**Apprendimento e Incertezza**

# **Apprendimento probabilistico**

Il Problema principale nell’apprendimento è apprendere la distribuzione dei dati che è sconosciuta, per cui l’obiettivo che ci si pone è di riuscire ad approssimare nel migliore dei modi la distribuzione dei dati.

Quando la distribuzione dei dati è sconosciuta, è possibile effettuare la scelta del modello più probabile sulla base dei dati: considerato il dataset **E**, il modello **m** che massimizza la probabilità

**P(m | E)**

è detto modello della **massima probabilità a posteriori** (**MAP**), ottenuto tramite il teorema di Bayes che utilizziamo poiché calcolare la probabilità a posteriori può essere complicato e quindi sfruttiamo il teorema di Bayes

* P(E | m) → **verosimiglianza** (*likelihood*) del fatto che **E** sia stato generato da **m**, se è bassa allora vuol dire che ***m*** avrebbe predetto dati diversi, mentre se alta allora sta a significare che il modello predice dati pressappoco simili a quelli visti.
* P(m) → **probabilità a priori**, codifica il **learning bias (risulta essere il bias del modello)**, indicando i modelli più verosimili a priori. Viene usato per indurre a scegliere modelli più semplici (forma di regolarizzazione)
* P(E) → **funzione di partizione**, costante di normalizzazione indipendente dal modello, per cui può essere ignorata quando si sceglie il modello più probabile.

Il modello MAP massimizza il numeratore P(E | m) P(m) della formula data dal teorema di Bayes.

**Modello di massima verosimiglianza** (**ML**): è un’alternativa al modello MAP ed è dato dal modello che massimizza la verosimiglianza P(E | m). Qui cerchiamo di capire quale sia il modello che spiega meglio i dati osservati.

Problema: se lo spazio dei modelli è ricco, esiste ***m’***  tale che **P(E | m’) = 1**.

* Tale modello può essere molto poco probabile a priori
* Non andrebbe comunque escluso perché potrebbe essere quello vero e, se i dati sono sufficienti, potrebbe risultare anche il migliore

**Il modello ML corrisponde al modello MAP se viene assegnato uno spazio di probabilità a priori uniforme sui modelli.**

In generale, il rasoio di Ockham suggerisce di preferire modelli (ipotesi) più semplici rispetto ad altri più complessi.

*Nota: stimatore di massima verosimiglianza → derivando la log likelihood e impostandola a 0 per ottenere p nel caso della bernoulliana.*

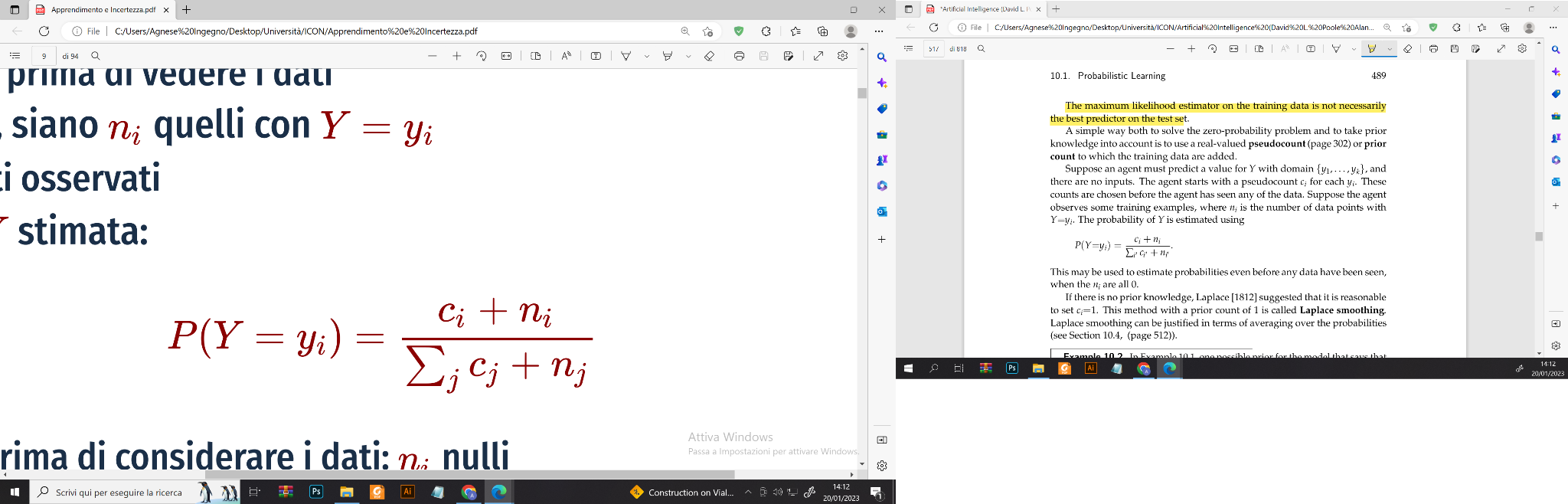
Lo stimatore ML sui dati di training non è necessariamente il miglior predittore sui dati di test:

* si pone il problema della probabilità nulla, nel caso in cui certo valore non si è verificato nel training set
* serve poter considerare conoscenza a priori

Per risolvere questi aspetti si utilizzano degli **pseudo-conteggi** (o *conteggi a priori*) a valori reali cui aggiungere i conteggi sui dati di training.

Per predire un valore Y, con dominio {y1, …, yk}

1. Prima che l’agente veda alcun dato, si inizializza uno pseudo-conteggio col valore ci per ogni yi
2. Dal training set, siano ni il numero di esempi osservati con Y=yi
3. La probabilità di Y viene stimata con



* Permette di poter stimare probabilità anche prima di aver osservato qualsiasi dato
* La **correzione di Laplace** consiste nell’impostare ci = 1, in modo tale che, anche in assenza di conoscenza pregressa, si possano ottenere probabilità diverse da 0 per valori che non occorrono nel training set.
  + La correzione di Laplace può essere giustificata in termini di *media delle probabilità*.

Per una bernoulliana, un modello a priori può essere P(p) = pc1 (1 − p) c0, per cui per massimizzare la probabilità a posteriori

**P(p | E) ∝ pc1+n1 (1 − p)c0+n0**

* una motivazione per l’utilizzo del modello a priori riguarda il fatto che stime di p = 0 oppure 1 non sembrano ragionevoli, per cui vengono esclusi
* il modello a priori costituisce una KB da usare in mancanza di dati oppure da integrare con i dati disponibili
* se la distribuzione a priori ha la stessa forma di quella a posteriori allora sono dette **coniugate**.

Per determinare lo pseudo conteggio appropriato, considerare “*avendo osservato un esempio con yi, di quanto aumenta il credito di yi*”

* non avendo mai osservato alcun esempio con yi, si riterrebbe yi impossibile: ci ← 0
* altrimenti, la proporzione da rispettare è (1+ci) : ci
* ci = 1 → osservando yi allora diventa due volte più probabile
* ci = 10 → 10% più probabile
* ci = 0.1 → 11 volte più probabile
* impostare i ci tutti uguali se non c’è alcun motivo per preferire un valore agli altri.

Grazie agli pseudo conteggi si introduce nella stima della probabilità, della distribuzione che vogliamo approssimare, della conoscenza di background la quale ci viene fornita dalla conoscenza dell’esperto del dominio. Per cui l’opinione dell’esperto è integrata con l’uso dei **pseudo-conteggi.**

**Opinione degli esperti**: gli pseudo-conteggi rappresentano l’integrazione delle opinioni degli esperti con i dati. Problematiche:

1. riluttanza a dare valori esatti di probabilità non modificabili in seguito
2. rappresentazione dell’incertezza: stima d’una probabilità
3. combinazione di numeri provenienti da più esperti
4. interazione tra opinione degli esperti

Se invece che fornire probabilità P(A), gli esperti fornissero una coppia di numeri interi <n, m>, dove n è il numero di occorrenze di A su m prove, allora sarebbe possibile effettuare una stima delle dimensioni dei dati (non sempre credibile) su cui si fonda il parere.

Mentre la proporzione riflette la probabilità, i diversi gradi di confidenza degli esperti si riflettono nei valori assoluti, ad esempio diversi modi per rappresentare la probabilità ⅔

* <2, 3> → bassa confidenza, tale conoscenza viene facilmente soppiantata da stime di altri esperti
* <20, 30> → maggiore confidenza
* <2000, 3000> → alta confidenza (anche centinaia di esempi non modificherebbero i conteggi a priori).

