Progetto Finale di Statistica Computazionale

Davide Torlo & Luca Venturi

Università degli Studi di Trieste
Dipartimento di Matematica e Geoscienze

21 Gennaio 2016

Professore
Prof. Luca Bortolussi

Sintesi

1. Task 1 - Regressione
2. Task 2 - Classificazione

Task 1 - Regressione

- Dataset composto da 99 parametri di 1994 città americane.
- La funzione dataci è il tasso di criminalità nelle singole città.
- Oltre ai 99 parametri, un centinaio di città avevano altri 20 parametri.
- Obiettivo: creare un modello di regressione che preveda il tasso di criminalità in altre città.

Regressione - Preprocessing

- Abbiamo usato i processi Gaussiani per costruire un modello di regressione.
- Per preprocessare i dati abbiamo provato a rinormalizzarli (erano forniti tutti tra 0 e 1) su un compatto di $\mathbb{R}$ centrato in 0, ma senza ottenere grandi risultati.
- Fondamentale è stata la riduzione di dimensione. Usando la PCA siamo passati a dimensione dalle 2 alle 30 a seconda degli algoritmi usati.

Regressione - Homemade GP Regression

- Abbiamo usato l'algoritmo dei processi Gaussiani visto a lezione.
- Il primo problema trovato è stato quello di invertire la matrice di Gram K con numeri di dati troppo elevati.
- Quindi abbiamo diviso il training set, in diversi training set da circa 100/200 dati ciascuno.
- Per ognuno di questi avevamo parametri da ottimizzare.

- Per ogni training set usato abbiamo usato dei validation set per ottimizzare i parametri di lengthscale, amplitude e noise che usa il nostro metodo.
- Per far ciò abbiamo usato la function `fmincon` di Matlab su una funzione che prende in ingresso i parametri e restituisce l'errore quadratico medio dei punti del validation set dalla regressione stimata.
- Con i parametri ottenuti abbiamo fatto la regressione su nuovi punti di test e infine mediato sulle diverse soluzione ottenute dai diversi training set.

Regressione - Homemade GP Regression

- Questo metodo porta con se un po' di problemi.
- Primo di tutti ottiene dei risultati non molto ottimali, in quanto il massimo che siamo riusciti a raggiungere è stato una media quadratica dell'errore di 0.6532 (con $t \in [0, 1]$) e con varianza molto elevata (varianza media attorno a 1.)
- Inoltre i parametri ottimali trovati da Matlab sono davvero molto sensibili ai dati iniziali e al training set.
- Ciò ci fa supporre che la funzione abbia tanti minimi locali e che sia facile non trovare il minimo assoluto.

Regressione - Matlab GP Regression

- Abbiamo provato ad usare la funzione di Matlab che implementa la regressione con processi Gaussiani.
- L'algoritmo ricerca in maniera automatica i parametri ottimali per il modello.
- L'abbiamo testato con differenti funzioni di kernel e per diverse dimensioni della PCA.

- Abbiamo deciso di non usare tutti il training set ma di suddividerlo in più training set, calcolare la predizione con ognuno di questi e poi prenderne una media.

- Dato un dato del test set X_{test}, il modello fornisce la distribuzione di probabilità del relativo t_{test}. Essa è una distribuzione Gaussiana di media e varianza:

$$\mu(i, t_{test}), \qquad \sigma^2(i, t_{test})$$

dove l'indice i indica che sono state calcolate usando l'i-esimo training set.

Regressione - Matlab GP Regression

- Infine si considera la media di queste predizioni, andando a descrivere la distribuzione di t_{test} come una Gaussiana di parametri:

$$\mu(t_{test}) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \mu(i, t_{test}),$$

$$\sigma^2(t_{test}) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \sigma^2(i, t_{test}).$$

Regressione - Matlab GP Regression

- Alcuni dati ottenuti:

$m = 8$	0.4213	Gaussian
$m = 8$	0.4138	GaussianARD
$m = 8$	0.4190	Matern52
$m = 15$	0.4155	Matern52
$m = 17$	0.4119	Matern52ARD

- Inoltre la percentuale di dati di test che stanno in un'intervallo di confidenza del 97% è sempre circa del 95%.

Regressione - Osservazioni finali

- Una considerazione importante riguarda le dimensioni.
- Infatti passando da PCA con $d = 2$ a $d = 30$ non si nota quasi alcuna differenza nell'errore quadratico medio (meno dell'1%).
- Perciò supponiamo che già a dimensione 2 si possa avere un'idea di come questo set di dati sia fatto.
- Plottandolo e disegnando la nostra funzione di regressione si nota che la nuvola di punti circonda in tutte le direzioni la superficie.
- Plot Homemade GP
- Plot Matlab GP

Task 2 - Classificazione

- Dataset composto da immagini 20x20 pixel.
- Ogni immagine rappresenta una cifra da 0 a 9 disegnata a mano.
- Dataset composto da 60000 immagini di training e 10000 di test.
- Le immagini sono fornite già centrate e ritagliate.
- Obiettivo: creare un modello che riesca a classificare questo tipo di immagini.

