

# Stacking utilizado nos experimentos

*Raphael Rodrigues Campos*

*15 abril, 2016*

## Stacking

Stacking também conhecido como “Stacked Generalization” é um método para combinar múltiplos classificadores usando algoritmos de aprendizagem heterogêneos  $L_1, \dots, L_N$  sobre um único conjunto de dados  $D$ , que consiste de exemplos  $e_i = (x_i, y_i)$ , onde  $x_i$  é o vetor de atributos e  $y_i$  sua classificação.

## Stacking Framework

O stacking framework utilizado é baseado no descrito em [1] David H. Wolpert, “Stacked Generalization”, Neural Networks, 5, 241–259, 1992. Foi utilizado um stacking de dois níveis (o framework não se limita a apenas dois níveis, é possível fazer o stacking de quantos níveis julgar necessário), que pode ser dividido em duas fases. Na primeira fase, um conjunto de classificadores do nível base  $C_1, C_2, \dots, C_N$  é gerado, onde  $C_i = L_i(D)$ . Na segunda fase um classificador do meta-nível aprende a combinar as saídas dos classificadores do nível base.

Para gerar o conjunto de treino para o aprendizado do classificador do meta-nível, pode-se aplicar o procedimento **leave-one-out** ou **cross validation**. Por questões óbvias de custo computacional, é utilizado nesse relatório cross validation, mais especificamente **5-fold cross validation**. Cada classificador do nível base aprende usando  $D - F_k$  deixando o  $k$ -ésimo *fold* para teste:  $\forall i = 1, \dots, N : \forall k = 1, \dots, 5 : C_i^k = L_i(D - F_k)$ . Agora, os classificadores recém aprendidos são usados para gerar as previsões para  $\forall x_j \in F_k : \hat{y}_j^i = C_i^k(x_j)$ . O conjunto de treino do meta-nível consiste de exemplos da seguinte forma  $((\hat{y}_i^1, \dots, \hat{y}_i^N), y_i)$ , onde os atributos são as previsões dos  $s$  classificadores do nível base e a classe é a classe correta sabida de antemão.

## Exemplo

Esse procedimento pode parecer complicado, mas na verdade é simples. Como um exemplo, vamos gerar alguns dados sintéticos com a função “saída = soma dos três componentes de entrada”. Nosso conjunto de treino  $D$  consiste de 5 pares de entrada e saída  $\{((0, 0, 0), 0), ((1, 0, 0), 1), ((1, 2, 0), 3), ((1, 1, 1), 3), ((1, -2, 4), 3)\}$ , todas as entradas sem ruídos. Vamos rotular esses 5 pares de entrada e saída como  $F_1$  até  $F_5$  (Então por exemplo  $D - F_2$  consiste dos quatros pares  $\{((0, 0, 0), 0), ((1, 2, 0), 3), ((1, 1, 1), 3), ((1, -2, 4), 3)\}$ ). Nesse exemplo, temos dois classificadores do nível base  $C_1$  e  $C_2$ , e um único classificador do meta-nível  $\Gamma$ . O conjunto de treino do meta-nível  $D'$  é dado pelo cinco pares de entrada e saída  $\{((C_1^k(F_k), C_2^k(F_k)), \text{componente de saída de } F_k) : \forall k \in \{1, \dots, 5\} \text{ e } C_i^k = L_i(D - F_k)\}$  (Esse espaço do meta-nível possui duas dimensões de entrada e uma de saída). Ou seja, a instância do conjunto de treino do meta-nível correspondente a  $k = 1$  tem o componente de saída 0 e entrada  $(C_1^1((0, 0, 0)), C_2^1((0, 0, 0)))$ . Agora nos é dado um exemplo de teste no formato do nível base  $(x_1, x_2, x_3)$ . Nós predizemos seu valor com  $\Gamma((C_1((x_1, x_2, x_3)), (C_2((x_1, x_2, x_3))))$ , onde  $C_1$  e  $C_2$  foram treinados com todo  $D$ , e  $\Gamma$  com  $D'$ . Em outras palavras, nós predizemos o valor da entrada de teste  $q = (x_1, x_2, x_3)$  treinando  $\Gamma$  em  $D'$  e assim predizendo a entrada formada pelas previsões do valor do exemplo de teste  $q$ , de ambos classificadores do nível base  $C_1$  e  $C_2$ , que por suas vezes foram treinados com todo  $D$ .