# **Classificatori probabilistici**

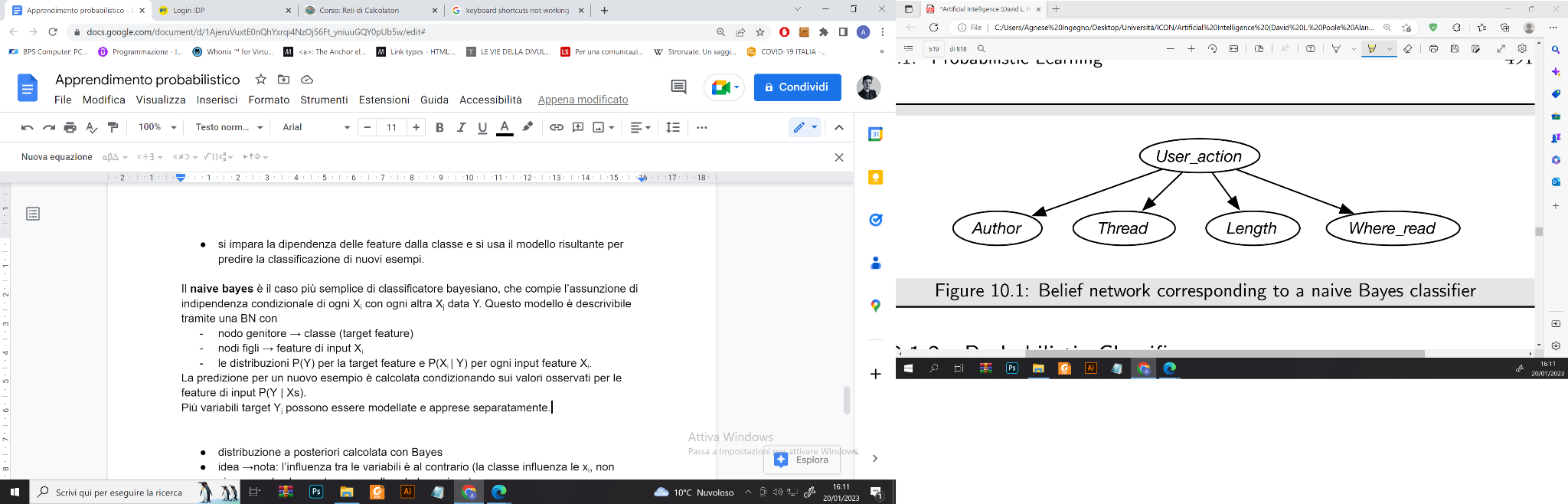
Un **Classificatore di Bayes** è un modello per la classificazione probabilistico, basato sull’idea che il ruolo di una classe influenza/predice i valori delle feature per i suoi membri. Quindi un classificatore bayesiano risulta un rete bayesiana dove la classe è il nodo genitore che influenza i figli che risultano essere le feature che caratterizzano gli esempi, per cui gli esempi dipendono dalla classe a cui appartengono ( e poiché dipendono ad una classe avranno caratteristiche tipiche di quella classe ed è per questo che dipendono da essa e si sfrutta tale dipendenza).

* gli esempi sono raggruppati in classi perché hanno valori comuni delle feature
* le classi sono anche dette **tipi naturali**
* si apprende la dipendenza delle feature dalla classe e si usa il modello risultante per predire la classificazione di nuovi esempi.

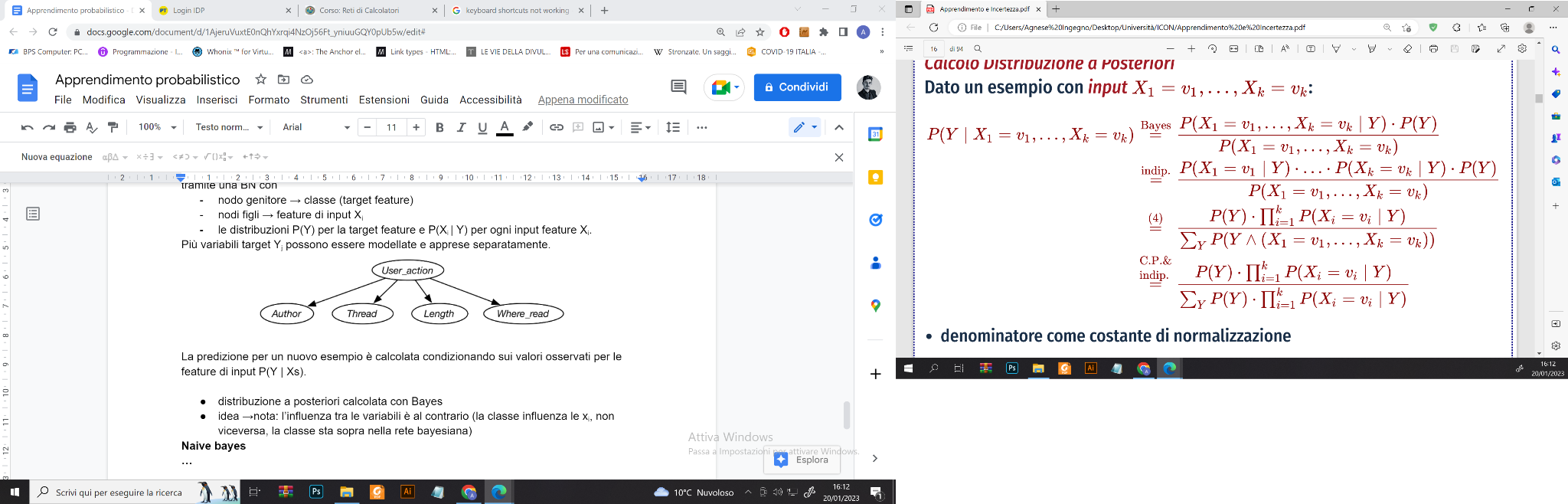
Il **naive bayes** è il caso più semplice di classificatore bayesiano, che compie l’assunzione di indipendenza condizionale di ogni Xi con ogni altra Xj data Y. Questo modello è descrivibile tramite una BN con

* nodo genitore → classe (target feature)
* nodi figli → feature di input Xi
* le distribuzioni P(Y) per la target feature e P(Xi | Y) per ogni input feature Xi.

Più variabili target Yj possono essere modellate e apprese separatamente.



La predizione per un nuovo esempio è calcolata condizionando sui valori osservati per le feature di input P(Y | Xs).



*denominatore: costante di normalizzazione*

**Osservazioni**

* Può trattare esempi con dati mancanti per alcuni attributi, condizionando solo sulle feature osservate
* è *ottimale* se Xi è l’unica feature osservata (per cui non si effettuano assunzioni di indipendenza) → al crescere del numero di Xi osservate, l’accuratezza dipende dalla reciproca indipendenza delle Xi data Y
* Se Xi sono tutte osservate, questo modello corrisponde alla regressione logistica (la probabilità è proporzionale a un prodotto, quindi il logaritmo è proporzionale a una somma)
* non considera le dipendenze tra variabili (diversamente dalla regressione lineare)

### **Apprendimento di classificatori bayesiani**

Per apprendere un classificatore, bisogna determinare dai dati le distribuzioni P(Y) e P(Xi | Y) per ogni i: P(Xi | Y) può essere trattato come l’obiettivo di un sotto-problema di apprendimento, uno per ogni valore che Y può assumere.

* Caso semplice (**Stima ML**): si usa la proporzione empirica osservata nei dati di training

**P(Xi = xi | Y = y) := #casi(Xi = xi ∧ Yi = y) / #casi(Y=y)**

I **casi di *probabilità nulle*** possono avere effetti indesiderati

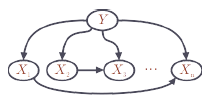
* alcune features diventano molto predittive → con un solo valore si può tagliare un’intera categoria (un insieme finito di dati potrebbe essere insufficiente a fornire l’evidenza necessaria per supportare una tale conclusione).
* alcune combinazioni di osservazioni possono risultare impossibili, portando a una divisione per zero nel classificatore
  + questo problema riguarda l’utilizzo delle frequenze empiriche come probabilità
  + si può risolvere il problema incorporando gli pseudo-conteggi.

***Conclusioni***

1. *classificatore NB dà in genere buoni risultati*
2. *è facile da implementare e funziona bene quando l’assunzione di indipendenza è appropriata e anche se l’assunzione non è rispettata non si discosta di molto dal ottenere buoni risultati.*

**Estensioni del naive bayes**: modelli in cui alcune Xi possono essere genitori di altre features purché non si instaurino dei cicli, per cui Y ovvero la classe sarà genitore di tutte le features prese in considerazione e in aggiunta ci sarà tra queste features qualcuna che sarà anche essa genitore di una delle features prima considerate purché la struttura della rete Bayesiana risulti essere un DAG

* la probabilità della classificazione data i genitori può essere rappresentata come un albero di decisione, una funzione lineare appiattita o una rete neurale
* Le figlie di Y non devono essere necessariamente indipendenti
  + **tree augmented naive Bayes (TAN) network**, oltre alla classe, le Xi possono avere un solo altro genitore (purché la rete sia aciclica), considerando interdipendenze tra gli Xi

****

* + **Latent Tree**: decompone Y in un numero di variabili latenti, connesse ad albero (ogni variabile osservata è figlia di una variabile latente). Le variabili latenti consentono di avere un modello di dipendenza tra le Xi osservate.

