Stacking utilizado nos experimentos

Raphael Rodrigues Campos

26 abril, 2016

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.

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.

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.

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.

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.

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**:

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

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:

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

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 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 $b = Bw_b$:

$$\bar{w}^T \bar{x_i} = w^T x_i + Bw_b \tag{3}$$

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

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.

$$w_{i,j} = \begin{cases} tf \times idf & , f_{i,j} > 0 \\ 0 & , f_{i,j} = 0 \end{cases} \tag{4}$$

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

Variações de TF-IDF

A table 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	$k + (1-K)\frac{f_{i,j}}{max_i f_{i,j}}$

Table 1: Variações do TF

A tabela 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	$\log \frac{N+1}{n_i+1}$

Table 2: Variações do IDF

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

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	$\frac{w_j}{\|w_j\|_2}$

Table 3: Normalizações

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).

SVM

% latex table generated in R 3.2.4 by xtable 1.8-0 package % Tue Apr 26 21:09:39 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**

Table 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.

KNN

% latex table generated in R 3.2.4 by xtable 1.8-0 package % Tue Apr 26 21:09:47 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

Table 5: Comparação entre as normalizações aplicada ao calssificador KNN