Aula 16 - *Ensembles*

João Florindo

Instituto de Matemática, Estatística e Computação Científica
Universidade Estadual de Campinas - Brasil
florindo@unicamp.br

Outline

- IDEIA BÁSICA: Combinar vários classificadores de acurácia baixa para formar um único classificador de acurácia alta.
- Se os classificadores-base tiverem baixa correlação, o modelo combinado vai ter baixa variância.
- Árvores de decisão são os modelos-base mais usados.
- Estratégias: votação, *bagging*, *boosting* e *stacking*.

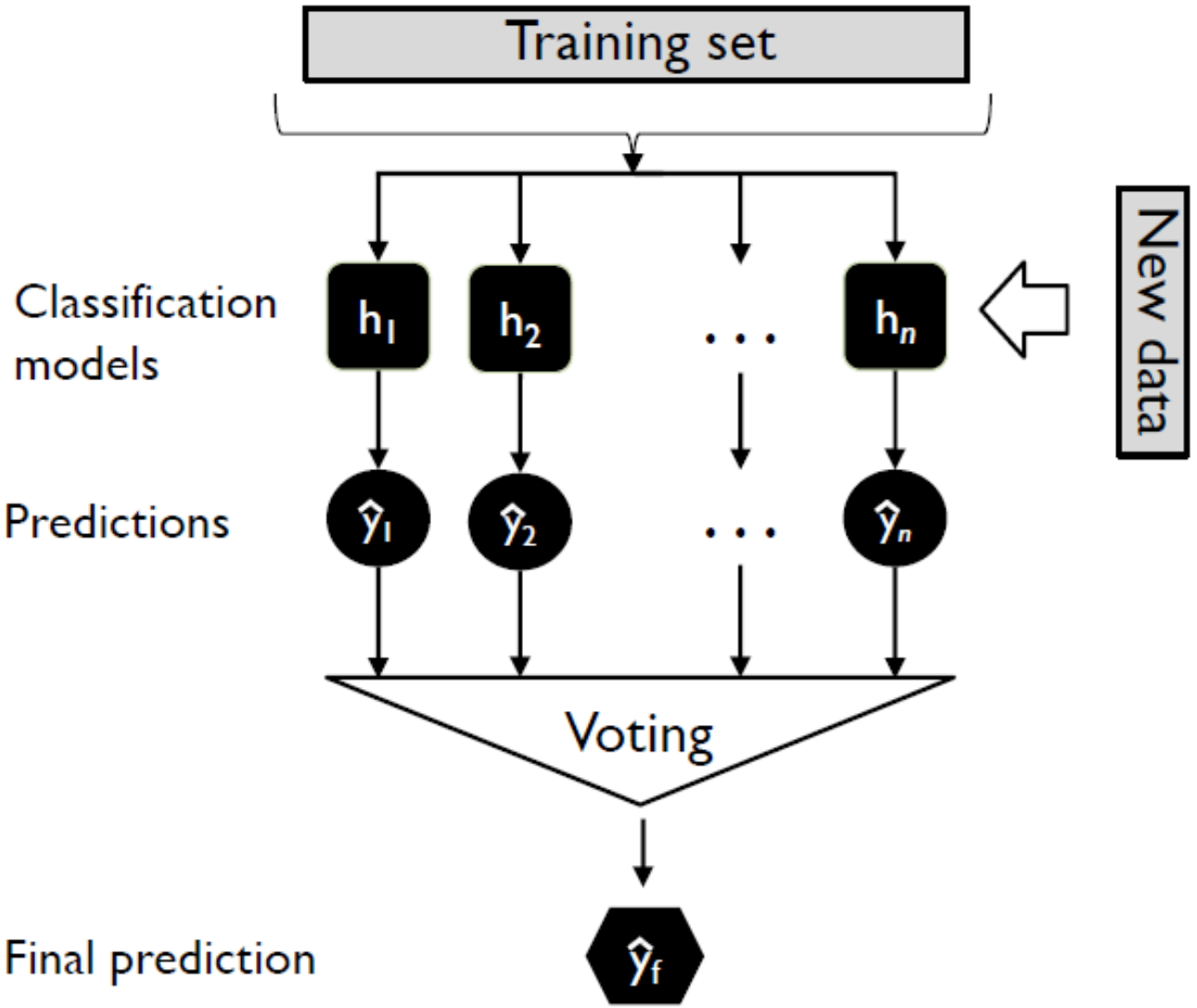
- IDEIA BÁSICA: Combinar vários classificadores de acurácia baixa para formar um único classificador de acurácia alta.
- Se os classificadores-base tiverem baixa correlação, o modelo combinado vai ter baixa variância.
- Árvores de decisão são os modelos-base mais usados.
- Estratégias: votação, *bagging*, *boosting* e *stacking*.

- IDEIA BÁSICA: Combinar vários classificadores de acurácia baixa para formar um único classificador de acurácia alta.
- Se os classificadores-base tiverem baixa correlação, o modelo combinado vai ter baixa variância.
- Árvores de decisão são os modelos-base mais usados.
- Estratégias: votação, *bagging*, *boosting* e *stacking*.

- IDEIA BÁSICA: Combinar vários classificadores de acurácia baixa para formar um único classificador de acurácia alta.
- Se os classificadores-base tiverem baixa correlação, o modelo combinado vai ter baixa variância.
- Árvores de decisão são os modelos-base mais usados.
- Estratégias: votação, *bagging*, *boosting* e *stacking*.

Outline

Votação por Maioria



Votação *Soft*

$$\hat{y} = \underset{j}{\text{argmax}} \sum_{i=1}^{n} w_i p_{ij},$$

em que p_{ij} é a probabilidade atribuída ao rótulo j pelo i-ésimo classificador e w_i são pesos (opcional). Poderíamos ter, por exemplo:

$$w_i = \frac{1}{n}, \quad \forall w_i \in \{w_1, \cdots, w_n\}.$$

NOTA

Os classificadores devem estar **calibrados**: Brier score.