## Stacking com distribuições de probabilidade

Usar probabilidade para gerar o conjunto de treino do meta-nível é mais vantajoso já que disponibiliza mais informação acerca das previsões feitas pelos classificadores do nível base. Essa informações adicionais

permitem que não seja usado somente a predição, mas também a confiança de cada classificador do nível base.

Nessa abordagem, cada classificador do nível base prediz uma Distribuição de Probabilidade (DP) sobre todas as classes possíveis. Então, a predição do classificador do nível base  $C$  aplicado a um exemplo  $x$  é a DP:  $p^C(x) = (p^C(c_1|x), \dots, p^C(c_m|x))$ , onde  $\{c_1, \dots, c_m\}$  é o conjunto de possíveis valores para as classes e  $p^C(c_i|x)$  descreve a probabilidade do exemplo  $x$  ser da classe  $c_i$  estimado pelo classificador  $C$ . A classe  $c_j$  com maior probabilidade será classe predita por  $C$ . Dessa forma, os atributos do meta-nível serão as probabilidades previstas para cada classe possível por cada classificador do nível base. O número total de atributos no conjunto de treino do meta-nível seria  $Nm$ ,  $m$  atributos para cada classificador do nível base.

Os experimentos rodados até então utilizaram o stacking framework com DPs.

## Stacking com DP, Entropia e probabilidade máxima

No artigo [2] Is combining classifiers better than selecting the best one, os autores propõem uma extensão para esse framework com DP expandindo o número de meta-atributos. Esses novos meta-atributos seriam:

- A distribuição de probabilidade multiplicada pela probabilidade máxima:  $p_{C_j} = p^{C_j}(c_i|x) \times M_{C_j}(x) = p^{C_j}(c_i|x) \times \max_{i=1}^m(p^{C_j}(c_i|x))$ ,  $\forall i \in \{1, \dots, m\}$  e  $\forall j \in \{1, \dots, N\}$ .
- As entropias das distribuições de probabilidade:  $E_{C_j}(x) = -\sum_{i=1}^m p^{C_j}(c_i|x) \cdot \log_2(p^{C_j}(c_i|x))$ .

O número total de atributos do meta-nível é  $N(2m + 1)$ .

A ideia é obter ainda mais informações em relação à predição feita pelos classificadores do nível base. Como Ting and Witten (1999) disseram: o uso de distribuição de probabilidades tem a vantagem de capturar não apenas as predições dos classificadores do nível base, mas também, suas certezas. Os atributos adicionais tentam capturar a certeza de forma mais explícita.

Entropia é uma medida de incerteza. Quanto maior a entropia da distribuição menor é a certeza sobre a predição. A probabilidade máxima de uma DP  $M_{C_j}$  também contém informação sobre certeza da predição: quanto maior  $M_{C_j}$  for mais certo daquela resposta o classificador do nível base está, e vice versa.

Essa é uma ideia para aplicarmos futuramente. Nesse momento continuarei utilizando somente a DP.

## SVM

Nessa seção, vamos dar uma visão geral sobre Support Vector Machine (SVM) utilizado nos experimentos.

Seja  $x_i \in R^d$  um vetor de características. Nosso objetivo é projetar um classificador, por exemplo, que associa a cada vetor  $x_i$  um rótulo positivo ou negativo baseado no critério desejado.

