Introduction to machine learning

Giacomo Fantoni
Telegram: @GiacomoFantoni
Github: https://github.com/giacThePhantom/intro2ml

23 aprile 2021

Indice

Capitolo 1

Introduzione

1.1 Definizioni

Si intende per machine learning lo studio di algoritmi che migliorano autonomamente attraverso l'esperienza. È un campo dell'intelligenza artificiale. Stravolge il paradigma convenzionale della programmazione: un algoritmo di machine learning infatti prende come input un insieme di dati e risultati in modo da produrre un programma che fornisce un risultato appropriato. Coinvolge pertanto la scoperta automatica di regolarità nei dati attraverso algoritmi in modo da poter compiere azioni basate su di essi.

1.2 Processo

Il machine learning permette ai computer di acquisire conoscenza attraverso algoritmi che inferiscono e imparano da dati. Questa conoscenza viene rappresentata da un modello che può essere utilizzato su nuovi dati.

1.2.1 Il processo di apprendimento

Il processo di apprendimento in particolare coinvolge diversi passaggi:

- Acquisizione dei dati dal mondo reale attraverso dispositivi di misurazione come sensori o database.
- Preprocessamento dei dati: filtraggio del rumore, estrazione delle feature e normalizzazione.
- Riduzione dimensionale: selezione e proiezione delle feature.
- Apprendimento del modello: classificazione, regressione, clustering e descrizione.
- Test del modello: cross-validation e bootstrap.
- Analisi dei risultati.

1.3 Modello

Un algoritmo di machine learning impara dall'esperienza E in rispetto di una classe di compiti T e di misurazione delle performance P, se la P di T aumenta con E. Si nota pertanto come un compito

di machine learning ben definito possiede una tripla:

$$< T, P, E >$$

1.4 Deep learning

Il deep learning è un sottoinsieme del machine learning che permette a modelli computazionali composti di multipli strati di imparare la rappresentazione di dati con multipli livelli di astrazione. Si utilizza pertanto una rete neurale con diversi strati di nodi tra input e output. Questa serie di strati tra input e output computa caratteristiche rilevanti automaticamente in una serie di passaggi. Questi algoritmi sono resi possibili da:

- Enorme mole di dati disponibili.
- Aumento del potere computazionale.
- Aumento del numero di algoritmi di machine learning e della teoria sviluppata dai ricercatori.
- Aumento del supporto dall'industria.

Capitolo 2

Machine learning basics

2.1 Introduzione

Il machine learning permette ai computer di acquisire conoscenza attraverso algoritmi che imparano e inferiscono dai dati. Tale conoscenza viene rappresentata da un modello che viene poi utilizzato su dati futuri.

2.1.1 Processo di learning

Si individua un processo di learning:

- Acquisizione di dati dal mondo reale attraverso sensori.
- Preprocessamento dei dati: eliminazione del rumore, estrazione delle features e normalizzazione.
- Riduzione di dimensionalità attraverso selezione e proiezione di features.
- Learning del modello: classification, regression, clustering e description.
- Test del modello attraverso cross-validation e bootstrap.
- Analisi dei risultati.

2.2 Dati

I dati disponibili ad un algoritmo di machine learning sono tipicamente un insieme di esempi. Questi esempi sono tipicamente rappresentati come un array di features, caratteristiche dei dati di interesse per lo studio in atto.

2.2.1 Training e test set

In particolare per questi algoritmi si assume sempre che il training e il test set siano distribuiti secondo variabili indipendenti e identicamente distribuite ($i.i.d$) La distribuzione P_{data} è tipicamente sconosciuta ma si può campionare, attraverso un modello probabilistico di learning. In particolare la distribuzione di probabilità di coppie di esempio e label viene detta data generating distribution e sia il training data che il test set sono generati basandosi su di essa.

2.3 Task

Si intende per task una rappresentazione del tipo di predizione che viene svolta per risolvere un problema su dei dati. Viene identificata con un insieme di funzioni che possono potenzialmente risolverla. In generale consiste di una funzione che assegna ogni input $x \in \mathcal{X}$ a un output $y \in \mathcal{Y}$:

$$f : \mathcal{X} \to \mathcal{Y} \qquad \mathcal{F}_{task} \subset \mathcal{Y}^{\mathcal{X}}$$

La natura di $\mathcal{X}, \mathcal{Y}, \mathcal{F}_{task}$ dipende dal tipo di task.

2.4 Modello

Un modello è un programma per risolvere un problema. È cioè l'implementazione di una funzione $f \in \mathcal{F}_{task}$ che può essere computata. UN insieme di modelli formano uno spazio di ipotesi:

$$\mathcal{H} \subset \mathcal{F}_{task}$$

L'algoritmo cerca una soluzione nello spazio di ipotesi.

2.4.1 Target ideale

Il target ideale del modello è quello di minimizzare una funzione di errore (generalizzazione)

$$E(f; P_{data})$$

Questa funzione determina quanto bene una soluzione $f \in \mathcal{F}_{task}$ fitta dei dati. Guida perrtanto la selezione della migliore soluzione in $\mathcal{F}_{task}$. Pertanto:

$$f^* \in arg \min_{f \in \mathcal{F}_{task}} E(f; P_{data})$$

2.4.2 Target feasible

Si deve restringere il focus sul trovare funzioni che possono essere implementate e valutate in maniera trattabile. Si definisce pertanto uno spazio di ipotesi del modello $\mathcal{H} \subset \mathcal{F}_{task}$ e si cerca la soluzione all'interno di quello spazio:

$$f^*_{\mathcal{H}} \in arg \min_{f \in \mathcal{H}} E(f; P_{data})$$

Si noti come questa funzione non possa essere computata correttamente in quanto P_{data} è sconosciuta.

