples of Bayesian Belief Networks

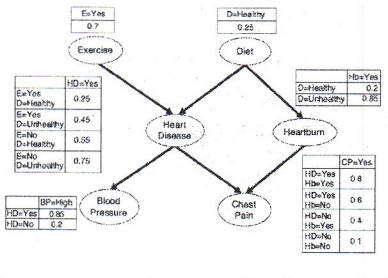
- Variable A and B are independent, but each one of them has an influence on the variable **C**
- A is conditionally independent of both B and D, given C
- The configuration of the typical naïve Bayes classifier



This representation means that we can evaluate $P(y)$ without anything else, instead we cannot evaluate $P(x_j)$, we can only evaluate $P(x_j|y)$ $\forall j$

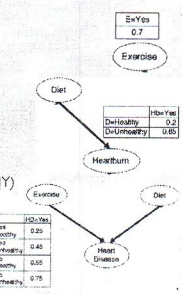
An Example of Bayesian Belief Network

all the combinations of **E** and **D** (where E = yes/no, while D = healthy/unhealthy)



Probability Tables

- Each node is associated with probability table
- If a node X does not have any parents, then the table contains only the prior probability P(X)
- If a node X has only one ~~any~~ parent Y, then the table contains the conditional probability P(X|Y)
- If a node X has multiple parents (Y1, …, Yk) table contains the conditional probability P(X|Y1…Yk)

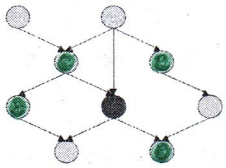


Probability Tables

- The network topology imposes conditions regarding the variable conditional independence
- Each node is associated with a probability table
  - If a node X does not have any parents, then the table contains only the prior probability P(X)
  - If a node X has only one ~~any~~ parent Y, then the table contains only the conditional probability P(X|Y)
  - If a node X has multiple parents, $Y_1, \ldots, Y_k$ the the table contains the conditional probability $P(X|Y_1 \ldots Y_k)$

Bayesian Belief Networks

- Values of variables in Network can be known or unknown.
- Idea is to estimate probabilities over unknown variables given known values



evidence, observed

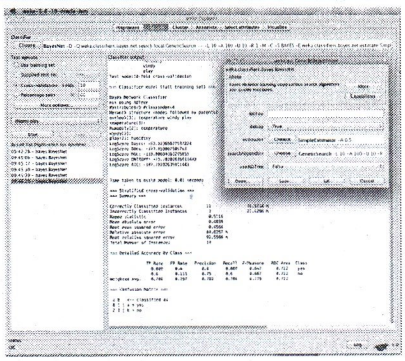
unobserved

to be predicted

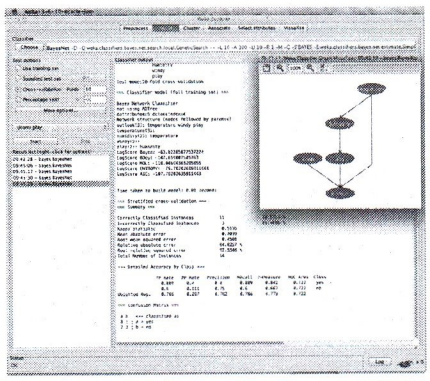
- In general the inference is NP-complete but there are approximating methods, e.g. Monte Carlo

There is not a real concept of class we're focusing on: we're trying to model the entire distribution of the data.

WEKA:



http://www.cs.waikato.ac.nz/~remco/weka.bn.pdf



Learning in Bayesian Belief Networks

- Several cases of learning Bayesian belief networks
  - Given both network structure and all the variables is easy
  - Given network structure but only some variables
  - When the network structure is not known in advance

Bayesian Belief Networks

- **Bayesian Networks**
  - Encode causal dependencies among variables
  - Well suited to dealing with incomplete data
  - Robust to overfitting
  - When used to build classifier, they extend Naïve Bayes by removing conditional independence assumption
- **Other benefits**
  - Can be used to capturing prior knowledge from domain experts
  - Graphical model describing relationship between variables is often useful for understanding / visualizing data
- **Building the network is time consuming!**
  - Involves searching huge space of possible conditional independence relationships
  - But adding variables to an existing network is straightforward

if we know some relations among data we can use it to build the network

POLITECNICO DI MILANO

k-Nearest Neighbors

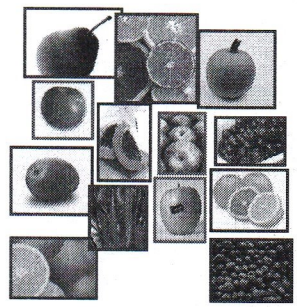
Data Mining and Text Mining

Prof. Pierluca Lanzi

k Nearest-Neighbors Classification

To classify an example x, select the k data points in the training set that are most similar to x

Assign the most frequent class among the k selected



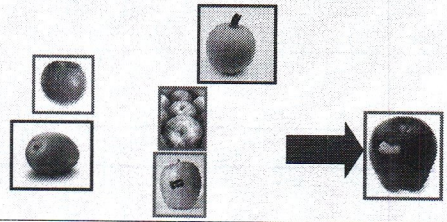
is this an apple?