O vetor  $x_i$  é classificado olhando o sinal do resultado da função  $f(x_i, w) = w^T x_i$ . O objetivo é aprender a estimar os parâmetros  $w \in R^d$  de tal forma que o sinal é positivo se o vetor  $x_i$  pertence a classe positiva e negativo caso contrário. De fato, na formulação padrão do SVM o objetivo é ter o valor da função no mínimo 1 no primeiro caso, e no máximo -1\* no segundo, impondo uma margem.

O parâmetro  $w$  é estimado ou aprendido ajustando a função a um conjunto de treino de  $n$  pares de exemplos  $(x_i, y_i)$ ,  $i = 1, \dots, n$ , onde  $y_i \in \{-1, 1\}$  são os rótulos dos correspondentes vetores de características. A qualidade do ajuste é mensurada pela função de perda que, em SVMs padrões, é a **hinge loss**:

$$l_i(w, x_i) = \max(0, 1 - y_i w^T x_i) \quad (1)$$

Note que a **hinge loss** é zero apenas se o valor de  $w^T x_i$  é no mínimo 1 ou no máximo -1, dependendo do rótulo de  $y_i$ .

Somente ajustar ao treino é normalmente insuficiente. Para que a função seja capaz de generalizar para dados nunca visto, é preferível um **trade off** entre a acurácia do ajuste e complexidade do modelo. Dessa forma, é adicionado um termo de regularização a formula, o regularizador na formulação padrão é mensurado pela normado vetor de pesos  $\|w\|^2$ . Tirando-se a média da perda de todo os exemplos de treino e adicionando-se a ela o regularizador ponderado pelo parâmetro  $\lambda$  produz uma função objetiva de perda regularizada:

$$E(w) = \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i f(w, x)) \quad (2)$$

Note que a função objetiva é convexa, desse modo existe um único ótimo global.

A função  $f(x_i, w)$  considerada até aqui é linear em sem viés. A subseção seguinte discute como um termo de viés pode ser adicionado ao SVM.

## Adicionando viés

É comum adicionar a função do SVM um termo de viés  $b$ , e considerar a nova função  $f(x_i, w) = w^T x_i + b$ . Na prática termo de viés pode ser crucial para ajustar os dados de treino de forma ótima, já que não há razão que o produto interno  $w^T x_i$  devesse ser naturalmente centrado em zero. Alguns algoritmos de aprendizado do SVM podem estimar ambos  $w$  e  $b$  diretamente. Porém, outros algoritmos como **Stochastic Gradient Descent (SGD)** e **Stochastic Dual Coordinate Ascent (SDCA)** não podem (ambos usados pelo LIBLINEAR). Nesse caso, uma simples solução é adicionar um termo constante  $B > 0$  ao dado, por exemplo, considere os vetores estendidos:

$$\bar{x}_i = \begin{bmatrix} x_i \\ B \end{bmatrix}, \bar{w} = \begin{bmatrix} w \\ w_b \end{bmatrix}.$$

De modo que função incorpore implicitamente o termo de viés  $b = Bw_b$ :

$$\bar{w}^T \bar{x}_i = w^T x_i + Bw_b \quad (3)$$

A desvantagem dessa redução é que o termo  $w_b^2$  torna parte do regularizador do SVM, que encole o viés  $b$  em direção a zero. Esse efeito pode ser aliviado tomando valor de  $B$  suficientemente grandes, por causa disso  $\|w\|^2 \gg w_b^2$  e o efeito de encolimento pode ser ignorado. Infelizmente, fazer  $B$  muito grande faz o problema numericamente desbalanceado, assim uma troca justa entre encolimento e estabilidade é buscada. Tipicamente, isso é obtido normalizando o dado, para que tenha uma norma Euclidiana unitária e, então, escolhendo  $B \in [1, 10]$ .

Referências:

- <http://www.vlfeat.org/api/svm-fundamentals.html>
- <https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf>