Stacking utilizado nos experimentos

Raphael Rodrigues Campos

28 abril, 2016

Capítulo 1

Overview

1.0.1 Stacking

Stacking também conhecido como "Stacked Generalization" é um método para combinar multiplos classificadores usando algoritmos de aprendizados heterogêneos $L_1, ..., L_N$ 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.

1.0.1.1 Stacking Framework

O staking 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 $C_1, C_2, ..., C_N$ é gerado, onde $C_i = L_i(D)$. Na segunda fase um classidicador 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: $\forall i = 1, ..., N : \forall k = 1, ..., 5 : C_i^k = L_i(D - F_k)$. Agora, os classificadores recém aprendidos são usados para gerar as predições 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 $((\hat{y}_i^1, ..., \hat{y}_i^N), y_i)$, onde os atributos são as predicóes do s classificadores do nível base e a classe é a classe correta sabida de antemão.

1.0.1.1.1 Exemplo Esse procedimento pode parecer complicado, mas na verdade é simples. Como um exemplo, vamos gerar alguns dados sintéticos com a função "saída = soma do 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 Γ. O conjunto de treino do meta-nível D' é dado pelo cinco pares de entrada e saída $\{$ $((C_1^k(F_k), C_2^k(F_k))$, componente de saída de $F_k) : \forall k \in \{1, ..., 5\}$ 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 componetne 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 Γ com D'. Em outras palavras, nós predizemos o valor da entrada de teste $q = (x_1, x_2, x_3)$ treinando

Γ em D' e assim predizendo a entrada formada pelas predições 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.

1.0.1.2 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 predições feitas pelos classificadores do nível base. Essa informações adicionais permitem que não seja usado somente a predição, mas também o 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 apliacado a um exemplo x é a DP: $p^C(x) = (p^C(c_1|x), ..., p^C(c_m|x))$, onde $\{c_1, ..., c_m\}$ 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 probalidade será classe predita por C. Dessa forma, os atributos do meta-nível serão as probabilidade preditas 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.

1.0.1.3 Stacking com DP, Entropia e probabilidade máxima

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

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

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

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

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.

Esse é uma ideia para aplicarmos futuramente. Nesse momento continuarei utilizanto somente a DP.

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

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

$$l_i(w, x_i) = \max(0, 1 - y_i w^T x_i)) \tag{1.1}$$

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 adionado um termo de regularição a formula, o regularizador na formulação padrão é mensurado pela normado vetor de pesos $\mid w \mid^2$. Tirando-se a média da perda de todo os exemplos de treino e adicionando-se a ela o regularizador ponderado pelo paramêtro λ produz uma função objetiva de perda regularizada:

$$E(w) = \lambda ||w||^2 + \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i f(w, x) \tag{1.2}$$

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.

1.1.1 Adicionando viés

É comum adicionar a funcão 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 Descente(SGC)** e **Stochastic Dual Coordinate Ascent (SDCA)** não podem (ambos usados pelo LIBLINEAR). Nesse caso, uma simples solução é adcionar 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 $b = Bw_b$:

$$\bar{w}^T \bar{x_i} = w^T x_i + B w_b \tag{1.3}$$

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 $||w||^2 >> w_b^2$ e o efeito de encolimento pode ser ignorado. Infelizmente, fazer B muito grande faz o problema numericamente desbalanciado, assim uma troca justa entre encolhimento 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

1.2 Esquema de ponderação TF-IDF

Um dos mais populares esquemas de ponderação de termos em recuperação de informação é baseado na combinação da frequência do termo (TF) e o fator IDF.

$$w_{i,j} = \begin{cases} tf \times idf & , f_{i,j} > 0 \\ 0 & , f_{i,j} = 0 \end{cases} \tag{1.4}$$

Há várias variações dos fatores TF e IDF, tais variações são mais adequadas que outras para certos algoritmo de classificação.

1.2.1 Variações de TF-IDF

A table 1.1 mostra cinco variações do fator TF. O esquema binário atribui 1 ao TF se o termo ocorre no documento, e 0 caso contrátio. A frequência pura é o uso da contagem do número de vezes que o termo ocorre no documento. A normalização log diminui a impacto do crescimento do frequência $f_{i,j}$. A normalização 0.5 introduz dois efeitos: 1) ele normaliza o peso pela frequência máxima no documento and 2) normaliza o peso mantendo-o entre 0.5 e 1. A normalização K é simplesmente uma generalização da anterior.

