Logistic Regression

Seminario Statistica Superiore

Simone Manti

Dipartimento di Matematica
Università di Pisa

22 Febbraio 2022

Piano della presentazione

Cos'è la regressione logistica

Per chiarezza espositiva fissiamo alcune notazioni:

1. $(X, y) = \{(x_i, y_i)\}_{i \in \{1,\dots,N\}}$ dataset e supponiamo inizialmente $y_i \in \{0, 1\}$ per ogni i (binary classification).
2. Indicheremo un generico ogni iperpiano con la notazione $w^T x$, dove $w = [1, w_1, \dots, w_N]^T$ e $x = [b, x_1, \dots, x_N]^T$.

Parliamo di **logistic regression** quando $p(y|x, w) = \text{Ber}(\sigma(w^T x))$, dove $\sigma(x) := \frac{1}{1+e^{-x}}$ è la **funzione logistica** e w è un iperparametro.
Cerchiamo l'iperparametro w che minimizza la **negative log-likelihood**

$$\text{NLL}(w) := -\frac{1}{N} \sum_{i=1}^{N} [y_n \log(\mu_n) + (1 - y_n) \log(1 - \mu_n)],$$

con $\mu_n = \sigma(w^T x_n)$.

Osservazioni importanti

- Si mostra facilmente che $\nabla NLL(w) = \frac{1}{N} X^T \text{diag}(\mu - y)$ e $\nabla^2 NLL(w) = \frac{1}{N} X^T S X$ con $S := \text{diag}([\mu_1(1-\mu_1), \ldots, \mu_N(1-\mu_N)])$. In particolare, dato che per ogni i si ha $0 < \mu_i < 1$ allora l'hessiana è definita positiva, ovvero NLL è convessa.
- È facile estendere tutto ai problemi di classificazione con più di 2 classi. Difatti, se $y_i \in \{1, \ldots, C\}$ si pone $p(y|x, \theta) = \mathcal{M}(S(Wx + b))$, dove $S(z) := [\frac{e^{z_1}}{\sum_{i=1}^{C} e^{z_i}}, \ldots, \frac{e^{z_C}}{\sum_{i=1}^{C} e^{z_i}}]$ è la funzione **softmax** e $\theta = (W, b)$. Anche in questo caso NLL è convessa.
- Per trattare problemi non linearmente separabili un'idea può essere definire (a priori, i.e. un altro iperparametro) $\phi : \mathbb{R}^N \to \mathbb{R}^K$ (con K > N) e considerare come dataset $\{(\phi(x_i), y_i)\}$

Gradient descent

Vogliamo

$$w = \arg\min_{w} \mathrm{NLL}(w)$$

e il più facile learning algorithm è il classico **gradient descent**: dopo aver inizializzato w_0 casualmente poniamo per ogni $t \geq 0$

$$w_{t+1} = w_t - \eta_t \nabla NLL(w_t),$$

dove $\eta_t > 0$ è il **learning rate**. Sorgono però alcuni problemi:

1. Come scegliere η_t affinchè l'algoritmo converga?
2. Quando N è molto grande (i.e. il dataset è molto grande), il calcolo del gradiente ad ogni passo potrebbe essere molto oneroso e occupare molta memoria. Come gestire questa situazione?

SGD e mini-batch

Il problema precedente è stato formulato **offline**, nel senso che cerchiamo un minimo di una funzione $f(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{\theta}, z_i)$, dove per ogni i $z_i := (x_i, y_i)$. Per risolvere il problema della dimensione del dataset si potrebbe pensare di aggiornare il valore degli iperparametri **per ogni** termine della somma, in altri termini il learning algorithm diventa (nel nostro caso)

$$w_{t+1} = w_t - \eta_t \nabla NLL(w_t, z_i).$$

L'algoritmo prende il nome di **stochastic gradient descent**.
Si potrebbe pensare di optare per una via di mezzo tra l'online learning e l'offline learning, vale a dire fissare un $B > 0$ naturale con $B \leq N$ ed aggiornare la somma su B addendi. Questa variante prende il nome di **mini-batch gradient descent**.
Per $B = 1$ otteniamo l'SGD, per $B = N$ si ottiene il classico gradient descent.