Classificazione - Preprocessing

- Immagine $\rightarrow$ vettore 400-dimensionale.
- Lavorare con queste dimensioni è costoso computazionalmente.
- Abbiamo utilizzato l'algoritmo di PCA per ridurre le dimensioni.

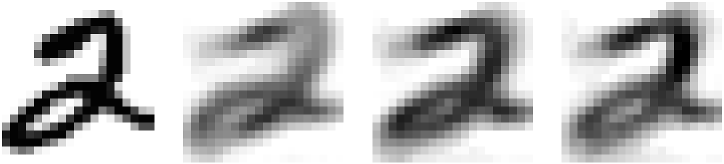

Figura: Immagine originale e proiettata in dimensione 2, 6 e 10.

Classificazione - Idea 1

- $X_{train} = [X^0_{train}, \dots, X^9_{train}] \quad \rightarrow \quad$ PCA
- $[U_{red}(0), \dots, U_{red}(9)], \quad [\lambda^0, \dots, \lambda^9]$
- $X_{test} \quad \rightarrow \quad [a(0, X_{test}), \dots, a(9, X_{test})]$
- Idea: se $t_{test} = i \quad \rightarrow \quad a(i, X_{test}) > a(j, X_{test}) \quad \text{per } i \neq j$

Classificazione - Idea 1

- Per ogni test X_{test} valutiamo le funzioni

$$f_i(X_{test}) = \frac{\sum_{j=1}^{d} \lambda_j^i a(i, X_{test})_j^2}{\sum_{j=1}^{d} \lambda_k^i}$$

- Prendo

$$i = \arg\max_j f_j(X_{test})$$

come predizione del nostro modello.

Classificazione - Idea 1

- Training set molto grande $\rightarrow$ lo dividiamo in diversi training set più piccoli.
- Ognuno di questi fornisce una predizione sul test.
- Abbiamo implementato due metodi diversi che ottengono una predizione finale basata sulle varie predizioni precedenti.

Classificazione - Idea 1 - Davide

- La convinzione su cui si basa questo metodo è che non tutti i training set siano ugualmente buoni.

- Per validare questi set ho preso diversi validation set e di volta in volta ho scartato i training set che si comportavano peggio.

$$\text{Se} \quad \frac{\text{cifre azzeccate dal train } j}{\text{cifre totali}} < \text{tol} \quad \rightarrow \quad \text{scarto il training set } j$$

- La tol l'ho fissata usando un altro validation set a 0.56

Classificazione - Idea 1 - Davide

- Con i training set rimasti ho usato come predizione finale la moda dei vari train.
- Così facendo ho ottenuto il valore ottimo di $80,05\%$ di cifre riconosciute, usando come dimensione della PCA 8 e 24 training set validati partendo da 100 totali, ciascuno di 1000 elementi.

```
0.8005

1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000         0
0.9903    0.7432    0.8228    0.3615    0.7803    0.9791    0.9144    0.8614    0.8394    0.6816
```

Classificazione - Idea 1 - Davide

- Per rendere il risultato confrontabile con altri (quello di Luca), ho dovuto creare una funzione probabilità che assegni ad ogni test una probabilità di essere assegnata alla cifra k.

- Perciò, invece di prendere la moda dei train, ho fatto le medie delle funzioni f_i dei vari train selezionati:

$$p(\text{classe } j|x) = \frac{\sum_{i=1}^{N_{train}} f_j^i(x)}{\sum_{i=1}^{N_{train}} \sum_{k=1}^{10} f_k^i(x)}.$$

Classificazione - Idea 1 - Luca

- L'idea è fare una media delle predizioni fornite dai vari training set.
- Per ogni train j calcolo la probabilità che X_{test} sia assegnato alla cifra i come:

$$P(X_{test} = i|\text{train } j) = \frac{f_i^j(X_{test})}{\sum_{k=0}^{9} f_k^j(X_{test})}.$$

- La predizione finale che ottengo è una probabilità che X_{test} sia assegnato alla cifra i, ottenuta mediando le precedenti sui vari training set:

$$P(X_{test} = i) = \frac{1}{n_{train}} \sum_{j=1}^{n_{train}} P(X_{test} = i|\text{train } j).$$

Classificazione - Idea 1 - Luca

- Altra idea: implementare un metodo che per ogni cifra non utilizzi i train set che funzionano male per imparare tale cifra.