Outline

- *Bagging* vem de *bootstrap aggregation*.

Definição - *Bootstrap*

Bootstrap é uma técnica para medir a incerteza de algum estimador (por exemplo, a média). A ideia básica é fazer reamostragens aleatórias com repetição.

- Seja uma população P e um conjunto de treinamento S amostrado de P ($S \sim P$). Suponha que queiramos saber o erro de uma estimativa da média de P usando apenas S.

- Geram-se então conjuntos de *bootstrap* $Z_1, Z_2, \cdots, Z_M$ a partir de S com repetição ($Z_m \sim S, |Z| = |S|$).

- *Bagging* vem de *bootstrap aggregation*.

Definição - *Bootstrap*

Bootstrap é uma técnica para medir a incerteza de algum estimador (por exemplo, a média). A ideia básica é fazer reamostragens aleatórias com repetição.

- Seja uma população P e um conjunto de treinamento S amostrado de P ($S \sim P$). Suponha que queiramos saber o erro de uma estimativa da média de P usando apenas S.

- Geram-se então conjuntos de *bootstrap* $Z_1, Z_2, \cdots, Z_M$ a partir de S com repetição ($Z_m \sim S, |Z| = |S|$).

- *Bagging* vem de *bootstrap aggregation*.

Definição - *Bootstrap*

Bootstrap é uma técnica para medir a incerteza de algum estimador (por exemplo, a média). A ideia básica é fazer reamostragens aleatórias com repetição.

- Seja uma população P e um conjunto de treinamento S amostrado de P ($S \sim P$). Suponha que queiramos saber o erro de uma estimativa da média de P usando apenas S.

- Geram-se então conjuntos de *bootstrap* $Z_1, Z_2, \cdots, Z_M$ a partir de S com repetição ($Z_m \sim S, |Z| = |S|$).

Seja n o número de amostras de *bootstrap*

for $i = 1$ **to** n **do**

  Extraia uma amostra de *bootstrap* $\mathcal{D}_i$ de tamanho m

  Treine um classificador base h_i sobre $\mathcal{D}_i$

end

$\hat{y} = mode\{h_1(\mathbf{x}), \cdots, h_n(\mathbf{x})\}$

Seja n o número de amostras de *bootstrap*

for $i = 1$ **to** n **do**

  Extraia uma amostra de *bootstrap* $\mathcal{D}_i$ de tamanho m

  Treine um classificador base h_i sobre $\mathcal{D}_i$

end

$\hat{y} = mode\{h_1(\mathbf{x}), \cdots, h_n(\mathbf{x})\}$

Definição - *Bagging*

No *ensemble*, treinamos um modelo de aprendizado G_m sobre cada conjunto Z_m. O **preditor agregado** é dado pela média:

$$G(x) = \sum_m \frac{G_m(x)}{M}.$$

- Preditores gerados desta forma tendem a ser menos correlacionados.
- Queda na variância compensa aumento no viés de cada preditor individual.
- Permite calcular o *out-of-bag error*: erro estimado usando a fração de S que não entra em Z_m.

Definição - *Bagging*

No *ensemble*, treinamos um modelo de aprendizado G_m sobre cada conjunto Z_m. O **preditor agregado** é dado pela média:

$$G(x) = \sum_m \frac{G_m(x)}{M}.$$

- Preditores gerados desta forma tendem a ser menos correlacionados.
- Queda na variância compensa aumento no viés de cada preditor individual.
- Permite calcular o *out-of-bag error*: erro estimado usando a fração de S que não entra em Z_m.

Definição - *Bagging*

No *ensemble*, treinamos um modelo de aprendizado G_m sobre cada conjunto Z_m. O **preditor agregado** é dado pela média:

$$G(x) = \sum_m \frac{G_m(x)}{M}.$$

- Preditores gerados desta forma tendem a ser menos correlacionados.
- Queda na variância compensa aumento no viés de cada preditor individual.
- Permite calcular o *out-of-bag error*: erro estimado usando a fração de S que não entra em Z_m.

Bagging + Árvores de Decisão

- Árvores são modelos com baixo viés e alta variância: propícios para *bagging*.
- Tratamento de valores faltantes: árvores que dividem naquele atributo são excluídas do *ensemble*.
- Custo computacional mais alto e menor interpretabilidade.
- Atributo muito "poderoso" tende a aparecer em muitas árvores do *ensemble*: aumenta a correlação!
- SOLUÇÃO: Florestas aleatórias - Só um sub-conjunto aleatório dos atributos podem ser usados em cada divisão.

Bagging + Árvores de Decisão

- Árvores são modelos com baixo viés e alta variância: propícios para *bagging*.
- Tratamento de valores faltantes: árvores que dividem naquele atributo são excluídas do *ensemble*.
- Custo computacional mais alto e menor interpretabilidade.
- Atributo muito "poderoso" tende a aparecer em muitas árvores do *ensemble*: aumenta a correlação!
- SOLUÇÃO: Florestas aleatórias - Só um sub-conjunto aleatório dos atributos podem ser usados em cada divisão.

Bagging + Árvores de Decisão

- Árvores são modelos com baixo viés e alta variância: propícios para *bagging*.
- Tratamento de valores faltantes: árvores que dividem naquele atributo são excluídas do *ensemble*.
- Custo computacional mais alto e menor interpretabilidade.
- Atributo muito "poderoso" tende a aparecer em muitas árvores do *ensemble*: aumenta a correlação!
- SOLUÇÃO: Florestas aleatórias - Só um sub-conjunto aleatório dos atributos podem ser usados em cada divisão.

Bagging + Árvores de Decisão

- Árvores são modelos com baixo viés e alta variância: propícios para *bagging*.
- Tratamento de valores faltantes: árvores que dividem naquele atributo são excluídas do *ensemble*.
- Custo computacional mais alto e menor interpretabilidade.
- Atributo muito "poderoso" tende a aparecer em muitas árvores do *ensemble*: aumenta a correlação!
- SOLUÇÃO: Florestas aleatórias - Só um sub-conjunto aleatório dos atributos podem ser usados em cada divisão.