Pseudocodice di SGD

Algorithm Stochastic Gradient Descent

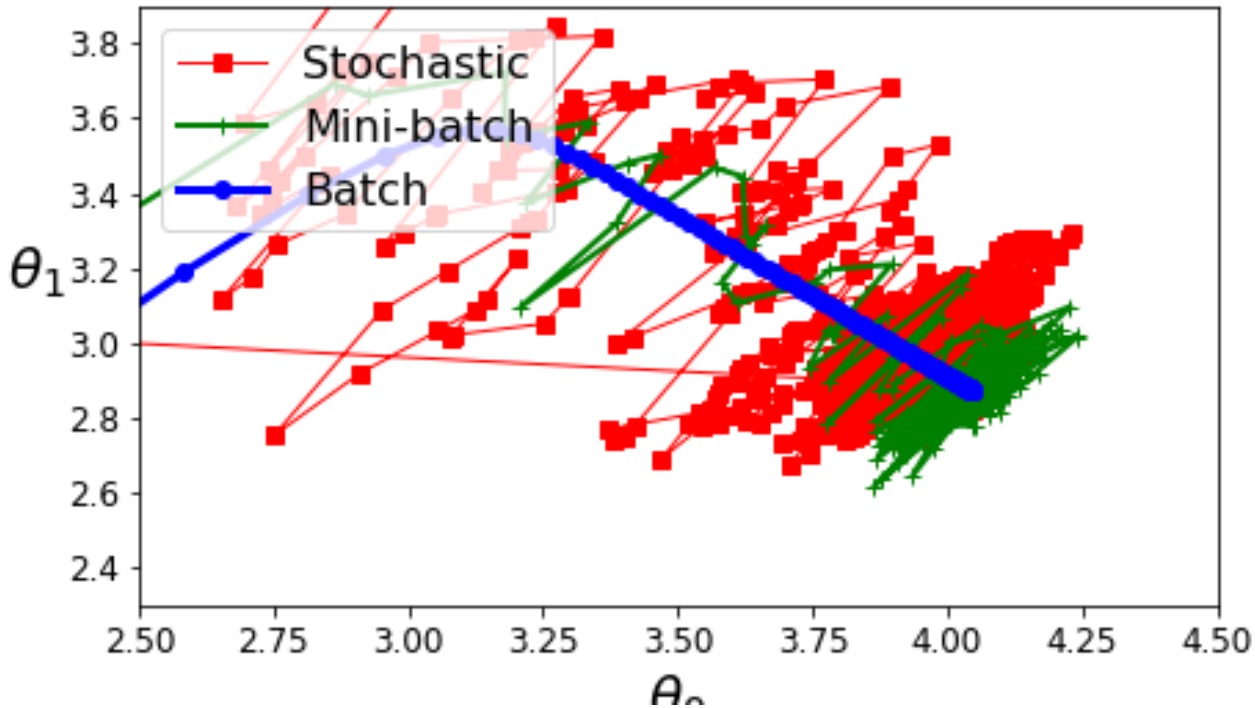
Input: Inizializzare $\boldsymbol{\theta}$ e η

1: **repeat**
2: **for** $i = 1 : N$ **do**
3: $\quad \boldsymbol{g} = \nabla f(\boldsymbol{\theta}, \boldsymbol{z}_i)$
4: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{g}$
5: $\quad$ Aggiornare η
6: **end for**
7: **until** convergence

Le **condizioni di Robbins-Monro**[1] ci assicurano che SGD converge se la successione dei learning rate $\{\eta_k\}_k$ soddisfa $\sum_{k=1}^{+\infty} \eta_k = \infty, \sum_{k=1}^{+\infty} \eta_k^2 < \infty$.