2.4.3 Target attuale

Per trovare il target attuale si deve lavorare su un campione di dati o il training set

$$\mathcal{D}_n = \{z_1, \ldots, z_n\}$$

Dove

$$z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$$
$$z_i \sim P_{data}$$

Pertanto in:

$$f^*_{\mathcal{H}} \in arg \min_{f \in \mathcal{H}} E(f; P_{data})$$

$E(f; P_{data})$ è il training error.

2.4.4 Funzione di errore

Le funzioni di generalizzazione e di training error possono essere scritte in termini di una pointwise loss $\updownarrow(f; z)$ che misura l'errore che avviene a f su un esempio di training z.

$$E(f; P_{data}) = \mathbb{E}_{z \sim P_{data}}[\updownarrow(f; z)]$$

$$E(f; \mathcal{D}_n) = \frac{1}{n} \sum_{i=1}^{n} \updownarrow(f; z_i)$$

Si nota pertanto come l'algoritmo di learning risolve il problema di ottimizzazione con target:

$$f^*_{\mathcal{H}}(\mathcal{D}_n)$$

2.4.5 Tipi di errore

- Overfitting.
- Estimation error, indotto imparando da un campione di dati.
- Approximation error, indotto dallo spazio di ipotesi $\mathcal{H}$.
- Irreducible error a causa della variabilità intrinseca.

2.4.6 Stimare l'errore di generalizzazione

L'errore di geenralizzazione può essere stimati utilizzando diversi insiemi di training, validation e test.

2.4.6.1 Migliorare la generalizzazione

La generalizzazione può essere migliorata:

- Evitando di ottenere il minimo sul training error.
- Riducendo la capacità del modello.
- Cambiando l'obiettivo con un termine di regolarizzazione.
- Iniettando rumore nell'algoritmo.
- Fermando l'algoritmo prima che converga.
- Aumentantdo la quantità di dati.
- Aggiungendo più campioni di training.
- Aumentando il training set con trasformazioni.
- Combinando predizioni da più modelli decorrelati o ensembling.

2.4.6.1.1 Regolarizzazione Si intende per regolarizzazione la modifica della funzione di training error con un termine $\Omega(f)$ che penalizza soluzioni complesse:

$$E_{reg}(f; \mathcal{D}_n) = E(f; \mathcal{D}_n) + \lambda_n \Omega(f)$$

2.5 Tipi di learning

2.5.1 Supervised learning

Nel supervised learning vengono dati in input a un modello o predittore un insieme di esempi che possiedono una label. Il modello poi impara a creare delle predizioni su un nuovo esempio.

2.5.1.1 Dati

Nel caso del supervised learning i dati creano una distribuzione:

$$p_{data} \in \Delta(\mathcal{X} \times \mathcal{Y})$$

2.5.1.2 Classificazione

In un problema di classificazione si trova un insieme finito di label discrete. In particolare dato un training set $\mathcal{T} = \{(x_1, u_1), \dots (x_m, y_m)\}$, si deve imparare una funzione f per predirre y dato x. f sarà pertanto:

$$f : \mathbb{R}^d$$

Dove d è la dimensionalità di x e k il numero di labels distinte.

2.5.1.2.1 Task Si deve pertanto trovare una funzione $f \in \mathcal{Y}^{\mathcal{X}}$ che assegna ogni input $x \in \mathcal{X}$ a una label discreta.

$$f(x) \in \mathcal{Y} = \{c_1, \dots, c_k\}$$

2.5.1.3 Regression

Un problema di regressione presenta un insieme di label continue. Dato un training set $\mathcal{T} = \{(x_1, y_1), \dots, (x_m, y_m)\}$, si deve imparare una funzione f per predirre y dato x. f sarà pertanto:

$$f : \mathbb{R}^d \to \mathbb{R}$$

Dove d è la dimensionalità di x.

2.5.1.3.1 Task Si deve trovare una funzione $f(x) \in \mathcal{Y}$ che assegna ogni input a una label continua.

2.5.1.4 Ranking

Il ranking è un tipo particolare di classificazione in cui una label è un ranking.

2.5.2 Unsupervised learning

Nel unsupervised learning vengono dati in input a un modello o predittore un insieme di esempi senza label. Il modello impara a creare delle predizioni su un nuovo esempio.

2.5.2.1 Dati

Nel caso del supervised learning i dati creano una distribuzione:

$$p_{data} \in \Delta(\mathcal{X})$$

2.5.2.2 Clustering

Nel clustering, data $\mathcal{T} = \{x_1, \dots, x_m\}$ si deve trovare la struttura nascosta che intercorre tra le x o i clusters.

2.5.2.2.1 Task Si deve trovare una funzione $f \in \mathbb{N}^{\mathcal{X}}$ che assegna ogni input $x \in \mathcal{X}$ a un indice di cluster $f(x) \in \mathbb{N}$. Tutti i punti mappati sullo stesso indice formano un cluster.

2.5.2.3 Dimensionality reduction

Nella dimensionalty reduction si tenta di ridurre il numero di variabili sotto considerazione ottenendo un insieme di variabili principali.

2.5.2.3.1 Task Si deve trovare una funzione $f \in \mathcal{Y}^{\mathcal{X}}$ che mappa ogni input di molte dimensioni $x \in \mathcal{X}$ a un output a dimensione minore $f(x) \in \mathcal{Y}$, dove $dim(\mathcal{Y} \ll dim(\mathcal{X})$

2.5.3 Reinforcement learning

Nel reinforcement learning un agente impara dall'ambiente interagendo con esso e ricevendo premi per lo svolgimento di azioni particolari. In particolare, data una sequenza di esempi o stati e una reward dopo il completamento di tale sequenza si impara a predirre l'azione da svolgere per uno stato o esempio individuale.