### **Apprendimento MAP di alberi di decisione**

Nell’apprendimento di alberi di decisione serve un bias, tipicamente in favore degli alberi più piccoli, imposto attraverso la distribuzione a priori sui modelli.

In generale, più alberi possono adattarsi perfettamente ai dati di training e, se c’è rumore nel dataset, nessuno di questi potrebbe essere il migliore: conviene scegliere anche tra quelli che si adattano meno perfettamente, per cui si può utilizzare l’**apprendimento MAP**.

* Alberi *m* che si adattano accuratamente ai dati → P(E | m)=1
* la preferenza per uno di essi dipende dalla probabilità a priori, la quale incorpora il **learning bias**: alberi semplici devono avere una probabilità a priori più alta.

Il teorema di bayes fornisce un modo per contemperare semplicità e abilità nel trattare il rumore: si creano alberi le cui foglie presentano probabilità

* alberi più grandi possono modellare regolarità casuali (date dal rumore) presenti nei dati di training
* la likelihood favorisce gli alberi più grandi, la distribuzione a priori quelli più piccoli

### **Lunghezza delle descrizioni MDL**

Passando al logaritmo e negando il criterio MAP otteniamo:

**(-log2 P(E | m)) + (-log2 P(m))**

In termini di teoria dell’informazione, il primo termine è il numero di bit per descrivere i dati dato il modello m, il secondo termine è il numero di bit per descrivere il modello m. Un modello che minimizza questa somma è detto **modello di minima lunghezza di descrizione** (*minimum description length model -* **MDL**).

**Principio MDL**: scegliere il modello che minimizza il numero di bit necessari a descrivere il modello e i dati forniti ad esso (prima va comunicato questo e in seguito i dati in termini del modello). ***Per la monotonia del logaritmo, il modello MAP coincide con quello MDL.***

***Quello che il criterio della minima lunghezza descrittiva ci dice è che dobbiamo preferire quei modelli che riescono a classificare correttamente gli esempi con il minor quantitativo di informazione compattata.*** *Questo criterio equivale a quello della MAP dove li cerchiamo di massimizzare la probabilità a posteriori, qui poiché usiamo i logaritmi cerchiamo di minimizzare la funzione per ottenere la minore lunghezza descrittiva la quale equivale a massimizzare la probabilità a posteriori. Per minimizzare la lunghezza descrittiva serve avere il modello più semplice il quale è da preferire al modello più complesso e qui ritroviamo la teoria del rasoio di Occham che ci dice di preferire i modelli più semplici a quelli complessi giustificata dalla teoria del minimum description length.*

*Questi principio si può utilizzare nell’apprendimento delle reti Bayesiane per andarne a minimizzare la complessità.* La complessità della struttura di una rete bayesiana la si può misurare in termini di misura dell’informazione tenendo conto di quanto in precedenza è stato detto sul principio del MDL.

Serve definire un codice per descrivere un albero di decisione: questo in generale è difficile e spesso si ricorre ad una misura approssimata della complessità di un modello, ove per risalire a questa misura approssimata si sfrutta quanto abbiamo detto sulla teoria del minimum description length, per cui avendo:

* |m|: numero di parametri del modello
  + albero (con probabilità alle foglie) → |m| è il numero di foglie
  + funzione lineare o rete neurale → |m| è il numero di parametri numerici
* |E|: numero di esempi del training
  + al più |E|+1 diverse probabilità da distinguere nel modello
  + servono quindi log2(|E| + 1) bit

L’approssimazione del MDL da minimizzare è detto indice **Bayesian information criteria** (**BIC**) ed è calcolata come segue e serve a dare una buona approssimazione del MDL e quindi data l’equivalenza con la MAP stiamo dando una buona approssimazione della probabilità a posteriori del modello che abbiamo:

**- log2 P(E | M) + |m| log2 (|E|)**

# **Apprendimento non Supervisionato**

Prevede un training set senza feature-target, l’obiettivo è (ri)costruire una classificazione naturale dei dati in cui la **feature-obiettivo è *latente***.

Un metodo generale per l’apprendimento non supervisionato è il **clustering**, che prevede il partizionamento degli esempi in classi, dette **cluster**, che predicono i valori delle feature per gli esempi contenuti. Il clustering può essere visto come un problema di ottimizzazione che cerca di minimizzare la funzione di errore.

Ogni raggruppamento (**sistema di cluster**) ha un **errore di predizione** associato, per cui il clustering migliore è quello che minimizza tale errore.

* **hard clustering**: ogni esempio viene assegnato a una sola classe precisa, che può essere usata per predire i valori delle feature dell’esempio.
* **soft clustering**: ogni esempio ha una distribuzione associata sulle classi e le predizioni dei valori delle feature è costituita dalla media pesata (dalla probabilità) delle predizioni date le classi, per cui ogni esempio avrà diverse percentuali di probabilità di appartenere alle varie classi in gioco.

# **K-means**

È un algoritmo iterativo di hard clustering che converge verso minimi locali e per questo necessita di più run per trovare il minimo globale, l’esecuzione di più run è giustificata dal primo passo casuale che questo algoritmo effettua per la costrizione e inizializzazione dei k centroidi che costituiranno i centroidi delle k classi o raggruppamenti che esso andrà a formare e saranno aggiornati nei successivi passi, a seconda della scelta iniziale si possono avere variazioni nei risultati di clusterizzazione ottenuti e per ottimizzare si usa il random restart per trovare la clusterizzazione ottimale:

* **input**: esempi di training e numero di cluster *k*
  + si assume che il dominio di ogni feature sia cardinale, in modo tale che abbia senso considerare differenze di valori
* **output**: k cluster, ogni esempio di training è assegnato a un cluster
  + viene anche fornita la predizione di ogni feature per ogni cluster

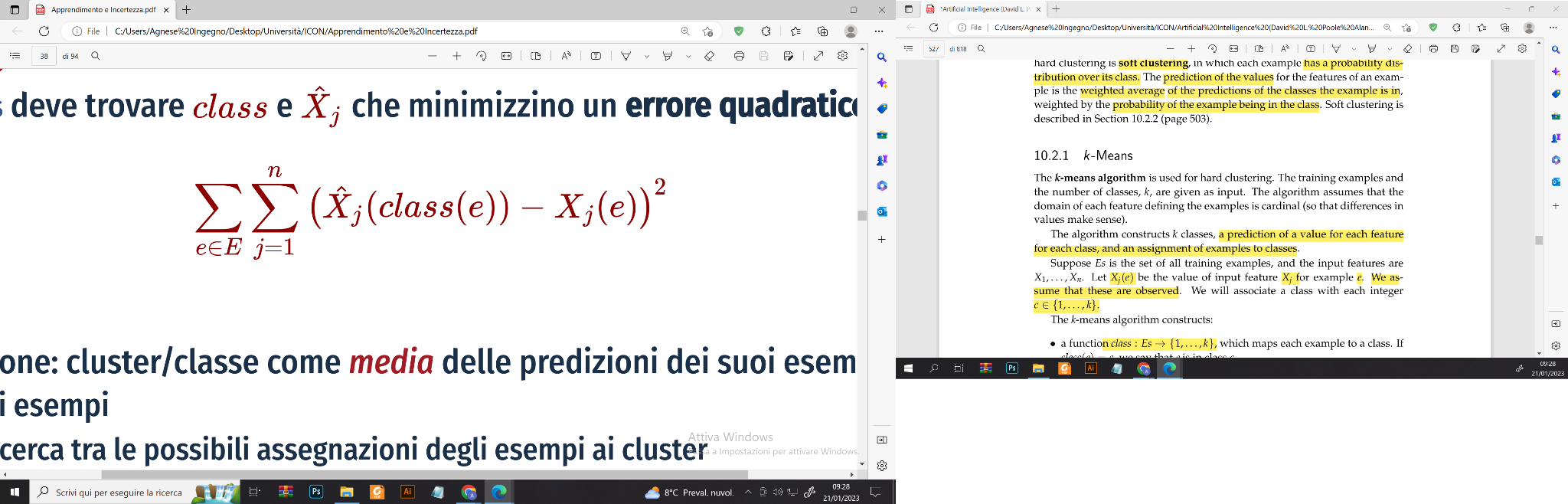
Dato il training set E, con X1, …, Xn feature di input

* sia Xj(e) il valore di Xj per e in E
* ogni classe è c in {1, …, k}

Il k-means costruisce

1. La funzione *class*: E → {1, …, k}, che assegna una classe (cluster) a ogni esempio
2. Per ogni feature Xj, una funzione Xpredj : {1, …, k} → dom(Xj). Il valore Xpredj(class(e)) è la predizione del valore di Xj per l’esempio e.