To decide the label for an unseen example, takes the most similar k examples from the training data

Classify the unknown example using the most frequent class

Nearest Neighbor in Action

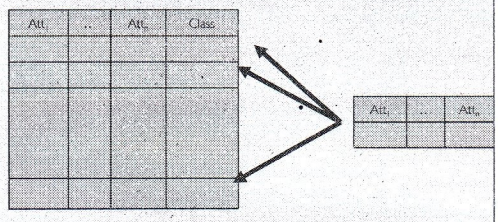
- If k=5, these 5 fruits are the most similar ones to the unclassified example
- Since, the majority of the fruits are labeled as positive examples, we decide that the unknown fruit is an apple



K-nearest neig. is also called (sometimes) instance-Based method because the method is not based on a model, it based on instances

Instance-Based Learning in General

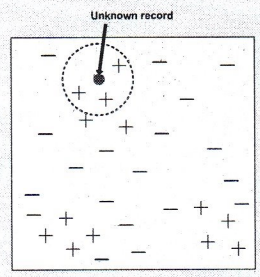
- Store the training records only, no model is computed
- Use the training records to predict an unknown class label



Instance-Based Methods

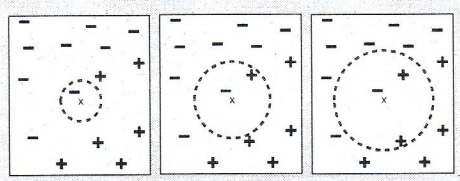
- They are the simplest form of learning
- The training dataset is searched for the instances that are more similar to the unlabeled instance
- The training dataset is the model itself, it is the knowledge
- The similarity function defines what's "learned"
- They implement a "lazy evaluation" (or lazy learning) scheme, nothing happens until a new unlabeled instance must be classified
- Known methods include "Rote Learning", "Case Base Reasoning", "k-Nearest Neighbors"

K-Nearest-Neighbors Classifier



- Three elements
  - The training dataset
  - Similarity function (or distance metric)
  - The value of k, the number of nearest neighbors to retrieve
- Classification
  - Compute distance to other training records
  - Identify the k nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

How Many Neighbors?



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

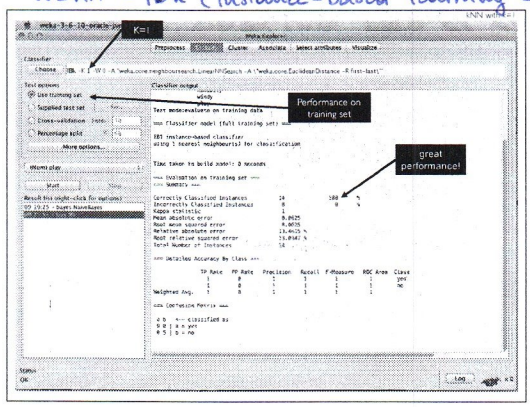
How many neighbors?

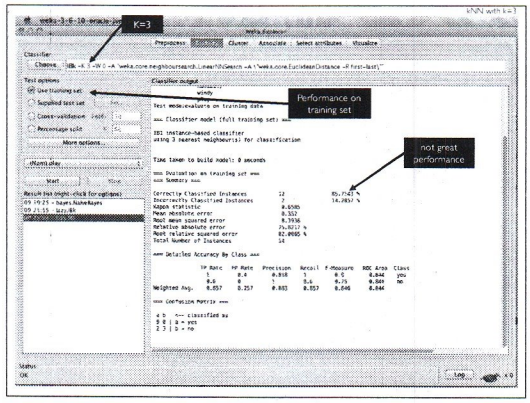
If k is too small, classification might be sensitive to noise points

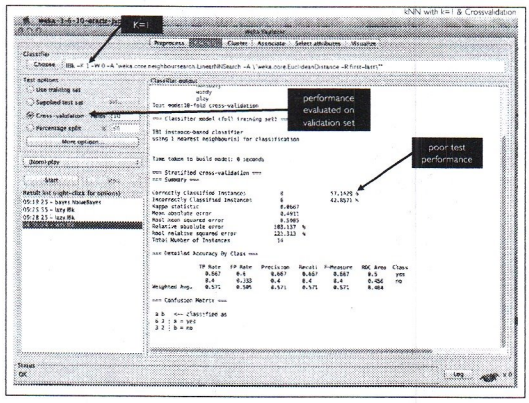
If k is too large, neighborhood may include quite dissimilar examples

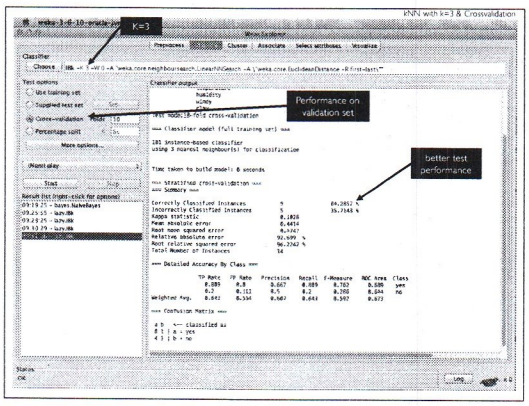
it's a hyperparameter! Cross-validation changing k at each iteration. (Or nested cross-validation)

WEKA - IBk (Instance-Based learning = k nearest neig.)









What similarity measures?

The same ones we applied for clustering!

Similarly to clustering, we must apply normalization when needed!

Finding Nearest Neighbors Efficiently

- Basic Approach
  - Linear scan of the data
  - Classification time for a single distance depends on the number of data points and the number of variables O(nd)
  - This can be prohibitive when the training set is large
- Nearest-neighbor search can be speeded up by using
  - KD-Trees
  - Ball-Trees
  - ...

methods that rely on the construction of a structure

The fact is that we would like to have something local. In k-nearest neighbors we always have to go through all the dataset to find the k nearest to the given one. can we speed this up?