Capitolo 3

KNN

3.1 Introduzione

Si possono considerare gli esempi come punti in uno spazio n dimensionale dove n è il numero di features. Per classificare un esempio d si può mettere a d una label uguale a quella dell'esempio più vicino a d nel training set. Questo concetto viene esteso nel K-nearest neighbour o *K-NN* in cui per classificare un esempio d si trovano i k esempi più vicini di d e si sceglie la label in maggior numero tra i k vicini più prossimi.

3.2 Misurare la distanza

Misurare la distanza tra due esempi è specifico al problema, ma un modo possibile è la distanza euclidea:

$$D(a,b) = \sqrt{(a_1 - b_1)^2 + \cdots + (a_n - b_n)^2}$$

3.3 Decision boundaries

I decision boundaries sono posti nello spazio delle features dove la classificazione di un punto o un esempio cambia. In particolare *K-NN* definisce dei decision boundaries localmente tra le classi.

3.4 Il ruolo di K

I fattori che determinano la bontà di un algoritmo di machine learning sono la sua abilità di minimizzare il training error e minimizzare il gap tra il training error e il test error. Questi due fattori corrispondono a underfitting e overfitting.

3.4.1 Underfitting

L'underfitting avviene quando il modello non è capace di ottenere un valore di errore abbastanza piccolo sul training set.

3.4.2 Overfitting

L'overfitting avviene quando il gap tra il training error e il test error è troppo grande.

3.5 Scelta di K

Il valore di K comprende euristiche comuni come $3, 5, 7$ o un numero dispari per evitare pareggi. Può essere scelto utilizzando dati di sviluppo. Per classificare un esempio d si trovano i k vicini più prossimi di d e si sceglie la classe più presente nei k scelti.

3.6 Variaizoni di K

Invece di scegliere i K vicini più prossimi si possono contare tutti gli esempi in una distanza fissata.

3.6.1 *K-NN* pesata

Si può pesare il voto di tutti gli esempi in modo che esempi più vicini pesino di più. Si usa speso qualche tipo di decadimento esponenziale.

Capitolo 4

Modelli lineari

4.1 Introduzione

Alcuni approcci del machine learning fanno delle forti assunzioni riguardo i dati. Questo avviene in quanto se le assunzioni sono vere si possono raggiungere performance migliori. In caso contrario l'approccio può fallire miseramente. Altri approcci che non fanno molte assunzioni riguardo i dati invece permettono di imparare da dati più vari ma sono più proni a overfitting e richiedono più dati di training.

4.1.1 Bias

Il bias di un modello è quanto forte le assunzioni del modello sono. I classificatori a low-bias fanno delle assunzioni minime riguardo i dati come *k-NN* (che assume unicamente che la vicinanza è correlata alla classe) e *DT*. I classificatori a high-bias fanno assunzioni forti riguardo ai dati.

4.2 Linear separability

L'assunzione strong-bias dei modelli lineari è la linear separability, ovvero che in due dimensioni le classi si possano separare attraverso una linea, mentre in dimensioni maggiori da un hyperplane. Un modello lineare è pertanto un modello che assume che i dati sono linearmente separabili.

4.2.1 Definire una linea

Ogni coppia di valori (w_1, w_2) definisce una linea attraverso l'origine:

$$0 = w_1 f_1 + w_2 f_2$$

Si può inoltre vedere il vettore $\overrightarrow{w} = (w_1, w_2)$ come il vettore dei pesi perpendicolare alla linea. Per classificare i punti rispetto alla linea si considera il segno sostituendo i punti a f_1 e f_2. La positività o negatività indica il lato della linea. Si può estendere l'equazione con:

$$a = w_1 f_1 + w_2 f_2$$

In queto modo la linea interseca l'asse delle y in a.

4.3 Definizione di modello lineare

Si definisce un modello lineare in uno spazio n dimensionale, dove n è il numero di features attraverso $n+1$ pesi.

$$0 = b + \sum_{i=1}^{n} w_i f_i$$

; Si classifica un modello lineare controllando il segno di un vettore $\overrightarrow{f}$ n dimensionale di features.

4.3.1 Training

Il training di un modello lineare avviene online, ovvero a differenza del modo in batch in cui vengono dati i training data come $\{(x_i, y_i) : 1 \leq i \leq n\}$, i data points arrivano uno alla volta. L'algoritmo allora riceve un esempio x_i senza label, predice la classificazione di questo esempio e confronta la predizione con l'effettivo y_i. Infine aggiorna il proprio modello.

4.3.2 Perceptron

4.3.2.1 Numero di iterazioni

Il numero di iterazioni del perceptron viene deciso in base alla convergenza. Inoltre può essere limitato in modo da ridurre l'overfitting.

4.3.2.2 Ordine dei campioni

I campioni da considerare nel perceptron sono considerati in ordine casuale.

4.3.2.3 Linear separable sets

Le istanze di training sono lienarmente separabili se esiste un hyperplane che separa le due classi.