[1] Robbins, H.; Monro, S. (1951). "A Stochastic Approximation Method"

Gd vs Sgd vs Mini-batch



[1] Il codice che ho scritto è presente qui nella sezione "Confronto tra i learning algorithm"

Metodo di Newton e IRLS

Un altro possibile learning algorithm è il **metodo di Newton**

$$w_{k+1} = w_k - \eta_k H^{-1} \nabla NLL(w_k),$$

con $H = \nabla^2 NLL(w_k)$. Se sostituiamo i valori del gradiente e dell'hessiana trovati prima, otteniamo

$$w_{k+1} = (X^T S_k X)^{-1} X^T S_k z_k,$$

con $z_k := X w_k + S_k^{-1}(y - \mu_k)$.
Questo algoritmo è noto come **iteratively reweighted least squares** (o IRLS in breve).

Metodi Quasi-Newton (cenni)

Purtroppo può risultare molto oneroso il calcolo di H. I **metodi Quasi-Newton** costruiscono un'approssimazione dell'hessiana iterativamente. Il metodo più famoso è il **BFGS** (Broyden, Fletcher, Goldfarb e Shanno)

$$B_{k+1} = B_k + \frac{y_k^T y_k}{y_k^T s_k} - \frac{(B_k s_k)(B_k s_k)^T}{s_k^T B_k s_k}$$

$$s_k = w_k - w_{k-1}$$

$$y_k = \nabla NLL(w_k) - \nabla NLL(w_{k-1})$$

Di solito si pone $B_0 = I$: in questo caso BFGS può essere pensato come un'approssimazione dell'Hessiana tramite una matrice diagonale più una di rango piccolo.

Regolarizzazione ℓ_2

Spesso il fenomeno di overfitting è causato da un valore molto grande delle componenti di w. Questo può anche portare ad instabilità numerica.
Per far fronte a questo problema l'idea è semplificare il modello, dunque si cerca un vettore w con componenti non troppo elevate. In altri termini, cerchiamo

$$w = \arg\min_{w} \mathrm{NLL}(w) + \boldsymbol{\lambda}||\boldsymbol{w}||_2^2.$$

Osserviamo che λ è un ulteriore iperparametro e diverse varianti sono possibili, e.g. norma 1 anzichè norma 2 (per indurre più sparsità nei coefficienti) oppure entrambe insieme (chiamata **elastic net**).

Bayesian Logistic regression

- Finora ci siamo concentrati sulla stima degli iperparametri tramite la non negative log likelihood loss. Può tuttavia essere molto utile stimare $p(w|\mathcal{D})$, ad esempio nelle situazioni in cui vogliamo associare degli intervalli di confidenza alle nostre predizioni.
- Tuttavia, a differenza della regressione lineare non è possibile calcolare esattamente la probabilità a posteriori: serve approssimarla.
- In questo seminario vediamo solo una delle approssimazioni più famose di $p(w|\mathcal{D})$, chiamata **approssimazione di Laplace**.

Approssimazione di Laplace

Sia $\theta \in \mathbb{R}^D$, sappiamo che

$$p(\theta|\mathcal{D}) = \frac{1}{Z} e^{-E(\theta)},$$

dove $Z = p(\mathcal{D})$ e la **funzione energia** $E(\theta) = -log(p(\theta, \mathcal{D}))$.
Sviluppando al second'ordine con Taylor nella moda θ^* (i.e. nel punto di minimo di E) otteniamo

$$E(\theta) \approx E(\theta^*) + (\theta - \theta^*)^T g + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*),$$

dove g e H sono rispettivamente il gradiente e l'hessiana calcolati nella moda θ^*. Dato che θ^* è un punto di minimo si ha $g = 0$ e quindi

$$E(\theta) \approx E(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*),$$

Approssimazione di Laplace per la regressione logistica

Dunque otteniamo

$$p(\theta|\mathcal{D}) \approx \frac{1}{Z} e^{-E(\theta^*)} e^{-\frac{1}{2}(\theta-\theta^*)^T H(\theta-\theta^*)}, \tag{3.1}$$

che è a meno di costanti una $\mathcal{N}(\theta^*, H^{-1})$. In particolare, otteniamo

$$Z = p(\mathcal{D}) \approx e^{-E(\theta^*)} (2\pi)^{\frac{D}{2}} |H|^{-\frac{1}{2}}. \tag{3.2}$$