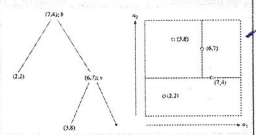
KD-Trees

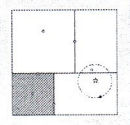
Split the space hierarchically using a tree generated from the data

To find the neighbor of a specific example, navigate the tree using the example

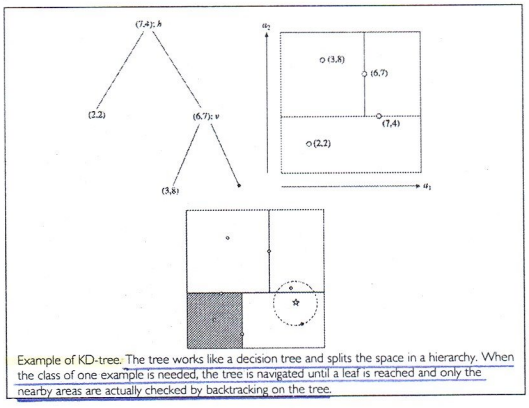
KD-Tree construction and search

- Tree construction
  - Add points iteratively to tree
  - Each new point falls in a leaf and splits region around it based on value of one of its attributes
- Search for nearest neighbors
  - Navigate tree to reach leaf and check
  - Backtrack up tree to check nearby regions
  - Until all k-nearest neighbors found
    i.e. stop when closest region is further than k-th closest point found so far





we randomly partionate the space (we randomly generate a point and from it we split in one direction). This can be translated (at the end) in a tree structure.



Example of KD-tree. The tree works like a decision tree and splits the space in a hierarchy. When the class of one example is needed, the tree is navigated until a leaf is reached and only the nearby areas are actually checked by backtracking on the tree.

Effectiveness of KD-trees

- Search complexity depends on depth of tree. It is the logarithm of number of nodes for balanced tree O(log(n))
- Occasional rebalancing of tree may be needed randomizing order of data is another option
- But amount of backtracking required depends on quality of tree some nodes are square (good) while others are skinny (bad)

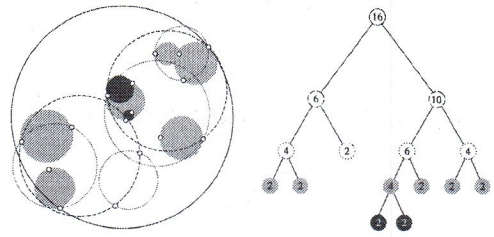
More on kD-trees

- Complexity depends on depth of the tree, given by the logarithm of number of nodes for a balanced tree
- Amount of backtracking required depends on quality of tree ("square" vs. "skinny" nodes)
- How to build a good tree?
  - Need to find good split point and split direction
  - Possible split direction: direction with greatest variance
  - Possible split point: median value along that direction
- Using value closest to mean (rather than median) can be better if data is skewed
- Can apply this split selection strategy recursively just like in the case of decision tree learning

One problem of this method are the corners.
One possible alternative →

Ball Trees

- Corners in high-dimensional space may mean query ball intersects with many regions and this is a potential limitation for KD-Tree
- Note that there is no need to make sure that regions do not overlap, so they do not heed to be hyperrectangles !
- Can use hyperspheres (balls) instead of hyperrectangles
- A ball tree organizes the data into a tree of k-dimensional hyperspheres
- Balls may allow for a better fit to the data and thus more efficient search



Example of Ball Tree.

Example using the loan dataset

Evaluation using 10-fold crossvalidation

kd-tree

t=11.163 Accuracy=0.806 Std=0.00

(seconds)

Brute force

t=567.895 Accuracy=0.806 Std=0.001

k-Nearest Neighbor Regression

- The regression methods we previously discussed were parametric
  - They assume an approximation function parametrized by a weight vector
  - The regression algorithm search for the best weight vector
  - E.g., linear regression assumes that the target is computed as a linear combination of the weight vector w and the feature vector h(x)
- Nearest Neighbor regression fits each point locally
- Basic Algorithm
  - Given a dataset $(x_1,y_1) \dots (x_N,y_N)$
  - Given a query point x_q
  - The value y_q associated to x_q is computed as a local interpolation of the targets associated to neighbor points
- Prediction can use the plain average of the k nearest targets or a weighted average of the same targets
- Prediction can use kernel functions that take the distance as input and return a weight (e.g., a Gaussian function of the distance)

Discussion on k-Nearest Neighbor

34

- K-Nearest Neighbor is often very accurate but slow, since simple version scans entire training data to derive a prediction
- Assumes all attributes are equally important, thus may need attribute selection or weights
- Possible remedies against noisy instances:
  - Take a majority vote over the k nearest neighbors
  - Remove noisy instances from dataset (difficult!)
- Statisticians have used k-NN since early 1950s. If $n \to \infty$ and $k/n \to 0$, error approaches minimum
- Data Structures
  - kD-trees can become inefficient when the number of attributes is too large
  - Ball trees are may help

Study Material

35

- Mining of Massive Datasets Section 12.3, 12.4
- Data Mining and Analysis: Fundamental Concepts and Algorithms - Chapter 18 & 19
- Chapter 4 of Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques - Ian H. Witten et al.

Comparison:

- if logistic regression is the more effective then the points are separable
- if k-nearest neighbors is the more effective then locality is more important
- if naive bayes is more effective then the hp. that variables are ⊥ is not bad, moreover it tells that the probabilistic view of the data is more effective than geometrical view of the data (logistic regression) or locality (k nearest neigh.)

The comparison of these 3 gives a lot of informations.

POLITECNICO DI MILANO

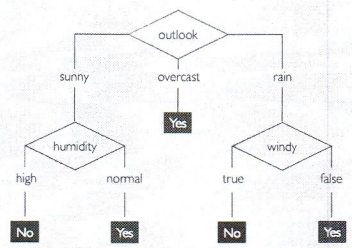
Decision Trees

Data Mining and Text Mining

The Weather Dataset

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

The Decision Tree for the Weather Dataset

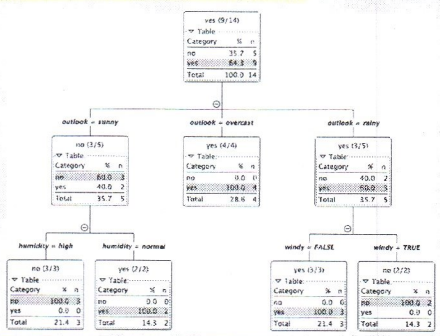


What is a Decision Tree?