Bagging + Árvores de Decisão

- Árvores são modelos com baixo viés e alta variância: propícios para *bagging*.
- Tratamento de valores faltantes: árvores que dividem naquele atributo são excluídas do *ensemble*.
- Custo computacional mais alto e menor interpretabilidade.
- Atributo muito "poderoso" tende a aparecer em muitas árvores do *ensemble*: aumenta a correlação!
- SOLUÇÃO: Florestas aleatórias - Só um sub-conjunto aleatório dos atributos podem ser usados em cada divisão.

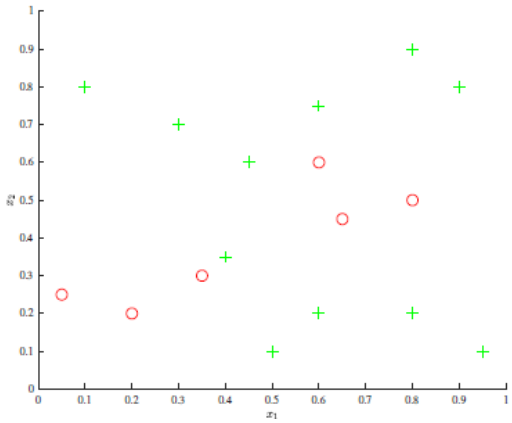
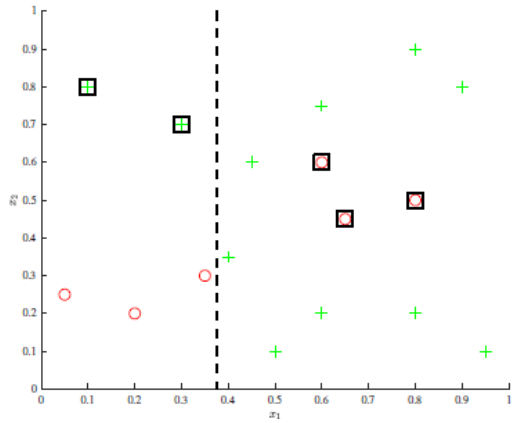
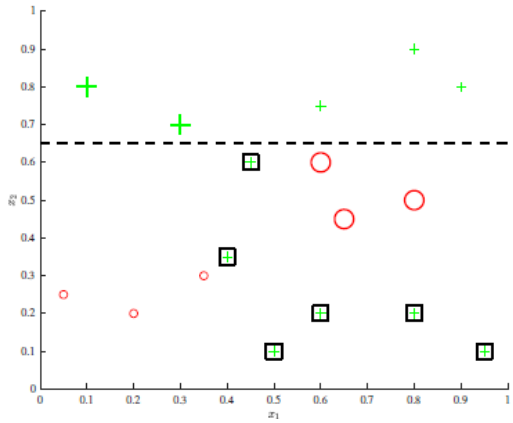
Outline

- Enquanto *bagging* reduz variância, *boosting* **reduz viés**.
- Classificadores-base com alto viés e baixa variância (**classificadores fracos**).
- Árvores de decisão se tornam classificadores fracos se puderem tomar apenas uma decisão antes da predição (*decision stump*).

- Enquanto *bagging* reduz variância, *boosting* **reduz viés**.
- Classificadores-base com alto viés e baixa variância (**classificadores fracos**).
- Árvores de decisão se tornam classificadores fracos se puderem tomar apenas uma decisão antes da predição (*decision stump*).

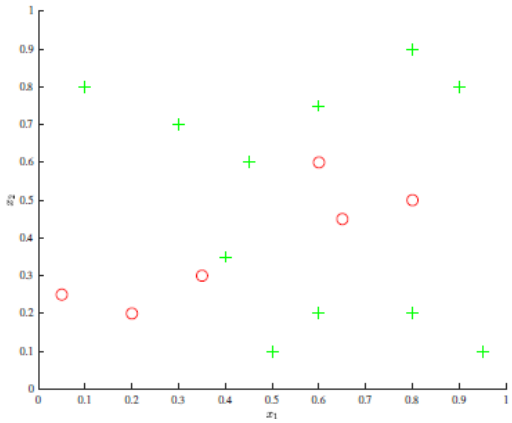
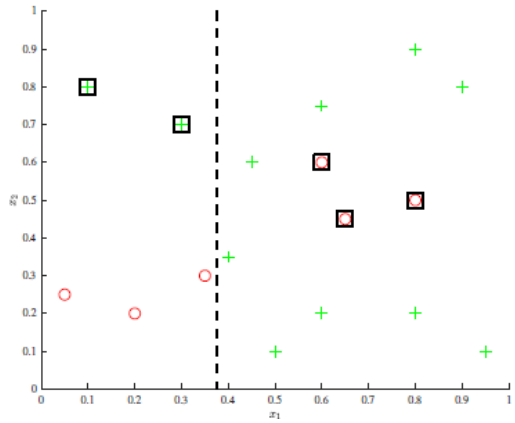
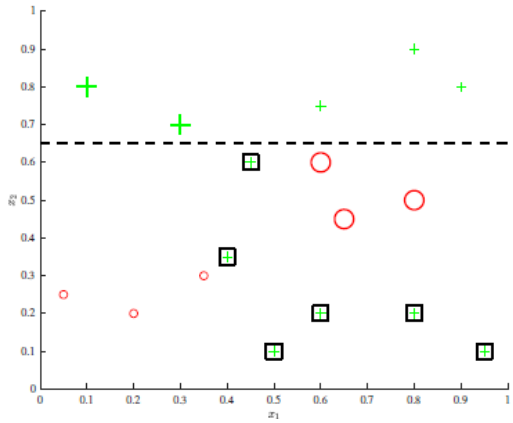
- Enquanto *bagging* reduz variância, *boosting* **reduz viés**.
- Classificadores-base com alto viés e baixa variância (**classificadores fracos**).
- Árvores de decisão se tornam classificadores fracos se puderem tomar apenas uma decisão antes da predição (*decision stump*).