4.3.2.4 Algoritmo

```
: Perceptron()
repeat
    foreach training example (f_1, f_2, ..., f_n, label) do
        %Label = ±1
        check if it is correct based on the current label
        if not correct then
            %update all the weights
            foreach w_i do
                w_i = w_i + f_i * label
                b = b + label
until Convergence
%Or some number of iteration
```

4.3.2.5 Calcolo della predizione

```
: Perceptron()
repeat
    foreach training example (f_1, f_2, ..., f_n, label) do
        %Label = ±1
        prediction = b + Σ_{i=1}^{n} w_i f_i
        if prediction is not label then
            %update all the weights
            foreach w_i do
                w_i = w_i + f_i * label
                b = b + label
until Convergence
%Or some number of iteration
```

Capitolo 5

Decision Trees

5.1 Struttura

Un decision tree è un modello di predizione con struttura ad albero. È composto da nodi terminali o foglie e nodi non terminali. I nodi non terminali hanno da due a più figli e implementando la funzione di routing. I nodi foglia non hanno figli e implementano la funzione di predizione. Non ci sono cicli e tutti i nodi hanno al massimo un genitore (con esclusione del nodo radice).

5.2 Funzionamento

Un decision tree prende un input $x \in \mathcal{X}$ e lo routa attraverso i nodi fino a che raggiunge un nodo foglia dove avviene la predizione. Ogni nodo non terminale

$$Node(\phi, t_L, t_R)$$

Contiene una funzione di routing $\phi \in \{L, R\}^{\mathcal{X}}$, un figlio destro t_L e un figlio sinistro t_R. Quando x raggiunge il nodo viene spostato sul figlio destro o sinistro in base al valore di $\phi(x) \in \{L, R\}$. Ogni nodo foglia

$$Leaf(h)$$

Contiene una funzione di predizione $h \in \mathcal{F}_{task}$, tipicamente una costante.

5.2.1 Inferenza

Sia f_t la funzione che ritorna la predizione per l'input $x \in \mathcal{X}$ secondo il decision tree t. Questa viene definita come:

$$f_t(x) = \begin{cases} h(t) & \textbf{if} t = Leaf(h) \\ f_{t_{\phi(x)}}(x) & \textbf{if} t = Node(\phi, t_L, t_R) \end{cases}$$

5.3 Decision trees learning algorithm

Dato un training set $\mathcal{D}_n = \{z_1, \ldots, z_n\}$ si deve trovare f_{t^*} dove:

$$t^* \in arg\min_{t \in \mathcal{T}} E(f_t; \mathcal{D}_n$$

Dove $\mathcal{T}$ è l'insieme dei decision trees. Il problema di ottimizzazione è facile se non si impongono constraints, altrimenti potrebbe diventare *NP-hard*. Una soluzione può essere trovata utilizzando una strategia greedy. Pertanto si assume:

$$E(f_t; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{z \in \mathcal{D}} \updownarrow(f; z)$$

Ora fissato un insieme di predizioni di foglie

$$\mathcal{H}_{leaf} \subset \mathcal{F}_{task}$$

E fissato un insieme di possibili funzioni di routing o split

$$\Phi \subset \{L, R\}^{\mathcal{X}}$$

La strategia di crescita dell'albero partiziona ricorsivamente il training set e decide se crescere foglie o nodi non terminali.

5.3.1 Crescere una foglia

Sia $\mathcal{D} = \{z_1, \ldots, z_m\}$ il training set che raggiunge un nodo. Il predittore di foglia ottimale viene computato come:

$$h^*_{\mathcal{D}} \in arg \min_{h \in \mathcal{H}_{leaf}} E(h; \mathcal{D})$$

Il valore di errore ottimale o misura di impurità:

$$I(\mathcal{D} = E(h^*_{\mathcal{D}}; \mathcal{D})$$

Se si raggiungono dei criteri si cresce una foglia $Leaf(h^*_{\mathcal{D}})$ Esempi di questi criteri sono la purezza $I(\mathcal{D}) < \epsilon$ o la cardinalità minima $|\mathcal{D}| < k$.

5.3.2 Crescere un nodo

Se non si raggiunge il criterio si deve trovare una funzione di split ottimale:

$$\phi^*_{\mathcal{D}} \in arg \min_{\phi \in \Phi} I_\phi(\mathcal{D})$$

L'impurità $I_\phi(\mathcal{D})$ di una funzione di split ϕ dato il training set $\mathcal{D}$ viene computata nei termini di impurità dei dati splittati:

$$I_\phi(\mathcal{D}) = \sum_{d \in \{L,R\}} \frac{|\mathcal{D}^\phi_d|}{|\mathcal{D}|} I(\mathcal{D}^\phi_d)$$

Dove

$$\mathcal{D}^\phi_d = \{(x, y) \in \mathcal{D}; \phi(x) = d\}$$

L'impurità di una funzione di split è il più basso errore di training che può essere ottenuto da un albero che consiste di una radice e due figlie. Si cresce pertanto un nodo $Node(\phi^*, t_L, t_R)$ dove ϕ^* è lo split ottimale, mentre t_L e t_R sono ottenuti applicando ricorsivamente l'algoritmo di learning ai training set splits.

5.3.3 Algoritmo

$$Grow(\mathcal{D}) = \begin{cases} Leaf(h^*_{\mathcal{D}}) & raggiunto\ criterio\ di\ stop \\ Node(\phi^*_{\mathcal{D}}, Grow(\mathcal{D}^*_L), Grow(\mathcal{D}^*_R)) & altrimenti \end{cases}$$

Dove $\mathcal{D}^*_d = \{(x,y) \in \mathcal{D}; \phi^*_{\mathcal{D}}(x) = d\}$.