- An internal node is a test on an attribute
- A branch represents an outcome of the test, e.g., outlook=windy
- A leaf node represents a class label or class label distribution
- At each node, one attribute is chosen to split training examples into distinct classes as much as possible
- A new case is classified by following a matching path to a leaf node

A structure that allows to make a decision

Decision Tree Representations (Graphical)



Computing Decision Trees

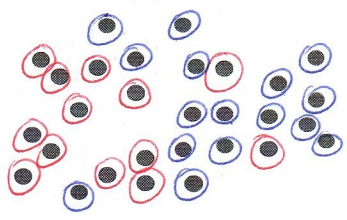
Computing Decision Trees

- **Top-down Tree Construction**
  - Initially, all the training examples are at the root
  - Then, the examples are recursively partitioned, by choosing one attribute at a time
- **Bottom-up Tree Pruning**
  - Remove subtrees or branches, in a bottom-up manner, to improve the estimated accuracy on new cases.

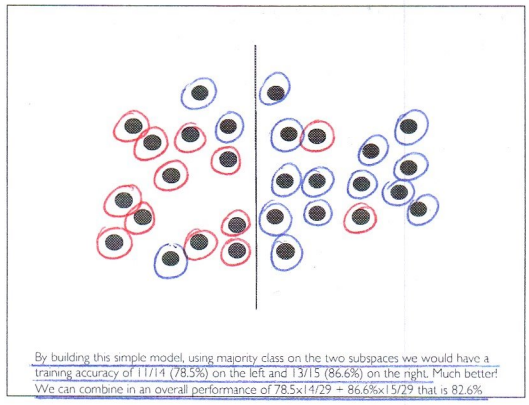
We build up a model and then we simplify it.

What Splitting Attribute?

We are looking for a way to separate example to create areas that are "pure" and contain mainly examples of one class



There are 13 red labels and 16 blue labels, using the majority class unlabeled point would be labeled as blue. Training accuracy for baseline classifier is 16/29 that is, 55%



By building this simple model, using majority class on the two subspaces we would have a training accuracy of 11/14 (78.5%) on the left and 13/15 (86.6%) on the right. Much better! We can combine in an overall performance of 78.5x14/29 + 86.6%x15/29 that is 82.6%

Which Attribute for Splitting?

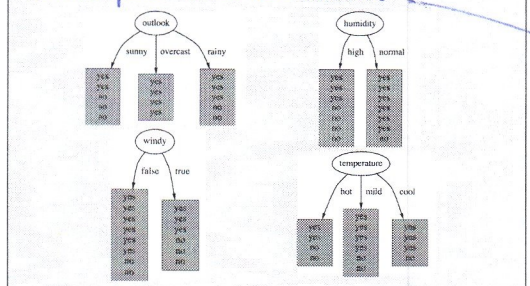
12

- At each node, available attributes are evaluated based on separating the classes of the training examples using either a purity or impurity measure
- Typical measures used are the information gain (ID3), information gain ratio (C4.5), gini index (CART)
- Information Gain increases with the average purity of the subsets that an attribute produces. It selects the attribute with the highest information gain

Which Attribute Should We Select?

13

Each variable gives a different partition of the space. Which one is better?



We rely on entropy to decide: if the entropy after the splitting is less than the entropy before then the split is good!
(this is because we create order from confusion
entropy = measure of confusion)

Computing Information

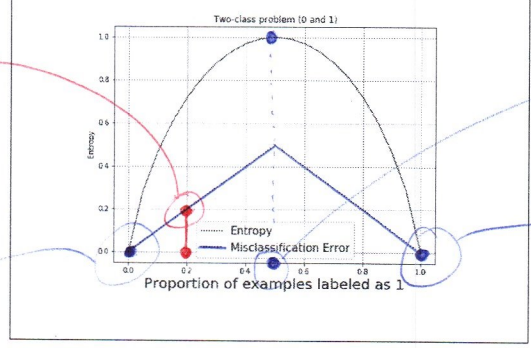
14

- Information is measured in bits
  - Given a probability distribution, the info required to predict an event is the distribution's entropy
  - Entropy gives the information required in bits (this can involve fractions of bits!)
- Formula for computing the entropy

$$entropy(p_1, p_2, \ldots, p_n) = -p_1 \log_2 p_1 - p_2 \log_2 p_2 \cdots - p_n \log_2 p_n$$

- Note that, since entropy refers to the amount of information expressed in bits, it uses the logarithm with base 2

We have two labels: 0/1



What does this mean?
If we're here (x=0.2) then we have 20% of points "1" and 80% of points "0": the majority voting is label 0 so the accuracy of label 0 is 80%.
⇒ the error is going to be 20%.
(that's why y=0.2)

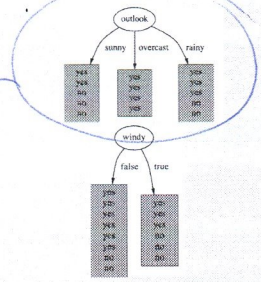
If all the points are 0 then the entropy is 0 because there is no confusion

if we have 50% of points labeled as 1 and 50% of the points labeled as 0 then the entropy is 1 (maximum) because there's a lot of confusion

if all the points are 1 then the entropy is 0 because there is no confusion

Which Attribute Should We Select?

Let's measure the entropies!