L'equazione (3.1) è l'approssimazione di Laplace della probabilità a posteriori, mentre la (3.2) è l'approssimazione di Laplace della probabilità marginale.
Per la regressione logistica, dai risultati di prima otteniamo (con le stesse notazioni) $p(w|\mathcal{D}) \approx \mathcal{N}(w^*, H^{-1})$

BIC (Bayesian Information Criterion)

Possiamo usare l'approssimazione di Laplace per scrivere la log likelihood marginale. Infatti, scartando costanti irrilevanti

$$\log(p(\mathcal{D})) \approx \log(p(\mathcal{D}|\theta^*)) + \log(p(\theta^*)) - \frac{1}{2}\log|H|.$$

Se abbiamo scelto una distribuzione a priori uniforme allora $p(\theta) \propto 1$ possiamo scartare il secondo termine della somma al RHS e sostituire θ^* con la MLE $\hat{\theta}$. [2]
Per approssimare il terzo termine scriviamo $H = \sum_{i=1}^{N} H_i$ dove $H_i = \nabla\nabla \log(p(D_i|\theta))$ e stimando ogni H_i con una matrice costante $\hat{H}$ otteniamo

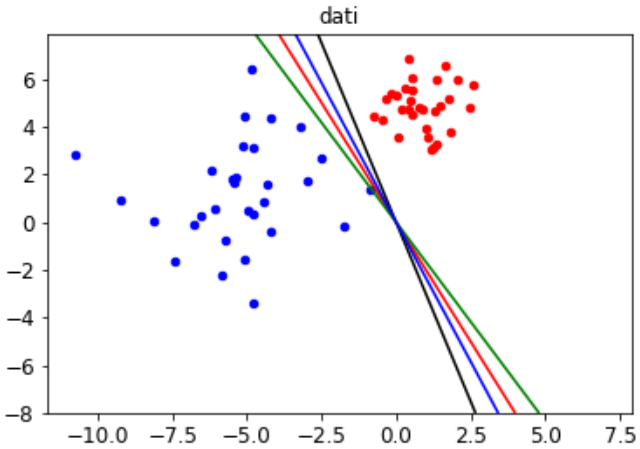
$$\log(p(\mathcal{D})) \approx \log\left(p(\mathcal{D}|\hat{\theta})\right) - \frac{D}{2}\log N$$

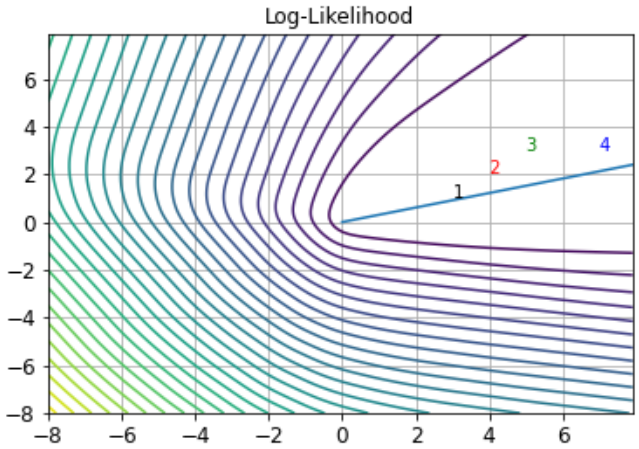
[2] $\theta_{MLE} = \arg\max_\theta p(\mathcal{D}|\theta)$, $\theta_{MAP} = \arg\max_\theta p(\theta|\mathcal{D})$

