Dalla prima alla quarta lezione sono lezioni introduttive che ai fini dell'esame non servono

1) PRIMA LEZIONE solo introduttiva, nessun appunto

2) SECONDA LEZIONE

Data / Dato -> set di valori di variabili che possono essere di due tipi:

Dati Qualitativo -> attributo, caratteristica che puo essere osservata e generalmente non misurabile con un numero (colore, sesso, tutti i tipi di diversi caratteri di una persona(generoso, vivace))

Dati Quantitativo -> Valutata come entità DISCRETA, un numero (età, altezza)

Incertezza -> Qualità di un processo che non ci dà la sicurezza di quello che succederà (utilizziamo i dati al fine di prevedere con perfetta accuratezza, ma essendo sempre una predizione essa gode di incertezza (gioco dei pacchi)). Non c'è sempre incertezza nelle predizioni come ad esempio se dobbiamo descrivere la velocità di oggetti utilizzando le leggi di newton, siamo sicuri del risultato che otterremo.

Deterministico -> Fenomeno che non possiamo prevedere. (Il lancio della monetina, se so vento, forza etc. so al 100% come andrà a cadere, ma il calcolo di tutte queste previsioni è veramente difficile).
Noi dobbiamo pensare in maniera stocastica (probabilistico)

Stocastico/Probabilistico -> Fenomeno che possiamo in qualche modo andare a prevedere tramite calcoli e formule matematiche.

Teoria delle probabilità -> è quella teoria che si occupa di lavorare con eventi incerti. Ci permette di quantificare l'incertezza tramite un set di assiomi.

Variabile Scalari -> assume un solo valore

Variabili multidimensionali -> assume piu valori; Possiamo o avere più variabili che assumono piu valori oppure una variabile singola che racchiude tutti i valori es. x=3, y=5, z=7 oppure data=[3,5,7] oppure anche x=1 scalare, x=1,3,5 multi-dimensionale.

Variabile Continua -> assume tutti i valori in un determinato intervallo andando a "riempire". [0,1] allora sono tutti i valori 0,1 0,22 0,99 0,2 1 0,11111 0,222 etc. Considerando l'altezza di una classe di scuola, [1.60;1.90], studente=1.75 è una variabile continua in quanto puo prendere tutti i valori dell'intervallo.

Variabile Discreta -> Sono invece un insieme finito di valori, non contigui tra loro. [0,10] sono i valori 0 1 2 3 4 etc. (monetina, variabile discreta (1-0); lancio dado, variabile discreta(1-2…-6)).

Random Variabile -> variabili i cui valori dipendono da un fenomeno causale/stocastico. L'insieme di possibili valori assumibili è detto **ALFABETO, SPAZIO DI Probabilità, SPAZIO CAMPIONARIO**

X = {H, T} -> {ALFABETO,RISULTATO}

DATO -> Valori assunti da una variabile random [**Definizione Formale**]
il dato è rappresentato dalla coppia <variabile,valore>

Esempio orange and apple [pagina 11] --> B{r,b} F{a,o}; scegliamo casualmente da dove estrarre, scegliamo casualmente il frutto da estrarre da quel box scelto in

precedenza, osserviamo il frutto preso e lo posiamo.
if we pick orange from blue boxe the output is F=o, B=b that are already data
X=[o,b];

Probabilità P(X=x) -> non posso dire con sicurezza ciò che succederà ma posso dare una probabilità che un determinato evento accadrà. 0 = impossibile, 1 = certo. E possiamo dire che la possibilità che si verifichi un evento b è descritto come P(b) = P(B=b) dove B è il vocabolario e b è l'evento.

Abbiamo due tipi di approccio alla probabilità:

-**Approccio Frequentista** -> utilizzando questa formula [P(X=x) = casi favorevoli / tutti i casi] , come l'estrazione di una carta in un mazzo: posso dire per esempio che l'estrazione di un determinato numero ha probabilità p di uscire se ripetendo l'esperimento infinte volte, c'è una proporzione p di output di quel numero. Nel lancio della moneta se esce 499 volte testa, su 1000 lanci allora avremo 499/1000=0.49 come probabilità di ottenere testa. L'approccio frequentista è basato sul fare ESEPRIMENTI.

-**Approccio Bayesiano** -> Non posso ripetere l'esperimento quando ragiono in questi termini: l'evento è caratterizzato da incertezza, che puo essere ridotta tramite la ricerca/osservazione di alcuni dati.
Dato che non posso ripetere l'esperimento, non posso neanche contare i casi favorevoli
Es. un paziente con dei determinati sintomi ha una determinata patologia. Se un nuovo sintomo dovesse manifestarsi potrebbe cambiare la patologia determinata precedentemente. Non possiamo ovviamente ripetere questo esperimento sullo stesso paziente. È importante il "degree if belief" in quanto non abbiamo il numero di casi favorevoli e percio dobbiamo utilizzare le regole di bayles per calcolarlo.

Probabilità Congiunte/Joint Probability -> P(x,y) ; z = [x,y] ; P(z) è una probabilità congiunta dove z equivale a scrivere (x,y)

La probabilità Congiunta è simmetrica P(X=x,Y=y) = P(Y=y,X=x)

	O	a
b	5	4
r	2	2

P(b,a) = 4/13 data la casella di pos(b,a) = 4/13
P(b) = 9/13 dato da 5+4;
P(O) = 7/13 dato da 5+2

PROBABILITA' MARGINALE -> Possiamo calcolare la probabilità di un evento P(x) partendo da una probabilità congiunta P(x,y). Probabilità che x assuma un valore variando tutte le y (vedere formula, infatti c'è la sommatoria dato che le y variano) -> $P(X = x) = \Sigma y\ P(X = x, Y = y)$. La probabilità P(x) è chiamato **MARGINAL PROBABILITY.** Sarebbe calcolata prendendo una riga e scorrendola tutta, così come fatto sopra per p(b). Quindi dato P(x) e P(y) è sicuro che possiamo calcolare pure P(x,y) (= P(y,x))

PROBABILITA' CONDIZIONALE -> probabilità che un evento accade dato un altro evento, quindi P(X=x|Y=y) ovvero probabilita che x accada dato y

P(X=x|Y=y) = (P(X=x, Y=y)) / P(Y=y) uguale anche a dire P(X=x, Y=y) = P(X=x|Y=y) P(Y=y)

Due regole utilizzate spesso negli esercizi di probabilità:

$$\textbf{Sum Rule: } P(X) = \sum_Y P(X, Y)$$
$$\textbf{Product Rule: } P(X, Y) = P(X|Y)P(Y)$$

Catene di regole di Probabilità condizionale, il tensore generalizza la matrice (come la sommatoria).

$$P(X_1, \dots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i|X_1, \dots, X_{i-1})$$

Teorema di Bayes

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Come ci si arriva?
P(A,B) = P(A|B) P(B)
P(B,A) = P(B|A) P(A)
sapendo che P(A,B)=P(B,A) con una semplice sostituzione P(A|B) = P(B|A)P(A) / P(B)

ESEMPIO PAGINA 29 PDF ed **ESERCIZIO PAGINA 31**

Probabilità incondizionata/Variabili Indipendenti -> Probabilità che qualcosa accada indipendentemente da tutto ciò che è successo precedentemente (pescare da un mazzo di carte e dopo aver pescato re-inserire le carte nel mazzo). La ricchezza e l'altezza di una persona sono due variabili indipendenti percio diciamo che ricchezza ⊥ altezza dove ⊥ sta per indipendenti.
$X \perp Y \mid Z$, vuol dire che X e Y sono condizionatamente indipendenti data la variabile Z fissata.
È la stessa cosa di scrivere P(X,Y|Z) = P(X|Z) P(Y|Z).

Due variabili condizionatamente indipendenti sono indipendenti se il valore di una terza è fissato

Probabilità Condizionata/Variabili Dipendenti -> Come per esempio altezza e peso: spesso chi è più alto è anche più pesante percio le due variabili dipendono l'una dall'altra.

REGOLA DEL PRODOTTO -> è utilizzata per fattorizzare l'unione delle probabilità.

REGOLA DELLA SOMMA -> è utilizzata per computare le probabilità marginali

3) TERZA LEZIONE

DISTRIBUZIONE DI PROBABILITA' -> è una funzione che assegna una probabilità ad ogni possibile valore di una variabile randomica.

Si dividono in PDF o PMF (viste successivamente) in base al fatto che le variabili siano continue o discrete:

X = {H , T} ; P(H)=1/2 ; x ~ P -> si legge "x segue P" dove x è la variabile random, P la prob P(x)

La distribuzione di probabilità è un insieme che associa un possibile valore dell'alfabeto X a un valore compreso tra [0,1] P : X -> [0,1]

PROBABILITY MASS FUNCTIONS (PMF)

Una distribuzione di probabilità su variabili discrete casuali

Se X è discreto allora P(X), ovvero la funzione, è PMF.

Devono essere rispettate le seguenti condizioni:

- Per ogni P(X) deve verificarsi 0 <= P(X) <= 1
- La somma di tutte le x appartenenti all'alfabeto X deve essere =1 -> $\sum_{x \in \chi} P(x) = 1$.

PROBABILITY DENSITY FUNCTIONS (PDF)

Quando le variabili sono continue allora parliamo di PDF (Prob. Density Function) P : X -> [0,1]. Supponiamo per esempio di avere una variabile h=altezza di una persona. L'insieme X allora lo possiamo scegliere in diversi modi per esempio X = R+ (numeri reali (il + indica solo i positivi).
Devono essere soddisfatte le seguenti caratteristiche:
- P(x) >=0 ma non puo essere negativo
- ∫ P(x) dx =1; quando parliamo di insiemi continui utilizziamo l'integrale e non la sommatoria per far la somma di tutti gli elementi dell'insieme X.

In generale la probabilità sarà ∫ tra (b,a) P(x) dx; l'integrale con b in alto e a in basso Il generatore di numeri casuali = ∫ (b,a) 1/b-a dx =1, questa cosa deve risultare sempre affinché siano distribuzioni uniformi

EXPECTATION -> la media pesata di tutti i valori possibili, ottenuta dalla probabilità dei valori (media in termini probabilistici). Se ho un insieme con l'altezza di tutti gli uomini, fare un expectation vuol dire fare la media.

E[X] = ∑x(sotto) x P(x)
$E_{X\sim P}[f] = \sum$x(sotto) x P(x)f(x) rappresenta l'expectation di una funzione sotto una probabilità P $E_{X\sim P}[f] = \int P(x)f(x)\,dx$ se le variabili sono continue l'expectation prende questa forma
$E_{X\sim P}[f(x)] = E[X] = \sum$x(sotto) x P(x) se la f(x)=x identità, possiamo calcolare l'expectation della variabile

VARIANZA -> è il valore atteso da E[(x' – E[x])^2] dove E è l'expectation , che ci fa capire come sono distribuiti i valori nel grafico. La varianza misura quanta

variabilità/dispersione c'è in una funzione f(x) intorno il valore intermedio E[f(x)] (esempio grafico 1 come un quadratino, grafico 2 come un cono gelato).

Var[f] = E[((f(x) – E [f(x)]) ^2] = E [f(x)^2] – E[f(x)]^2
oppure possiamo direttamente ottenere tramite la variabile direttamente
var[X] = E[X^2] – E[X]^2

COVARIANZA -> quanto due variabili si influenzano (sono correlate) a vicenda (se crescono i valori di una, quanto crescono i valori dell'altra). È un numero alto se al cresce di una cresce l'altra, basso se crescono poco, 0 se sono indipendenti quindi al crescere di una l'altra non cresce.
x - E[x] , y - E[y] se sono tutti e due a sinistra allora vuol dire che sono negativi, se sono a destra sono positivi. Valore positivo se sono concordi, valore negativo se sono discordi.

Cov (f(x),g(y)) = E[(f(x) – E[f(x)]) g(y) – E[g(y)])] con f e g due funzioni definite in X e Y rispettivamente
Cov (X,Y) = E[(x-E[x])(y-E[y])] con f e g identità e con X e Y due variabili random

DISTRIBUZIONE DI BERNOULLI, è una distribuzione (insieme di dati) su una variabile X che puo prendere come valori solamente 0,1: P(X=1) = Ø; P(X=0) = 1- Ø;
E[x]= Ø; Var(x)= Ø(1- Ø);
Es. qual è la probabilità di ottenere testa, lancio un dado -> ovviamente il lancio nel lancio del dado o otteniamo testa oppure otteniamo croce.