Ideia Básica

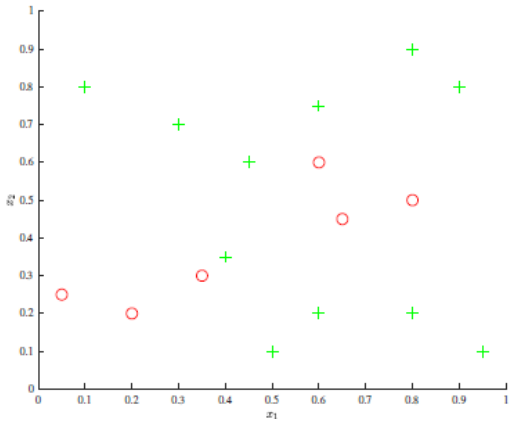
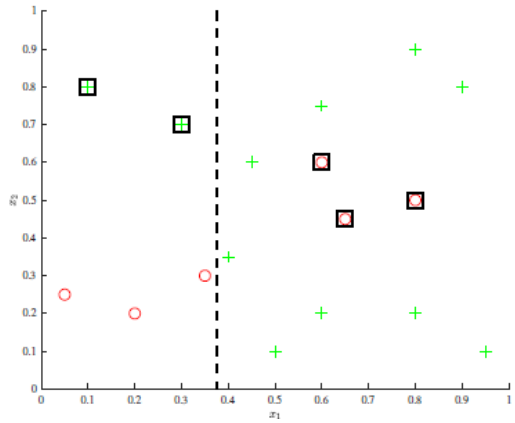
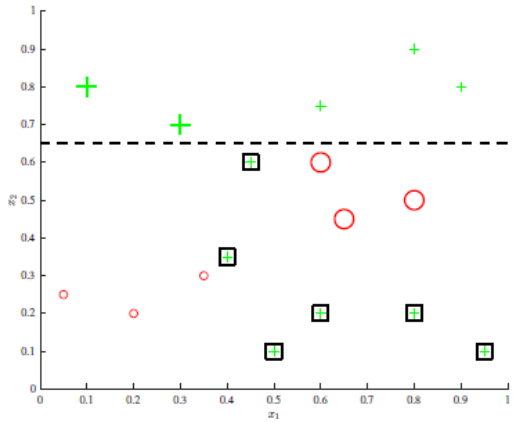
1. Partimos do conjunto à esquerda e treinamos um único *decision stump* no centro.
2. Observamos os exemplos classificados incorretamente e aumentamos o peso relativo desses exemplos.
3. Treina-se então um novo *decision stump* (direita) que é incentivado a acertar estes "exemplos mais difíceis".
4. Continua o processo e vai incrementalmente reajustando estes pesos, chegando-se ao final a um *ensemble* de classificadores fracos.

Ideia Básica

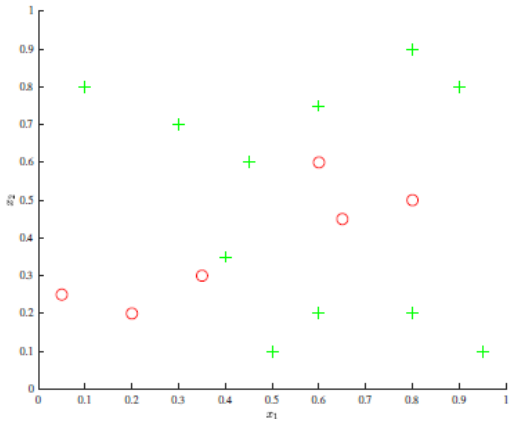
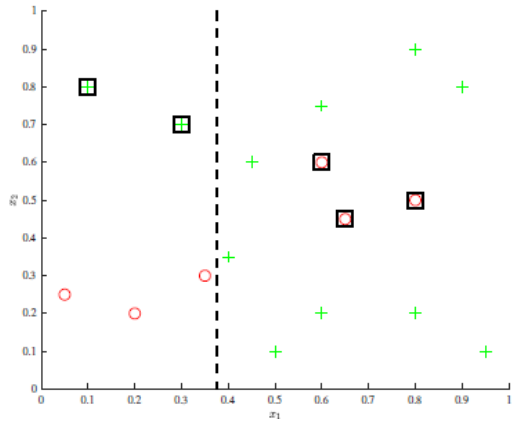
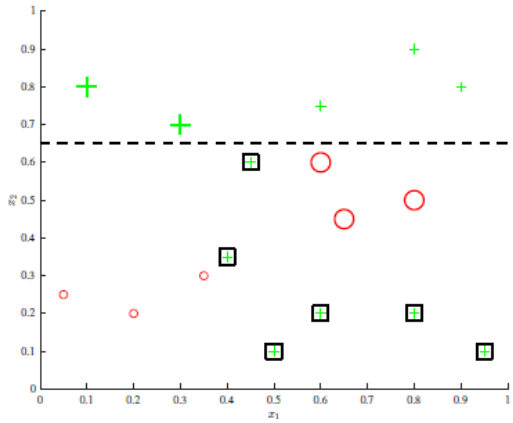
1. Partimos do conjunto à esquerda e treinamos um único *decision stump* no centro.
2. Observamos os exemplos classificados incorretamente e aumentamos o peso relativo desses exemplos.
3. Treina-se então um novo *decision stump* (direita) que é incentivado a acertar estes "exemplos mais difíceis".
4. Continua o processo e vai incrementalmente reajustando estes pesos, chegando-se ao final a um *ensemble* de classificadores fracos.

Ideia Básica

1. Partimos do conjunto à esquerda e treinamos um único *decision stump* no centro.
2. Observamos os exemplos classificados incorretamente e aumentamos o peso relativo desses exemplos.
3. Treina-se então um novo *decision stump* (direita) que é incentivado a acertar estes "exemplos mais difíceis".
4. Continua o processo e vai incrementalmente reajustando estes pesos, chegando-se ao final a um *ensemble* de classificadores fracos.