Esquema	TF
binário	0,1
frequência pura	$f_{i,j}$
normalização log	$1 + \log f_{i,j}$
normalização 0.5	$0.5 + 0.5\frac{f_{i,j}}{max_i f_{i,j}}$
normalização K	$k + (1-K)\frac{f_{i,j}}{max_i f_{i,j}}$

Tabela 1.1: Variações do TF

A tabela 1.2 mostra três variações do fator TF. O esquema unário fixa o valor do IDF como 1 (IDF é ignorado). A frequência inversa é a formulação padrão para IDF. A frequÊncia inversa suave soma 1 ao denominador e numerador para evitar comportamento inesperado quando n_i atingir valores extremos.

Esquema	IDF
unária	1
frequência inversa	$\log \frac{N}{n_i}$
frequência inversa suave	$\log \frac{N+1}{n_i+1}$

Tabela 1.2: Variações do IDF

As combinações das variações de TF e IDF produzem vários esquemas TF-IDF. Nesse trabalho focaremos apenas no esquema $(1 + \log f_{i,j}) \times \log \frac{N+1}{n_i+1}$.

1.3 Normalização dos documentos

Há várias formas de normalizar documentos representados no espaço vetorial como **bag-of-words**. Seja w_j um vetor de pesos, criado a partir de algum dos esquemas de ponderação mencionado acima, que representa um documento d_j no espaço vetorial. Temos as seguintes formas normalizações utilizadas nesse trabalho:

Normalização	fórmula
None	w_j
Max	$\frac{w_j}{\max_i w_{i,j}}$
L1	$\frac{w_j}{\|w_j\|_1}$
L2	$\frac{w_j}{\|w_j\|_2}$

Tabela 1.3: Normalizações

1.4 Efeitos dos esquemas de ponderação e normalizacao em classificação de texto

Nos experimentos subsequentes, é feito um estudo sobre os efeitos dessas normalizações e dos esquemas de ponderação quando aplicados a classificadores vetorias tais como Support Vector Machine (SVM) e K Nearest Neighbors (KNN).

1.4.1 SVM

% latex table generated in R 3.2.4 by xtable 1.8-0 package % Thu Apr 28 10:22:27 2016

V1	V2	20NG	4UNI	ACM	REUTERS90
L2	microF1	**90.06 ± 0.43**	**83.48 ± 1.08**	**75.4 ± 0.66**	**68.19 ± 1.15**
	macroF1	**89.93 ± 0.43**	**73.39 ± 2.17**	**63.84 ± 0.55**	**31.95 ± 2.59**
L1	microF1	**89.8 ± 0.4**	78.23 ± 1.49	**75.31 ± 0.74**	**68.25 ± 1.2**
	macroF1	**89.59 ± 0.43**	67.47 ± 3.01	**62.33 ± 1.76**	**31.37 ± 2.22**
MAX	microF1	88.35 ± 0.37	81.36 ± 1.01	73.82 ± 0.78	**67.6 ± 1.1**
	macroF1	88.3 ± 0.38	68.01 ± 2.39	**62.55 ± 1.53**	**31.73 ± 3.13**
NONE	microF1	83.47 ± 0.46	80.55 ± 0.72	71.34 ± 1.01	66.6 ± 1.06
	macroF1	83.37 ± 0.42	**71.04 ± 2.06**	61.08 ± 0.67	**31.68 ± 3.32**

Tabela 1.4: Comparação entre as normalizações aplicada ao calssificador SVM

Como pode-se observar, a normalização tem um papel fundamental no desempenho do SVM. A normalização $L2$ teve o melhor desempenho em todos os conjuntos de dados testados.

1.4.2 KNN

% latex table generated in R 3.2.4 by xtable 1.8-0 package % Thu Apr 28 10:22:36 2016

V1	V2	20NG	4UNI	ACM	REUTERS90
KNN-L2	microF1	**87.39 ± 0.68**	**75.51 ± 1.25**	**70.67 ± 1.1**	**69.39 ± 1.46**
	macroF1	**87.1 ± 0.66**	**60.15 ± 1.26**	**55.39 ± 0.96**	**34.23 ± 2.75**
KNN-NONE	microF1	**87.45 ± 0.67**	**75.73 ± 1.2**	**71.06 ± 1.03**	**68.13 ± 1.01**
	macroF1	**87.15 ± 0.65**	**60.02 ± 0.8**	**55.82 ± 0.92**	**31.53 ± 3.42**
KNN-L1	microF1	**87.46 ± 0.69**	**75.74 ± 0.86**	**71.02 ± 1.08**	67.8 ± 1.02
	macroF1	**87.16 ± 0.66**	**60.29 ± 0.55**	**55.83 ± 1.15**	**30.91 ± 2.7**
KNN-MAX	microF1	**87.53 ± 0.69**	**75.63 ± 0.94**	**70.99 ± 0.96**	**68.07 ± 1.07**
	macroF1	**87.22 ± 0.66**	**60.34 ± 1.36**	**55.85 ± 0.97**	29.93 ± 2.48

Tabela 1.5: Comparação entre as normalizações aplicada ao calssificador KNN

Como pode-se observar, para o KNN a normalização teve pouco efeito sobre a efetividade do classificador. A normalização $L2$ empata estatisticamente com o esquema de ponderação sem normalização(None), que praticamente empatou com as outras normalizações.