5.3.4 Split selection

Tipicamente la migliore funzione di split viene data in termini di minimizzazione dell'impurità dello split, ma altre volte nella massimizzazione del guadagno di informazioni o

$$\Delta_\phi(\mathcal{D}) = I(\mathcal{D}) - I_\phi(\mathcal{D})$$

Essendo $\Delta_\phi(\mathcal{D}) \geq 0$ per ogni $\phi \in \{L,R\}^{\mathcal{X}}$ e ogni training set $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$, l'impurità non aumentera mai per ogni split scelto casualmente.

5.3.5 Predizione delle foglie

La predizione delle foglie fornisce una soluzione a un problema semplificato coinvolgendo solo dati che la raggiungono. Questa soluzione può essere una funzione arbitraria $h \in \mathcal{F}_{task}$, ma in pratica si restringe a un sottoinsieme di $\mathcal{H}_{leaf}$. Il predittore più semplice è una funzione che ritorna una costante (come una label). L'insieme di tutte le possibili funzioni costanti può essere scritto come:

$$\mathcal{H}_{leaf} = \bigcup_{y \in \mathcal{Y}} \{y\}^{\mathcal{X}}$$

5.4 Misure di impurità per la classificazione

Si consideri per la classificazione:

$$\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$$

$$\mathcal{Y} = \{c_1, \ldots, c_k\}$$

Sia $\mathcal{D}^y = \{(x,y') \in \mathcal{D} : y = y'\}$, che denota il sottoinsieme di training samples in $\mathcal{D}$ con label y. Considerando la funzione di errore:

$$E(f,\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{z \in \mathcal{D}} \updownarrow(f;z)$$

Se $\updownarrow(f;(x,y)) = 1_{f(x) \neq y}$ e $\mathcal{H}_{leaf} = \bigcup_y \in \mathcal{Y}\{y\}^{\mathcal{X}}$ la misura di impurità è allora il classification error:

$$I(\mathcal{D}) = 1 - \max_{y \in \mathcal{Y}} \frac{|\mathcal{D}^y|}{|\mathcal{D}|}$$

Se invece $\updownarrow(f;(x,y)) = \sum_{x \in \mathcal{Y}} [f_c(x) - 1_{c=y}]^2$ e $\mathcal{H}_{leaf} = \bigcup_{\pi \in \Delta(\mathcal{Y})} \{\pi\}^{\mathcal{X}}$ allora la misura di impurità è l'impurità di Gini:

$$I(\mathcal{D}) = 1 - \sum_{y \in \mathcal{Y}} \left(\frac{|\mathcal{D}^y|}{|\mathcal{D}|} \right)^2$$

Infine se $\updownarrow(f;(x,y)) = -\log f_y(x)$ e $\mathcal{H}_{leaf} = \bigcup_{\pi\in\Delta(\mathcal{Y})} \{\pi\}^{\mathcal{X}}$, con una distribuzione costante di label come predizione di foglie, allora la misura di impurità è l'entropia:

$$I(\mathcal{D}) = -\sum_{y\in\mathcal{Y}} \frac{|\mathcal{D}^y|}{|\mathcal{D}|} \log \frac{|\mathcal{D}^y|}{|\mathcal{D}|}$$

5.5 Misure di impurità per la regressione

Si consideri per la regressione:

$$\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$$

$$\mathcal{Y} \subset \mathbb{R}^d$$

Se $\updownarrow(f;(x,y)) = ||f(x) - y||_2$ e $\mathcal{H}_{leaf} = \bigcup_{y\in\mathcal{Y}} \{y\}^{\mathcal{X}}$ allora la misura di impurità è la varianza:

$$I(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} ||x - \mu_{\mathcal{D}}||^2$$

Dove $\mu_{\mathcal{D}} = \frac{1}{\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} x$

5.6 Data features e attributi

Un data point $x \in \mathcal{X}$ potrebbe essere d dimensionale con ogni dimensione con tipi di valori eterogenei come discreti o continui e avere un ordinamento o no, rispettivamente ordinali o nominali.

5.7 Funzioni di split o routing

Il routing o split $\phi \in \{L, R\}^{\mathcal{X}}$ determina se un data point $x \in \mathcal{X}$ deve muoversi a destra o sinistra. La possibile funzione di split è ristretta in un insieme predefinito $\Phi \subset \{L, R\}^{\mathcal{X}}$ in base alla natura dello spazio di features. La funzione di split prototipica per un input d dimensionale prima seleziona una dimensione e poi applica un criterio di split 1 dimensionale.

5.7.1 Features discrete e nominali

Si assumano features discrete e nominali con valori in $\mathcal{K}$. La funzione di split può essere implementata data una partizione di $\mathcal{K}$ in $\mathcal{K}_R$ e $\mathcal{K}_L$:

$$\phi(x) = \begin{cases} L & \textbf{if} x \in \mathcal{K}_L \\ R & \textbf{if} x \in \mathcal{K}_R \end{cases}$$

Trovare lo split ottimale richiede testare $2^{|\mathcal{K}|-1} - 1$ bi-partizioni.

5.7.2 Features ordinali

Si assumano features ordinali con valori in $\mathcal{K}$. La funzione di split può essere implementata dando una soglia $r \in \mathcal{K}$:

$$\phi(x) = \begin{cases} L & \textbf{if} x \leq r \\ R & \textbf{if} x > r \end{cases}$$

Se $|\mathcal{K}| \leq |\mathcal{D}|$ trovare lo split ottimale richiede il test di $|\mathcal{K} - 1$ soglie. Se $|\mathcal{K} > |\mathcal{D}|$ si deve ordinare i valori di input in $\mathcal{D}$ dove $\mathcal{D}$ è il training set che raggiunge il nodo e testare $|\mathcal{D}| - 1$ soglie.