**Obiettivo**: trovare queste funzioni in modo tale che minimizzino **l’errore quadratico** ove questo errore quadratico che si calcola con la sottostante formula è l’errore che si commette quando andiamo ad associare un esempio visto di training ad uno dei centroidi di cui disponiamo e il calcolo dell’errore quadratico viene fatto feature per feature, ovvero si calcola la differenza tra i valori che assumono le feature dell’esempio visto rispetto ai valori che assumono i centroidi che il k-means sta costruendo a tempo di running:



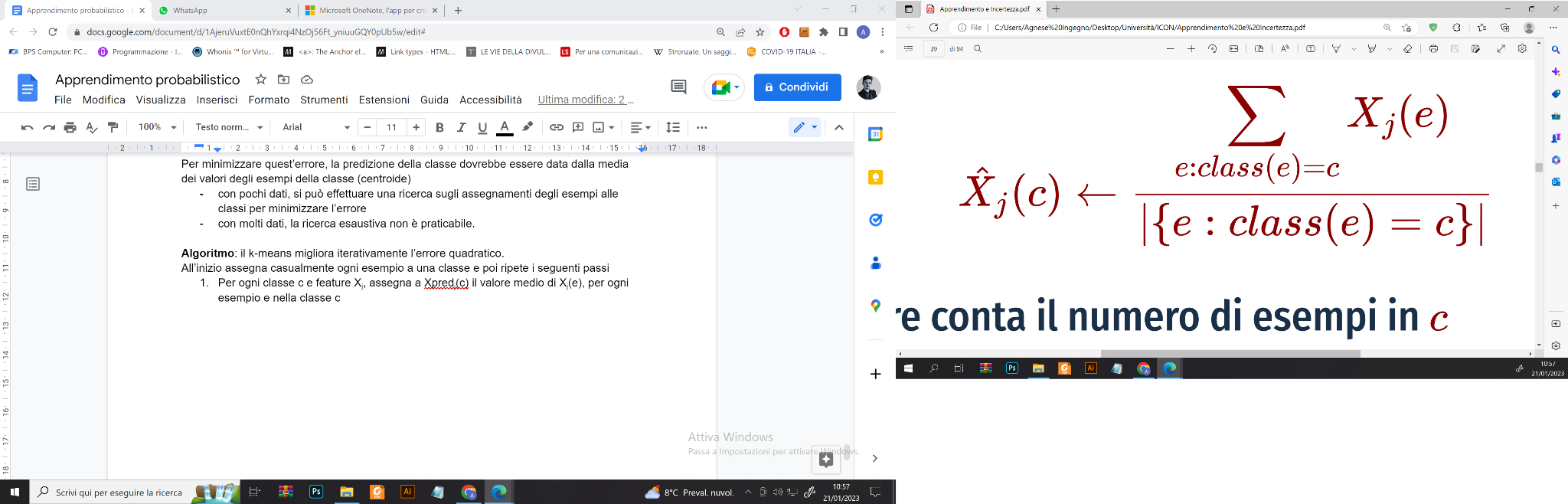
Per minimizzare quest’errore, la predizione della classe dovrebbe essere data dalla media dei valori degli esempi della classe (centroide)

* con pochi dati, si può effettuare una ricerca sugli assegnamenti degli esempi alle classi per minimizzare l’errore
* con molti dati, la ricerca esaustiva non è praticabile.

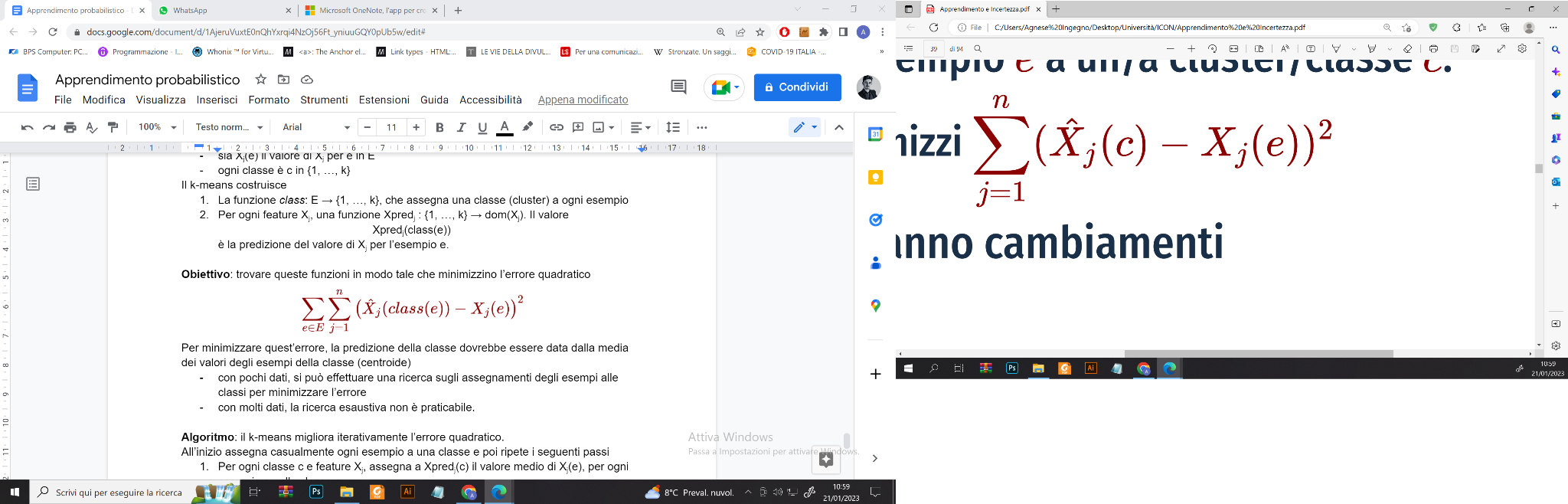
**Algoritmo**: il k-means migliora iterativamente l’errore quadratico.

All’inizio assegna casualmente ogni esempio a una classe e poi ripete i seguenti passi

1. Per ogni classe c e feature Xj, assegna a Xpredj(c) (*nota: scritto X-hat nell’img*) il valore medio di Xj(e), per ogni esempio e nella classe c



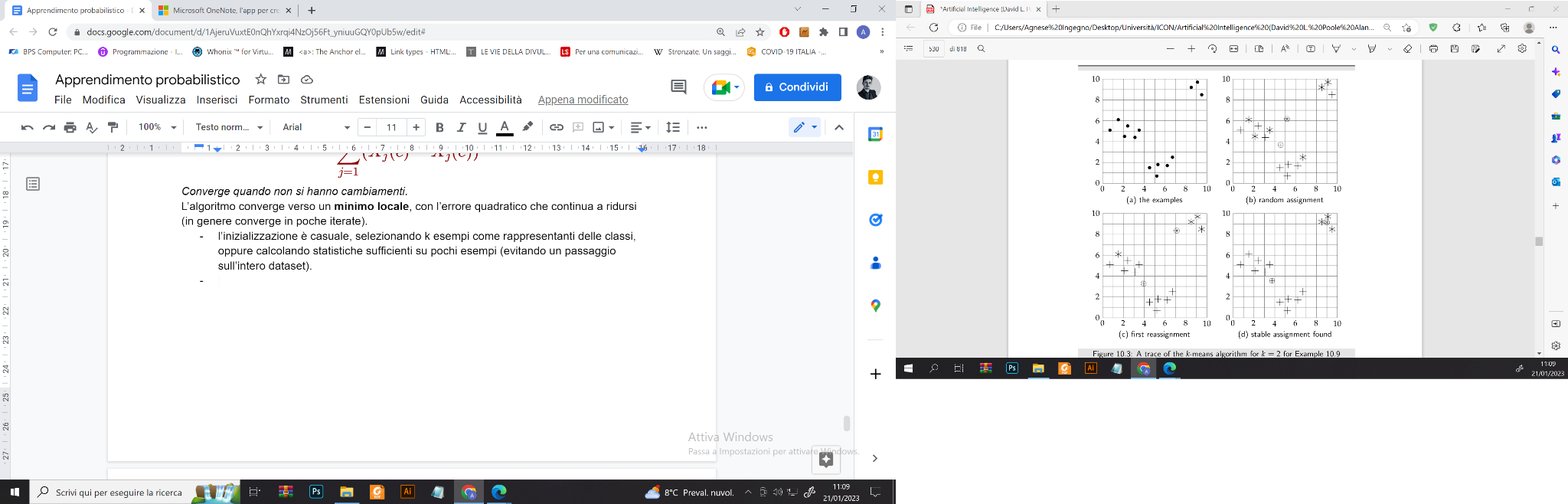
1. Riassegna ogni esempio e alla classe c che minimizzi