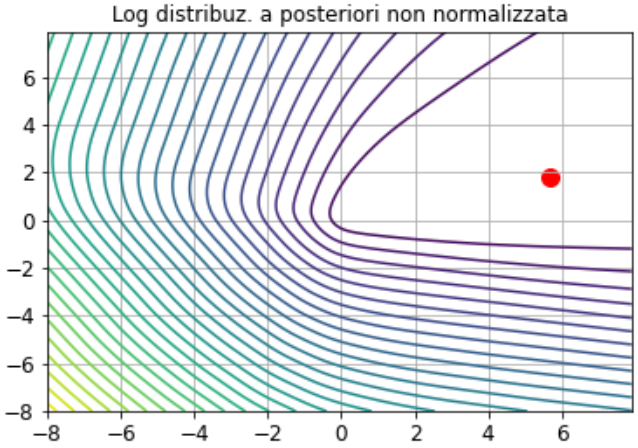
Un esempio

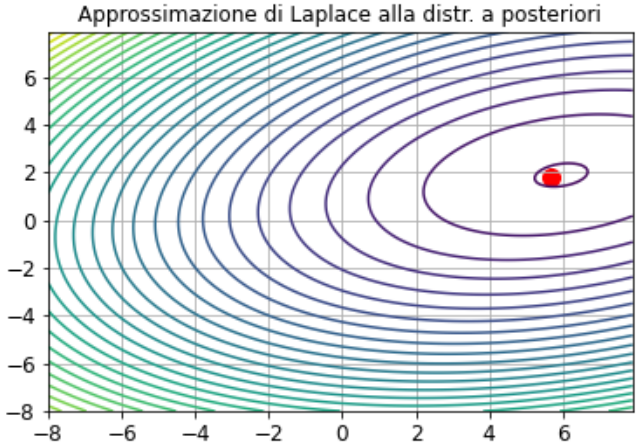
- Consideriamo dei dati in dimensione 2 linearmente separabili: ho rappresentato 4 iperpiani di separazione in alto a sx.
- In alto a dx ho rappresentato la Log-Likelihood: i 4 punti colorati sono i vettori dei parametri per ogni iperpiano. La semiretta rappresentata va dall'origine verso la direzione del MLE (che è all'infinito).
- Per regolarizzare il problema (in modo da assicurare una soluzione unica) ho utilizzato una distribuzione a priori Gaussiana centrata in 0 e con matrice di covarianza $100I$. Moltiplicando la distribuzione a priori per la likelihood si ottiene la distribuzione a posteriori non normalizzata in basso a sx. Il punto evidenziato è il MAP.
- Infine in basso a dx ho riportato l'approssimazione di Laplace alla distribuzione a posteriori.

Grafici









[2] Il codice che ho scritto è presente qui nella sezione "Bayesian Logistic Regression"

Approssimare la distribuzione predittiva a posteriori

Sicuramente in machine learning siamo interessati alle predizioni. La distribuzione predittiva a posteriori è

$$p(y|x, D) = \int p(y|x, w)p(w|D)dw.$$

Tuttavia quest'integrale è intrattabile.
L'approssimazione più semplice di quest'integrale è la cosiddetta **approssimazione plug-in** che ha la forma:

$$p(y = 1|x, D) \approx p(y = 1|x, \mathbb{E}[w]),$$

dove $\mathbb{E}[w]$ è la media a posteriori.

Approssimazione Monte-Carlo

Un approccio migliore è quello di usare un'**approssimazione Monte Carlo** come segue

$$p(y = 1|x, D) \approx \frac{1}{S} \sum_{i=1}^{S} \text{sigm}((w^s)^T x),$$

dove $w^s \sim p(w|D)$ sono campionati dalla distribuzione a posteriori.

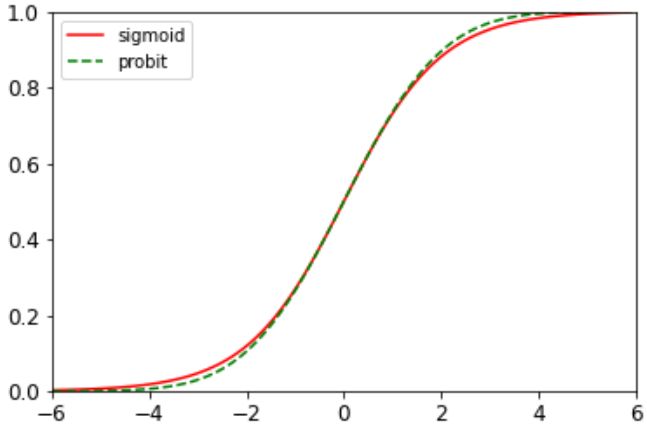
Proibit approximation per la regressione logistica

