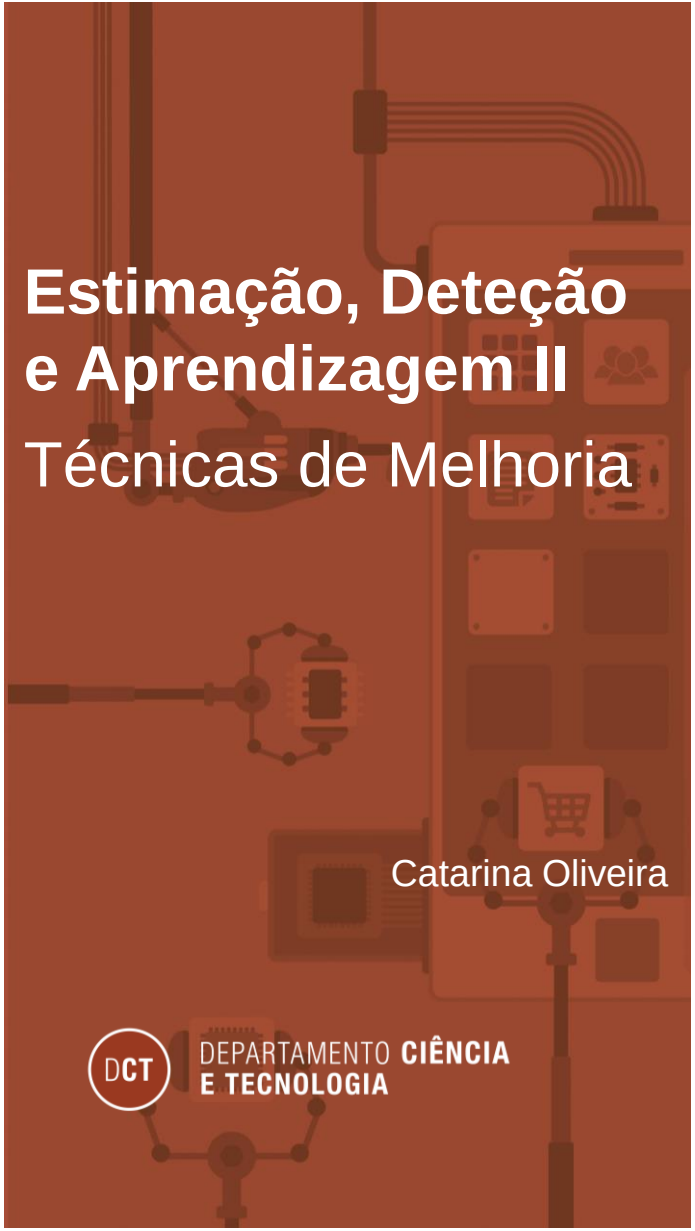


UNIVERSIDADE PORTUCALENSE

CONTEÚDO

1. Ensemble Learning
2. Bagging
3. Random Forests
4. Boosting

Ensemble Learning

Ensemble Learning

A performance preditiva em tarefas de classificação pode ser melhorada combinando as previsões de múltiplos modelos

→ *Ensemble of classifiers*

- **Homogéneo**: modelos criados com a mesma técnica
- **Heterogéneo**: modelos criados com técnicas diferentes

***Base classifier*:**

- *classifier* individual cujas previsões são combinadas no *ensemble*
- Cada um pode ser criado usando
  - O *trainset* original
  - Partes do *trainset* original

Requisitos dos *base classifiers* dos ensembles:

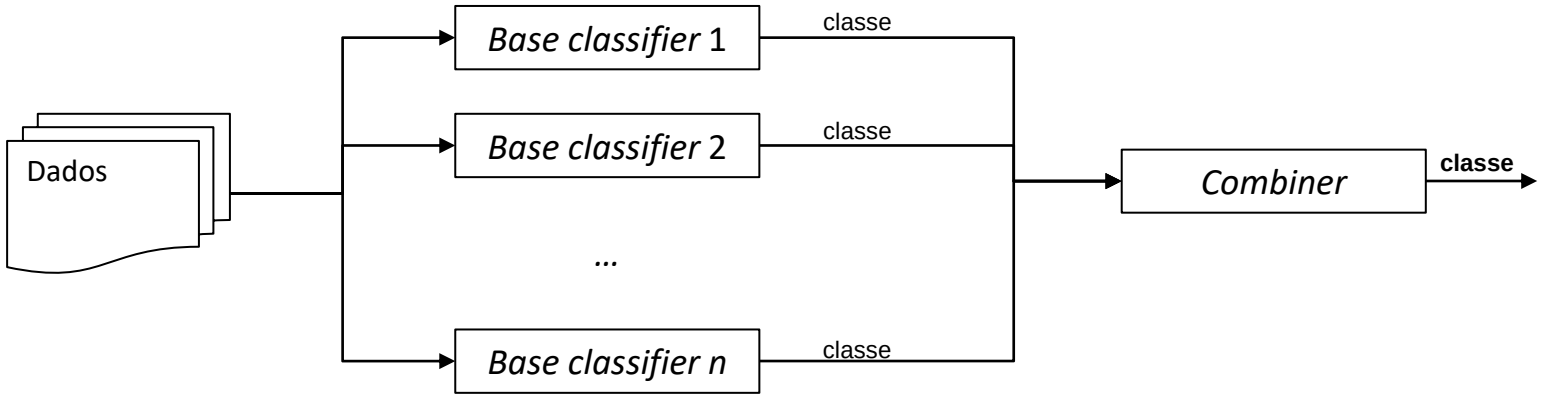
- **Performance preditiva**: devem ter uma performance superior à do modelo que prevê a classe maioritária
- **Diversidade preditiva**: devem ser independentes, idealmente cometendo erros em partes diferentes dos dados