The Attribute "outlook"

- "outlook" = "sunny" $info([2,3]) = entropy(2/5, 3/5) = 0.971$
- "outlook" = "overcast" $info([4,0]) = entropy(1,0) = 0.000$
- "outlook" = "rainy" $info([3,2]) = entropy(3/5, 2/5) = 0.971$
- Expected information for attribute (weighted sum: the weights is the proportion of data that we have in that division)

$$info([2,3][4,0][3,2]) = 5/14 \times 0.971 + 4/14 \times 0 + 5/14 \times 0.971$$

Here we have 14 observations: 5 in "sunny", 4 in "overcast" and 5 in "rainy"

- As previously noted, since entropy refers to the amount of information expressed in bits, it uses the logarithm with base 2

this value alone says nothing: we have to compute the difference with the original one

Information Gain

- Difference between the information before split and the information after split

$$gain(A) = info(D) - info_A(D)$$

- The information before the split is the entropy of the class distribution in D,

$$info(D) = -p_1 \log_2 p_1 \cdots - p_n \log_2 p_n$$

- The information after the split using attribute A is computed as the weighted sum of the entropies on each split, given n splits,

$$info_A(D) = \frac{|D_1|}{|D|} info(D_1) + \cdots + \frac{|D_n|}{|D|} info(D_n)$$

Information Gain

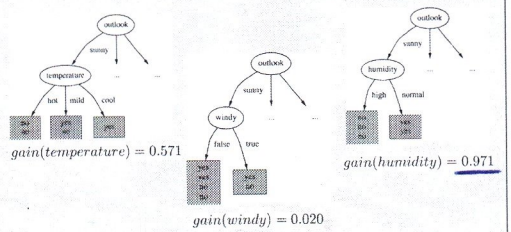
- Difference between the information before split and the information after split

$$\begin{aligned} gain(outlook) &= info([9,5]) - info([2,3][4,0][3,2]) \\ &= 0.940 - 0.693 \\ &= 0.247 \end{aligned}$$

- Information gain for the attributes from the weather data:
  - gain("outlook")=0.247 ← highest gain
  - gain("temperature")=0.029
  - gain("humidity")=0.152
  - gain("windy")=0.048

It's a greedy process

Going Further

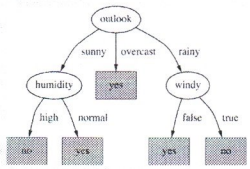


$gain(temperature) = 0.571$ $gain(windy) = 0.020$ $gain(humidity) = 0.971$

Since outlook is categorical, once we use it we don't use it anymore! In the subset where outlook = "sunny" there's no way we can make a division using again outlook, they're all "sunny"! That's not true for continuous variables!

The Final Decision Tree

- Not all the leaves need to be pure
- Splitting stops when data can not be split any further

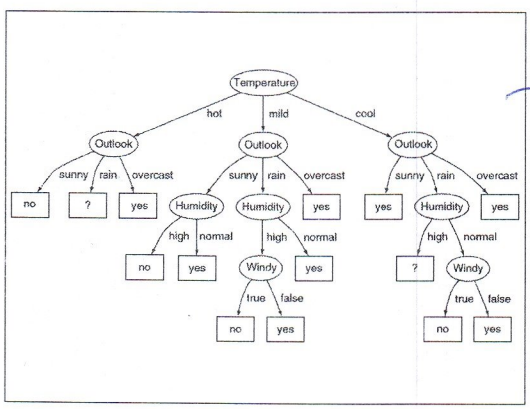


When Should Building Stop?

- There are several possible stopping criteria
- If all samples for a given node belong to the same class
- If there are no remaining attributes for further partitioning, majority voting is employed
- If there are no samples left (we may want a minimum number of elements)
- If there is nothing to gain in splitting

We Can Always Build a 100% Accurate Tree

But do we want it?



This tree in every leaf has 1 case (the decision is taken from 1 single point) → it doesn't generalize

Many-valued Attributes

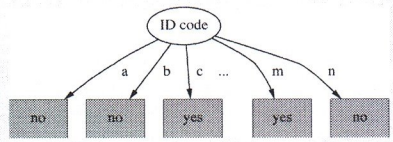
The attributes that produce a lot of splits are generally not good to be used: they cause overfitting!

E.g. if we use the persona_id as an attribute we end up having 1 case in every subset

Another Version of the Weather Dataset

ID code	Outlook	Temp	Humidity	Windy	Play
A	Sunny	Hot	High	False	No
B	Sunny	Hot	High	True	No
C	Overcast	Hot	High	False	Yes
D	Rainy	Mild	High	False	Yes
E	Rainy	Cool	Normal	False	Yes
F	Rainy	Cool	Normal	True	No
G	Overcast	Cool	Normal	True	Yes
H	Sunny	Mild	High	False	No
I	Sunny	Cool	Normal	False	Yes
J	Rainy	Mild	Normal	False	Yes
K	Sunny	Mild	Normal	True	Yes
L	Overcast	Mild	High	True	Yes
M	Overcast	Hot	Normal	False	Yes
N	Rainy	Mild	High	True	No

Decision Tree for the New Dataset



- Entropy for splitting using "ID Code" is zero, since each leaf node is "pure"
- Information Gain is thus maximal for ID code

However it's overfitting!

Highly-Branching Attributes

- Attributes with a large number of values are usually problematic
- Examples: id, primary keys, or almost primary key attributes
- Subsets are likely to be pure if there is a large number of values
- Information Gain is rewards attributes with many values
- This may result in overfitting (selection of an attribute that is non-optimal for prediction and thus overfits)

Information Gain Ratio

Information Gain Ratio

- Modification of the Information Gain that reduces the bias toward highly-branching attributes
- Information Gain Ratio should be
  - Large when data is evenly spread
  - Small when all data belong to one branch
- Information Gain Ratio takes number and size of branches into account when choosing an attribute
- It corrects the information gain by taking the intrinsic information of a split into account

Information Gain Ratio and Intrinsic information

31

- Intrinsic information

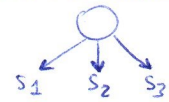
$$IntrinsicInfo(S, A) = -\sum \frac{|S_i|}{|S|} \log \frac{|S_i|}{|S|}$$

computes the entropy of distribution of instances into branches

- This is the entropy of A, independent of the class
- Information Gain Ratio normalizes Information Gain by

$$GainRatio(S, A) = \frac{Gain(S, A)}{IntrinsicInfo(S, A)}$$

we have the data splitted:



we don't care about the classes themselves, we care about the distribution of the data in the classes, so we take into account $|S_1|/|S|$, $|S_2|/|S|$, $|S_3|/|S|$.