Se abbiamo $p(w|\mathcal{D}) \approx \mathcal{N}(m_N, V_N)$ possiamo anche calcolare un'approssimazione deterministica della distribuzione predittiva a posteriori (almeno nel caso di binary classification). Si ha

$$p(y = 1|x, \mathcal{D}) = \int \text{sigm}(w^T x) p(w|\mathcal{D}) dw \approx \int \text{sigm}(a) \mathcal{N}(a|\mu_a, \sigma_a) da,$$

dove $a := w^T x, \mu_a := \mathbb{E}[a] = m_N^T x, \sigma_a := \text{Var}(a) = x^T V_N x$.
Quindi dobbiamo stimare la media di una sigmoide rispetto a una distribuzione gaussiana. Per farlo useremo il fatto che la sigmoide è simile alla **proibit function** (i.e alla funzione di ripartizione di una gaussiana di media 0 e varianza 1) opportunamente riscalata.
Precisamente $\text{sigm}(a) \approx \Phi(\lambda a)$ con $\lambda^2 = \frac{\pi^2}{8}$

Proibit approximation per la regressione logistica

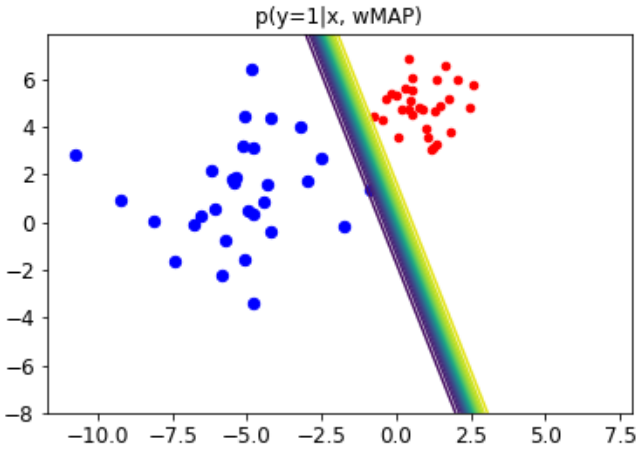


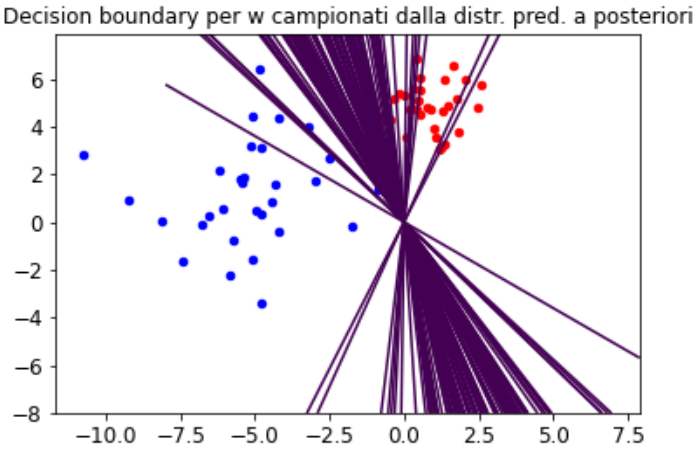
Il vantaggio di questa approssimazione è che sappiamo calcolare esattamente $\int \Phi(a)\mathcal{N}(a|\mu_a, \sigma_a)da = \Phi(\frac{a}{(\lambda^{-2}+\sigma^2)^{1/2}})$. Mettendo tutto insieme otteniamo