DISTRIBUZIONE BINOMIALE -> è una PMF su numeri naturali con parametri n e p. $P(k) = (^{n}_{k})p^{k}\ (1 - p)^{n-k}$ dove k numero di successi(risultato sperato), n numero di prove indipendenti, p probabilità di successo in una singola prova.
Es. La probabilità di ottenere 3 testa lanciando 3 volte? K=3; n=3; p=0.5;

DISTRIBUZIONE CATEGORICA -> è una distribuzione di una singola variabile discreta con k stati differenti dove k è un valore finito. Probabilità degli eventi p(x=i) = pi dove: p $\in$ [0,1]^k-1; la probabilità dell'i-esimo stato è dato da $1 - \sum_i p_i$; Questa formula è la stessa formula di Bernoulli però distribuita in casi di multi stati.

DISTRIBUZIONE MULTINOMIALE -> è una distribuzione estensione della distribuzione binomiale nel caso in cui l'esperimento non è binario ma si ha multiplo output. Ci dice qual è la probabilità, dando un'ipotetico numero di volte che è uscito l'1, il 2, il 3, etc., ottenuta da un lancio di un dado(6facce) lanciato (15) volte, tramite la formula: $P(n_1 \ldots n_k) = [\ (n!) / (n_1! \ldots n_k!)\ ]\ \ p_1^{n1} \cdot \ldots \cdot p_k^{nk}$

DISTRIBUZIONE GAUSSIANA/NORMALE -> è una PDF utilizzata quando si parla di valori reali. Vedere la formula slide.

DISTRIBUZIONE GAUSSIANA DEL PARAMETRO STIMATO -> avendo dei dati possiamo ottenere i parametri della distribuzione gaussiana correlati ai dati con una stima. Vedere la formula

Teoria dell'informazione, ci permette di "trattare" l'incertezza però non ci permette di quantificarla e di separare dati informativi da dati non informativi (secondo un relativo scopo da raggiungere: se mi servono le altezze di 30 uomini non vado a prendere le età).

Self-Information -> vogliamo quantificare l'informazione: eventi frequenti (pochi dati informativi), eventi poco frequenti(tanti dati informativi dato che accadono

raramente). Dato una variabile casuale X~P, la self-information di un evento X=x è data dalla formula I(x)=-logP(x). se la base logaritmica è 2 allora si misura in bits, se invece è e si misura in nats.

Una **proprietà / caratteristica dell'informazione** è che la quantità di informazione di due possibili risultati di variabili indipendenti è la somma di quantità di informazioni relative a risultati delle singole variabili.

Entropia
Self-Information è definito su un singolo risultato: possiamo utilizzare la **shannon entropy** per calcolare l'incertezza in una distribuzione di probabilità tramite la formula $H(X) = E_{X\sim P}[I(x)] = -E_{X\sim P}[\log P(x)]$. Sono le informazioni che ci si aspetta di un evento estratto da una distribuzione.

Nel caso di una variabile DISCRETA è $H(X) = -\sum_x P(x) log P(x)$ se la variabile è CONTINUA invece la formula è uguale solo che al posto della sommatoria c'è il logaritmo.
Abbiamo la **joint entropy H(X,Y)** in un set di variabili casuali

$$H(X,Y) = \sum_{x,y} p(x,y) \log p(x,y)$$

Possiamo definire l'entropia condizionale invece come

$$H(Y|X) = -\sum_{x,y} p(x,y) \log p(y|x)$$

4) QUARTA LEZIONE

API Permettono di fare richiesta verso un sito in maniera strutturata per l'estrapolazione dati.
Seguono il paradigma WEB SERVICES: Client -> richiesta verso il server -> Server -> risposta verso il client.

PROTOCOLLO REST (Representational State Transfer):
Richiesta del client a un server (generalmente) che ritorna una risposta in formato strutturato (html+css, protocollo HTTP).
Oppure Client (app) che invia richiesta al server e riceve una risposta in JSON o XML.
Non ci sono le sessioni quindi ogni chiamata è indipendente cosi come l'autentificazione dei parametri passati ad ogni chiamata.
JSON (JavaScript Object Notation) tra i piu utilizzati e molto semplice nella lettura del formato.

REST CALL:
Somiglia a un'URL. Vedere a pagina 6 come avviene la suddivisione del link
Https e http sono i due protocolli piu utilizzati. La chiamata ritorna un JSON con tutte le informazioni relative alla richiesta e possono essere decodificate con delle librerie → Vedere pagina 7 sull'estrazione dei dati dalla richiesta JSON.

In molti casi le richieste APIs richiedono autenticazione. Per le chiamate REST l'autenticazione, non si fa chiamata per chiamata, ma utilizziamo un principio di "autorizzazione" -> tramite OAuth (autorizza, non autentica), permette lo sharing delle risorse tra gli utenti tra i diversi siti; le credenziali non sono condivise ma vengono condivisi dei token che garantiscono: accesso al sito specifico, accesso alla risorsa specifica (ma non autenticità), durata (in caso di furto del token il ladro potrà usufruirne solo per un determinato tempo). Le credenziali vengono passati una volta sola, il token lo richiediamo quando ci serve. No token -> richiesta rifiutata.
Autenticazione -> avviene tramite id e password
Autorizzazione -> Utente idProvider -> tramite un token (es. google da un token e lo passiamo al sito/app a cui noi vogliamo accedere).

L'user richiede il token all'identity provider e all'API provider. Il provider controlla l'identità dell'utente e verifica se ci sono i giusti diritti di accesso alle risorse, rilasciando il token all'utente. Il token poi viene utilizzato nell'app/servizio garantendo accesso.

Pag23 tutte le varie API e in precedenza l'utilizzo delle API relative a twitter

Pag24 -> Student Tutorial

Quando utilizziamo grandi quantità di dati, dobbiamo spesso utilizzare un database. Database che usano SQL sono piu rigidi -> MONGODB è un database non relazionale (non usa SQL).

Tools di Analysis visual -> dopo aver ottenuto i dati, ci sono vari modi per visualizzarli per capire la situazione in maniera facile e chiara utilizzando dei tool per l'analisi di questi dati.

Highcharts/Google Charts/ D3js / Plotly sono tools di visualizzazione dati (vedere le slide per le differenze)

WEB SCRAPING -> Quando non sono disponibili le API in un sito web che contiene informazioni importanti che vogliamo estrarre. Possiamo analizzare la pagina HTML tramite formati strutturati, il DOM (Document Object Model) che quindi permette l'analisi automatizzata. Il processo quindi, quando non sono presenti le API, richiedono "esplorazione manuale del DOM attraverso il browser". [come facciamo noi nel sito degli ukulele, esempi da 41 in poi]

5) QUINTA LEZIONE - Lezione giorno 11/11/2019 – Inizio capitoli per esame

MACHINE LEARNING

Campo di studi che fornisce ai computer la possibilità di apprendere senza essere esplicitamente programmati.
Un algoritmo di ML è così descritto : Si dice che un computer impari dall'esperienza E rispetto a qualche compito T e qualche misura di prestazione P, se la sua prestazione

su Y, misurata da P, migliora con l'esperienza E. Un approccio con machine learning ha a che fare con:

-**Task T**, è il problema da risolvere (trovare i volti in una foto; capire se un'e-mail è spam).
Es. Dobbiamo capire se un post su facebook è relativo alla politica o meno quindi trovare se la label Y=post_politico oppure Y=post_non_politico
-**Esperienza E**, l'esperienza è rappresentata dai dati, che sono i valori assunti dalle variabili random. L'algoritmo impara quindi come comportarsi in una determinata occasione sfruttando i dati in suo possesso. L'esperienza è costituita quindi da una coppia $(X_i$=dato ,Y_i=label/valore corretto$)_i$.
Es. Dato un post di facebook relativo alla politica abbiamo X=contenuto_del_post e Y=post_politico o post_non_politico .
-**Misura di Performance P**, consiste in una funzione in grado di valutare quanto le predizioni fatte sono state corrette. Consideriamo di avere due set di valori, uno relativo alle labels predette $\{\hat{y}_i\}_i$ e l'altro set di label corretto $\{y_i\}_i$. Come facciamo a capire quanto è buono il nostro metodo di predizione? Utilizziamo quindi una misura di performance P: **$P(\{\hat{y}_i\}_i, \{y_i\}_i)$**, Questa funzione ritorna un valore: se alto significa predizione corretta e accurata, se basso significa predizione non accurata.

L'input/esempio per gli algoritmi di machine learning possiamo vederlo come un X (array) appartenete a R^n in cui **$x = (x_1,\dots,x_n)$** dove ogni x_i è detta **CARATTERISTICA/features (x)**. Un **esempio/input** è rappresentato quindi dal vettore X ovvero un insieme di features che è quantitativamente misurata da alcuni eventi.
Es. 1.1 -> (per esempio l'array X nel riconoscimento di un petalo ha come xi, 4 x che rappresentano le grandezze in orizzontale e verticale del petalo che possiamo chiamare x1,x2 per il primo petalo e x3,x4 per il secondo petalo. I possibili risultati quindi le **lables** y{1,2,3} sono solo 3 in quanto abbiamo solo 3 tipi diversi con cui possono essere identificati questi fiori)). Le x sono gli attributi, i valori degli attributi sono le feature (colore(att) = blu(feature)).

Features = CARATTERISTICA DEL DATO; possono essere categoriche ovvero un numero finito di variabili *discrete* di tipo nominale (non c'è ordine tra i valori) e ordinale(c'è ordine), continue solitamente quando non c'è "spazio" tra i possibili valori.

L'output viene detto invece **LABEL (y)**, spesso è possibile associare già un determinato output ad un certo input (non sempre) : x→y. Come? Tramite un esempio = input. Le labels le possiamo quindi vedere come y=f(x) in quanto trattando con funzioni le features otteniamo la labels piu simile.

RAPPRESENTAZIONE DEI DATI -> I dati spesso ci vengono dati in maniera non conveniente per il trattamento e dobbiamo quindi estrarre delle caratteristiche di essi in modo tale da renderli entità qualificabili.
Tramite una **funzione f** detta **RAPPRESENTAZIONE** che ci dà x=f(d) , dove d (**input raw data**(es. post di facebook)) sta per dato (l'input), f (**representation)** è la funzione che trasforma l'input preso d in modo tale da renderlo "leggibile" per il nostro algoritmo di ML e l'output x (**feature vector/rappresentazione**) che sarà utile come input per l'algoritmo di machine learning. Questa funzione rende il dato trattabile, esplicitando alcuni dati ed oscurandone altri non utili. Utilizziamo quello che è detto **FEATURE EXTRACTION** in cui identifichiamo quali dati sono importanti e quali meno per la nostra classificazione. Non ci sono quindi rappresentazioni universali

ma solo rappresentazioni specifiche di quei determinati dati.
Es. email 3 righe sotto

ATTENTI A NON CONFONDERSI in quanto **rappresentazione** è utilizzato sia per x che per la funzione f

Come capiamo se un'email è spam?

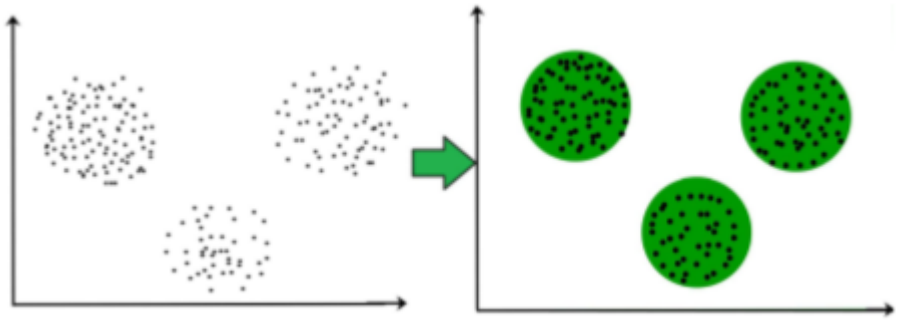
Dato il testo dell'email, facciamo intanto feature extraction per ottenere le varie features di X quindi andiamo a vedere per esempio due caratteristiche (x1,x2) dove x1=errore ortografico, x2=parole chiave in quanto spesso nelle email di spam abbiamo molti errori e molte parole chiave ripetute. Una volta ottenuti questi valori possiamo avviare il nostro algoritmo di ML. Se questi valori sono alti allora la probabilità che siano SPAM è alta. Nella rappresentazione grafica avremmo i pallini tutti in alto a dx; nelle email che non sono spam invece sarebbero tutti in basso a sx. Abbiamo un sacco di informazioni che vengono buttate, ma sono tutte informazioni che per il nostro scopo non sono importanti.

TASK CLASSIFICATION -> in questo tipo di classificazione si chiede a quale specifica categoria **k** l'input appartiene. I learning algorithm è comunemente fornito di un set di **SAMPLES** x che vanno da $x^1,...,x^n$,

con $x^j \in \Re$ (N.B. non sono feature, ma vettori) e un set di corrispondenti **LABELS** $y^j \in \{1...k\}$ che specificano a quale categoria k i samples appartengono.
Dato che dobbiamo capire se un'email è spam o meno la classificazione è binaria e quindi le lables $y \in \{0,1\}$ e non da 1 a k. Si ha alla fine una funzione del tipo $y^{(j)} = f(\bar{x^{(j)}})$.

TASK REGRESSION -> si chiede alla macchina di predire un valore numerico dati alcuni input. Utilizzabile se dato in input la grandezza (insieme x) e vogliamo dare in in output (insieme y) il prezzo. Non c'è ovviamente piu il bisogno di utilizzare livelli k. Si ha la stessa funzione della classificazione.

TASK CLUSTERING -> diamo in input all'algoritmo x^i e un parametro K che rappresenta il numero di gruppi che vogliamo ottenere: non riceviamo nessun tipo di label/output. Lo scopo dell'algoritmo è quello di raggruppare gli input in gruppi coerenti tra loro.
Es. dati un insieme di utenti e volendo proporre loro delle offerte, sfruttiamo per esempio un determinato like per alcuni libri/giochi, raggruppiamo quindi gli utenti in gruppi facendo clustering e poi facciamo l'offerta.



THE PERFORMANCE MEASURE P: ritorna un valore in grado di valutare un algoritmo di ML. È usata per 2 ragioni principalmente:

1) per capire se un algoritmo di ML sta migliorando su un determinato task da risolvere
2) per valutare la performance di un algoritmo di ML una volta che è stato allenato.
Calcolata su un set di label reali e un set di label predette $P((y_i)_i, (\hat{y}_i)_i) \to [0,1]$ specifico per il task -> $\hat{y}$ rappresenta la predizione del valore che ci aspettiamo. [Una misura di performance è considerabile anche L'**ERROR RATE** dato da 1-accuracy].
Un algoritmo di ML impara dall'esperienza come migliorare la misura di performance su un dato task.
L'**EXPERIENCE E** è un insieme di esempi x^j detto **INSIEME DI DATI DI TRAINING** (alcune volte legata con le labels y) utilizzati dall'algoritmo di machine learning per migliorare la misura di performance su un determinato task.

P(Y, $\hat{y}$) = 1/N sommatoria che va da N partendo j=1 [y^(j) = $\hat{y}$^(j)] → **ACCURACY**

Regressione: ho un input (es.5) e devo dare un output (es.8)

L'esperienza è utilizzata in maniera diversa in base al tipo di approccio di ML utilizzato:

- Approccio/insegnamento SUPERVISIONATO -> algoritmo è fornito di input samples e label corretto: (x,y) e Modeling: P(Y|X): l'algoritmo ha i samples e risolve i task adattando una funzione ai dati in input, dando le label y corretta per quell'x. Esempi di task di ML supervisionati sono classification e regression.
- Approccio/insegnamento NON SUPERVISIONATO -> algoritmo fornito solamente con i samples: x e Modeling: P(x). E' molto meno ben definita, sono più complicati dato che deve calcolare tutto da sé senza avere esempi. Ho solamente x, non ho y. Clustering è un esempio di learning non supervisionati in cui non sono fornite informazioni addizionali oltre quelle dei samples.

Lavorare con l'approccio supervisionato è generalmente piu facile ma il fatto di avere le lables molte volte puo provocare un problema di perdita di tempo in quanto ci vuole molto lavoro manuale.
ES. se dobbiamo costruire un sistema di spam-detection ci deve essere qualcuno che selezioni prima un insieme di email come spam/non-spam.
L'approccio non supervisionato invece è piu difficile da lavorarci e non richiede la presenza di labels.
ES. costruire un sistema di spam-detector utilizzando un algoritmo di clustering e poi automaticamente scoprire quale messaggio è buono e quale no.

Abbiamo il **DATASET** -> è il set di samples. puo essere sia un array che una matrice e viene passato in input all'algoritmo di ML.

LEARNING/TRAINING -> è un processo di "allenamento/migliorare le performance" del nostro algoritmo tramite un **training set**/**dataset di esempi** per un determinato task. L'algoritmo possiede dei **parametri** che vengono modificati dal training, utilizzando i samples, in maniera da ottimizzare il loro valore. Altri parametri detti **hyperparametri** invece non vengono modificati dal TR.
Non c'è comunque garanzia che dopo un training, l'algoritmo sarà performante con dati che non ha mai visto. Si fa affidamento in 2 set di dati:
- Training set, usato per il training
- Test set, utilizzando per valutare la performance dopo il training
TESTING/EVALUTATION fase successiva al training in cui tramite il **TEST SET** valutiamo/testiamo se l'algoritmo di ML è buono.
TRAINING SET + TEST SET = DATA SET → TR set e TE set sono frutto della divisione in maniera casuale del data set.

Tramite la divisione del dataset in **training, validation e test set (randomica)** al fine di cercare di sintonizzare quelli che sono gli hyperparametri in modo di cercare di avere un algoritmo performante su dati che non ha mai visto.
Possiamo utilizzare quello che è il metodo di **CROSS VALIDATION** -> Una procedura con varie azioni (leggere dalle slide) che è in grado di dare un buon risultato, anche se richiede un po di tempo, quando il dataset è piccolo. Dando una buona stima del modello hyperparametro.

La capacità di avere buone prestazioni trattando un input mai visto prima dall'algoritmo è detta **GENERALIZZAZIONE.** Utilizziamo due parametri che vogliamo siano bassi:
- training error, errore ottenuto nel training set;
- **test error or generalization,** errore ottenuto sul test set. Ci sono però due problemi comuni che sono:

Dobbiamo essere in grado di andare a cercare una figura geometrica in grado di andare a cercare i "punti" corretti da prendere per il training set.

-Underfitting -> lineare -> Training error e Test error elevati -> una retta non è in grado di andare a determinare i punti dello schema se sono per esempio agli angoli -> bassa capacità in quanto fa un taglio netto. È molto efficace nel caso in cui i dati si trovano in fila indiana o circa.

-Overfitting -> Training error basso, Test error alto -> ho figure geometriche elaborate in quanto non ho una retta ma una curva che è causa anche di prendere puntini che non ci interessano e che fanno sbagliare.

Queste due condizioni sono influenzate dalla **capacità** del modello: l'abilità di fittare un grande numero di funzioni. Un modello che computa solamente funzioni lineari ha sicuramente capacità minore rispetto ad un modello che modella pure funzioni non lineari. L'**Appropriate capacity** è la capacità del modello in grado di avere un'ottima generalizzazione.

Tramite algoritmi di regolarizzazione sono in grado di andare a limitare il problema dell'over/under Fitting
-> si preferisce l'overfitting all'under.

Vedere gli esempi nel pdf 5 alla fine con l'esempio dell'under e dell'over fittings.

6) SESTA LEZIONE 13/11/2019

Il più comune mezzo nel web per l'uso dei social media è il "TESTO" in diverse strutture semantiche come xml, json oppure scritte da parte dell'uomo come per esempio nei post di twitter. Prima di andare a estrarre il testo dobbiamo utilizzare delle regole in grado di capire il **linguaggio naturale**. Quando un testo non è strutturato abbiamo bisogno di vari modi per processarlo: si utilizza il **Natural Language Processing NLP**. Vedremo sotto le varie fasi della NPL (*).
Es:
- Information Estraction data una mail estrarre tutti gli appuntamenti o numeri di telefono. Dato un testo estrarre tutti gli argomenti. Dato un testo del tipo: domani il meeting sul ML è dalle 12 fino alle 13 in aula 350 -> dare un output strutturato del tipo Evento:xxx, Data:xxx, Start:xxx, End:xxx, Location:xxx
- Text Categorization Dato un testo capire l'argomento. Sentiment Analysis -> Date certe parole capire se il testo è piu o meno positivo (magari relativo ad un prodotto su

Amazon).
- Machine Translation -> Google traslator
- Summarization -> Fare un riassunto del testo che stiamo passando in input.
- Image Captioning -> Data un'immagine se ne deriva la descrizione.

ESPRESSIONE REGOLARE -> Permettono di risolvere i problemi in maniera semplice tramite linguaggi formali. Trovano dei pattern in grado di tirare fuori da un set di dati solo quello che noi stiamo cercando.

I **SET** (indicati con []) permettono di andare a fare match di un carattere/serie di caratteri che sono match possibili:
[Tt] -> nel testo verranno evidenziate tutte le t minuscole e le T maiuscole.
[Tt]he -> trova tutte le T maiuscole o minuscole seguite da he
[A-Z] -> trova tutte le lettere dell'alfabeto maiuscole
[a-z] -> trova tutte le lettere dell'alfabeto maiuscole
[0-9] -> trova tutti i numeri
[A-Z0-9] -> trova tutti i numeri singoli e tutte le lettere
[^A-Za-z] -> trova tutto cio che non è una lettera maiuscola/minuscola
day|end -> trova o la stringa day o la stringa end
t?he -> si matchano solamente the oppure he (es. The = te -> la T è maiuscola; tghe = te; he = he; the = the; ttttthe = he; hhhhe=he)
o+h -> numero infinito di o seguito da h (es. oooooh ooh ooh oh oooooh, h, li segniamo tutti tranne l'ultimo)
o*h -> la o puo non esserci (es. h verrà matchata lo stesso)
o{3}h -> matchiamo solo oooh
.{2}h -> 2 caratteri qualsiasi legati a una h (Es. segnati quelli matchati ohsdjshisnnhoieohihhhshohnihhhhh)
[oh]+! -> un qualsiasi numero di o e h concatenate anche a caso tipo hohhhoohohohhohhho!

Vedere esempi sulle espressioni regolari fino a pagina 22 – Quantificatori etc.

(*) L'NLP è diviso in diverse fasi:
WORD TOKENIZATION -> data una testo lo dividiamo in diversi pezzi chiamati token, cioè entità che possono fare matching in un vocabolario di simboli/parole per dare una struttura alla frase. Quando nel vocabolario non si trovano quei simboli/caratteri allora si incorre nell'**out of words OOV.** In ogni parola si tiene conto di :
-PREFIX, carattere iniziale
-SUFFISSI, caratteri finali
-INFISSI, caratteri di mezzo
-ECCEZIONI, eccezioni tipo N.Y. oppure let's, su cui è difficile definire una regola

STEMMING, data una parola che puo cambiare tipo run/running/runs in base al tempo verbale/singolare/plurale, viene utilizzata un'unica parola per raggruppare varie parole nello stesso STEM.
Lo stemmer piu famoso è il Porter Stemmer.
Es. run,running,runs -> run; stem,stemming,stems -> stem
LEMMATIZATION, diversamente dallo stemming considera il vocabolario di un linguaggio da applicare all'analisi morfologica delle parole. Il lemma di "was" è "be", il lemma di "knives" è "knife", il lemma di "meeting" puo essere sia "meet" che "meeting" in base al contesto. Uno stemmer non è in grado di capire questa variazione di significati. La lemmatization andando a guardare tutto il contesto

circostante quella parola, è in grado di risolvere problemi di ambiguità.
Lo stemming è meno complesso della lemmatization, però si perde in termine di efficacia.

STOP WORDS -> si vanno a cercare delle “stop words” all’interno di un testo ovvero tutte quelle parole che probabilmente posso essere rimosse in quanto danno dei dati ridondanti -> “the” “a”

PART OF SPEECH (POS)-> il linguaggio naturale è ambiguo in alcuni casi in quanto alcune parole prendono diversi significati in base alla loro posizione nel testo/frase. POS tramite l’analisi del testo ci permette di definire il ruolo di ogni parola (tag the words) come verbo, aggettivo, etc. e quindi di risolvere le ambiguità.
Ci sono due diverse maniera di tagging: Coarse-Grained POS tags e Fine-Grained POS tags

NAMED ENTITY RECOGNITION (**NER**) -> è in grado di evidenziare e quindi di taggare nel testo alcune parole al fine di menzionare un nome/entità/ammontare di soldi o percentuali. (pag 37)

SENTENCE SEGMENTATION -> analizzare l’intero testo è spesso difficile e quindi si divide il testo in frasi. Puo essere utile per contare il totale delle frasi o per fare la media di parole per ogni frase.
N.B. Non puo essere utilizzato il “.” come unico segno per spezzare la frase perche ci sono circostanze in cui il punto non ha quel significato es scrivendo 25.5 il punto divide centinaia e decine.

NLP PIPELINE -> generalmente segue i seguenti step:
-word tokenization
-stop words removal
-stemming o lemmatization
-POS tagging
-NER tagging

Sappiamo che per risolvere molti task dobbiamo rappresentare i dati nella maniera corretta quindi fare una giusta **data representation**. In particolare, dobbiamo ottenere dati in **rappresentazione a lunghezza fissa** così poi da essere processabili dal ML.
Possiamo utilizzare la rappresentazione **BAG OF WORDS** -> un algoritmo di rappresentazione in cui oltre la parola (salvate senza piu l’ordine del testo) vengono salvate pure le **FREQUENZE** per ogni parola. Si utilizza la funzione bow(d) = count(d,t) dove d è il documento e t è il termine considerato, quindi count ci torna il numero di volte che il termine t appare nel documento d. Avendo le parole così distribuite, senza un ordine, è piu facile derivare l’argomento del documento.

Il processo è quindi questo:
documento -> NLP Pipeline -> SET OF WORDS: creazione vocabolario V i cui termini sono presi dal documento -> ogni documento è rappresentato da una bow(d) che genera un array in cui l’i-esimo elemento raffigura quante volte quel termine è ripetuto in quel documento: c(d,t) -> la dim(bow(d))=|V| da qui deriva quindi la lunghezza fissa indipendentemente dalla lunghezza del documento in input.

FEATURE NORMALIZATION: in alcuni casi due istogrammi, relativi a due documenti diversi, possono far risultare i documenti simili anche se essi non lo sono affatto in quanto possono avere simile frequenza di termini ma diverso numero di parole. Es. doc1 = 38 parole, doc2 = 114 parole. Si effettua quindi una feature normalization per

vedere se effettivamente i due documenti sono “simili” seguendo i seguenti passaggi:

$nbow(d)_i = \frac{bow(d)_i}{\sum_i bow(d)_i} = \frac{c(d,t_i)}{\sum_i c(d,t_i)} = tf(d,t_i)$ e dato che il numero di parole è sempre >0 questa normalizzazione è equivalente a l1: $\overline{x} = \frac{x}{\sum_i |x_i|}$, where $x = (x_1, \dots, x_n)$; .
tf(d,t) è term frequency che indica la frequenza con il quale viene ripetuta una parola in un documento.

TF-IDF:
Le parole meno frequenti, cosi come i dati, contengono piu informazioni e quindi dobbiamo logicamente assegnare un peso diverso alle parole: possiamo farlo tramite normalizzazione **TF-IDF** term frequency - inverse document frequency.

IDF $p(d_i,t_j) = \begin{cases} 1 \text{ if } c(d_i,t_i) > 0 \\ 0 \text{ otherwise} \end{cases}$ $c(t_j) = \sum_i p(d_i,t_j)$ $idf(t_j) = \log \frac{n}{c(t_j)}$

$$bow(d_i)_j = tf(d_i,t_j) \cdot idf(t_j) = \frac{c(d_i,t_j)}{\sum_j c(d_i,t_j)} \log \frac{n}{c(t_j)}$$

$$idf(t_j) = -\log \frac{c(t_j)}{n} = -\log P(T = t_j) \text{ (self information)}$$

La funzione p(di,tj) indica se quel termine t è presente nel documento. Definiamo c(tj) ci dice il numero di documenti che contengono il termine tj. L’elemento j esimo della bag of world del documento di sarà dato dalla formula bow(di)j. Possiamo ricavare infine l’inverse document frequency con la funzione idf(tj) che rappresenta la self-information di T=tj ovvero la misura di quanta informazione contiene un valore: grande tanta informazione, poca se piccolo.

N-GRAMS -> BOW uguali possono appartenere a frasi che hanno significato diverso -> “Alessia lava la macchina di Diego” – “Diego lava la macchina di Alessia”. N-grams cerca di risolvere il problema delle ambiguità nelle frasi in cui i significati sono diversi. Settando:
- N=1 abbiamo un unigram (quello che si è fatto fin qui), ottenendo Alessia; lava; … ; Diego. Scomponendo quindi le due frasi non risolviamo le ambiguità utilizzando N=1.
- N=2 abbiamo un bi-grams che già inizia a ridurre le ambiguità. Creiamo un vocabolario di bigrams e rappresentiamo le frasi con bag of bi-grams.

N-grams aggiunge quindi molta flessibilità ma la computazione richiede molte risorse con N grandi.

Il BOW puo essere utilizzato in generale come “bag of thing” per ottenere la rappresentazione.
Es. bag of stems/lemmas/POS/NER/n-grams

7) SETTIMA LEZIONE 20/11/2019 (pdf7)

Tasks Classification : Partiamo dal presupposto che dato un input dobbiamo cercare di capire a quale label y appartiene (post facebook se y=politico o y=non_politico; email spam; object recognition; etc.).

Il primo passaggio fondamentale è la **Data Rappresentation** ovvero ottenere un vettore di lunghezza fissa per un determinato example. **DR Notation** -> Assumiamo

di avere un vettore $x \in \Re^n$ i cui elementi xi sono chiamati features e una classe di label y $\in \{0,\ldots,m-1\}$. Consideriamo anche 2 sets di examples : TR (Training Set) e TE (Test set): $TR = \{(x^{(j)}, y^{(j)})\}_j; TE = \{(x^{(j)}, y^{(j)})\}_j;$

C'è un gruppo opzionale VA (validation) $VA = \{(x^{(j)}, y^{(j)})\}_j;$
La rappresentazione dei dati avviene in 2D su un piano cartesiano in quanto possiamo sfruttare quelle che sono le equazioni.

Dobbiamo trovare quindi quella funzione f(x) tale che y=f(x)

Abbiamo diversi tipi di **classification**:

CLASSIFICATION NEAREST NEIGHBOR, la funzione y=f(x) che noi vogliamo trovare è una funzione tale che $P(\{y^{(j)}\}_j, \{f(x^{(j)})\}_j)$ torni dei valori alti (buoni), sia per il TR che per il TE.
Se memorizziamo il TR set abbiamo una funzione in grado di associare la corretta label per un data point: $f(x) = y\ s.t\ (x, y) \in TR$ che però non funziona nel momento in cui gli passiamo un sample che non $\in$ al TR. Quando lo spazio delle features è continuo è molto probabile che TR $\cap$ TE != $\varnothing$.
Quindi la soluzione è quella di trovare il TR molto simile al TE in quanto la probabilità che i due sample siano associati alla stessa classe è molto alta. La nostra funzione quindi diventa $f(\bar{x}) = \arg_y \min\{d(\bar{x}, x) | (x, y) \in TR\}$ dove d è $d(\bar{x}, x) = |\bar{x} - x|_2;$ la **distanza EUCLIDEA** in cui x segnato è un TR sample mentre x è un TE sample.

Es. (x1=2;y1=0) e (x2=1;y2=1) è quello che sappiamo -> dato x3=1.8 allora andiamo a calcolare il valore y3=0 dato che la distanza minima è piu vicina a x1 che a x2 quindi il risultato sarà lo stesso di y1 (Tutto ciò lo vediamo al livello grafico(come sempre) tramite l'utilizzo delle palline).
Es. Tutte le e-mail rappresentate utilizzando una bow di 50k parole allora $x \in \Re, n =$ 50000 mentre y $\in \{0,1\}$ in due classi spam e non spam. d è la distanza tra x segnato e x. Dato un x segnato si cerca il punto del TR piu vicino, assegnando la classe del punto nel TR piu vicino.

CLASSIFICATION K-NEAREST NEIGHBORS, K-NN -> assumiamo il fatto che un data point nella rappresentazione 2D avrà sicuramente come elemento piu vicino, un elemento appartenente alla sua stessa classe/label. Elementi lontani quindi corrispondono, probabilmente, ad elementi di classi differenti. Ovviamente ci sono casi in cui un data point è vicino ad un altro data point appartenente ad una classe che non è quella corretta come per esempio un'email in cui troviamo la parola viagra che non è spam, è molto probabile che come data point piu vicino ne avrà uno la cui classe di appartenenza è email di spam.
Per questo motivo non si guarda l'elemento piu vicino, ma tutto il suo intorno.
Di tutto l'intorno siamo noi a decidere la grandezza del raggio (quanti elementi selezionare) tramite un numero k che corrisponde al numero di elementi piu vicini al data point selezionato da selezionare. Definiamo il neighborhood di centro xsegnato e grandezza K come segue $N(\bar{x}; TR, K) = \{(x, y) \in TR \mid d(\bar{x}, x) \leq \epsilon_K(\bar{x}; TR, K))$. La funzione che finale è quindi $f(\bar{x}) = mode\{y | (x, y) \in N(\bar{x}; TR, K)\}$ in cui mode ritorna l'elemento piu

ripetuto (come moda in MMS) all'interno del set.
K rappresenta un hyperparametro e quindi per essere stabilito come parametro arbitrario oppure ottimizzato utilizzando un VA set in cui utilizzando vari valori di k scegliamo quel valore che ci ha dato prestazioni migliori. Notiamo che l'algoritmo Nearest Neighbor è l'algoritmo K-NN in cui K=1 e quindi 1-NN.

CLASSIFICATION MAP/DECISION BOUNDARY in cui la funzione assegna una classe ad ogni input x, dividendo lo spazio dell'input per la classificazione in diversi colori. Se il valore di k è:
- **PICCOLO** si fa overfitting e quindi si assegnano in modo errato classi a data point in maniera errata
- **GRANDE** si riduce l'overfitting.
- **ENORME (QUASI INFINITO)**, si fa underfitting ignorando alcuni data points senza assegnarli nessuna classe.
Con una buona rappresentazione e tanti dati, lavorare secondo la K-NN è un'ottima soluzione.

CLASSIFICATION: EVALUATING PERFORMANCE, importante per misurare la performance di un determinato algoritmo. Viene fatta sul TE in quanto rappresenta dati mai visti dall'algoritmo. Dobbiamo prendere in considerazione:
- Il set formato dalle vere labels Y{y}, **realtà di base labels**
- Il set formato dalle **labels previste** dall'algoritmo che vogliamo valutare $\hat{y} = \{\hat{y}\}$ che puo anche essere visto come ŷ=f(x)
La funzione di performance/evalutation P(Y, ŷ) →R è quindi in grado di darci un numero reale per valutare l'algoritmo avendo set di labels vere e il set di labels predette, valore spesso appartenente all'intervallo [0,1].

Valutiamo un classificatore tramite l'**accuracy**, un valore tra [0,1] secondo la seguente formula:

$$Accuracy(Y, \hat{Y}) = \frac{|\{i \mid y^{(i)} = \hat{y}^{(i)}\}|}{|Y|}$$

in cui si misura quanto sono simili i due set Y e ŷ.
L'ACCURACY tramite il suo valore ci dice quanto i due set (reale, previsto) siano simili e quindi se l'algoritmo è stato efficiente o meno. Funziona solamente quando il dataset è bilanciato: le classi hanno circa lo stesso numero di samples.
Es. Dato un dataset con 100k elementi e con 2 classi {0,1}, in cui 5k appartengono alla classe 0 e 95k appartengono alla classe 1, l'accuracy non è utilizzabile in quanto qualsiasi f(x) in cui x=0 verrà ugualmente computata come x=1 con un'accuracy f(x)=1 del 95%:

Con l'accuracy possiamo commettere due tipi di errori:
-Falsi negativi FN, classifichiamo elementi come non appartenenti ad una classe quando in realtà gli appartengono.
-Falsi positivi FP, classifichiamo elementi come appartenenti ad una classe quando in realtà non gli appartengono.
E due tipi di predizioni corrette:
-Veri positivi TP, elementi che appartengono giustamente alla classe a cui sono stati assegnati
-Veri negativi TN, elementi non appartenenti alla classe che giustamente sono stati classificati non appartenenti a quella classe.
Possiamo creare quella che è la **MATRICE DI CONFUSIONE,** che ci permette di capire se ci sono problemi con i classificatori nel caso di dati non bilanciati, utilizzando questi

elementi, in una matrice del tipo:
$\begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}$ righe indicano le corrette lables, le colonne le labels predette. In una matrice di confusione buona abbiamo alti i numeri della diagonale e bassi gli altri (gli errori). Possiamo ricavare l'accuracy dalla matrice di confusione andando a sommare i numeri nella diagonale e dividendoli per la somma di tutti gli altri numeri.

Possiamo calcolare due parametri importanti per vedere come i classificatori performano:
- **PRECISIONE**, misura quanti sample, classificati come positivi, sono veramente positivi secondo la formula Precision=TP/TP+FP
- **RECALL**, misura quanti samples positivi sono stati correttamente classificati come positivi secondo la formula Recall = TP/TP+FN
Preferiamo avere sempre un valore alto di questi valori che sono complementari tra loro.

-F1 Score ci permette di "raggruppare" precisione e recall in un unico numero che è media armonica tra questi due parametri secondo la formula F1=2* (precision*recall)/(precision+recall). Questo metodo non risente del problema dell'accuracy.

La matrice confusione puo essere generalizzata anche nel caso in cui ci sono m classi e non solamente binarie. Si puo costruire anche per la classificazione multi-classe. La matrice confusa MxM è composta da elementi M_{ij} in cui si intende l'elemento appartenente alla classe i che è stato classificato come appartenente alla classe j.

8) OTTAVA LEZIONE 25/11/2019
Possiamo calcolare che un determinato vettore x appartenga ad una determinata classe C sfruttando la distribuzione di probabilità condizionata P(c|x) tramite le regole di Bayes. Dato un input x percio possiamo associare la classe c corretta per esso, andando a prendere la probabilità P(c|x) piu alta -> per l'input x dobbiamo quindi calcolare la probabilità di quell'input con tutte le classi c. Questo processo in termini matematici utilizza la seguente formula: c **= arg$_c$ max**$P(c|x)$ **= arg$_c$ max P(x|c)P(c)/P(x).**
In termini Bayesiani P(c|x) è **una probabilità a posteriori** da cui deriva il nome **MAP maximum a posteriori.**

Modeling The Posterior: Calcolare P(c|x) è abbastanza difficile quindi sfruttando il teorema di BAYES andiamo a calcolare P(x|c), calcolo piu facile per noi ma con alcune restrizioni. Secondo la formula di Bayes:
P(C|X) = P(X|C)P(C)/P(X), consideriamo che P(x|c) la possiamo stimare ma abbiamo due problemi**:**
- P(C), la **probabilità a priori**, stimabile in diversi modi:
1) andando a considerare il numero di elementi nel nostro dataset. Es. dati 800non-spam e 200spam e-mail le probabilità saranno P(0)=0.2 e P(1)=0.8
2) P(C)=1/m dove m è il numero di classi, quindi assegnare equa probabilità a tutte le classi.
-P(X), che se è multi-dimensionale è piu difficile da stimare. Normalmente sappiamo che P(c|x) è proporzionale a P(x|c)P(c) e quindi c **= arg$_c$ max**$P(c|x)$**=P(x|c)P(c)**

P(c|x) = (P(x|c) P(c)) / (P(x)) -> P(c) è probabilità a priori mentre P(c|x) è a posteriori

CLASSIFICATORE NAIVE BAYES è una tecnica di classificazione, con features passate in input continue, basata sulla teoria di Bayes. È diviso in passaggi:
- Considerando di avere un vettore x=[x1,…,xn] in cui n è molto grande. Calcoliamo c = $\arg_c \max P(c|x)$P(c)= $\arg_c$ max P(x1,…,xn|c)P(c)
- Fattorizzazione secondo la regola del prodotto
$P(X_1, \dots, X_n|C) = P(X_1|C)P(X_2|X_1, C)P(X_3|X_2, X_1, C) \dots P(X_n|X_1, \dots, X_{n-1}, C)$
- 3 Facciamo **l'ASSUNZIONE NAIVE** ovvero assumiamo che le features in input sono condizionatamente indipendenti data la classe (assunzione che è quasi sempre mai vera). $X_i \perp X_j \mid C, \forall i \neq j \Rightarrow P(X_1, \dots, X_n|C) = P(X_1|C)P(X_2|C) \dots P(X_n|C)$
Questa assunzione è detta naive, poiché per esempio in un testo il numero di occorrenze di una parole non influenza il numero di occorrenze di un'altra parola, cosa che risulta falsa in quanto dove troviamo la parola "viaggio" è facile trovare anche la parola "bagaglio", andando quindi a "rompere" quella che è la condizionalità indipendente.
- Quindi possiamo riscrivere la MAP classification come
$\overline{c} = \arg_c \max P(x_1|c)(x_2|c) \dots P(x_n|c)P(c)$
- I singoli termini P(X1|C) sono ora facili da modellare in quanto X1 è monodimensionale. Sono modellati comunemente in due modi in base ai dati e al problema da risolvere: o tramite una distribuzione Gaussiana o tramite distribuzione multinomiale.
Nonostante la naïve assumption il classificatore funziona in maniera ottima in molti casi.

MULTINOMIAL NAIVE BAYES, le features non sono continue ma discrete:
- Trovare il vocabolario V di lunghezza n → |V|=n
- Le features in input x = (x_1,…,x_n)
- Quindi la probabilità di una x data la sua classe puo essere vista come $p(x_1, \dots, x_n|c) = p(x_1|c)p(x_2|c) \dots p(x_n|c)$ distribuzione multinomiale
- L'espressione finale per la text classification sarà quindi
$\overline{c} = \arg_c \max P(x_1|c)(x_2|c) \dots P(x_n|c)P(c)$

Il naive bayes lo possiamo quindi

riassumere cosi:

Pick the most proabable C given $P(C|X)$

Invert the conditional probability with Bayes: $P(C|X) \propto P(X|C)P(C)$

Assume a uniform prior for $P(C)$ or compute in a frequentist way

Assume the naive assumption and model $P(x_i|c_j)$

x_i continuous and Gaussian -> $P(x_i|c_j)$ Gaussian

x_i discrete -> $P(x_i|c_j)$ estimated in a frequentist way

9) NONA LEZIONE 27/11/2019

La **REGRESSIONE** è un algoritmo supervisionato che permette di imparare a mappare dei valori numeri presi in input in valori numeri come output.
Se volessimo andare a predire il valore di case dato il loro numero di stanze è possibile andare a farlo: Fare una predizione, quindi utilizzare una regressione lineare la cui funzione prende in input la grandezza della casa x1 e ci torna in output il prezzo.
Es. dobbiamo avere nel dataset il numero di stanze in una casa (X) e il prezzo della casa (Y), per poi andare a fare predizione nel caso in cui sapendo il numero di stanze della casa ne andiamo a prevedere il prezzo. Calcoliamo la covarianza tramite cov(x,y) = 4.49 che conferma il fatto che all'aumentare del numero di stanze, si ha l'aumento del prezzo della casa.

La **NOTAZIONE** utilizzata è :

Input features $x^{(i)}$ Target values $y^{(i)}$

Training samples $(x^{(i)}, y^{(i)})$

Training set $TR = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ $X = x^{(i)}$ when $Y = y^{(i)}$

$x^{(i)} \in \mathcal{X}, \quad y^{(i)} \in \mathcal{Y}$

Learn $f: \mathcal{X} \to \mathcal{Y}$ so that $f(x)$ is a good predictor for y

Quindi dato il TR set la regressione lineare tramite la funzione a cui viene passato una variabile indipendente X riesce a darci il valore y in modo che f(x)=y dove y è il buono valore predetto.

Es. **ESEMPIO REGRESSIONE LINEARE A VARIABILE MULITPLA / REGRESSIONE POLINOMIALE**
Consideriamo di avere oltre al numero di stanze della casa (x1) anche il tasso di criminalità del quartiere (x2) che va a influire sul prezzo della casa (y) -> si tratta di multi-variabili.
Quindi abbiamo che una determinata x è ora del tipo x=(x1,x2) (o anche maggiore, in base alle features che vogliamo considerare) e quindi la funzione f(x) è del tipo $f:R^2$ -> R in quanto alla fine dovrà dare un valore y.
Quindi regressione semplice f:R->R mentre regressione multipla $f:R^n$ -> R con n>=2

(Interpretazione Statica) La **REGRESSIONE LINEARE** è in grado di dare la correlazione tra le variabili ma non è in grado di spiegarla. Nel caso delle case e delle stanze non possiamo dire che aggiungendo una stanza il prezzo aumenta di 9100 perche ci sono altri parametri da valutare (es. criminalità) però comunque l'aumento di una stanza aumenta il prezzo. Possiamo dire che la crescita di un parametro causa crescita di un altro parametro percio si vede che i due fenomeni capitano insieme ma non si sa il perché -> **la correlazione non significa causalità**

(interpretazione geometrica) Nel caso di una figura geometrica o una linea con più curve, non è possibile descriverla tramite un'equazione lineare in quanto questa rappresenta appunto una linea semplice.

Nel computer questa funzione lineare è del tipo $f_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ in cui le varie x_n sono le feature, gli 0i sono i **parametri/pesi/coefficienti** appartenenti al $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$.

Nel caso di **regressione semplice** la formula sarà del tipo $f(x)=0_0+0_1x_1$: questa formula riporta a quella che è l'esplicita equazione di una retta y=mx+q in cui $m=0_1$ e $q=0_0$, dove **m** è il **coefficiente angolare** che ci dà l'orientamento della retta rispetto l'asse x mentre **q** è **l'intercetta** e ci dice dove si trova la linea nello spazio.

Come vengono scelti questi pesi? Un modo per farlo è scegliere dei parametri in modo da rendere f(x) molto simile ad y tramite la seguente **funzione costo della regressione lineare, il LEAST MEAN SQUARES LMS** $J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}\left(f_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2$
dove la differenza rappresenta la distanza media tra il valore f0(x) e il rispettivo valore label y. Se la distanza è di basso valore anche il "costo" sarà di basso valore. Quindi il "learning" viene fatto in modo da minimizzare la funzione costo in modo tale da trovare $\boldsymbol{\theta} = \arg_{\theta} \min J(\boldsymbol{\theta})$. Per trovare questi valori di 0 abbiamo bisogno di calcolare il min J(0) senza calcolarlo per tutti i possibili valori di 0, irrealizzabile quando 0 è multi-dimensionale. La soluzione è la l'algoritmo della discesa del gradiente che permette di minimizzare i parametri della funzione in questi 4 passi iterativi:
1) Scegliere un punto casuale x
2) calcolare la derivata f' del punto x -> f'(x)
3) Aggiornare la posizione del punto usando la formula x = x – yf'(x) fino ad ottenere ilq minimo
4) Ripetere i punti 2 e 3 fino ad un numero di volte che era stato prestabilito oppure fino a quando si raggiungere una f'(x) molto piccola tale che non possiamo piu procedere. Quello sarà il valore finale ottimizzato $x_n=\arg_x\min f(x)$

Quando si tratta di piu variabili seguiamo questi procedimenti (sceglierne uno, sono equivalenti):

1.Initialize $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ randomly;
2.Compute the gradient at the current point

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} f_{x_1}(\boldsymbol{x}) \\ f_{x_2}(\boldsymbol{x}) \\ \dots \\ f_{x_n}(\boldsymbol{x}) \end{pmatrix}$$

3.Update the current point using the formula:

$$\boldsymbol{x} = \boldsymbol{x} - \gamma \nabla f(\boldsymbol{x})$$

4.Repeat 2-3 until some termination criteria are met;

1.Initialize $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ randomly;
2.For each variable x_i:

- Compute the partial derivative at the point:

$$\frac{\partial}{\partial x_i} f(\boldsymbol{x})$$

- Update the current variable using the formula:

$$\boldsymbol{x_i} = \boldsymbol{x_i} - \gamma \frac{\partial}{\partial x_i} f(\boldsymbol{x})$$

3.Repeat 2 until some termination criteria are met;

L'algoritmo della discesa del gradiente è utilizzato per **L'OTTIMIZZAZIONE** del problema di regressione lineare e quindi per trovare $\boldsymbol{\theta} = \arg_{\theta} \min J(\boldsymbol{\theta})$:
- inizializziamo in maniera random 0 per poi applicare ad ogni variabile la regola di aggiornamento $\boldsymbol{\theta_j} = \boldsymbol{\theta_j} - \gamma \cdot \frac{\partial}{\partial \theta_j} J(\theta)$. Per implementarlo abbiamo bisogno di computare le derivate parziali del costo ella funzione rispetto a ogni parametro.
- scriviamo la funzione costo esplicitando i parametri 0

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}\left(f_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{2}\sum_{i=1}^{N}\left(\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)}\right)^2$$

- possiamo calcolare la derivata parziale del costo della funzione rispetto la j-esima variabile con la seguente formula

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N} 2\left(f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)\frac{\partial}{\partial \theta_j}\left(\theta_0 x_0^{(i)} + \cdots + \theta_n x_n^{(i)} - y^{(i)}\right)$$

- tramite varie osservazioni possiamo scrivere la regola dell'aggiornamento come

$$\theta_j = \theta_j - \gamma \sum_{i=1}^{N}\left(f_{\theta}(\boldsymbol{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

La **VALUTAZIONE** dell'algoritmo di regressione lineare può essere svolta in vari modi:

$$MSE(Y, \hat{Y}) = \frac{1}{n}\sum_{i}(y_i - \hat{y_i})^2$$

- MSE (**Mean Square Error**)

$$RMSE(Y, \hat{Y}) = \sqrt{\frac{1}{n}\sum_{i}(y_i - \hat{y_i})^2}$$

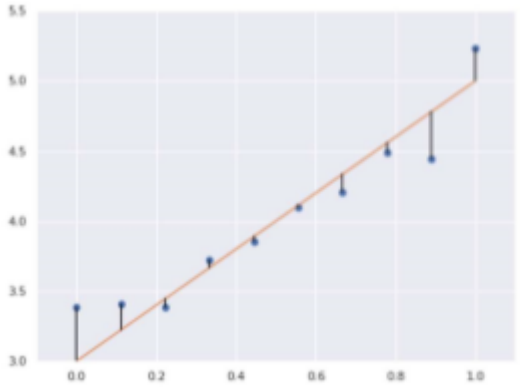
- **RMSE** (**ROOT**)
Questi due errori danno poca importanza ai piccoli errori e grande importanza ai grandi errori

$$MAE(Y, \hat{Y}) = \frac{1}{n}\sum_{i}|y_i - \hat{y_i}|$$

- **MAE** (**Mean Absolute Error**)

Quest'ultimo è piu intuitivo, e rappresenta l'errore medio che si fa quando si stima y. Ha un'interpretazione geometrica in quanto misura quanto è imperfetta la linea di regressione calcolata rispetto all'effettiva posizione dei punti. Esso fa una media delle distanze tra i punti e la retta. Questi segmenti (punto y reale e punto y predetto) sono

chiamati **residuals**, dati dalla formula y-ŷ. Per un buon fitting il segmento deve essere piccolo.



$$MAE(Y,\hat{Y}) = \frac{1}{n}\sum_{i}|y_i - \hat{y}_i|$$

residuals

10) DECIMA LEZIONE 29/11/2019

Una **regressione semplice** è una regressione in cui si ha una sola variabile indipendente.
Una **regressione multipla** è una regressione in cui si hanno piu variabili indipendenti.
Una **regressione multi-variabile** è una regressione in cui ci sono piu variabili dipendenti nel caso in cui $\mathcal{X} = \mathbb{R}^n$ and $\mathcal{Y} = \mathbb{R}^m$;
Ovviamente si tratta di mappare le variabili in input x multi-dimensionali con variabili multi-dimensionali y:

$$\begin{pmatrix} y_1 \\ \dots \\ y_m \end{pmatrix} = \begin{pmatrix} f_1(\boldsymbol{x}) \\ \dots \\ f_m(\boldsymbol{x}) \end{pmatrix}$$

La regressione lineare presenta però un limite di utilizzo in quanto dato una serie di dati che seguono una relazione non lineare (sono messi a caso nel piano, non riconducibili a una retta) non possono fittare una regressione lineare in quanto otterremo un fit chiaramente non ottimale. Questo per il fatto che il nostro modello ha capacità ridotta in quanto stiamo cercando di far fittare ad una funzione lineare, una relazione input-output non lineare e quindi dire che il nostro modello è **under-fitting**.

Problema risolto dalla **REGRESSIONE POLINOMIALE**, facilmente ottenibile dalla regressione lineare semplice (può essere pesante con tante features) senza aumentare la complessità dell'algoritmo, replicando le feature m volte e fino all'elevazione m-esima:

es. $f(x) = 0_0+0_1x$ ------> $f(x) = 0_0+0_1x+0_2x^2+\dots+0_mx^m$

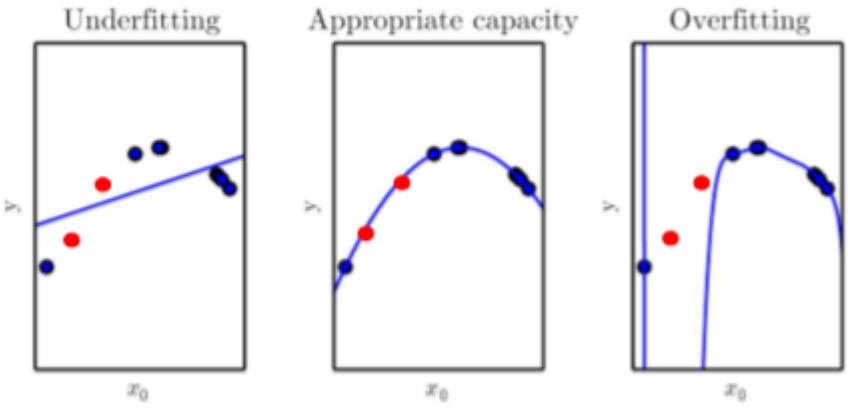
Il vantaggio della regressione polinomiale è quello di ottenere nel grafico di rappresentazione, non piu una linea ma una figura geometrica complessa (es. curva) andando a risolvere in alcuni casi il problema relativo alla capacità ridotta che causa under-fitting nella regressione semplice.
La **regressione polinomiale con features multiple,** ovvero con piu termini è:

$$f(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$f(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2$$

In quanto date due variabili [x1,x2] si ottiene [x1,x2,x1^2,x1x2,x2^2]

Quando quindi il modello ha capacità ridotta (esempio in cui si cerca di far fittare su una linea un set di punti non lineare) si tratta di **under-fitting**.
Quando la **capacità è appropriata** si ha una linea in cui tutti i punti giacciono su di essa.
Quando utilizziamo una polinomiale di grado troppo grande, con un numero di parametri troppo alto, il modello puo perfettamente fittare i dati di training, ma non quelli del test data
Nell'esempio sotto possiamo vedere:
- Underfitting derivato da una regressione lineare con capacità ridotta
- Capacità appropriata con corretta regressione polinomiale
- Overfitting derivato da una regressione polinomiale di troppo alto grado che fitta bene i TR data ma generalizza i TE data.



Possiamo prevenire l'overfitting tramite la **REGOLARIZZAZIONE**, tramite i termini λ detti **PENALIZZATORI** applicati a tutti i parametri tranne che al primo θ_0.

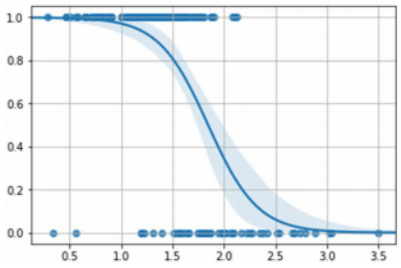
La formula da applicare è $$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}\left(f_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

Es. Partendo dalla funzione $f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$ applichiamo la formula per ottenere $$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}\left(f_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda\theta_1 + \lambda\theta_2 + \lambda\theta_3 + \lambda\theta_4$$

Ottimizzare il costo darà soluzioni piu semplici, se i parametri θ sono veramente piccoli (tendenti a 0) in quanto nell'equazione per es. faremo θx1 dove θ=0 e quindi θx1 = 0 cosi da poterlo levare nell'equazione e riducendo il numero di parametri di essa.
Quindi l'ottimizzazione porta dei risultati vantaggiosi solamente quando un gran numero di parametri θ tende a 0 e specialmente quando ad essere eliminate sono le x di grado elevato in quanto sono quelle che alla fine non ci permettono di far fittare

bene la funzione. Ovviamente potrebbe anche essere che non esiste una funzione in grado di fittare bene i dati e quindi abbiamo sempre un minimo di over o under fitting.

Possiamo predire la classe ŷ dall'input x direttamente come fatto con la RLin quindi y = f(x) = 0^Tx ma tutto cio con dei problemi: per esempio considerando di avere y binario quindi {0,1} tramite RLin otteniamo una linea vicino l'1 quando i valori sono positivi e una linea vicino lo zero quando i valori sono negativi. Cosa succede però se alcuni valori sono >1 e <0? Il codominio è discreto quindi del tipo y{0,1}. Dobbiamo cercare di avere un output percio che da [0,1] deve diventare [-inf,+inf].
Cerchiamo di regredire verso una trasformata versione della probabilità e definiamo l'odd come:
odd(p) = p/1-p in cui **p=P(C|X)** per brevità, osservando che ciò risolve parte dei problemi:
-è definita tra 0 e +inf quindi a piccole probabilità corrispondono piccole odds e a grandi probabilità corrispondono grandi odds (fino a +inf) -> rimane ancora un problema ovvero comprimere tra -inf e +inf e lo facciamo tramite il **Logodd(p) = log(p/1-p)**.
La funzione Logodd è ottimale per un fitting lineare e puo essere riscritta come **Logodd(p) = 0_0+0_1x_1+…+0_nx_n** a noi interessa, una volta allenato il modello, classificare il sample e trovare la probabilità. Dobbiamo trovare quindi la relazione tra logodd e probabilità attraverso l'inversione del logodd chiamata **FUNZIONE LOGISTICA** secondo cui $p = \frac{e^{logodd(p)}}{1+e^{logodd(p)}}$ per poi arrivare a dire che $p = \frac{1}{1+e^{-logo\quad(p)}}$ cioè quindi $logistic(x) = \frac{1}{1+e^{-x}}$.

Una volta risolto il problema della regressione allora andiamo a dire che $logodd(P(y|\boldsymbol{x})) = \boldsymbol{\theta}^T\boldsymbol{x}$ da cui possiamo stimare la **REGRESSIONE LOGISTICA** $P(y|\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T x}}$ che non fitta una curva ma una SIGMOIDE di dati che ha questa forma 

Esso assegna probabilità che va da 1 (elemento certo) a 0.

Regressione Logistica – NEGATIVE LOG LIKELIHOOD, dobbiamo cercare ora una funzione costo per ottimizzare i parametri della RLog in cui mentre in precedenza la RLin regrediva la probabilità P(y|x), ora la probabilità non è coinvolta in quanto si hanno coppie (x,y) dove y è la classe associata ad x**.** I passaggi da seguire sono i seguenti:

$$f_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$P(y = 1|x;\theta) = f_\theta(x)$$

$$P(y = 0|x;\theta) = 1 - f_\theta(x)$$

$$P(y|x;\theta) = \left(f_\theta(x)\right)^y \left(1 - f_\theta(x)\right)^{1-y}$$ dalla prima, sapendo che la seconda e la terza, allora possiamo arrivare a dire la 4a sostituzione in cui se y=1 allora abbiamo $P(y = 1|x;\theta) = f_\theta(x)$ mentre se y=0 P(y=0|x;0)= $1 - f_\theta(x)$.
Possiamo stimare i parametri tramite il **massimo likelihood**, scegliendo i parametri che lo massimizzano sulle variabili random: L(0) = P(Y|X;0); Assumendo che i TR sample sono tutti indipendenti, il likelihood puo essere espresso come:

$$L(\theta) = \prod_{i=1}^{N} P(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{N} f_\theta\left(x^{(i)}\right)^{y^{(i)}} \left(1 - f_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}$$ massimizzare questa espressione equivale a minimizzare il logaritmo negativo del L(0) e quindi

$$nll(\theta) = -\log L(\theta) = -\sum_{i=1}^{N} y^{(i)} \log f_\theta(x^{(i)}) + \left(1 - y^{(i)}\right) \log(1 - f_\theta\left(x^{(i)}\right))$$

Regressione Logistica – DERIVATIVE, possiamo ottimizzare il costo della funzione della RL usando un **gradiente discendente**.
Computiamo prima il gradiente rispetto il coefficiente θ, considerando la derivata della func logistica: $g(x) = \frac{1}{1 + e^{-x}}$ che sarà **g'(x) = g(x)(1-g(x)).**

Regressione Lineare – OPTIMIZZATA, tramite vari passaggi possiamo arrivare a dire che
nll(θ)= $-\sum_{i}^{N} x_j^{(i)} \left(y^{(i)} - g\left(\theta^T x^{(i)}\right)\right)$ e che il gradiente discendete per ogni parametro è

$$\theta_j = \theta_j - \gamma \sum_{i=1}^{N} x_j^{(i)} \left(g\left(\theta^T x^{(i)}\right) - y^{(i)}\right)$$ uguale a quella del caso della regressione lineare.

Cosi come la regressione lineare trova una retta/curva che fitta i dati, la regressione logistica trova un iperpiano che separa i dati.

Consideriamo quindi una $x \in R^2$. Se fittiamo una $f(x) = \frac{1}{1 + e^{-\theta_0 + \theta_1 x_1 + \theta_2 x_2}}$ troviamo i valori dei parametri θ che come sempre ci permettono di calcolare la probabilità. Basandoci su questo valore di probabilità andiamo a classificare i samples. Ci dobbiamo concentrare ovviamente nei valori in cui la funzione f(x) è incerta. Andando a portare tutti i dati a equazione della retta ovvero del tipo

$0 = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ possiamo quindi esplicitare in questo modo: $x_2 = -\frac{\theta_1}{\theta_2} x_1 - \frac{\theta_0}{\theta_2}$ in cui il termine a moltiplicare la x è il coefficiente angolare e l'altro è l'intercetta. Se tracciassimo questa equazione su un piano, otterremo il **decision boundary**, una linea, che separa gli elementi in due classi. Essa è una linea in quanto la Regressione Logistica è un classificatore lineare, nonostante la Funzione Logistica non sia lineare.

Se avessimo piu classi, la regressione logistica non è in grado di risolvere il problema in quanto è lineare quindi puo risolvere solo problemi legati alla classificazione binaria. Tramite alcune modifiche però possiamo estendere i compiti della RLog:
-**ONE vs REST (One vs All) Classification**, permette di classificare tutte le multi-classi in un set binario di classificazione utilizzando un classificatore binario e un **valore di confidenza** come es. una probabilità.
Data una classificazione task T con n classi, lavora in questo modo:

1) Divide i problemi di classificazioni multiclassi in n classificazion di task T_i in cui ogni i è una classe contro tutte le altre. Quindi avendo n=3=classificazioni binarie T0,T1,T2 si fa T0vsResto, T1vsResto, T2vsResto
2) Fitta un classificatore binario f_i per ogni problema di classificazione
3) Per classificare un sample x in una delle n classi, prima lo classifichiamo usando tutti i classificatori binari f_i e dopo lo assegniamo ad una classe che ha ottenuto lo score piu grande (nel caso della regressione logistica questo valore corrisponde alla grande probabilità): $\hat{y} = \arg_i \max\{f_i(\boldsymbol{x}), \forall i\}$
-**ONE VS ONE CLASSIFICATION**, non c'è descrizione
-SOFTMAX REGRESSION, non c'è descrizione

11) UNDICESIMA LEZIONE 04/12/2019

Possiamo applicare quelli gli algoritmi di machine learning per quelli che sono scopi di regressione e di classificazione oltre che per i testi, anche per le immagini. Anche qua abbiamo bisogno di una rappresentazione adeguata dei dati, in grado di soddisfare i tasks del ML.
Mentre un testo è rappresentato da parole che hanno un significato, le immagini sono rappresentate da pixel che non ha significato da soli, se non attraverso l'utilizzo di **patches** (sottoinsieme dell'immagine) ovvero insieme di pixel appartenenti alle sotto immagini che compongono l'immagine.

Nel caso dei testi utilizziamo i token per convertire una sequenza di caratteri in una sequenza di parole basata sul fatto dei suffissi, prefissi, etc. Nel caso invece delle immagini utilizziamo una **GRIGLIA** in grado di dividere l'immagine in patches**.**
Dobbiamo considerare alcuni parametri: **GRANDEZZA DELLA PATCH e FASE DI CAMPIONAMENTO/SAMPLING STEP**¸ quest'ultimo rappresenta il numero di pixel di cui ci dobbiamo spostarci per campionare il quadratino successivo. Basandoci su questi 2 parametri possiamo ottenere **overlapping**, se la size/2 del patch è piu grande del sampling, o **non-overlapping**, se la size/2 del patch è piu piccola del sampling.
Consideriamo quindi, dopo l'estrazione delle patches, l'immagine come una **"bag of patches"**.
Un'analisi statistica puo rilevare cosa raffigura un'immagine cosi come un'analisi statistica del testo puo rilevare l'argomento del testo.
Dopo aver diviso l'immagine in patches dobbiamo seguire i seguenti step:
- **DEFINIRE UN VOCABOLARIO DI PATCHES**. Se abbiamo un numero di patches molto alto dobbiamo scegliere un numero limitato da scegliere nel vocabolario. Nel vocabolario non vengono ripetute fatches uguali. Es. un'immagine in cui è presente il cielo nella parte superiore, avrà magari 1/2 patches nel vocabolario per la rappresentazione del cielo, su quelle che invece sarebbero state 20+ patches in cui il cielo era presente nell'immagine.
- **CREARE UNA BOW** per contare il numero di occorrenze di ogni patch nel

vocabolario
Abbiamo però un problema di rappresentazione in quanto non siamo ancora in grado di dire se due patches sono simili tra loro e quindi dobbiamo trovare uno spazio di rappresentazione in grado di farlo che magari nella rappresentazione risultano vicini se simili e lontani se diverse.

Per la rappresentazione utilizziamo la funzione φ che mappa patches di immagini in uno spazio di rappresentazione secondo le seguenti caratteristiche:
- patches simili sono mappate vicine nello spazio di rappresentazione
- patches non simili sono mappate lontane.
Possiamo performare 2 tipi di classificazione di tasks:
- **CLUSTERING**, in cui ogni cluster (gruppo di patches simili) rappresentare una parola del vocabolario.
- MATCHING, data una nuova patch possiamo mapparla nel cluster corretto data la sua rappresentazione nello spazio.

Un metodo di rappresentazione delle patches è **L'ISTOGRAMMA PONDERATO DEI BORDI ORIENTATI** che segue i seguenti step:
-dato un input di patches, lo converte in scala di grigi
-applica i due filtri sobel X e sobel Y all'immagine e tramite essi ottenere un numero che viene utilizzato per calcolare la grandezza e l'orientamento dei bordi
-Selezioniamo le orientazioni principali che sono [0,45,90,135,180,225,270,315] espresse in gradi in quanto sono angoli, computando l'istogramma di orientamento dei bordi su tutta l'intera patch.
-ogni bin (rettangolo) dell'istogramma è pesato dal valore di ogni bordo. Tutto ciò al fine di ridurre il contributo dei bordi "deboli" per la rappresentazione finale.
-In un istogramma regolare ogni volta che troviamo un pixel che abbia orientamento compreso tra [0,45] aggiungiamo 1 al bin corrispondente. In un istogramma pesato invece per un pixel con orientamento tra [0,45] con **magnitude** m, sommiamo m al corrispettivo bin.
In ogni caso l'istogramma è normalizzato dividendo ogni bin per la somma di tutti i bin.
Abbiamo definito un modo semplice per mappare le patches delle immagini con una rappresentazione a lunghezza fissa = 8 (numero di bins)

L'istogramma pesato serve quindi a descrivere in maniera semplice le patches e sicuramente per immagini simili avremo un istogramma simile.

Descrittore SIFT sfrutta sempre gli istogrammi pesati ma rispetto quelli semplici è una maniera piu strutturata e potente in quanto permette di mappare il patch di un'immagine ad un vettore di 128 dimensioni, in cui la parola descrittore indica che l'algoritmo descrive le patches. I principi sono:
- Il patch è diviso in una griglia 4x4
- Per ogni cella della griglia, viene visualizzato un istogramma ponderato degli orientamenti del gradiente calcolato quindi si fa 4x4x8 = 128 elementi, dove 4x4 rappresentano le patches della griglia, mentre 8 rappresentano i vari angoli visti precedentemente.

Rappresentazione = descrittore -> Bisogna creare il Vocabolario in modo tale da inserire solo alcuni descrittori. Raggruppo tutti i vettori delle rappresentazioni in base a quanto essi sono simili. Inoltre, quando una nuova patch arriva, dobbiamo essere in

grado di assegnarle un descrittore incluso nel vocabolario. Possiamo utilizzare l'algoritmo di clustering chiamato **K-MEANS** per raggruppare i descrittori in un numero di gruppi K che è il parametro specificato dall'utente da cui dipende l'efficacia dell'algoritmo. L'output dell'algoritmo è un gruppo S = {Si,…,Sk} che rappresenta la partizione del TR set.
È un algoritmo non-supervisionato quindi funzione f(x) in cui a x in input non è associata nessuna label y.
Il funzionamento è il seguente:

Al K-MEANS viene dato in input un set di TR examples $TR = \{x^j\}_j$. Il mean point di ogni cluster è detto **CENTROIDE** calcolato secondo la formula: $\mu_i = \frac{1}{|S_i|}\sum_{x \in S_i} x$

Vogliamo all'interno di ogni partizione che la varianza (la lunghezza del diametro) sia minima quindi $\hat{S} = \arg_S \min \sum_{i=1}^{K} \sum_{x \in S_i} ||x - \mu_i||^2$
Si segue un **ALGORITMO ITERATIVO**:
1) Facciamo **l'INIZZIALIZZAZIONE** prendendo un numero k di punti in maniera casuale che vengono usati come centroidi.
2) Assegniamo agli elementi il gruppo, calcolando la distanza dell'elemento preso in considerazione con il centroide piu vicino.
3) Si aggiornano i centroidi computando i punti medi per ogni centroide e muovendoli nella loro nuova posizione. Ricalcolo ora le distanze minime e ho un PROBLEMA -> alcuni elementi cambiano gruppi di appartenenza -> rifaccio l'assegnazione degli elementi ai gruppi corretti.
N.B. Ci sarà un momento in cui l'aggiornamento non farà piu muovere i centroidi e ciò vorrà dire che l'algoritmo è giunto a convergenza.
Se dovessi scegliere per sfiga dei centroidi "sbagliati" che non convergono mai ma che con l'aggiornamento cambiano sempre posizione l'algoritmo si loopa, la posizione di essi non sarà mai ottimale e quindi è uso comune bloccarlo dopo ad dato numero di volte es. 1k di prove, si sceglie un numero alto in quanto magari l'algoritmo potrebbe aver bisogno di tanti aggiornamenti per convergere.
4) Per un nuovo data point inserito (non contenuto nel TR set ovviamente) andiamo a calcolare la distanza minima con i diversi centroidi e il piu vicino prende il nuovo elemento nel suo gruppo.

Ora che abbiamo fatto i seguenti passaggi → estrazione delle patches -> descrizione patches con SIFT -> clustering con K-Means -> Creazione del vocabolario -> possiamo creare la **BOVW Bag Of Visual Words** seguendo i seguenti passaggi:
1) Creazione del vocabolario di **VISUAL WORDS** in cui dobbiamo considerare tutte le immagini del TR -> estraiamo le patches da tutte le immagini del TR -> ogni patch sarà descritta e diventa un vettore multidimensionale utilizzando SIFT -> Clusteriamo i vettori utilizzando il K-MEANS -> ogni centroide è considerato quindi una visual word e il vocabolario costituito da tutti i centroidi.
NB. Len(Centroidi) = len(Descrittori) -------- Il numero di visual words nel vocabolario è = num di clusters K

2) Estraiamo i patches dall'immagine, descriviamo i patches usando lo stesso descrittore utilizzato in precedenza (SIFT) per creare il vocabolario, assegnare ogni descrittore a uno dei visual words usando l'assegnamento NN , ogni immagine puo

essere vista come un insieme di words prese da un vocabolario e possiamo applicare la tecnica utilizzata per i testi per ottenere una rappresentazione BOW e quindi ad esempio la normalizzazione o TF-IDF.

Possiamo sfruttare la bag of Visual Words per ottenere una rappresentazione di lunghezza fissa da un'immagine in input. Possiamo quindi a questo vettore di lunghezza fissa applicare un algoritmo di ML come classification e regressione.

Content-Based Image Retrieval (**CBIR**), tecnica per la rappresentazione di immagini → data un'immagine, tramite una query ad un database grande, chiediamo dare in output un'altra simile.
Task facile da risolvere nel caso dei testi.
Per quanto riguarda le immagini abbiamo di una buona rappresentazione di esse.
Possiamo strutturare le immagini del database e l'immagine passata alla query in modo tale da ottenere tramite distanza minima e quindi sfruttando l'algoritmo di K-NN solo per le prime k immagini piu vicine (quindi piu simili) all'immagine passata.