Ideia Básica

1. Partimos do conjunto à esquerda e treinamos um único *decision stump* no centro.
2. Observamos os exemplos classificados incorretamente e aumentamos o peso relativo desses exemplos.
3. Treina-se então um novo *decision stump* (direita) que é incentivado a acertar estes "exemplos mais difíceis".
4. Continua o processo e vai incrementalmente reajustando estes pesos, chegando-se ao final a um *ensemble* de classificadores fracos.

Adaboost

Algoritmo de *boosting* mais popular:

Entrada: Conjunto de dados rotulados $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$
Saída : *Ensemble* classificador $f(x)$
$w_i \leftarrow \frac{1}{N}$ para $i = 1, 2, \cdots, N$
for $m = 0$ **to** M **do**
  Ajustar o classificador fraco G_m ao conjunto de treino ponderado por w_i
  Calcular o erro ponderado $err_m = \frac{\sum_i w_i \mathbb{1}_{(y_i \neq G_m(x_i))}}{\sum w_i}$
  Calcular o peso $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
  $w_i \leftarrow w_i * \exp(\alpha_m \mathbb{1}_{(y_i \neq G_m(x_i))})$
end
$f(x) = \text{sign}(\sum_m \alpha_m G_m(x))$

Forward Stagewise Additive Modeling

Entrada: Conjunto de dados rotulados $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$
Saída : *Ensemble* classificador $f(x)$
Inicializar $f_0(x) = 0$
for $m = 0$ **to** M **do**
  Calcular $(\beta_m, \gamma_m) = \text{argmin}_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta G(x_i; \gamma))$
  Fazer $f_m(x) = f_{m-1}(x) + \beta_m G(x; y_i)$
end
$f(x) = f_m(x)$

- Adaboost é um caso particular com 2 classes e perda exponencial:

$$L(y, \hat{y}) = \exp(-y\hat{y}).$$

Forward Stagewise Additive Modeling

Entrada: Conjunto de dados rotulados $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$
Saída : *Ensemble* classificador $f(x)$
Inicializar $f_0(x) = 0$
for $m = 0$ **to** M **do**
  Calcular $(\beta_m, \gamma_m) = \text{argmin}_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta G(x_i; \gamma))$
  Fazer $f_m(x) = f_{m-1}(x) + \beta_m G(x; y_i)$
end
$f(x) = f_m(x)$

- Adaboost é um caso particular com 2 classes e perda exponencial:

$$L(y, \hat{y}) = \exp(-y\hat{y}).$$

Gradient Boosting

- Métodos como **xgboost** usam otimização numérica no *Forward Stagewise Additive Modeling*.
- Passos do gradiente descendente devem ser feitos dentro da classe de modelos, não são arbitrários.
- No **gradient boosting**, o gradiente em cada ponto é calculado em relação ao preditor atual (tipicamente um *decision stump*):

$$g_i = \frac{\partial L(y, f(x_i))}{\partial f(x_i)}.$$

- Treina-se então um novo preditor de regressão que se ajusta a este gradiente e é usado como passo do gradiente:

$$\gamma_i = \text{argmin}_\gamma \sum_{i=1}^{N} (g_i - G(x_i; \gamma))^2.$$

Gradient Boosting

- Métodos como **xgboost** usam otimização numérica no *Forward Stagewise Additive Modeling*.

- Passos do gradiente descendente devem ser feitos dentro da classe de modelos, não são arbitrários.

- No **gradient boosting**, o gradiente em cada ponto é calculado em relação ao preditor atual (tipicamente um *decision stump*):

$$g_i = \frac{\partial L(y, f(x_i))}{\partial f(x_i)}.$$

- Treina-se então um novo preditor de regressão que se ajusta a este gradiente e é usado como passo do gradiente:

$$\gamma_i = \text{argmin}_\gamma \sum_{i=1}^{N} (g_i - G(x_i; \gamma))^2.$$

Gradient Boosting

- Métodos como **xgboost** usam otimização numérica no *Forward Stagewise Additive Modeling*.
- Passos do gradiente descendente devem ser feitos dentro da classe de modelos, não são arbitrários.
- No **gradient boosting**, o gradiente em cada ponto é calculado em relação ao preditor atual (tipicamente um *decision stump*):

$$g_i = \frac{\partial L(y, f(x_i))}{\partial f(x_i)}.$$

- Treina-se então um novo preditor de regressão que se ajusta a este gradiente e é usado como passo do gradiente:

$$\gamma_i = \text{argmin}_\gamma \sum_{i=1}^{N} (g_i - G(x_i; \gamma))^2.$$

Gradient Boosting

- Métodos como **xgboost** usam otimização numérica no *Forward Stagewise Additive Modeling*.

- Passos do gradiente descendente devem ser feitos dentro da classe de modelos, não são arbitrários.

- No **gradient boosting**, o gradiente em cada ponto é calculado em relação ao preditor atual (tipicamente um *decision stump*):

$$g_i = \frac{\partial L(y, f(x_i))}{\partial f(x_i)}.$$

- Treina-se então um novo preditor de regressão que se ajusta a este gradiente e é usado como passo do gradiente:

$$\gamma_i = \text{argmin}_\gamma \sum_{i=1}^{N} (g_i - G(x_i; \gamma))^2.$$

Gradient Boosting

Em geral temos 3 passos:

1. Construir uma árvore base (apenas nó raiz)
2. Construir a próxima árvore com base nos erros da árvore anterior
3. Combinar árvores dos passos 1 e 2 e voltar ao passo 2.

Exemplo (Regressão):

x_1# Rooms	x_2=City	x_3=Age	y=Price (in million US Dollars)
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

Gradient Boosting

Em geral temos 3 passos:

1. Construir uma árvore base (apenas nó raiz)
2. Construir a próxima árvore com base nos erros da árvore anterior
3. Combinar árvores dos passos 1 e 2 e voltar ao passo 2.