5.7.3 Obliquo

A volte è conveniente fare split considerando più features alla volta. Tali funzioni lavorano con features continue e sono dette oblique in quanto generano decision boundaries obliqui. Se $x \in \mathbb{R}^d$ allora la funzione di split può essere implementata dato $w \in \mathbb{R}^d$ e $r \in \mathbb{R}$:

$$\phi(x) = \begin{cases} L & \textbf{if} w^T x \leq r \\ R & \textbf{altimenti} \end{cases}$$

Si nota come questa funzione sia più difficile da ottimizzare.

5.8 Decision trees e overfitting

I decision trees sono modelli non parametrici con una struttura determinata dai dati. Per questo sono flessibili e possono facilmente fare fit sul training set, con un alto rischio di overfitting. Tecniche standard per migliorare la generalizzazione si applicano ai decision trees:

- Early stopping.
- Complexity reduction.
- Regularization.
- Data augmentation.
- Ensembling.

Una tecnica per ridurre la complessità a posteriori è detta pruning.

5.9 Random forest

Le random forest sono ensembles di decision trees. Ogni algero è tipicamente trained con una versione bootstrapped del training set campionata con sostituzione. Le funzioni di split sono ottimizzate su features campionate a caso o completamente a caso (extremely randomized trees). Questo aiuta ad ottenere decision trees decorrelati. La predizione finale della foresta è ottenuta facendo la media delle predizioni per ogni albero nell'ensemble $\mathcal{Q} = \{t_1, \ldots, t_T\}$.

$$f_{\mathcal{Q}}(x) = \frac{1}{T} \sum_{j=1}^{T} f_t(x)$$

Capitolo 6

Multi class classification

6.1 Introduzione

6.1.1 Classificazione binaria

Si è definita la classificazione binaria come la task in cui dati:

- Uno spazio di input $\mathcal{X}$.
- Una distribuzione sconosciuta $\mathcal{D}$ su $\mathcal{X} \times \{-1, +1\}$.
- Un training set D campionato da $\mathcal{D}$

Si deve computare una funzione f che minimizza $\mathbb{E}_{(x,y)\sim\mathcal{D}}[f(x) \neq y]$

6.1.2 Classificazione multi classe

La classificazione multiclasse è l'estensione naturale della classificazione binaria. L'obiettivo è quello di assegnare una label discreta a degli esempi. La differenza è che ora ci sono $k > 2$ classi da cui scegliere. Dati pertanto:

- Uno spazio di input $\mathcal{X}$ e un numero di classi K. $[K]$.
- Una distribuzione sconosciuta di $\mathcal{D}$ su $\mathcal{X} \times [K]$.
- Un training set D campionato da $\mathcal{D}$.

Si deve computare una funzione f che minimizza $\mathbb{E}_{(x,y)\sim\mathcal{D}}[f(x) \neq y]$.

6.1.2.1 K nearest neighbours

Si noti come una *K-NN* per classificare un esempio d trova i k vicini di d e sceglie la label in presenza maggiore tra i k vicini più prossimi. Non necessita pertanto di cambi algoritmici nel caso della multi class classification.

6.1.2.2 Decision tree

I decision tree non richiedono cambi algoritmici per la multi class classification.

6.1.3 Approccio black box alla multi class classification

Dato un classificatore binario questo si può usare per risolvere il problema della multiclass classification. Si noti come un perceptron oltre al risultato può anche dare un punteggio di confidenza. Inoltre siccome una linea non è sufficiente per dividere le classi se ne possono usare diverse.

6.2 One versus all *OVA*

Nell'approccio *OVA* nel training si definisce per ogni label L un problema binario in cui:

- Tutti gli esempi con la label L sono positivi.
- Tutti gli altri esempi sono negativi.

In pratica si imparano L diversi modelli di classificazione. Si ricordi come il classificatore divide il piano in due semipiani.

6.2.1 Ambiguità

In questo caso si formano pertanto delle zone in cui si creano delle ambiguità. Se il classificatore non fornisce confidence e c'è ambiguità si sceglie una delle label in conflitto. Nella maggior parte dei casi i classificatori forniscono confidence, allora in questo caso si:

- Si sceglie il positivo con confidence maggiore.
- Se nessuno è positivo si sceglie il negativo con confidence minore.

La confidence nel perceptron si calcola come distanza dall'iperpiano stabilito dalla prediction.

6.2.2 Algoritmi

: `OneVersusAllTrain(`$D^{multiclass}$`, BinaryTrain())`

for $i = 1$ **to** K **do**
  D^{bin} = relabel $D^{multiclass}$ in modo che i è positivo e $\neq i$ è negativo
  f_i = `BinaryTrain(`D^{bin}`)`
Return $f_1, \ldots, f_K$

: `OneVersusAllTest(`$f_1, \ldots, f_K$`,` $\hat{x}$`)`

score = $\langle 0, \ldots, 0 \rangle$ %`Inizializza` K `score a` 0
for $i = 1$ **to** K **do**
  y = $f_i(\hat{x})$
  $score_i = score_i + y$
Return `max(`*score*`)`

6.3 All versus all *AVA*

Un approccio alternativo consiste nel gestire il problema della classificazione multi classe decomponendolo in problemi di classificazione binaria. Questo approccio viene detto anche all pairs. Si classificano $\frac{K(K-1)}{2}$ classificatori in modo che

$$F_{ij}, 1 \leq i < j \leq K$$

Sia il classificatore che discrimina la classe i contro la classe j. Questo classificatore riceve tutti gli esempi della classe i come positivi e tutti gli esempi della classe j come negativi. Quando arriva un punto di test si valuta su tutti i classificatori F_{ij}. Ogni volta che F_{ij} predice positivo la classe i prende un voto, altrimenti lo prende j. Dopo aver eseguito tutti i $\frac{K(K-1)}{2}$ classificatori la classe con più voti decide la label.