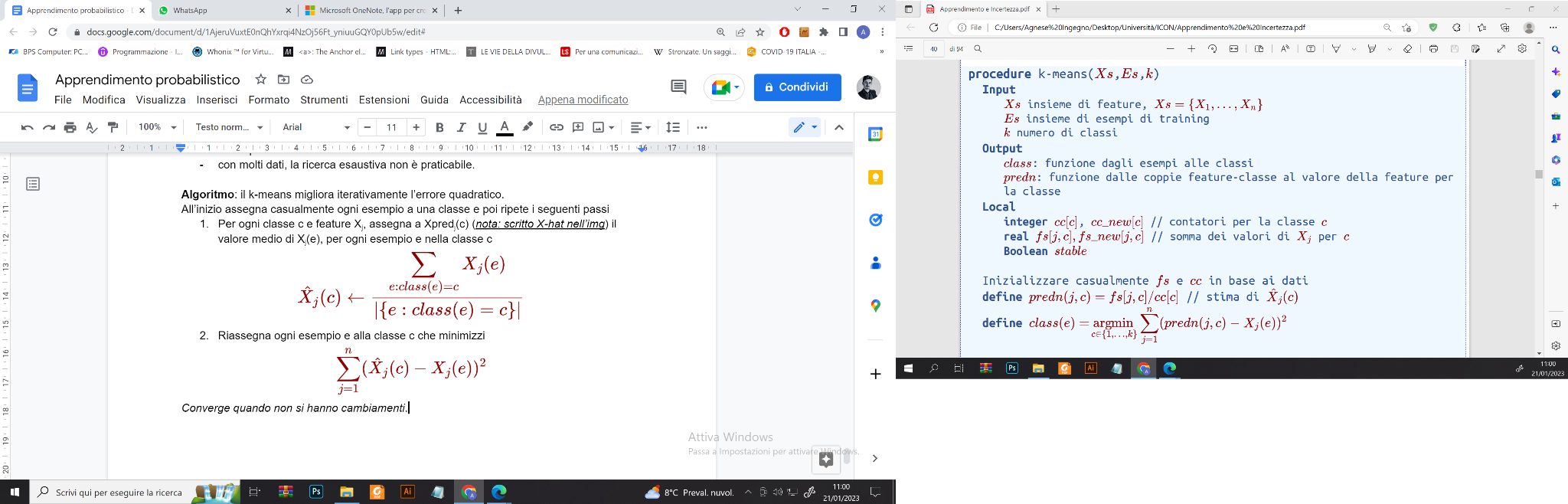


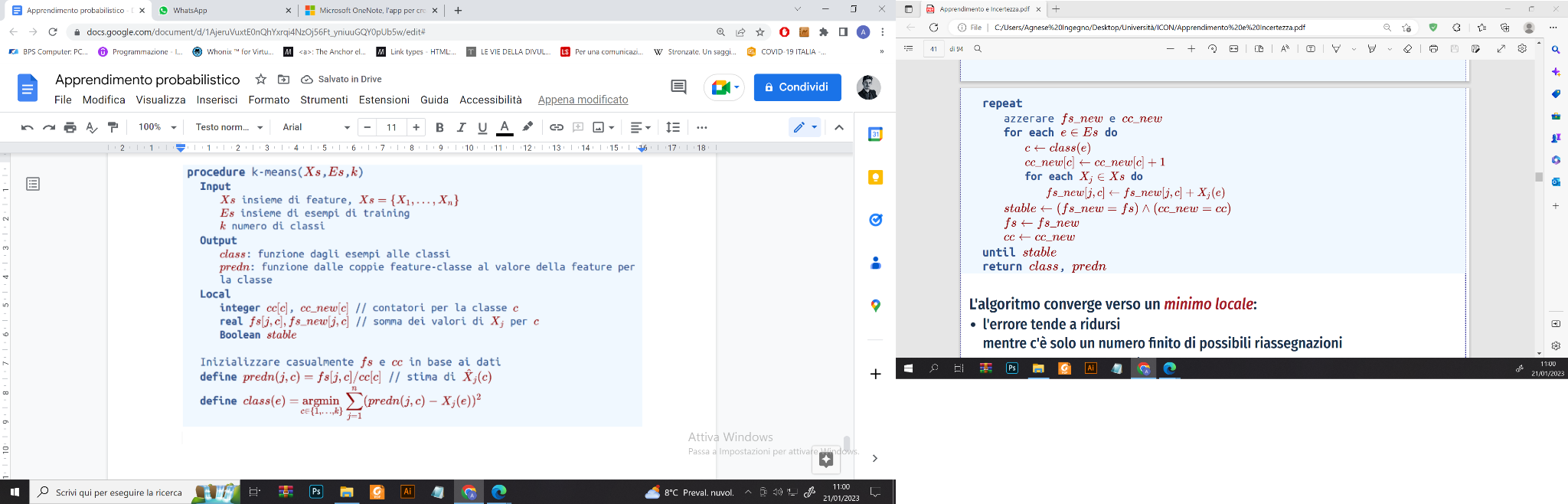
*Converge quando non si hanno cambiamenti*.

L’algoritmo converge verso un **minimo locale**, con l’errore quadratico che continua a ridursi (in genere converge in poche iterate).

* L’inizializzazione è casuale, selezionando k esempi come rappresentanti delle classi, oppure calcolando statistiche sufficienti su pochi esempi (evitando un passaggio sull’intero dataset).
* Differenti assegnazioni iniziali possono portare a diversi clustering
* Per trovare il migliore clustering, è spesso utile provare diverse inizializzazioni (**random restart**) e scegliere l’assegnazione stabile con il minimo errore.







**Problemi**

* L’algoritmo è sensibile alla scala relativa delle dimensioni, per cui i valori andrebbero riscalati per poter essere confrontati, per cui è essenziale standardizzare o normalizzare l’intero dataset prima di adoperare il k-means.
* Per trovare il numero appropriato di classi k, si effettua la ricerca al variare di k:
  + all’aumentare di k, gli errori saranno sempre più bassi
  + il k appropriato risulta essere il valore che porta a una grande riduzione dell’errore rispetto al precedente e a una piccola riduzione ulteriore dell’errore per il k successivo, e questo passaggio lo si può notare plottando un grafico con asse y errore quadratico commesso dal k-means e sull’asse numero k di raggruppamenti che si vogliono ottenere, quando il grafico curva simulando la figura di un gomito quello è il k migliore per la clusterizzazione tale procedura prende il nome di elbow-method.

**[il k migliore è dato dal elbow method]**

**Ricordiamo che i metodi per valutare una clusterizzazione come efficiente o meno sono l’elbow-method e il silhouette score.**

# **Expectation maximization EM e Soft Clustering**

Una **variabile latente** è una variabile aleatoria che non è osservata nel dataset: un classificatore Bayesiano può servire da base per l’apprendimento non supervisionato, rendendo la classe una variabile latente.

L’**expectation maximization** (**EM**) **algorithm** permette di apprendere modelli probabilistici con variabili latenti

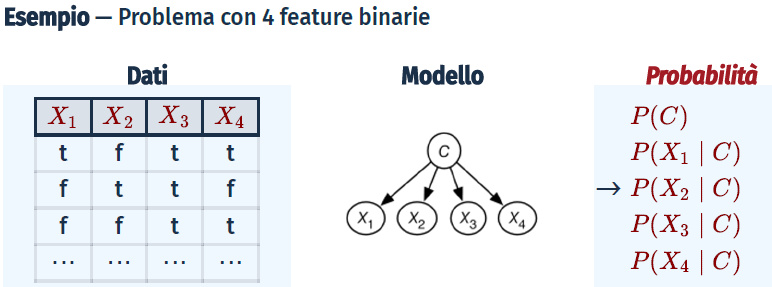
* combinato con *naive Bayes classifier*, effettua soft clustering
* è analogo al k-means, ma gli esempi sono assegnati probabilisticamente alle classi

Agisce costruendo un modello uguale al naive bayes per la classificazione e lo fa per ottenere una distribuzione di probabilità sulle classi ove una volta ottenuta questa distribuzione cercherà di massimizzare la ML o MAP come proprio come farebbe un classificatore Bayesiano per andare a determinare le probabilità di appartenenza ad una classe.