We use this information (the distribution among classes) as the penalty in the gain

This tries to counterbalance the problem of attributes with many values. However, this may not be enough in the case of primary keys!

Computing the Information Gain Ratio

32

- The intrinsic information for ID code is

$$info([1/14, \ldots, 1/14]) = 14 \times (-1/14 \log_2 1/14) = 3.807$$

- Importance of attribute decreases as intrinsic information gets larger
- The Information gain ratio of "ID code",

$$GainRatio(ID\ code) = \frac{0.940}{3.807} = 0.246$$

original information gain

Information Gain Ratio for Weather Data

33

Outlook		Temperature	
Information after split:	0.693	Information after split:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.362
Gain ratio: 0.247/1.577	0.156	Gain ratio: 0.029/1.362	0.021

(still the best

Humidity		Windy	
Information after split:	0.788	Information after split:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

More on Information Gain Ratio

34

- "outlook" still best compared to other original attributes
- however "ID code" has even greater gain ratio (0.246)
  - despite likely not being a reliable attribute to branch on
  - standard fix is an ad-hoc test to prevent splitting on that type of attribute
- In practice, tree learners often use both IG and IGR:
  - First, only consider attributes with greater than average Information Gain;
  - Then, compare them using the Information Gain Ratio
- Why?
  - Information Gain Ratio may overcompensate and choose an attribute just because its intrinsic information is very low

!

Gini Index

Another Splitting Criteria: The Gini Index

- The Gini index, for a data set T contains examples from n classes, is defined as

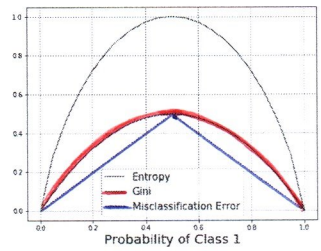
$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in T

- gini(T) is minimized if the classes in T are skewed
- gini measures error rate of random classifier that assigns classes to instances according to their prior frequencies:

$$gini(T) = \sum_j p_j(1-p_j) = \sum_j p_j(1-p_j) = \sum_j p_j - p_j^2 = 1 - \sum_j p_j^2$$

The Gini Index vs Entropy



The Gini Index

- If a data set D is split on A into two subsets D_1 and D_2 then,

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- The reduction of impurity is defined as,

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest Gini splitting D over A (or the largest reduction in impurity) is chosen to split the node (need to enumerate all the possible splitting points for each attribute)

Gini index is applied to produce binary splits

It's always one branch against the other, it's never one branch against multiple others

The Gini Index for the Outlook Attribute

- The dataset has 9 tuples labeled "yes" and 5 labeled "no"

$$gini(D) = 1 - (\frac{9}{14})^2 - (\frac{5}{14})^2 = 0.459$$

- The outlook attribute has three values (overcast, rainy, sunny), thus we have to evaluate three possible partitions
  - {overcast, rainy} and {sunny}
  - {sunny, rainy} and {overcast}
  - {sunny, overcast} and {rainy}

!

The Gini Index for the Outlook Attribute

- {overcast, rainy} and {sunny}

$$\begin{aligned} Gini(D_{o,r}, D_s) &= \frac{9}{14}Gini(D_{o,r}) + \frac{5}{14}Gini(D_s) \\ &= \frac{9}{14}Gini([7,2]) + \frac{5}{14}Gini([2,3]) \\ &= \frac{9}{14}0.346 + \frac{5}{14}0.480 \\ &= 0.394 \end{aligned}$$

- {rainy, sunny} and {overcast}

$$\begin{aligned} Gini(D_{r,s}, D_o) &= \frac{10}{14}Gini(D_{r,s}) + \frac{4}{14}Gini(D_o) \\ &= \frac{10}{14}Gini([5,5]) + \frac{4}{14}Gini([4,0]) \\ &= \frac{10}{14}0.5 + \frac{4}{14}0.0 \\ &= 0.357 \end{aligned}$$

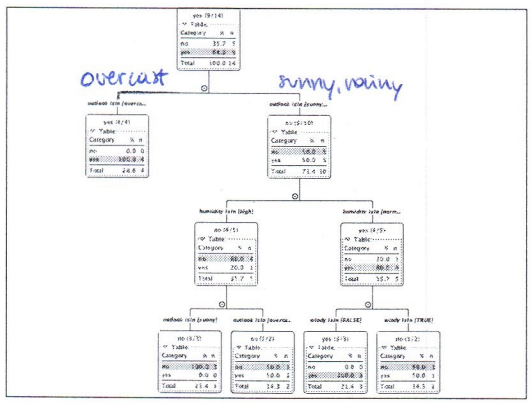
The Gini Index for the Outlook Attribute

- {sunny, overcast} and {rainy}

$$\begin{aligned} Gini(D_{s,o}, D_r) &= \frac{9}{14}Gini(D_{s,o}) + \frac{5}{14}Gini(D_r) \\ &= \frac{9}{14}Gini([6,3]) + \frac{5}{14}Gini([3,2]) \\ &= \frac{9}{14}0.444 + \frac{5}{14}0.480 \\ &= 0.457 \end{aligned}$$

- The attribute partition with the largest gain is {rainy, sunny} and {overcast} that correspond to a Gini of 0.357 and an overall gain of

$$Gini(D) - Gini(D_{r,s}, D_o) = 0.459 - 0.357 = 0.102$$



Numerical Attributes

The Weather Dataset (Numerical)