6.3.1 *AVA* training

Per ogni coppia di label si addestra un classificatore che le distingue:

```
: AllVersusAllTrain()
for i = 1 to number of labels do
    for j = i + 1 to number of labels do
        train a classifier F_ij to distinguish between label_j and label_i
        create a dataset with all examples with label_j labeled positive and with label_i
         negative
        Train classifier F_ij su questo sottoinsieme di dati
```

6.3.2 *AVA* classification

Per classificare un esempio x lo si classifica per ogni classificatore F_{ij}. Per scegliere la classe finale si può:

- Considerare la maggioranza.
- Considerare un voto pesato basato sulla confidence:
  - $y = F_{ij}(x)$.
  - $score_j += y$.
  - $score_i -= y$.

Lo score viene cambiato in quanto se y è positivo il classificatore lo pensa di tipo j, se negativo lo pensa di tipo i e pertanto lo score viene aggiornato di conseguenza.

6.3.3 Algoritmi

6.4 Confronto tra *OVA* e *AVA*

6.4.1 Tempo di training

AVA impara più classificatore ma il training set è omlto più piccolo pertanto tende ad essere più veloce se le label sono equamente bilanciate.

: `AneVersusAllTrain(`$D^{multiclass}$`, BinaryTrain())`

$f_{ij} = \emptyset, \forall 1 \leq i < j \leq K$
for $i = 1$ **to** $K - 1$ **do**
  $D^{pos} =$ all $x \in D^{multiclass}$ labeled i
  for $j = j + 1$ **to** K **do**
    $D^{neg} =$ all $x \in D^{multiclass}$ labeled j
    $D^{bin} = \{(x, +1) : x \in D^{pos}\} \cup \{(x, -1) : x \in D^{neg}\}$
    $f_{ij} =$ `BinaryTrain(`D^{bin}`)`
Return all f_{ij}

: `AllVersusAllTest(`*all* f_{ij}, $\hat{x}$`)`

score $= \langle 0, \ldots, 0 \rangle$ %`Inizializza` K `score a` 0
for $i = 1$ **to** $K - 1$ **do**
  for $j = i + 1$ **to** K **do**
    y $= f_{ij}(\hat{x})$
    $score_i = score_i + y$
    $score_j = score_j - y$
Return `max(`*score*`)`

6.4.2 Tempo di test

Avendo *AVA* più classificatori è tipicamente più lenta.

6.4.3 Errori

AVA fa training con data sets più bilanciata, ma avendo più classificatori i test tendono ad avere più possiblità di errori.

6.5 Riassunto

Se vengono usati classificatori binari viene tipicamente utilizzata *OVA*, altrimenti si usa un classificatore che permette label multiple come *DT* o *K-NN*, nonostante altri metodi più sofisticati siano meglio.

6.6 Multiclass evaluation

6.6.1 Microaveraging

Nel microaveraging si fa la media sugli esempi.

6.6.2 Macroaveraging

Nel macroaveraging si calcola lo score di valutazione o accuratezza per ogni label e poi si fa la media tra le label. Questo in quanto dà più enfasi a label più rare e permette un'altra dimensione di analisi.

6.6.3 Confusion matrix

La confusion matrix è una matrice in cui (i, j) rappresenta il numero di esempi con label i predetti avere label j. Viene spesso espressa come percentuale.

Capitolo 7

Gradient descent

7.1 Model based machine learning

Nel model based machine learning si sceglie un modello definito da un insieme di parametri. In particolare si nota come:

- Per i decision trees i parametri sono la struttura dell'albero, quali features ogni nodo divide e le predizioni delle foglie.
- Per il perceptron i parametri sono i pesi e il valore di b.

Dopo aver scelto il modello si deve scegliere un criterio da ottimizzare o la funzione obiettivo come per esempio il training error. Infine si sviluppa un algoritmo di learning che deve cercare di minimizzare il criterio, spesso in maniera euristica.

7.1.1 Modelli lineari

Nei modelli lineari il modello è:

$$0 = b + \sum_{j=1}^{m} w_j f_j$$

Si deve scegliere il criterio da ottimizzare.

7.1.1.1 Notazioni

7.1.1.1.1 Funzione indicatrice Una funzione indicatrice trasforma valori di *Vero* e *Falso* in numeri e conte.

$$1[x] = \begin{cases} 1 & \textbf{if} x = True \\ 0 & \textbf{if} x = False \end{cases}$$

7.1.1.1.2 Dot-product Utilizzando una notazione vettoriale si rappresenta un esempio $f_1, \ldots, f_m$ come un vettore singolo $\overrightarrow{x}$ in cui j indicizza la feature e i indicizza un dataset di esempi. Si possono rappresentare anche i pesi $w_1, \ldots, w_m$ come un vettore $\overrightarrow{w}$. Il dot-product tra due vettori a e b viene definito come:

$$a \cdot b = \sum_{j=1}^{m} a_j b_j$$

7.1.1.2 Funzione obiettivo

Il criterio da ottimizzare o funzione obiettivo può essere:

$$\sum_{i=1}^{n} 1[y_i(w \cdot x_i + b) \leq 0]$$

Si devono pertanto trovare w e b tali che minimizzano questa funzione, ovvero:

$$argmin_{w,b} \sum_{i=1}^{n} 1[y_i(w \cdot x_i + b) \leq 0]$$

7.2 Loss functions

7.2.1 Loss $0/1$

Una funzione di loss $0/1$ è una funzione nella forma:

$$\sum_{i=1}^{n} 1[y_i * w \cdot x_i + b) \leq 0]$$

Dove tra le quadre si trova se la predizione e la label sono d'accordo, con vero se non lo fanno e tra le tonde la distanza dall'iperpiano, di cui il segno è la predizione. Questa funzione ritorna il numero di sbagli.