Dato E ovvero gli esempi di training, costruisce un modello NB con

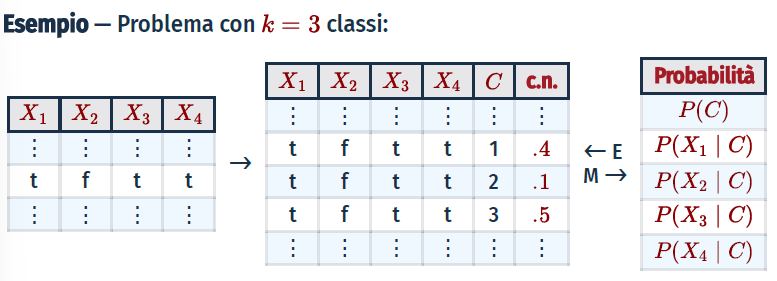
* **nodo genitore** → variabile latente C della classe, con dominio {1, …, k}
* **nodi figli** → feature Xi dei dati osservati in E

basato sulle distribuzioni P(C) e P(Xi | C) che si adattino meglio a E.



Innanzitutto, viene esteso il dataset con una nuova feature “C” e una colonna “count” tramite un **meccanismo di augmentation** che va a creare queste nuove due colonne virtuali.

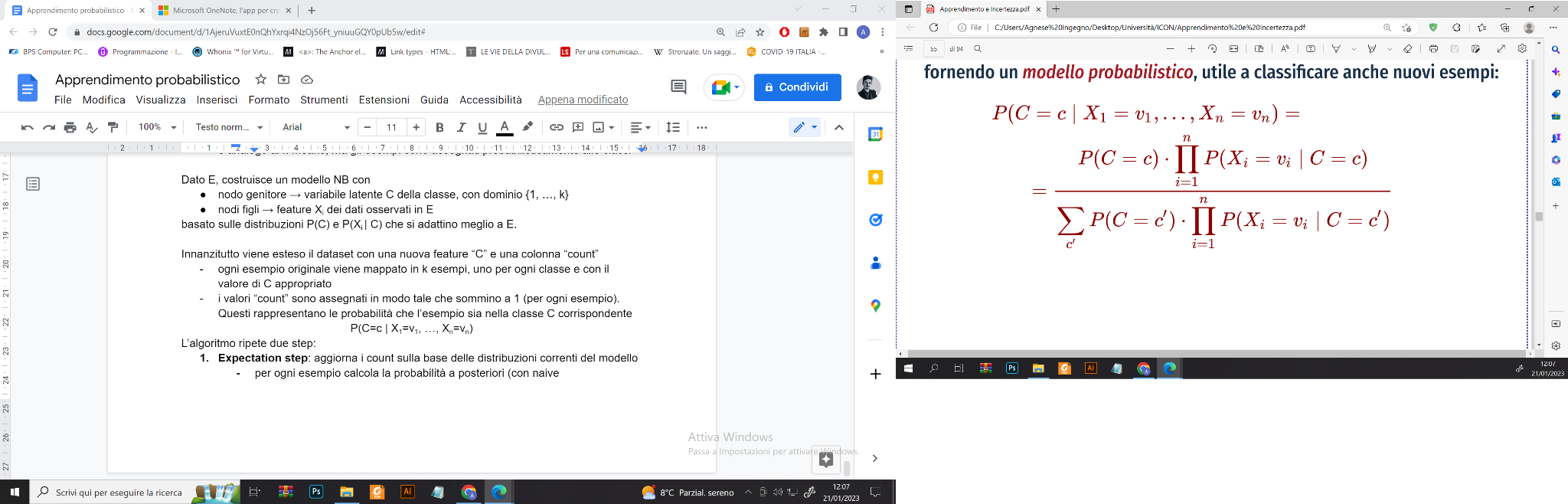
* ogni esempio originale viene mappato in k esempi, uno per ogni classe e con il valore di C appropriato
* i valori “count” sono assegnati in modo tale che sommino a 1 (per ogni esempio) ovvero **sono contatori normalizzati.**
  + corrispondono alla probabilità a posteriori P(C=c | X1=v1, …, Xn=vn).



L’algoritmo ripete due step iterando fino a convergere:

1. **Expectation step**: aumenta i **count** sulla base delle distribuzioni correnti del modello

* per ogni esempio calcola la probabilità a posteriori (*naive bayes*)



e il campo count associato all’esempio, con C=c, viene aggiornato con il valore P(C=c | X1=v1, …, Xn=vn).

Questo step richiede di effettuare inferenza probabilistica (calcola valori attesi, per questo è detto **expectation step**).

1. **Maximization step**: inferisce le probabilità per il modello dai dati estesi

* dato che sono presenti esempi con valori per tutte le variabili, **l’apprendimento delle probabilità avviene come in un classificatore *naive bayes***
* è uno step di massimizzazione perché calcola lo stimatore di massima verosimiglianza o della massima probabilità a posteriori

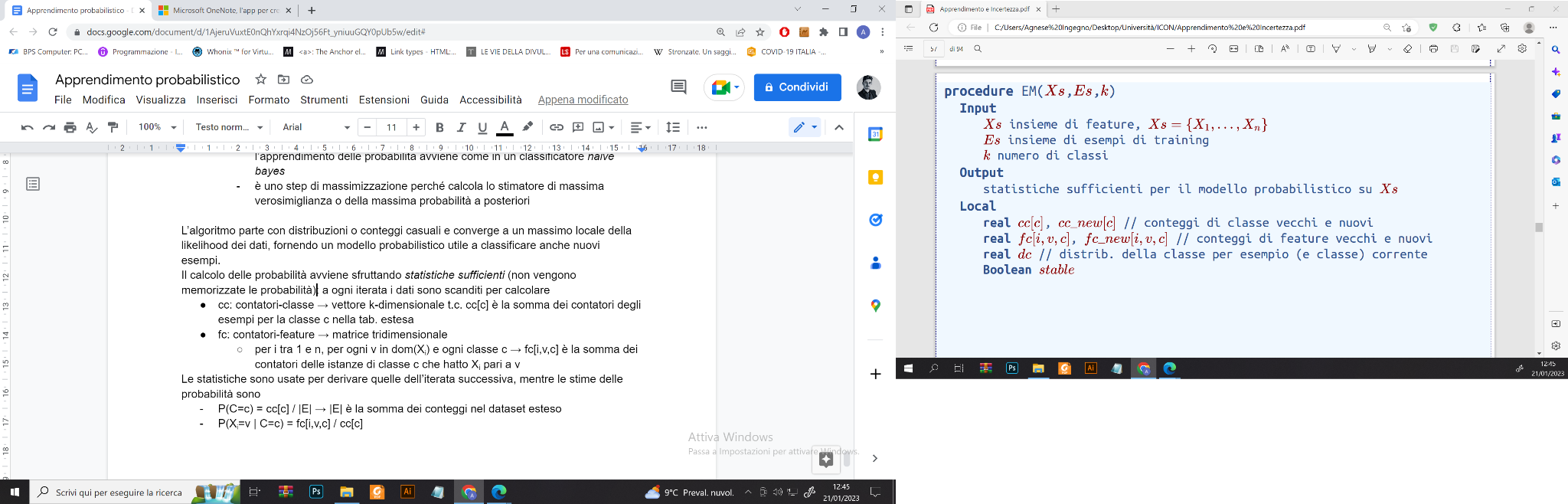
**L’algoritmo parte con distribuzioni o conteggi casuali e converge a un massimo locale della likelihood dei dati**, fornendo un modello probabilistico utile a classificare anche nuovi esempi.

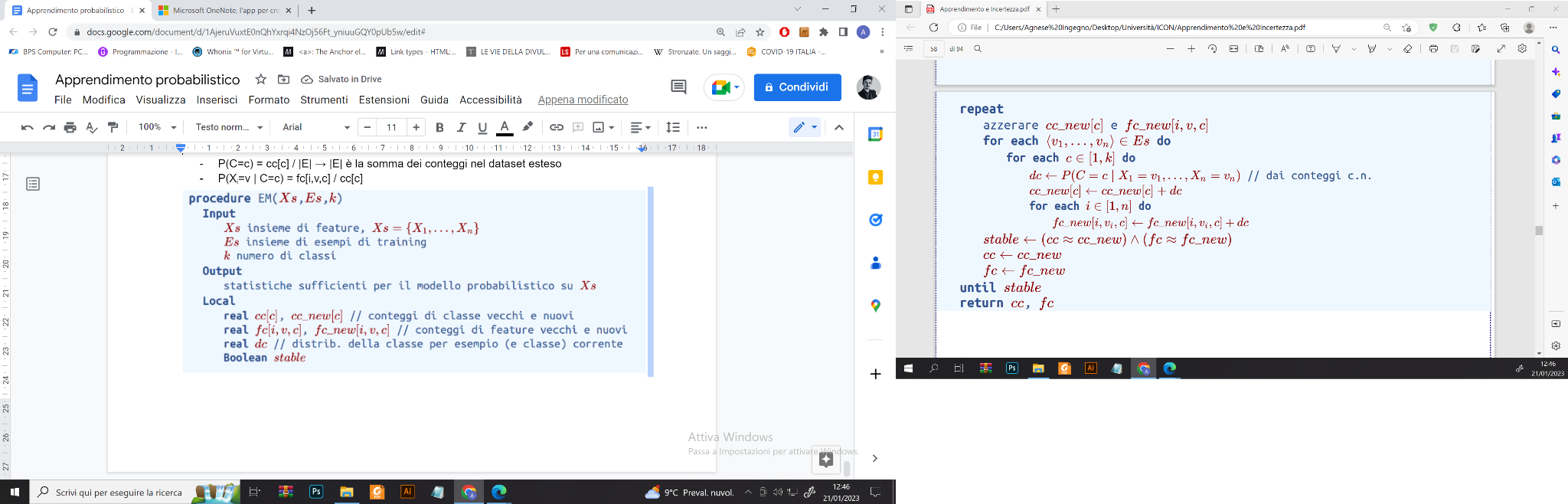
Il calcolo delle probabilità avviene sfruttando *statistiche sufficienti* (non vengono memorizzate le probabilità): a ogni iterata i dati sono scanditi per calcolare