Outlook	Temp	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	78	False	Yes
Rainy	70	96	False	Yes
Rainy	68	80	False	Yes
Rainy	65	70	True	No
Overcast	64	65	True	Yes
Sunny	72	95	False	No
Sunny	69	70	False	Yes
Rainy	75	80	False	Yes
Sunny	75	70	True	Yes
Overcast	72	90	True	Yes
Overcast	81	75	False	Yes
Rainy	71	80	True	No

The Temperature Attribute

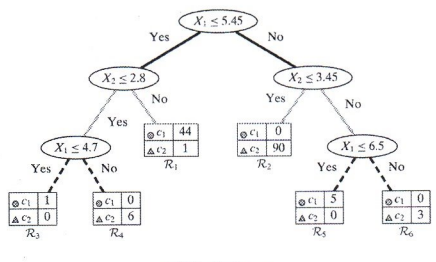
- First, sort the temperature values, including the class labels
- Then, check all the feasible cut points and choose the one with the best information gain

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

- E.g. temperature ≤ 70.5: yes/4, no/1
  temperature > 70.5: yes/5, no/4
- Info([4,1],[5,4]) = 5/14 info([4,1]) + 9/14 info([5,4]) = 0.894
- Information gain for temperature at 70.5 is thus 0.045

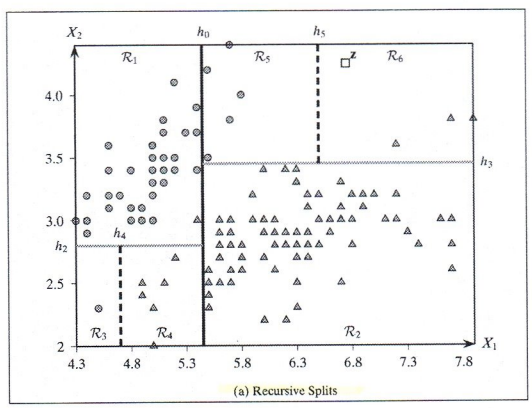
we look for splits that lead to purity (e.g. it doesn't make much sense to split in 68/69 or 69/70 since we have 3 "yes")
→ we consider feasible points those for which we have a change in the output

Geometric Interpretation



(b) Decision Tree

Figure 19.1. Decision trees: recursive partitioning via axis-parallel hyperplanes.



(a) Recursive Splits

What does it mean in the problem space?
We're creating hyperplanes that split the problem space.
It's a sort of logistic regression/neuron behavior.
What is the difference between these hyperplanes and the ones produced by logistic regression?
Logistic regression builds hyperpl. that are combination of all the variables (⇒ the hyperpl. could also be diagonal).
In this case all the internal nodes make the test on only one variable at the time
⇒ the resulting hyp. are parallel to the axis of the problem space

We can avoid repeated sorting

- Sort instances by the values of the numeric attribute takes O(n log n)
- Does this have to be repeated at each node of the tree?
- No, since the sort order for children can be derived from sort order for parent
- The time complexity of derivation is O(n)
- We need to create and store an array of sorted indices for each numeric attribute

Binary vs Multiway splits

- Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
- A nominal attribute is tested (at most) once on any path in the tree
- Numeric attribute may be tested several times along a path in the tree and the tree can become hard to read and interpret
- Possible solutions
  - Pre-discretize numeric attributes, or
  - Use multi-way splits instead of binary ones

however it leads to a lost of informations

ALGORITHM 19.1. Decision Tree Algorithm

```
DECISIONTREE (D, η, π):
n ← |D| // partition size
n_i ← |{x_j | x_j ∈ D, y_j = c_i}| // size of class c_i
purity(D) ← max_i {n_i / n}
if n ≤ η or purity(D) ≥ π then // stopping condition
    c* ← arg max_{c_i} {n_i / n} // majority class
    create leaf node, and label it with class c*
    return
(split point*, score*) ← (∅, 0) // initialize best split point
foreach (attribute X_j) do
    if (X_j is numeric) then
        (v, score) ← EVALUATE-NUMERIC-ATTRIBUTE(D, X_j)
        if score > score* then (split point*, score*) ← (X_j ≤ v, score)
    else if (X_j is categorical) then
        (V, score) ← EVALUATE-CATEGORICAL-ATTRIBUTE(D, X_j)
        if score > score* then (split point*, score*) ← (X_j ∈ V, score)
// partition D into D_Y and D_N using split point*, and call recursively
D_Y ← {x ∈ D | x satisfies split point*}
D_N ← {x ∈ D | x does not satisfy split point*}
create internal node split point*, with two child nodes, D_Y and D_N
DECISIONTREE(D_Y); DECISIONTREE(D_N)
```

Generalization and overfitting in trees

till now we went top-down, now we have to go bottom-up. As we go down we increase the complexity of the model: the prediction becomes really detailed and almost single-samples oriented (overfitting). A decision at the top of the tree is a decision for the whole dataset. A decision at the second level is a decision on a portion of data. Going down we reduce the cardinality of the space the decision concerns: at some point it'll be too detailed.