- Si calcola, usando diversi validation set, il seguente indice:

$$\mathrm{ind}(i,j) = \begin{cases} 1 & \text{se perc}(i|\text{train } j) > \text{tol per ogni validation set} \\ 0 & \text{altrimenti} \end{cases}$$

  dove tol è una tolleranza fissata e $\mathrm{perc}(i|\mathrm{train}\ j)$ é la percentuale di cifre i azzeccate usate il j-esimo train.

- Definisco quindi $\mathrm{pos}(i,j) = \inf\{k \geq 1 : \sum_{l=1}^{k} \mathrm{ind}(i,l) = j\}$.

Classificazione - Idea 1 - Luca

- Definisco, pero ogni train j e cifra i:

$$l_i^j(X_{test}) = \frac{f_i^{\text{pos}(i,j)}(X_{test})}{\sum_k f_k^{\text{pos}(i,j)}(X_{test})}.$$

- Si calcola quindi infine la probabilità che a X_{test} sia associata la cifra i come:

$$P(X_{test} = i) = \frac{\sum_j l_i^j(X_{test})}{\sum_{k,j} l_k^j(X_{test})}.$$

Classificazione - Idea 1 - Luca

- In realtà, testando i due metodi, si trova che funziona meglio il primo, ovvero quello che usa tutti i train, senza scartarli.
- Con tale metodo si ottengono sui dati di test le seguenti percentuali di cifre azzeccate:

```
>> Task_2_test

perc =

    0.8022


ans =

    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000         0
    0.9938    0.7936    0.8059    0.2515    0.8128    0.8935    0.8959    0.7926    0.8057    0.9459
```

Classificazione - Idea 1 - Parametro 'Peso'

- Da entrambi i metodi risulta che alcuni numeri sono spesso misclassificati, ad esempio il 4 era la cifra peggio riconosciuta.
- Idea: pesare le f_i con dei coefficienti α_i in modo da favorire le cifre peggiori:
$$f_i^{new}(X_{test}) = \alpha_i \cdot f_i(X_{test}).$$
- Come scelgo α?

Classificazione - Idea 1 - Davide: Parametri Ottimi

- $\alpha = [0.88, 1.08, 1.05, 1.23, 1.13, 0.98, 1, 1.09, 1.05, 1.21]$.
- Percentuali di cifre azzeccate ottenute:

```
0.8574

1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000         0
0.9683    0.8014    0.7673    0.8106    0.8240    0.9144    0.8852    0.8840    0.7909    0.9153
```

Classificazione - Idea 1 - Luca: Parametri Ottimi

- $\alpha = [1, 1.07, 1.05, 1.25, 1, 1, 1, 1.05, 1.09, 1]$.
- Percentuali di cifre azzeccate ottenute:

```
perc =

    0.8646


ans =

    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000         0
    0.9938    0.8450    0.8386    0.7943    0.7915    0.8894    0.8599    0.8337    0.8375    0.9388
```

Classificazione - Idea 1 - Osservazioni

- Mediando le probabilità ottenute dai due metodi si ottiene una percentuale di cifre azzeccate dell'$87,37\%$.
- Abbiamo provato ad usare la funziona matlab `fmincon` per cercare gli α migliori, ma non funzionava.
- Aumentando la dimensione del PCA oltre un certo valore, il risultato peggiora (troppo rumore?).

Classificazione - Logistic Multiclass Regression

- L'approcio proposto finora può essere leggermente modificato per trasformarlo in un algoritmo di Logistic Multiclass Regression.
- Definiamo le funzioni $\Phi_i(\mathbf{x}) = f_i(\mathbf{x})$.
- Tale algoritmo calcola quindi le probabilità

$$P(C_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

dove $a_k = \mathbf{w}_k^T \cdot \Phi(\mathbf{x})$.

- Nel nostro approcio abbiamo utlizzato $\mathbf{w}_k = \alpha_k \mathbf{e}_k$.

Classificazione - Stochastic Gradient Descent

- Si possono calcolare i $\mathbf{w}_k$ migliori andando a minimizzare la funzione:

$$E(\mathbf{w}_0, \dots, \mathbf{w}_9) = -\sum_{n=1}^{N} \sum_{k=0}^{9} t_{nk} \log(y_{nk})$$

dove N è il numero di dati di training.

- Poichè la funzione da minimizzare è del tipo

$$f(\mathbf{x}) = \sum_{n=1}^{N} f_n(\mathbf{x})$$

si potrebbe utilizzare un algoritmo di stochastic gradient descent per la ricerca dei minimi dividendo i train (e quindi la somma su n) in vari 'blocchetti'.