* **cc: contatori-classe** → vettore k-dimensionale t.c. cc[c] è la somma dei contatori degli esempi per la classe c nella tab. estesa
* **fc: contatori-feature** → matrice tridimensionale
  + per i tra 1 e n, per ogni v in dom(Xi) e ogni classe c → fc[i,v,c] è la somma dei contatori delle istanze di classe c che hatto Xi pari a v

Le statistiche sono usate per derivare quelle dell’iterata successiva, mentre le stime delle probabilità sono

* P(C=c) = cc[c] / |E| → |E| è la somma dei conteggi nel dataset esteso
* P(Xi=v | C=c) = fc[i,v,c] / cc[c]





*Osservazioni*

* per valutare *dc* ci si basa su *cc* e *fc*, l’inizializzazione non è indicata nell’algoritmo ma si può partire con una distribuzione casuale nella prima iterata
* **criteri di stop**: se *cc* e *fc* cambiano poco dopo un’iterata (soglia regolabile, *convergenza*) oppure si fissa un numero di iterazioni.
* similarità con k-means:
  + passo E → assegna gli esempi alle classi (*probabilisticamente*)
  + passo M → (ri)determina le predizioni basate sulle classi

**Problema**

se K > 1 allora EM ha pressoché sempre massimi locali *multipli*: con qualsiasi permutazione delle etichette delle classi un massimo è ancora un massimo locale. Per trovare il massimo globale, si possono effettuare più *restart* e scegliere il modello con log-likelihood minimale.

# **Apprendimento di Belief Network**

È difficile per l’esperto fornire modelli accurati di Belief Network, spesso si vuole imparare una rete bayesiana dai dati.

Si possono usare diverse varianti di metodi per apprendere BN, sulla base della disponibilità di conoscenza a priori e della completezza del dataset.

* caso più semplice → la struttura è nota, per cui solo le probabilità condizionali devono essere imparate dai dati
* caso più difficile → le variabili o la struttura della rete non sono note e ci possono essere dati mancanti (senza assunzione *missing at random*).

### **Imparare le probabilità**

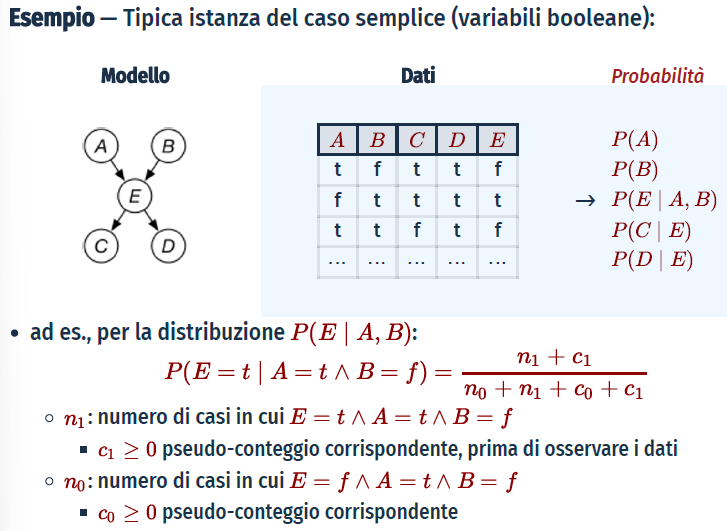
Nel caso semplice, l’obiettivo è imparare per ogni Xi la distribuzione P(Xi | parents(Xi))

* è una forma di apprendimento supervisionato con Xi feature target e parents(Xi) le feature di input
* per casi con pochi genitori, ogni probabilità condizionata viene appresa separatamente usando gli esempi di training e la conoscenza pregressa (come pseudo-conteggi) → si sfruttano le frequenze empiriche
  + per calcolare P(A=t | B=f) si usa

**P(A=t and B=f) / (P(A=t and B=f) + P(A=f and B=f))**

combinando le frequenze empiriche con la conoscenza pregressa

* se una variabile ha molti genitori, usare tale metodo porta spesso all’overfitting (in particolar modo quando ci sono pochi esempi per alcune combinazioni di variabili). In tal caso, utilizzare tecniche di apprendimento supervisionato
  + *alberi di decisione* per variabili discrete
  + *regressione logistica* e *reti neurali* per rappresentare probabilità condizionali di variabili binarie (dati i genitori).
  + per variabili discrete non binarie → si possono usare variabili indicatrici.



**!WARINING!** si può verificare overfitting utilizzando conteggi e pseudo conteggi specialmente quando una variabile ha molti genitori e si hanno pochi esempi di training a disposizione.

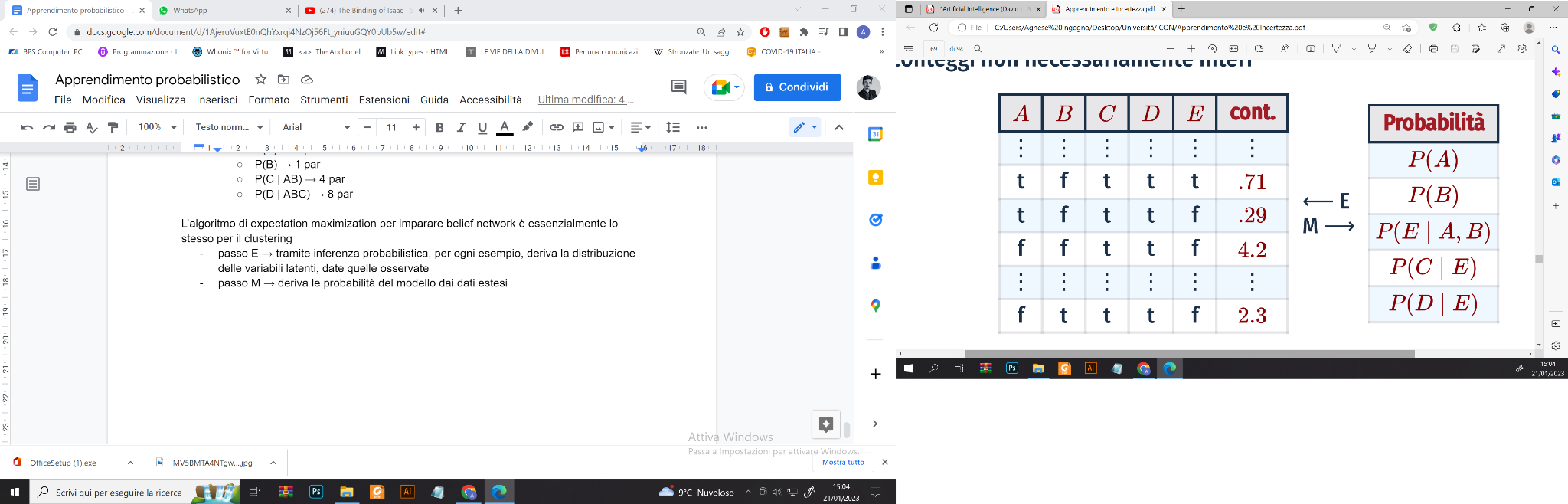
### **Variabili latenti**

Variabili non osservate per nessun esempio di training sono dette **latenti** (nella tabella non ci sono colonne per tali variabili): senza aggiungere queste variabili nel modello ci sarebbero più parametri, con un conseguente rischio maggiore di overfitting.