Exemplo (Regressão):

x_1# Rooms	x_2=City	x_3=Age	y=Price (in million US Dollars)
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

Gradient Boosting

Em geral temos 3 passos:

1. Construir uma árvore base (apenas nó raiz)
2. Construir a próxima árvore com base nos erros da árvore anterior
3. Combinar árvores dos passos 1 e 2 e voltar ao passo 2.

Exemplo (Regressão):

x_1# Rooms	x_2=City	x_3=Age	y=Price (in million US Dollars)
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

Gradient Boosting

Em geral temos 3 passos:

1. Construir uma árvore base (apenas nó raiz)
2. Construir a próxima árvore com base nos erros da árvore anterior
3. Combinar árvores dos passos 1 e 2 e voltar ao passo 2.

Exemplo (Regressão):

x_1# Rooms	x_2=City	x_3=Age	y=Price (in million US Dollars)
5	Boston	30	1.5
10	Madison	20	0.5
6	Lansing	20	0.25
5	Waunakee	10	0.1

Gradient Boosting

- **Passo 1**: Como vimos, na regressão a previsão em um nó é a média das saídas. Para apenas um nó raiz:

$$\hat{y}_1 = \frac{1}{m} \sum_{i=1}^{m} y^{(i)} = 0.5875.$$

- **Passo 2**: Calculamos os **pseudo-resíduos** - diferença entre o valor verdadeiro e o predito, na regressão:

$$r_1 = y_1 - \hat{y}_1.$$

x_1# Rooms	x_2=City	x_3=Age	y=Price	r_1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunakee	10	0.1	0.1 - 0.5875 = -0.4875

Gradient Boosting

- **Passo 1**: Como vimos, na regressão a previsão em um nó é a média das saídas. Para apenas um nó raiz:

$$\hat{y}_1 = \frac{1}{m} \sum_{i=1}^{m} y^{(i)} = 0.5875.$$

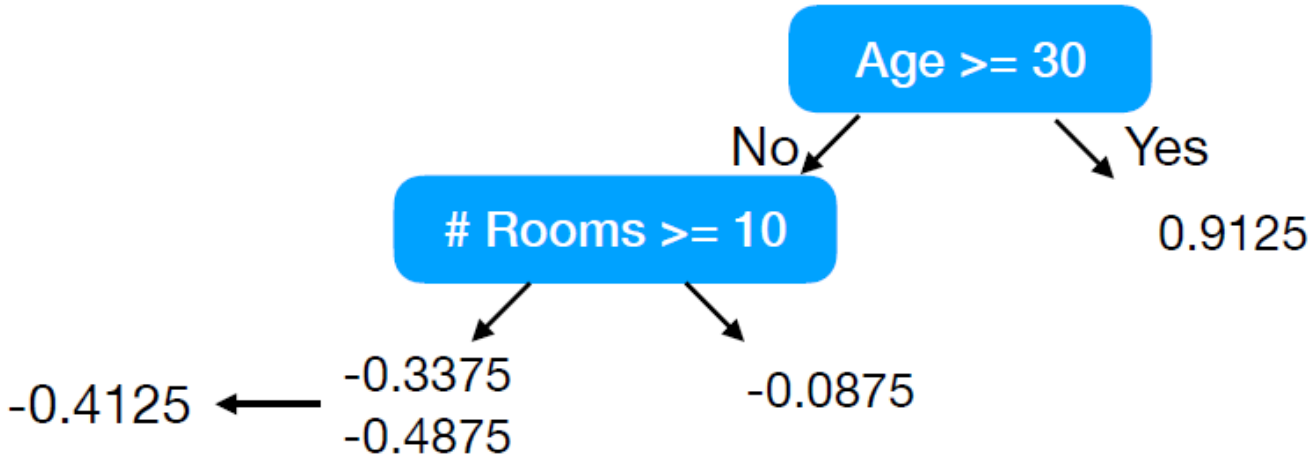
- **Passo 2**: Calculamos os **pseudo-resíduos** - diferença entre o valor verdadeiro e o predito, na regressão:

$$r_1 = y_1 - \hat{y}_1.$$

x_1# Rooms	x_2=City	x_3=Age	y=Price	r_1=Res
5	Boston	30	1.5	1.5 - 0.5875 = 0.9125
10	Madison	20	0.5	0.5 - 0.5875 = -0.0875
6	Lansing	20	0.25	0.25 - 0.5875 = -0.3375
5	Waunakee	10	0.1	0.1 - 0.5875 = -0.4875

Gradient Boosting

- Ajusta uma nova árvore **aos resíduos**. A profundidade máxima dessa árvore é um hiperparâmetro. EXEMPLO:



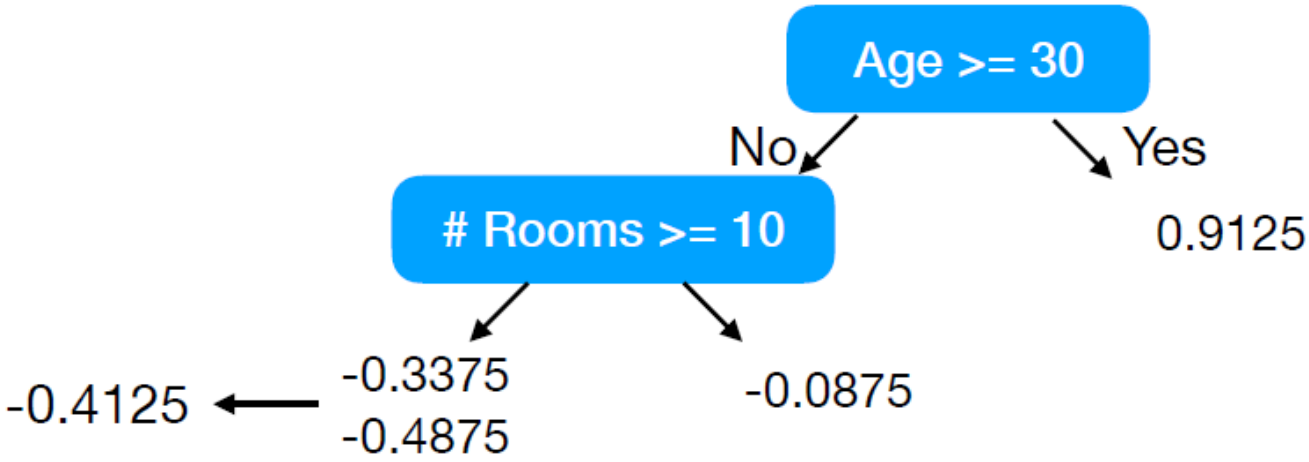
- **Passo 3**: Combinar árvore do Passo 1 (nó raiz) com a do Passo 2. Para *Lansing*, por exemplo, teríamos:

$$\hat{y}_{Lansing} = 0.5875 + \alpha(-0.4125),$$

em que $\alpha \in [0, 1]$ é a taxa de aprendizado (tipicamente 0.1 ou 0.01).