Avoiding Overfitting in Decision Trees

- The generated tree may overfit the training data
- Too many branches, some may reflect anomalies due to noise or outliers
- Result is in poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning
  - Postpruning

the more in depth we go the fewer cases are the ones that we base our decision on

Pre-pruning vs Post-pruning

- Prepruning (we stop before)
  - Halt tree construction early
  - Do not split a node if this would result in the goodness measure falling below a threshold
  - Difficult to choose an appropriate threshold !
- Postpruning
  - Remove branches from a "fully grown" tree
  - Get a sequence of progressively pruned trees
  - Use a set of data different from the training data to decide which is the "best pruned tree"

build an overfitted model and then semplify it

Prepruning

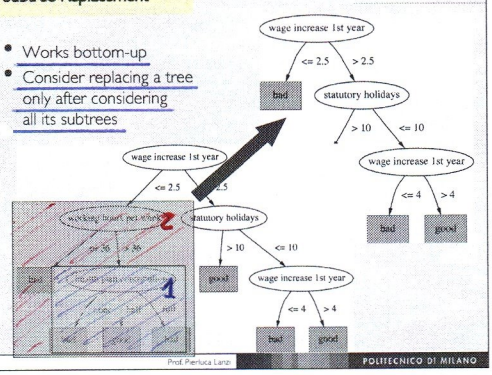
- Based on statistical significance test
- Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node
- The most popular approach is the chi-squared test
- Quinlan's classic tree learner ID3 used chi-squared test in addition to information gain
- Only statistically significant attributes were allowed to be selected by the information gain procedure

Post-Pruning

- First, build full tree, then prune it
- Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects
- Two pruning operations
  - Subtree raising
  - Subtree replacement
- Possible strategies
  - Error estimation
  - Significance testing
  - MDL principle

Subtree Replacement

- Works bottom-up
- Consider replacing a tree only after considering all its subtrees



We're reconsidering what we did. While we created the tree we were not using anything statistical, only precise measures. (exact)

Once the tree is done we go bottom-up and we consider confidence levels and probabilities. We ask ourselves: was **1** worth to create? In this case no. Then we ask: was **2** worth? Also no. We conclude that we can substitute the whole **2** with one leaf.

How do we decide if it's worth it?

Estimating Error Rates

- Prune only if it reduces the estimated error
- Error on the training data is NOT a useful estimator (Why it would result in very little pruning?)
- A hold-out set might be kept for pruning ("reduced-error pruning")
- Example (C4.5's method)
  - Derive confidence interval from training data
  - Use a heuristic limit, derived from this, for pruning
  - Standard Bernoulli-process-based method
  - Shaky statistical assumptions (based on training data)

C4.5's Pruning Method

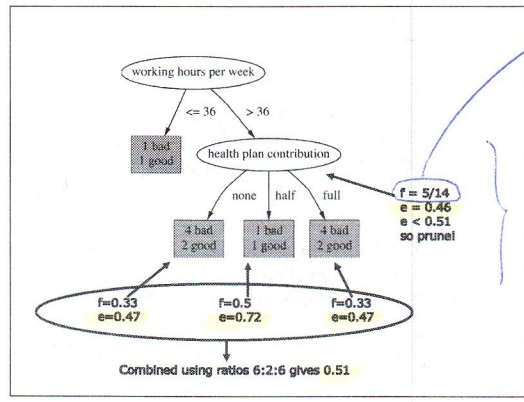
For every node we compute the error on the training data.

- Given the error f on the training data, the upper bound for the error estimate for a node is computed as

$$e = \left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \Big/ \left(1 + \frac{z^2}{N} \right)$$

- If c = 25% then z = 0.69
  - f is the error on the training data
  - N is the number of instances covered by the leaf

confidence level (we're kind of loose: we accept that we make an error with the 25% of probability) (confidence)



error = 5/14 because we have 9 bad and 5 good ⇒ the label is the majority ⇒ bad ⇒ the misclassified are the good : 5/14

we compute the error for every leaf, we average it (w.r.t. the number of elements per leaf) and we compare the result with the error evaluated in the root : if the error in the root is smaller ⇒ we prune. when we prune we replace with one leaf labeled as the majority of the labels.

Regression & Model Trees

Decision trees can also be used to predict the value of a numerical target variable

Regression and model trees work similarly to decision trees

They search for the best split that minimizes an impurity measure

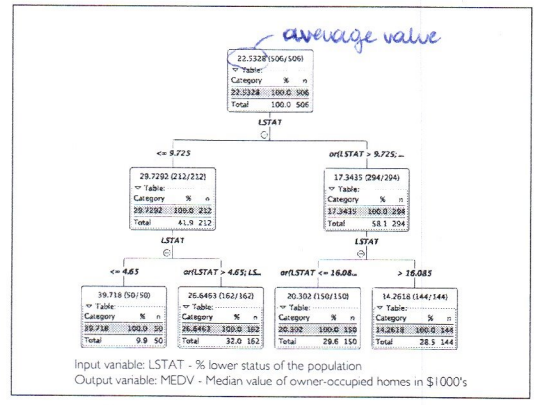
Regression & Model Trees

64

- What prediction?
  - Prediction is computed as the average of numerical target variable in the subspace (regression trees)
  - Alternatively leafs can contain a linear model to predict the target value in the corresponding subspace (model trees)
- What impurity measure?
  - It can be measured as the expected error reduction, or SDR (standard deviation reduction)

$$SDR = \sigma(D) - \sum_i \frac{|D_i|}{|D|}\sigma(D_i)$$

  - D is the original data D_i are the partitions and σ is the standard deviation of the target attribute in the set



Decision Stumps

Decision Stumps

- Decision stumps are one level decision trees
- They are the simplest decision trees possible and also the main building blocks for boosting methods
- Categorical Attributes
  - One branch for each attribute value
  - One branch for one value and one branch for all the others
  - Missing values sometimes are treated as a special value
- Numerical Attributes
  - Two leaves defined by a threshold value selected based on some criterion
  - Multiple splits (rarely used)

Other Approaches

- **ADTree**
  - Builds alternating decision trees
- **BFTree**
  - Builds decision trees using a best-first search
- **FT**
  - Builds a functional trees with oblique splits and linear functions at the leaves
- **LADTree**
  - Builds a multiclass alternating decision trees using LogitBoost
- **LMT**
  - Builds logistic model trees
- **NBTree**
  - Builds a decision tree classifier with a Naïve Bayes classifier at the leaves
- ...