Capítulo 2

Avaliação experimental

2.1 Caso de Estudo: Classificação Automática de Texto

2.1.1 Resultados e discussões

A Tabela 2.1 sumariza os resultados da avaliação empirica de algums algoritmos estado-da-arte para classificação de texto e os algoritmos propostos baseado na **Extremelly Randomized Tree**(doravante Extra-trees).

Primeiro aspecto que pode ser observado é que as *Extra-Trees* por si só melhoram a eficácia da *Random Forest* em dois conjuntos de dados(empatando nos outros dois). Esse resultado comprova que *Extra-Trees* também são mais robustas a ruídos e atributos irrelevante como mostrado em [Geurts et al., 2006], além disso, mostra que isso se mantém quando aplicadas a tarefas de classificação de texto. Todavia, está longe de figurar o cojunto dos melhores classificadores dentre os algoritmos analizados. O SVM continua sendo o melhor classificador, sendo o melhor classificador em todos os 4 conjuntos de dados. Isso não é supresa, já que o SVM é bom para aprender quando aplicado a dados de alta dimensionalidade e com natural robustês para atributos ruidosos e irrelevantes.

Logo em seguida vem *BERT (Boosted Extremelly Randomized Trees)*, o algoritmo é uma extensão do *BROOF* proposto em [Salles et al., 2015b]. Os resultados obtidos com a abordagem comprova nossa intuição de que a simples substituição da RF pela Extra-Trees traria ganhos expressivos no poder de generalização do *BROOF*, tornando a técnica ainda mais competitiva se comparada a outros classificadores.

Outra proposta foi a substituíção da RF pela *Extra-Trees* no algorimo *LazyNN_RF*, na esperação de ganho no poder de generalização do mesmo. É nítido que o uso das Extra-Trees no algoritmo LXT proporcionou um ganho sobretudo nos conjuntos de dados 20NG e REUTERS90, quando comparado ao algoritmo Lazy.

Esses resultados reforçam que as Extra-Trees são mais robustas a ruídos que as RF, todavia estão aquém se comparadas a capacidade do SVM de lidar com dados de alta dimensionalidade e ruidosos. Porém, com uso das técnicas apresentadas em [Salles et al., 2015a] pode-se aumentar a capacidade desses algoritmos de lidar com dados ruidosos tarnando-os assim mais competitivos ou muitas vezes melhores que a os algoritmos a presentados na Tabela 2.1.

% latex table generated in R 3.2.4 by xtable 1.8-0 package % Thu Apr 28 17:53:01 2016

2.1.1.1 Stacking

Nessa seção contrastamos o aumento do poder de generalização ao combinarmos técnicas altamente eficázes, tais como LazyNN_RF, BROOF, BERT e LazyExtraTrees(LXT), com o ganho proporcionado pela combinação de alguns classificadores estado-da-arte para classificação automática de texto.

V1	V2	20NG	4UNI	ACM	REUTERS90
SVM	microF1	**90.06 ± 0.43**	**83.48 ± 1.08**	**75.4 ± 0.66**	**68.19 ± 1.15**
	macroF1	**89.93 ± 0.43**	**73.39 ± 2.17**	**63.84 ± 0.55**	**31.95 ± 2.59**
BERT	microF1	88.93 ± 0.39	**84.61 ± 0.98**	**74.8 ± 0.59**	**67.33 ± 0.72**
	macroF1	88.59 ± 0.5	**73.61 ± 1.85**	**62.1 ± 0.99**	**29.24 ± 1.4**
BROOF	microF1	87.96 ± 0.24	**84.41 ± 1.07**	73.35 ± 0.79	**66.79 ± 0.97**
	macroF1	87.44 ± 0.28	**73.23 ± 1.1**	60.76 ± 0.8	**28.48 ± 2.17**
LAZY	microF1	87.96 ± 0.37	**82.34 ± 0.61**	**74.02 ± 0.79**	**66.3 ± 1.07**
	macroF1	87.39 ± 0.37	68.33 ± 1.6	59.46 ± 1.35	26.61 ± 2.12
KNN	microF1	87.53 ± 0.69	75.63 ± 0.94	70.99 ± 0.96	**68.07 ± 1.07**
	macroF1	87.22 ± 0.66	60.34 ± 1.36	55.85 ± 0.97	**29.93 ± 2.48**
NB	microF1	88.99 ± 0.54	62.63 ± 1.7	73.54 ± 0.71	65.32 ± 1.13
	macroF1	88.68 ± 0.55	51.38 ± 3.19	58.03 ± 0.85	**27.86 ± 0.79**
XT	microF1	85.94 ± 0.23	81.66 ± 1.03	71.94 ± 0.66	64.33 ± 0.86
	macroF1	85.57 ± 0.22	65.44 ± 2.41	57.4 ± 1.13	24.47 ± 2.22
LXT	microF1	88.39 ± 0.51	81.24 ± 0.71	69.63 ± 0.91	65.92 ± 0.82
	macroF1	88.05 ± 0.44	66.89 ± 1.23	57.33 ± 1.48	26.71 ± 2.53
RF	microF1	83.64 ± 0.29	81.52 ± 1	71.05 ± 0.31	63.92 ± 0.81
	macroF1	83.08 ± 0.35	65.44 ± 1.91	56.56 ± 0.45	24.36 ± 1.98