Abordagens:

- Paralela (ex: *bagging*, *random forests*)
- Sequencial (ex: *AdaBoost*)
- Hierárquica

Ensemble learning: abordagem paralela

- A mais comum
- Tenta explorar as semelhanças e diferenças das previsões feitas por diferentes *base classifiers*
- Cada *base classifier*
  - É induzido utilizando instancias do *trainset* original
    - Todas as instâncias | todas as *features*
    - Amostra das instâncias | todas as *features*
    - Todas as instâncias | amostra das *features*



Ensemble learning: abordagem paralela – combinação de previsões

Voting: a classe prevista pela maioria dos *classifiers* é a classe prevista pelo *ensemble*

Weighted voting: à classe prevista por cada base *classifier* é associado um peso, que representa quanto a previsão desse *classifier* deve ser considerada para a previsão final do *ensemble*

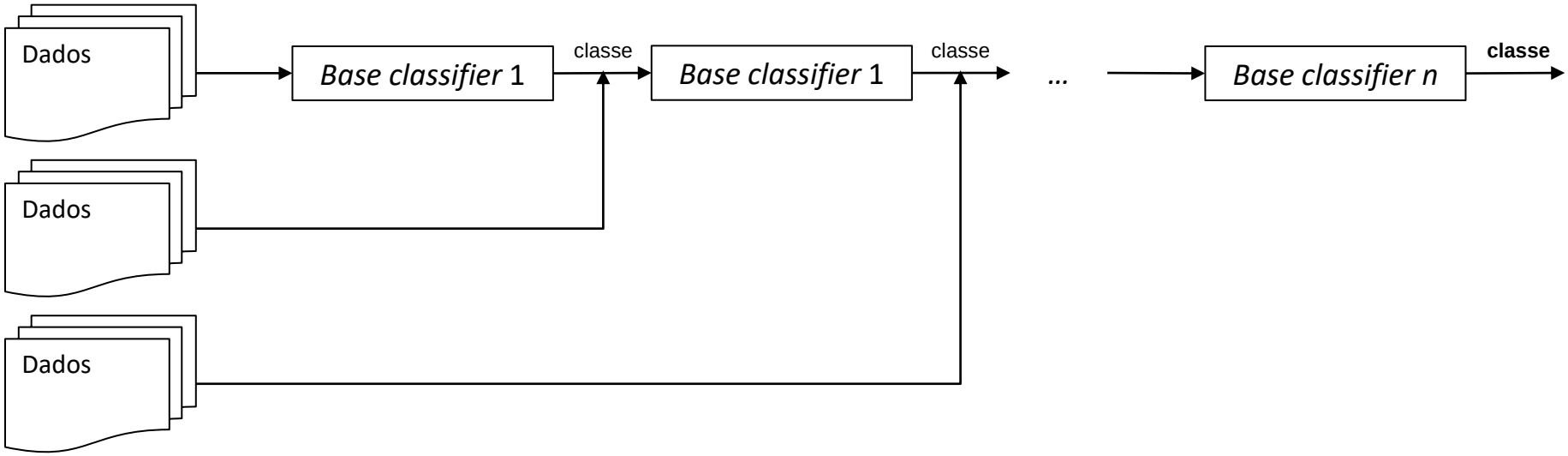
Stacking: um algoritmo de classificação é utilizado para prever a classe final do *ensemble*, tendo como *features* as previsões dos vários base *classifiers*

Ensemble learning: abordagem sequencial

A indução de um base *classifier* usa informação dos base *classifiers* previamente induzidos (ex: combinar previsões dos *base classifiers* previamente induzidos com as *features*)

Pode ser utilizado para:

- Tarefas de classificação hierárquicas
- Tarefas de *multilabel classification*



Bagging

Bagging

- Cada *base classifier* é induzido utilizando uma amostra do *trainset*
  - Amostras, definidas com uma abordagem *bootstrap**, têm o mesmo número de objetos que o *trainset*
- Combina as previsões dos base *classifiers* por *voting*
- Pode ser usado para técnicas de classificação instáveis (*unstable predictors*): a sua performance preditiva é afetada por alterações na composição do *trainset*. Ex: árvores de decisão, redes neuronais)
- Robusto a *overfitting* quando há ruído no *trainset*
- Número de modelos gerados é um *hyper-parameter* para a técnica de *bagging*
  - Quanto maior, menor a variância da previsão (e maior o custo computacional)
- Também pode ser utilizado para regressão
  - Combinação é feita pela média
- **Resultados:**
  - Previsões
  - *Base models* gerados

* Próximo capítulo

Bagging: definição de *hyper-parameters*

- **Número de *base models*** a gerar
  - Quanto maior melhor
    - Tendo atenção ao custo computacional
  - Geralmente, 100 é considerado uma boa opção

- ***Base learner*** a utilizar para gerar os modelos
  - Algumas abordagens usam árvores de decisão
  - Outras permitem que o utilizador escolha o *base learner*
  - Mais comuns: árvores de decisão, redes neuronais (devido à sua instabilidade)

Bagging: vantagens e desvantagens

Vantagens	Desvantagens

• Melhora a performance preditiva do *base learner*
  • dado que este é um *unstable predictor*
• Poucos *hyper-parameters* a definir
	• Devido à amostragem por *bootstrapping* tem uma componente aleatória
	  • mas a variabilidade dos resultados pode ser minimizada pela escolha do número de *base models* a gerar
	• Computacionalmente mais “caro” do que usar um modelo simples
	  • Mas pode ser executado em paralelo

Random Forests

Random Forests

- Combinam diversas árvores de decisão
- Semelhante ao *bagging*: Cada árvore de decisão é criada com uma diferente amostra obtida por *bootstrap*
- Diferente do *bagging*: em cada nó da árvore, em vez de escolher o *split* a partir de todas as *features*, é usado apenas um número predefinido de atributos aleatoriamente selecionados
- Boa escolha para *datasets* com muitas *features*
- **Resultados:**
  - Previsões
  - Estatísticas sore a importância das *features*

Random Forests: definição de *hyper-parameters*

- **Número de *base models*** a gerar
  - Recomendado: 1000
  - Para obter estatísticas mais confiáveis da importância das *features*: 5000

- **Número de *features*** a escolher aleatoriamente em cada nó
  - Depende do problema
  - Regra prática: $\sqrt{\#features}$

Random Forests: vantagens e desvantagens

Vantagens	Desvantagens

• Boa performance preditiva em diversos problemas
• Relativamente fáceis de interpretar
• Fácil de definir *hyper-parameters*
	• Computacionalmente "caro"
	  • Porque o número de árvores recomendado é elevado
	  • Mas pode ser executado em paralelo
	• Aleatoriedade
	  • Pode ser minimizado usando o número mínimo recomendado de árvores

Boosting

Boosting: algoritmo genérico

1. O *base learner* atribui a todas as instâncias pesos iguais

2. Repetir até ao limite do *base learner* ser atingido, ou a performance preditiva aumentar:

    1. Se houver algum erro de previsão causado pelo primeiro *base learner*, aumenta-se o peso das observações com erro

    2. Aplica-se o próximo *base learner*

Boosting: AdaBoost

- Um dos mais representantivos métodos de *boosting*

- Em cada iteração de treino, um *base classifier* é induzido utilizando o *trainset* e a cada instância é atribuído um peso de acordo com quão bem o modelo previu a sua classe
- Quanto mais difícil for a previsão da classe, maior será o peso associado à instância

- O peso de uma instância define a probabilidade de ser escolhido para o *trainset* do base *classifier* seguinte (sequencial)

- Bom para utilizar com *weak classifiers*: performance preditiva apenas ligeiramente superior à da previsão aleatória

- Pode ser utilizado para regressão: *gradient Boosting*, *KGBoost*

- **Resultados:**
  - Previsões

AdaBoost: definição de *hyper-parameters*

- **Número de iterações**
  - Autores do algoritmo utilizam: 100

  (*Freund, Y. and Shapire, R.E. (1996) Experiments with a new boosting algorithm, in Proceedings of the 13th International Conference on Machine Learning, ICML 1996, pp. 148–156.*)

AdaBoost: vantagens e desvantagens

Vantagens	Desvantagens

• Boa performance preditiva em diversos problemas
• Fácil de definir *hyper-parameters*
	• Computacionalmente “caro”
	  • Porque o número de modelos gerados depende do número de iterações
	  • Não é possível executar em paralelo (sequencial)
	• Difícil de interpretar

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.