7.2.1.1 Minimizzare la loss $0/1$

Per minimizzare una funzione $0/1$ si deve, ogni volta cambiare un valore di w in modo che l'esempio è corretto o scorretto la perdita aumenta o diminuisce. Si nota come a ogni feature aggiunta si aggiunge una nuova dimensione allo spazio. Il minimo si trova trovando w e b che minimizzano la perdita. Questo è un problema *NP-hard*. Sue difficoltà comprendono il fatto che piccoli cambi in ogni w possono portare a grandi cambi nella perdita in quanto il cambio non è continuo. Ci possono essere molti minimi locali. Ad ogni punto non si hanno informazioni che direzionano verso il minimo. Pertanto si nota come una loss function ideale sia continua e differenziabile in modo da avere un'indicazione verso la direzione di minimizzazione e un unico minimo.

7.2.2 Funzioni convesse

In una funzione convessa il segmento tra qualsiasi due punti della funzione si trova al di sopra della funzione.

7.2.3 Surrogate loss function

Per molte applicazioni si vuole minimizzare la loss $0/1$. Una surrogate loss function è una loss function che fornisce un limite superiore alla loss function attuale. Si vuole identificare un surrogato convesso della loss function in modo da facilitarne la minimizzazione. Chiave a una loss function è come verifica la differenza tra la label y effettiva e la predizione y'.

7.2.3.1 Alcune surrogate loss function

- 01 *loss*: $l(y, y') = 1[yy' \leq 0]$.
- *Hinge* $l(y, y') = \max(0, 1 - yy')$.
- Exponential: $l(y, y') = \exp(iyy')$
- *Squared loss*: $l(y, y') = (y - y')^2$.

7.3 Gradient descent

Il gradient descent è un modo per trovare il minimo di una funzione: le derivate parziali danno un slope o direzione dove muoversi in tale dimensione. Questo approccio consiste di scegliere un punto di partenza e a ripetizione di: scegliere una dimensione e muoversi di una piccola quantità verso il minimo utilizzando la derivata.

7.3.1 Spostamento in direzione della minimizzazione dell'errore

Il movimento in direzione della minimizzazione dell'errore è pertanto:

$$w_j = w_j - \eta \frac{d}{dw_i} loss(w)$$

Dove η è il learning rate.

7.3.1.1 Calcolo dello spostamento per la loss function esponenziale

Si deve pertanto calcolare:

$$\begin{aligned}\frac{d}{dw_j} loss &= \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) \\ &= = \sum_{i=1}^{n} \frac{d}{dw_j} [-y_i(w \cdot x_i + b)] \exp(-y_i(w \cdot x_i + b))\end{aligned}$$

Si consideri pertanto ora:

$$\begin{aligned}\frac{d}{dw_j}[-y_i(w \cdot x_i + b)] &= -\frac{d}{dw_j}[-y_i(w \cdot x_i + b)] = \\ &= -\frac{d}{dw_j} y_i(w_1 x_{i1} + \cdots + w_m x_{im} + b) = \\ &= -y_i x_{ji}\end{aligned}$$

Si nota pertanto come:

$$\frac{d}{dw_j} loss = \sum_{i=1}^{n} -y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$

Si aggiorna pertanto w_j:

$$w_j = w_j - \eta \sum_{i=1}^{n} -y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$

Questo viene fatto per ogni esempio x_i.

7.3.2 Learning algorithm del perceptron

Si nota pertanto come considerando il perceptron nell'ambito del gradient descent si aggiorna sempre il vettore dei pesi. Si noti come in questo caso η rappresenta il learning rate, y_i la label e $(w \cdot x_i + b))$

: `Perceptron()`

repeat
  foreach *training example* $(f_1, f_2, \ldots, f_n, label)$ **do**
    %`Label =` ± 1
    $prediction \ = \ b \ + \ \sum_{i=1}^{n} w_i f_i$
    foreach w_i **do**
      $w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) \ b \ = \ b \ + \ label$
until *Convergence*

la predizione. Questi generano una costante c

7.3.3 Costante c

Nella costante c se label e predizione hanno lo stesso segno quando gli elementi predetti aumentano gli aggiornamenti diventano minori. Se invece sono diversi più diversi lo sono, maggiore l'aggiornamento.

7.3.4 Gradiente

Il gradiente è il vettore delle derivate parziali rispetto a tutte le coordinate dei pesi:

$$\nabla L = \left[\frac{\delta L}{\delta w_1} \cdots \frac{\delta L}{\delta w_N} \right]$$

Ogni derivata parziale misura quanto veloce la perdita cambia in una direzione. Quando il gradiente è zero la perdita non sta cambiando in nessuna direzione.

: `i(`$\mathcal{F}$`, `K`, `η_1`, ...)`

GradientDescent
$z^{(0)} \ \rightarrow < 0, 0, \ldots, 0$%`Inizializza la variabile da ottimizzare`
for $k = 1$ **to** K **do**
  $g^{(k)} \ \rightarrow \ \nabla_z \mathcal{F}|_{z^{(k-1)}}$%`Computa il gradiente nella posizione corrente`
  $z^{(k)} \rightarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$%`scendi il gradiente`
return $z^{(k)}$