Tabela 2.1: Comparação entres métodos de base

Tabela 2.2: Legenda para os stacking. Os classificadores reportados na descrição são todos do nível base, para todos os stackings foi utilizado RF como classificador do meta-nível.

Stacking	Descrição
COMB1	BROOF + Lazy
COMB2	BERT + LXT
COMB3	BROOF + Lazy + BERT + LXT
COMBSOTA	SVM + KNN + XT + RF + NB
COMBALL	Stacking de todos

2) temos um a abordagem combinando stacking, bagging (arvores tradicionais e LAZY) e boosting (BROOF e BERT)
3) os resulatdos de stacking so são bons qdo os novos métodos de florestas entram (COMBSOTA não eh tao bom)
4) ha' um ganho em combinar tudo mas 5) nao precisa combinar tudo praobter bons resuatdos, o stacking de florestas jah eh competitivo

% latex table generated in R 3.2.4 by xtable 1.8-0 package % Thu Apr 28 18:21:38 2016

V1	V2	20NG	4UNI	ACM	REUTERS90
COMBALL	microF1	**91.67 ± 0.44**	**86.74 ± 1.17**	**78.46 ± 0.72**	**80.02 ± 1.24**
	macroF1	**91.43 ± 0.42**	**79.45 ± 2.23**	**63.72 ± 1.01**	**37.84 ± 3.14**
COMB3	microF1	90.63 ± 0.57	**86.79 ± 0.86**	**77.34 ± 0.6**	**79 ± 1.14**
	macroF1	90.4 ± 0.57	**79.63 ± 1.91**	**62.91 ± 0.92**	**33.93 ± 2.97**
COMB2	microF1	90.2 ± 0.51	**86.54 ± 1.06**	76.88 ± 0.55	**78.25 ± 1.17**
	macroF1	89.95 ± 0.52	**79.41 ± 1.63**	**62.66 ± 0.81**	**32.86 ± 2.23**
COMBSOTA	microF1	90.65 ± 0.45	**84.95 ± 1.15**	**77.78 ± 0.73**	74.63 ± 1
	macroF1	90.42 ± 0.44	**75.96 ± 1.78**	**63.04 ± 0.85**	27.66 ± 0.88
COMB1	microF1	89.32 ± 0.42	**86.52 ± 1.18**	76.74 ± 0.73	77.22 ± 1.14
	macroF1	89.01 ± 0.44	**78.66 ± 1.9**	**62.2 ± 1.01**	31.71 ± 2.7
BERT	microF1	88.93 ± 0.39	**84.61 ± 0.98**	74.8 ± 0.59	67.33 ± 0.72
	macroF1	88.59 ± 0.5	73.61 ± 1.85	**62.1 ± 0.99**	29.24 ± 1.4
SVM-L2	microF1	90.06 ± 0.43	83.48 ± 1.08	75.4 ± 0.66	68.19 ± 1.15
	macroF1	89.93 ± 0.43	73.39 ± 2.17	**63.84 ± 0.55**	31.95 ± 2.59

Tabela 2.3: Comparação entre os métodos de stackings

Bibliografia

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. ISSN 0885-6125. doi: 10.1007/s10994-006-6226-1. URL http://dx.doi.org/10.1007/s10994-006-6226-1.

Thiago Salles, Marcos Gonçalves, and Leonardo Rocha. Random forest based classifiers for classification tasks with noisy data. 2015a.

Thiago Salles, Marcos Gonçalves, Victor Rodrigues, and Leonardo Rocha. Broof: Exploiting out-of-bag errors, boosting and random forests for effective automated classification. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 353–362, New York, NY, USA, 2015b. ACM. ISBN 978-1-4503-3621-5. doi: 10.1145/2766462.2767747. URL http://doi.acm.org/10.1145/2766462.2767747.