Gradient Boosting

- Ajusta uma nova árvore **aos resíduos**. A profundidade máxima dessa árvore é um hiperparâmetro. EXEMPLO:



- **Passo 3**: Combinar árvore do Passo 1 (nó raiz) com a do Passo 2. Para *Lansing*, por exemplo, teríamos:

$$\hat{y}_{Lansing} = 0.5875 + \alpha(-0.4125),$$

em que $\alpha \in [0, 1]$ é a taxa de aprendizado (tipicamente 0.1 ou 0.01).

Gradient Boosting

- Em seguida, voltamos ao Passo 2 e repetimos os passos 2 e 3 T vezes.
- Usamos as predições do Passo 3 para calcular novos resíduos e ajustar a próxima árvore. Por exemplo, para Lansing, com $\alpha = 0.1$, teríamos:

$$r_{Lansing,2} = y_{Lansing} - \hat{y}_{Lansing} = 0.25 - 0.5875 + 0.1(-0.4125) = -0.37875$$

Gradient Boosting

Entrada: $\mathcal{D}$: Conjunto de treinamento $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$
T: número de iterações
Saída : *Ensemble* classificador $h_t(x)$
Escolher uma função de perda diferenciável $L(y^{(i)}, h(x^{(i)})$
Passo 1: Inicializar o modelo $h_0(x) = \text{argmin}_{\hat{y}} \sum_{i=1}^{m} L(y^{(i)}, \hat{y})$ `// nó raiz`
Passo 2:
for $t = 1$ **to** T **do**

- **A.** Calcule o pseudo-resíduo $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(x^{(i)})}{\partial h(x^{(i)})}\right]_{h(x)=h_{t-1}(x)}$, para $i = 1$ até m
- **B.** Ajuste uma árvore aos valores de $r_{i,t}$ criando nós-folhas $R_{j,t}$, para $j = 1, \cdots, J_t$
- **C.**
- **for** $j = 1$ **to** J_t **do**
  - $\hat{y}_{j,t} = \text{argmin}_{\hat{y}} \sum_{x^{(i)} \in R_{i,j}} L(y^{(i)}, h_{t-1}(x^{(i)}) + \hat{y})$
- **end**
- **D.** Atualize $h_t(x) = h_{t-1}(x) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} \mathbb{1}(x \in R_{j,t})$

end
Passo 3: Retorne $h_t(x)$

Gradient Boosting

- Lembre-se que na regressão a função de perda é

$$L(y^{(i)}, h(x^{(i)}) = \frac{1}{2}(y^{(i)} - h(x^{(i)})^2$$

e, como já vimos também:

$$\frac{\partial L(y^{(i)}, h(x^{(i)})}{\partial h(x^{(i)})} = -(y^{(i)} - h(x^{(i)}).$$

- Note que ao final temos uma combinação das T árvores:

$$h_0(x) + \alpha \hat{y}_{j,t=1} + \cdots + \alpha \hat{y}_{j,T}.$$

- O processo na classificação é similar, apenas trocando a função de perda pela log-likelihood negativa.

Gradient Boosting

- Lembre-se que na regressão a função de perda é

$$L(y^{(i)}, h(x^{(i)}) = \frac{1}{2}(y^{(i)} - h(x^{(i)})^2$$

e, como já vimos também:

$$\frac{\partial L(y^{(i)}, h(x^{(i)})}{\partial h(x^{(i)})} = -(y^{(i)} - h(x^{(i)}).$$

- Note que ao final temos uma combinação das T árvores:

$$h_0(x) + \alpha \hat{y}_{j,t=1} + \cdots + \alpha \hat{y}_{j,T}.$$

- O processo na classificação é similar, apenas trocando a função de perda pela log-likelihood negativa.

Gradient Boosting

- Lembre-se que na regressão a função de perda é

$$L(y^{(i)}, h(x^{(i)}) = \frac{1}{2}(y^{(i)} - h(x^{(i)})^2$$

  e, como já vimos também:

$$\frac{\partial L(y^{(i)}, h(x^{(i)})}{\partial h(x^{(i)})} = -(y^{(i)} - h(x^{(i)}).$$

- Note que ao final temos uma combinação das T árvores:

$$h_0(x) + \alpha \hat{y}_{j,t=1} + \cdots + \alpha \hat{y}_{j,T}.$$

- O processo na classificação é similar, apenas trocando a função de perda pela log-likelihood negativa.

Outline

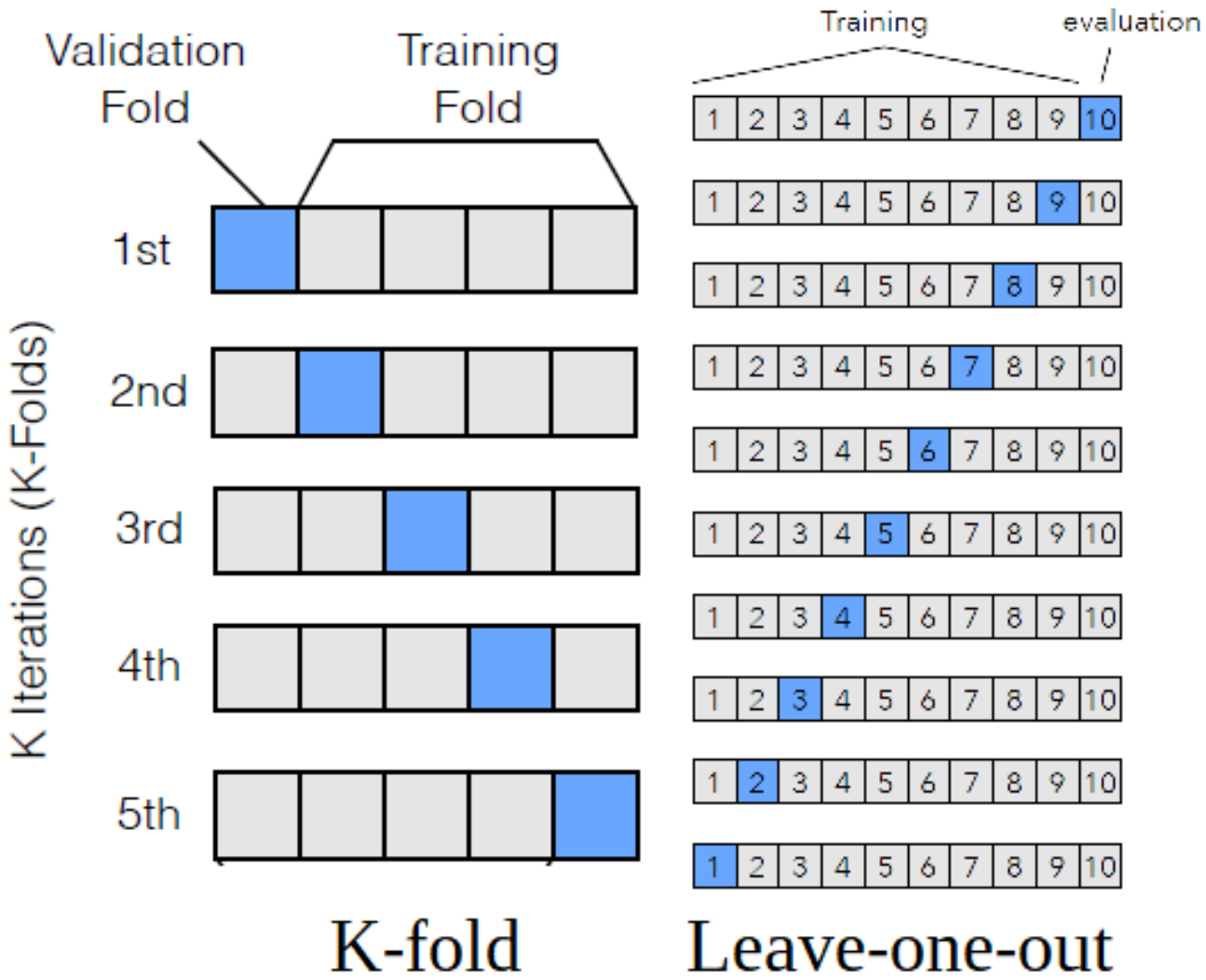
- Meta-aprendizado: usa a saída de modelos-base de aprendizado como entrada para um novo "meta-aprendiz".
- Abordagem ingênua:

Entrada: $\mathcal{D}$: Conjunto de treinamento $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$
Saída : *Ensemble* classificador h_E
Passo 1: Aprender classificadores-base
for $t = 1$ **to** T **do**
  Ajuste um modelo-base h_t sobre $\mathcal{D}$
end
Passo 2: Construa um novo conjunto $\mathcal{D}'$ a partir de $\mathcal{D}$
for $i = 1$ **to** m **do**
  Adicione $(x'^{(i)}, y^{(i)})$ ao novo conjunto, em que
  $x'^{(i)} = \{h_1(x^{(i)}), \cdots, h_T(x^{(i)})\}$
end
Passo 3: Aprenda um meta-classificador h_E
Retorne $h_E(\mathcal{D}')$

- Meta-aprendizado: usa a saída de modelos-base de aprendizado como entrada para um novo "meta-aprendiz".
- Abordagem ingênua:

Entrada: $\mathcal{D}$: Conjunto de treinamento $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$
Saída : *Ensemble* classificador h_E
Passo 1: Aprender classificadores-base
for $t = 1$ **to** T **do**
  Ajuste um modelo-base h_t sobre $\mathcal{D}$
end
Passo 2: Construa um novo conjunto $\mathcal{D}'$ a partir de $\mathcal{D}$
for $i = 1$ **to** m **do**
  Adicione $(x'^{(i)}, y^{(i)})$ ao novo conjunto, em que
  $x'^{(i)} = \{h_1(x^{(i)}), \cdots, h_T(x^{(i)})\}$
end
Passo 3: Aprenda um meta-classificador h_E
Retorne $h_E(\mathcal{D}')$

- PROBLEMA: Forte tendência a *overfitting*.
- Solução: usar K-fold ou *leave-one-out*.



Com validação cruzada:

Entrada: $\mathcal{D}$: Conjunto de treinamento $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$
Saída : *Ensemble* classificador h_E
Passo 1: Aprender classificadores-base
Construir novo conjunto $\mathcal{D}' = \{\}$
Particionar $\mathcal{D}$ aleatoriamente em k subconjuntos de tamanhos iguais
$\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_k\}$
for $j = 1$ **to** k **do**
  for $t = 1$ **to** T **do**
    Ajuste um modelo-base h_t sobre $\mathcal{D} \backslash \mathcal{D}_k$
  end
  for $i = 1$ **to** $m \in |\mathcal{D} \backslash \mathcal{D}_k|$ **do**
    Adicione $(x'^{(i)}, y^{(i)})$ ao novo conjunto $\mathcal{D}'$, em que
    $x'^{(i)} = \{h_1(x^{(i)}), \cdots, h_T(x^{(i)})\}$
  end
end
Passo 3: Aprenda um meta-classificador h_E
Retorne $h_E(\mathcal{D}')$