* data la rete bayesiana nell’immagine, un modello che include E (variabile latente) avrà i seguenti parametri
  + P(A) → 1 probabilità da imparare
  + P(B) → 1 probabilità da imparare
  + P(E | A, B) → 4 probabilità da imparare
    - P(E=t | A=t, B=t) e per le altre 3 combinazioni di valori per A e B
  + P(C | E) → 2 probabilità da imparare
    - P(C=t | E=t) e P(C=t | E=f)
* escludendo E dal modello, i parametri saranno invece 14
  + P(A) → 1 par
  + P(B) → 1 par
  + P(C | AB) → 4 par
  + P(D | ABC) → 8 par

L’algoritmo di expectation maximization per imparare belief network è essenzialmente lo stesso per il clustering

* passo E → tramite inferenza probabilistica, per ogni esempio, deriva la distribuzione delle variabili latenti, date quelle osservate
* passo M → deriva le probabilità del modello dai dati estesi



### **Dati mancanti**

Oltre al caso delle variabili latenti, i dati possono essere incompleti per altre cause:

* per alcune variabili in qualche tupla possono esserci valori mancanti
* serve cautela nell’uso di tali esempi: la mancanza potrebbe essere correlata con il fenomeno di interesse.
* È possibile affrontare la mancanza di tali dati con metodi di data mining per il loro completamento tramite un imputatore oppure scartare tali esempi che presentano la mancanza di dati.

L’assunzione **dati mancanti in modo casuale** (*missing-at-random*) prevede che la ragione del perché mancano i dati non è correlata ad alcuna variabile del modello

* in tal caso i dati mancanti possono essere ignorati o completati con algoritmi come EM
* è un’assunzione forte, per cui è consigliato indagare sul perché i dati mancano ed eventualmente costruire un modello che spieghi la mancanza dei dati.

### **Apprendimento della struttura**

Il caso di dati completi, nessuna variabile nascosta ma struttura della BN non disponibile è quello in cui serve **apprendere la struttura**. Gli approcci principali sono due

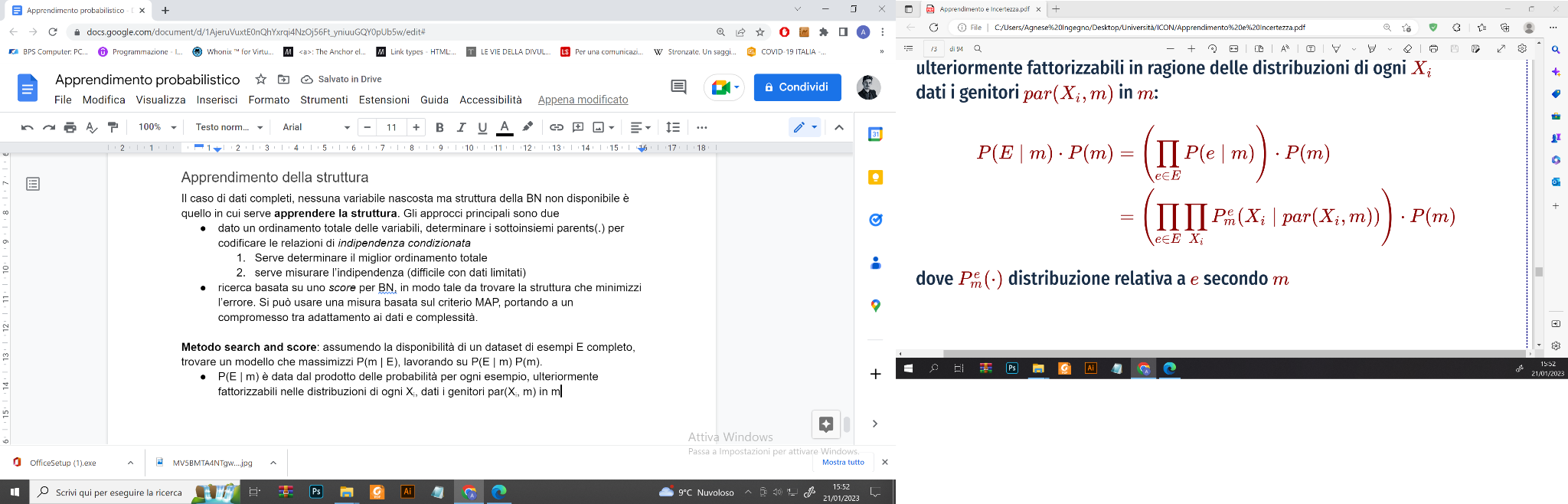
* dato un ordinamento totale delle variabili, determinare i sottoinsiemi parents(.) per codificare le relazioni di *indipendenza condizionata*

1. Serve determinare il miglior ordinamento totale
2. serve misurare l’indipendenza (difficile con dati limitati)

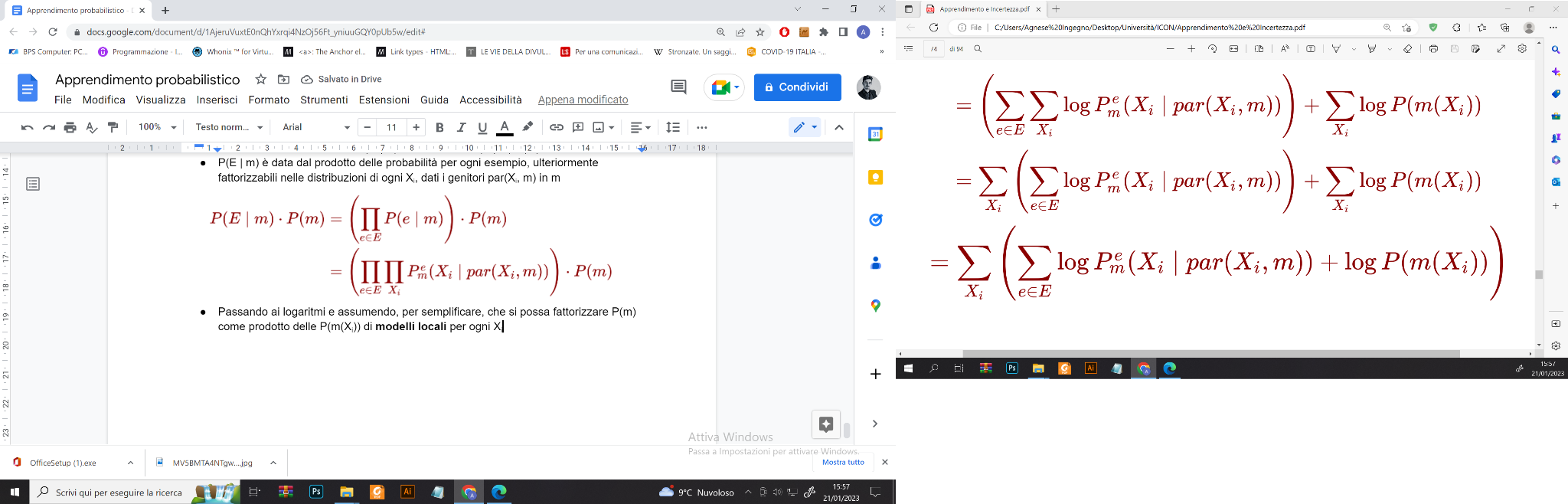
* ricerca basata su uno *score* per BN, in modo tale da trovare la struttura che minimizzi l’errore. Si può usare una misura basata sul criterio MAP, portando a un compromesso tra adattamento ai dati e complessità.

**Metodo search and score**: assumendo la disponibilità di un dataset di esempi E completo, trovare un modello che massimizzi P(m | E), lavorando su P(E | m) P(m).

* P(E | m) è data dal prodotto delle probabilità per ogni esempio, ulteriormente fattorizzabili nelle distribuzioni di ogni Xi, dati i genitori par(Xi, m) in m



* Passando ai logaritmi e assumendo, per semplificare, che si possa fattorizzare P(m) come prodotto delle P(m(Xi)) di **modelli locali** per ogni Xi



*Conclusione:* obiettivo da massimizzare separatamente su ogni variabile, assumendo la BN aciclica.

* una volta trovato un buon ordinamento totale per le variabili (tramite una ricerca locale o branch-and-bound), ci sono sottoproblemi di apprendimento supervisionato da risolvere indipendentemente
* serve determinare le distribuzioni Pme(Xi | par(Xi, m)) (*sottoproblemi supervised*).
* P(m(Xi)) è approssimabile come nel BIC score.

**Caso generale**: se la struttura è incognita, ci sono variabili latenti e dati mancanti

1. Per quanto riguarda questi ultimi si ripropone quanto detto prima
2. *complessità computazionale* → lo spazio è troppo esteso per provare le combinazioni possibili di ordinamenti e variabili latenti.

Come possibili soluzioni:

1. Selezionare il miglior modello (secondo il criterio MAP)
2. Effettuare la predizione mediando su tutti i modelli → in questo modo si forniscono migliori predizioni ma l’output risulta più difficile da spiegare/giustificare.