Classificazione - Idea 2

- Usare SVM per distinguere 2 cifre.
- Usare strategia tutti contro tutti o con scontri tra gruppi di cifre.
- Usare la PCA su tutti i dati di training con dimensioni maggiori.

Classificazione - Idea 2 - SVM: 1 vs 1

- Le percentuali di cifre ben classificate nell'1 vs 1 sono le seguenti:

	1	2	3	4	5	6	7	8	9	10
1	1	0.9940	0.9944	0.9986	0.9931	0.9938	0.9898	0.9872	0.9953	0.9976
2	0.9940	1	0.9736	0.9772	0.9730	0.9729	0.9714	0.9706	0.9824	0.9891
3	0.9944	0.9736	1	0.9960	0.9416	0.9914	0.9784	0.9647	0.9792	0.9945
4	0.9986	0.9772	0.9960	1	0.9867	0.9881	0.9846	0.9898	0.9518	0.9969
5	0.9931	0.9730	0.9416	0.9867	1	0.9697	0.9911	0.9502	0.9763	0.9861
6	0.9938	0.9729	0.9914	0.9881	0.9697	1	0.9889	0.9871	0.9939	0.9871
7	0.9898	0.9714	0.9784	0.9846	0.9911	0.9889	1	0.9835	0.9475	0.9950
8	0.9872	0.9706	0.9647	0.9898	0.9502	0.9871	0.9835	1	0.9642	0.9918
9	0.9953	0.9824	0.9792	0.9518	0.9763	0.9939	0.9475	0.9642	1	0.9945
10	0.9976	0.9891	0.9945	0.9969	0.9861	0.9876	0.9950	0.9918	0.9945	1

Classificazione - Idea 2 - SVM: Torneo

- Abbiamo diviso le cifre in gruppi (tenendo conto delle misclassificazioni) e abbiamo organizzato un torneo tra gruppi di cifre per ogni test.

- 1° Round: $[0, 1, 2, 6]$ vs $[3, 4, 5, 7, 8, 9]$
- 2° Round: $[0, 6]$ vs $[1, 2]$ e $[3, 8, 5]$ vs $[4, 7, 9]$
- 3° Round: $[0]$ vs $[6]$ e $[1]$ vs $[2]$ e $[3, 5]$ vs $[8]$ e $[4]$ vs $[7, 9]$
- 4° Round: $[3]$ vs $[5]$ e $[7]$ vs $[9]$

- Il risultato migliore ottenuto fornisce $85, 63\%$ cifre azzeccate, con dimensione del PCA 45.

Classificazione - Idea 2 - SVM: Tutti contro tutti

- Un'altra strategia provata è quella di confrontare per ogni test, ciascuna cifra contro l'altra.
- Per ogni test, si sceglie quindi la cifra che ha vinto più partite contro le altre 9.
- I risultati ottenuti con questa strategia arrivano a riconoscere il $94,04\%$ di cifre, con dimensione del PCA 60.
- Aumentando la dimensione i risultati migliorano leggermente ma i costi computazionali si alzano.

Classificazione - Idea 2 - SVM: Tempi

- Interessante notare che i tempi impiegati per fare SVM durante gli scontri tra le diverse coppie di cifre sono molto diversi: passano da un minimo di circa 1 secondo a un massimo di 25 secondi.

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	1.3989	0	0	0	0	0	0	0	0
3	5.8373	3.1187	0	0	0	0	0	0	0
4	3.7179	3.3769	25.6071	0	0	0	0	0	0
5	2.3860	0.9962	9.6599	5.0136	0	0	0	0	0
6	5.6139	1.9948	14.8517	9.9253	9.8443	0	0	0	0
7	5.4302	1.3077	7.7770	2.5489	4.0584	9.3773	0	0	0
8	1.6653	1.4744	6.5209	7.0928	3.8018	3.5742	1.0250	0	0
9	2.9108	7.5775	13.1604	14.9318	2.9870	15.8238	3.8334	7.5580	0
10	3.3430	1.5383	5.7947	6.8741	11.7840	6.0686	1.3373	12.9496	12.6500

Classificazione - Osservazioni finali

- Le cifre di training e test sono brutte e di bassa qualità.
- Se avessimo dei pixel in più probabilmente funzionerebbe meglio (in rete si trovano soprattutto esempi di 28×28 pixel).
- Per implementare un software di riconoscimento cifra, bisogna anche aggiungere un preprocessing che centri l'immagine e la ingrandisca sui pixel interessanti.
- Google inoltre usa l'informazione della direzione in cui si sta scrivendo.

Classificazione - Testiamo i metodi!

- Draw!