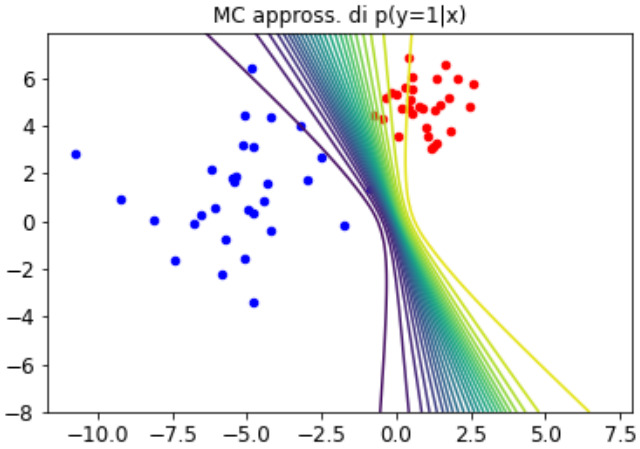
$$p(y = 1|x, D) \approx \text{sigm}(k(\sigma_a^2)\mu_a),$$

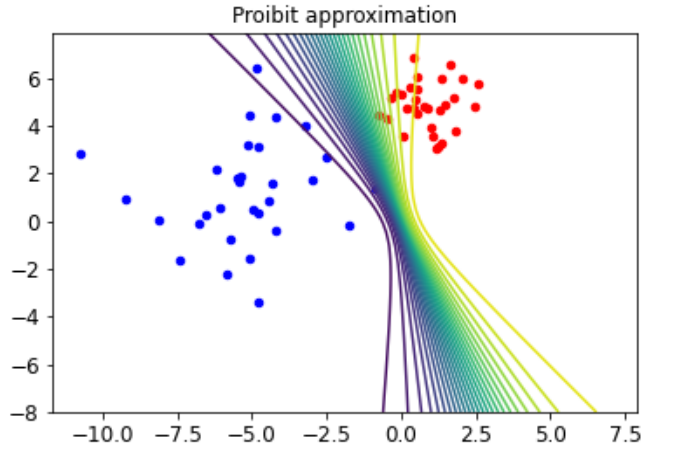
con $k(\sigma^2) := (1 + \pi\sigma^2/8)^{-1/2}$.

Un esempio









[2] Il codice che ho scritto è presente qui nella sezione "Bayesian Logistic Regression"

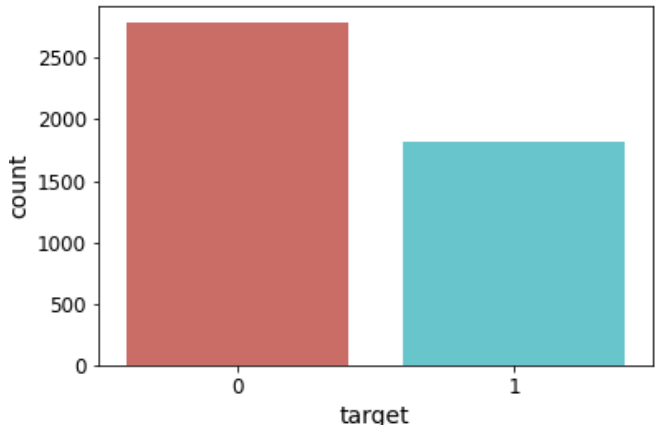
Esempio: Spam dataset

Infine, ho deciso di applicare parte delle cose viste ad un dataset famoso: quello riguardante le email di George Forman, reperibile qui `https://archive.ics.uci.edu/ml/datasets/spambase`.
La task è binary classification, il dataset consiste di 4601 email, da cui sono state estratte 57 features (più una colonna relativa al target).

- 48 feature tra $[0, 100]$ che danno la percentuale di parole in un dato messaggio che coincidono con una specifica parola.
- 6 feature tra $[0, 100]$ che danno la percentuale di alcuni caratteri in un dato messaggio. I caratteri sono ; ([! $ #
- Feature 55: lunghezza media di una sequenza ininterrotta di lettere in maiuscolo.
- Feature 56: lunghezza della più lunga sequenza ininterrotta di lettere in maiuscolo.
- Feature 57: la somma delle lunghezze di una sequenza ininterrotta di lettere in maiuscolo.

Esempio: Spam dataset

Come prima cosa ho visto se il dataset fosse bilanciato o meno, ovvero se presentasse o meno la stessa percentuale di osservazioni con target 0 e 1.



Il dataset non è bilanciato. In seconda battuta, dopo aver permutato il dataset, ho diviso il dataset in training set (80%) e test set (20%) in modo che fossero entrambi bilanciati come il dataset di partenza (**stratification**).

[2] Il codice che ho scritto è presente qui nella sezione "Problema dello spam"

Esempio: Spam dataset

- A questo punto sono passato alla fase di model selection, una volta compilato (i.e. inizializzato) il modello ho scelto i migliori iperparametri con una grid search che valutava ogni n-upla di iperparametri tramite una **3-fold cross validation**.
- Una volta scelti gli iperparametri migliori ho riallenato il modello in tutto il training set e ho stimato l'accuracy nel training set e nel test set: il modello finale ha un'accuracy di circa il 92% sia nel training che nel test set.
- Nel seguito, riporto il codice e le learning curve del modello finale, con alcune considerazioni aggiuntive.

Codice Spam Dataset (model selection)

```
def create_model_sgd(learning_rate, momentum, k_reg):
    model = Sequential()
    model.add(Dense(1,
                    input_dim=57,
                    activation='sigmoid',
                    kernel_initializer= "uniform",
                    kernel_regularizer=regularizers.l2(k_reg)))
    optimizer = tf.keras.optimizers.SGD(lr=learning_rate,
                                        momentum=momentum)
    model.compile(loss='binary_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
    return model
```

Codice Spam Dataset (model selection)

```
np.random.seed(42)
model = KerasClassifier(build_fn=create_model_sgd,
                        epochs = 150,
                        batch_size = 40)
param_grid ={"learning_rate": [0.01, 0.05, 0.1],
             "momentum": [0.1, 0.2, 0.3],
             "k_reg": [0.00001, 0.0001, 0.001]}
model_cv = GridSearchCV(model,
                        param_grid,
                        scoring = None,
                        n_jobs=-1,
                        cv = 3,
                        verbose = 10)
model_result = model_cv.fit(X_train, y_train)
```

Learning curve e accuracy

lr	mom	reg	AccTR	AccVL	AccTS
0.05	0.3	0.0001	92.6%	92.2%	92.2%

Tabella: Training, Validation e test accuracy

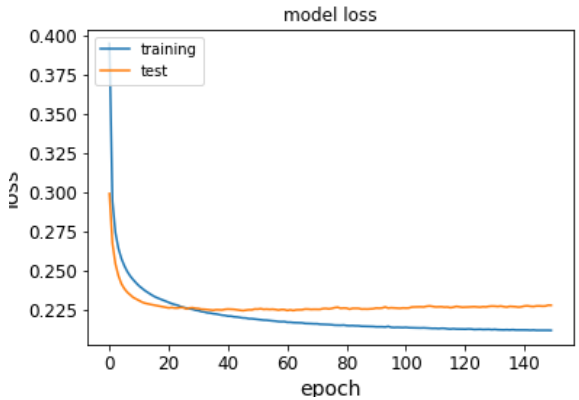


Figura: Learning curve

Alcune considerazioni

- È stato scelto come ottimizzatore l'SGD con learning rate costante e con momento: semplicemente ad ogni iterazione prima di calcolare il gradiente in w si somma ad esso α volte il gradiente al passo precedente. Si dimostra empiricamente che questo aiuta a "smorzare" leggermente le oscillazioni.
- È stato fissato il numero di epoche a 150 perchè dopo una fase di screening ho notato che la convergenza era piuttosto veloce.
- È stata fissata la batch size a 40 per motivi di tempo: l'SGD "full" era troppo lento a causa dell'elevata dimensione del dataset.
- È stato deciso di non raffinare ulteriormente la grid search dopo aver visto i risultati (molto buoni) che dava la cross validation per la terna migliore.

Grazie per l'attenzione!