12) DODICESIMA LEZIONE

SISTEMI DI RACCOMANDAZIONE sono classi degli applicativi web che posso predire quale sarà la scelta futura di un utente. Es. in un giornale consigliare un articolo che il lettore leggerà in base a quelli che sono i suoi interessi; consigliare una determinato prodotto in base ai prodotti comprati in precedenza o in base a quelli inseriti nei preferiti.
Questi sistemi sono in grado di predire i prodotti che ad un determinato utente potrebbero piacere. Vengono anche predetti dei prodotti che l'utente non ha mai visto ma che potrebbero piacergli.

Mentre il negozio fisico non puo mostrare tutti gli articoli in quanto soffre di problemi di "spazio fisico", ma cerca di mettere in mostra quelli che hanno sempre maggior vendita, il negozio online è in grado di eliminare questo problema in quanto possiede una quantità di spazio "infinita". Netflix > blockbuster ; Spotify > Physical Store. Gli elementi poco conosciuti non saranno mai presenti nello store fisico in quanto è piu difficile venderli, ma saranno sicuramente presenti in un digital store e magari potranno essere suggeriti agli utenti che hanno avuto determinate preferenze.

GETTING DOWN THE LONG TAIL, grafico in cui è facile vedere la differenza tra un digital store e un physycal store in termini di vendite: un prodotto che viene venduto piu facilmente si trova piu a sinistra nel grafico (prodotto nello store fisico) mentre un prodotto che viene venduto con poca frequenza si trova nella in fondo alla coda della curva (prodotto nello store digitale).

Tramite l'utilizzo delle **raccomandazione** vengono suggeriti all'utente tutti quegli items, simili ad un altro item che gli è piaciuto in passato, che potrebbero piacergli.
Es. "into Thin Air (1997)" raccomandava "Touching the void (1988)" che è diventato poi piu venduto di into.

L'obiettivo di un sistema di raccomandazione è predire l'interesse di un utente tramite alcuni items.
Definiamo un **set X** di utenti e un **set S** di items. L'obiettivo è quello di assegnare un valore reale ad una coppia (x,s) che quantifica quanto all'utente x potrebbe piacere l'item s, tramite la funzione: $u: X \times S \rightarrow R$ dove R è il **set di valutazione** (esempio le stelle di valutazione di un prodotto da 0-5 stelle).
Si introduce anche il concetto di **UTILITY MATRIX U** ovvero una matrice utente-items

in cui:
ogni riga è un utente x, ogni colonna è un utente s, ogni elemento della tabella r(x,s) rappresenta un rating di un utente su quell'item.
Lo scopo di questa matrice è quello di andare a prevedere un determinato valore r(x,s) negli spazi bianchi della matrice in modo da dare ad un item non conosciuto dall'utente un valore che fa capire se all'utente potrebbe piacere o meno quell'item.
La funzione scelta per la predizione è l'**UTILITY FUNZIONE U**.
Il problema di questo sistema è rappresentato da 2 problemi principali:
- Popolare la matrice -> per un nuovo utente abbiamo il problema del **cold start problem**. Non sappiamo niente dell'utente e quindi possiamo affrontare il problema chiedendo all'utente in maniera esplicita di valutare alcuni items (potrebbero seccarsi per questa cosa) oppure in maniera implicita (es. acquisto di un prodotto da alto punteggio)
- Trovare una corretta funzione utilità per prevedere i nuovi ratings da quelli che gia si sanno. Questo tipo di problemi vengono risolti dai **FEATURE BASED SYSTEMS** o da **COLLABORATIVE FILTERS**

CONTENT-BASE RECOMMENDATION SYSTEMS, sono quei sistemi che sono in grado di dare un "voto" ad un item senza che l'utente ci abbia mai interagito (es. nei film dato che si è visto un film con un attore, dare come consiglio gli altri film con quell'attore). Tutto ciò avviene secondo una procedura:
- La matrice utilità viene creata partendo da un soggetto che ha valutato gia diversi prodotti
- Ad ogni item vengono associate una o piu proprietà (es. ad un utente piacciono i triangoli e i cerchi rossi, allora gli verranno consigliati i triangoli e i cerchi o le figure di colore rosso)
- Costruiamo grazie a questi dati una "**USER PROFILE**" che raccoglie tutte le features degli oggetti liked dall'utente. Ogni item presenta un vettore di caratteristiche che lo descrivono. L'user profile e il vettore di feature dell'item sarà dello stesso tipo, ovver: se per un determinato item troviamo come feature Action_movie e romantic_movie allora nell'user profile ci saranno questi campi per specificare che all'user piacciono quelle feature.
- Ultimo step è fare match tra l'user profile e gli oggetti ancora non valutati dall'utente. Se ovviamente questi nuovi oggetti condividono delle caratteristiche in comune allora vuol dire che ci potrebbe essere interessamento.

Dobbiamo costruire anche quello che è un "**ITEM PROFILE**" identificando tutte le caratteristiche dell'item (es. film, attori/trici, regista, genere).

Nel caso di testi o immagini è difficile capire quali features dobbiamo prendere e come estrarle. Un approccio è l'utilizzo di TF-IDF per estrarre parole significative da un documento secondo i passaggi: tokenizzare tutti i documenti, eliminare le stop words, computare il TF-IDF per tutte le parole, prendere le n parole piu ripetute ed utilizzarle come features per il documento in quanto ci diranno qualcosa su esso.
Dopo aver assegnato ad ogni item un set di features dobbiamo trasformare questo insieme di set in **VETTORI DI RAPPRESENTAZIONE** in cui troviamo 1 se una determinata feature è presente, altrimenti 0.

Quando i dati non sono così ben strutturati ma sono rappresentati in maniera complessa si utilizza allora Bag Of Words (text) oppure Bag Of Visual Words (img), che nonostante non ci rendono in grado di comprendere i vettori delle caratteristiche, è uno strumento molto potente in molti casi.

Per computare il vettore di features di un item utilizziamo la **FUNZIONE DI RAPPRESENTAZIONE F** che ci ritorna il vettore tramite $f(s_i)$ in cui s_i è un item. Quando la matrice di utilità è binaria, possiamo calcolare un profilo tramite la formula:

$$profile(x_i) = \frac{1}{\sum_j U(x_i, s_j)} \sum_{j=1}^{n} U(x_i, s_j) f(s_j)$$

che fa una media pesata delle diverse features dei diversi items che contengono il valore 1.

Quando la matrice di utilità non è binaria, normalizziamo le utilità sottraendo il mean utility value. Se il ratings è sotto la media allora ha valore neg, se sopra ha valore positivo. Ci permette di contare lo "scale" di ratings per ogni utente. La utility media dell'utente x è

$$\overline{u_i} = \frac{1}{\sum_j [U(x_i, s_j) \neq 0]} \sum_{j=1}^{n} U(x_i, s_j)$$

Possiamo ottenere il profico come

$$profile(x_i) = \frac{1}{\sum_j [U(x_i, s_j) \neq 0]} \sum_{j=1}^{n} (U(x_i, s_j) - \overline{u}_j) f(s_j)$$

Avendo ora i profiles utente e degli item possiamo raccomandare gli item calcolando le similitudine tra il feature vectore di un item ed quello di un user profile in quanto essi sono con uguale lunghezza e uguale significato: una delle tecniche utilzzate per calcolare la similitudine è data dalla legge del coseno $\cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$ che calcola l'angolo compreso tra i due vettori. Il dominio è {-1;1} se i due vettori sono simili allora la similarità sarà circa 1; se sono completamente diversi vicino -1.

La funzione utilità verrà definita come $u(x_i, s_j) = \cos\left(profile(x_i), f(s_j)\right)$.
Possiamo quindi ora valutare tutti gli item che l'utente non ha valutato utilizzando la funzione utilità.

Quali sono i **PRO** del C-B Recom Syst:
-nessun dato sull'user -possibile raccomandazione all'user a cui piace solo un genere
-consigliare items non popolari -> non c'è il problema del first-rater
-possono dare spiegazione del perché di alcuni consigli

Quali sono i **CONS** del C-B Recom Syst:
-Trovare i features appropriati è difficile
-Cold Start problem per utenti che non hanno mai valutato items
-**Overspecialization**, tende a consigliare items simili ad altri già consigliati ma potrebbe essere che le persone abbiano interessi vari. Non sfruttano il giudizio di altri utenti su un determinato item.

COLLABORATIVE FILTERS sono dei filtri per la produzione di recommendations basati **sull'osservazione** che utenti con gusti simili daranno valutazioni simili agli items. Un filtro Collaborative funziona in questo modo:
-Si considera l'utente X e si trova un set S di utenti i cui ratings sono simili a quelli di x
-Si stimano i mancanti ratings di x dai rating degli users in S.

Dobbiamo quindi capire come misurare quanto sono simili gli utenti tra loro. Assumiamo di avere una **matrice di utilità binaria**. Possiamo andare a misurare quanto due utenti sono simili, contando quanti items i due sets hanno in comune,

tramite la **similarità di Jaccard** $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ in cui A e B sono i set di due utenti che hanno forma User A: {HP1, TW, SW1} User B: {HP1, HP2, HP3, TW, SW1}. Il numero similarità di Jaccard è compreso nel dominio [0,1] e conta il numero di items che due set hanno in comune. Se due set sono uguali la similarità è 1, 0 altrimenti.
Considerando gli user A e B si ottiene un J(A,B)=3/5 dove 5 è il numero totale di elementi mentre 3 è il numero di elementi in comune tra A e B.

Data una matrice non binaria (al posto di 1-0 troviamo il ranking di stelline assegnato a quell'item che va da 1-5 stelle) invece possiamo trasformarla in binaria. Si puo fare come vuole ma per esempio possiamo cambiare i valori <=3 a 0 e i valori >=4 a 1 ottenendo una matrice binaria e quindi potendo utilizzare la formula di Jaccard.
Sorge però un problema in quanto due utenti a cui non piacciono gli stessi film sono due utenti che potrebbero avere gusti simili, però andando a lavorare con la binarizzazione non è possibile fare questa distinzione.

Un modo per risolvere questo problema è tenere conto dei valori del rating andando a riempire gli elementi bianchi della matrice con 0. Qua nasce un altro problema che noi stiamo considerando elementi mai valutati come elementi valutati 0 e cio causa una limitazione. Quindi un modo per evidenziare la differenza tra alti e bassi è sottrarre in ogni riga il valore medio di tutti i ratings in modo da rendere negativi le valutazioni basse e positive le valutazioni alte.
In fine utilizziamo la funzione coseno e otteniamo un numero che va tra [-1,1].

L'obiettivo finale di un sistema di raccomandazione è predire quelli che sono i valori mancanti della matrice di utilità ovvero il valore della funzione $u(x_i, s_j);$ dove x è l'user e s l'item. Un metodo per ottenere questa predizione con un filtro collaborativo è:
- considerare l'user x e un item s che non è mai stato valutato da x
- trovare gli N set dei K utenti che hanno valutato s e che sono simili a x
- Dati i K utenti ottenere una media dei ratings tramite $u(x_i, s_j) = \frac{1}{K} \sum_{x_k \in N} r(x_k, s_j)$
- Alternativamente si puo pesare il raking per similarità:

$$u(x_i, s_j) = \frac{\sum_{x_k \in N} \cos(profile(x_i), profile(x_k))\, r(x_k, s_j)}{\sum_{x_k \in N} \cos(profile(x_i), porfile(x_k))}$$

È possibile oltre un user-user collaborative filtering (quelli visti fin qui) utilizzare anche un **item-item collaborative filtering**.
Consideriamo un utente x e un item s che non è stato valutato. Tramite profile(s) per ogni item consideriamo le colonne della matrice di utilità normalizzando sottraendo la colonna mean.
Troviamo un set N di items simili che sono stati valutati da x. Stimiamo u(x,s) calcolando una media pesata tramite la formula

$$u(x_i, s_j) = \frac{\sum_{s_k \in N} \cos\left(profile(s_j), profile(s_k)\right) r(x_i, s_k)}{\sum_{s_k \in N} \cos\left(profile(s_j), profile(s_k)\right)}$$

In pratica item-item funziona bene perche gli users tendono ad avere diversi gusti

Da pagina 32 a 39 per un esempio:
- sceglie i film 3 e 6 per valutare 1 in quanto sono i film piu simili all'1 e si vede

dall'indice di valutazione verde (s(1,3)=0,41 e s(1,6)=0,59) valutati secondo la formula sim(1,m) dove m non si sa cos'è.

Per valutare il rating che l'utente 1 darebbe al film 5 utilizziamo $r(x_5, s_1) = \frac{\sum_{s_j \in N} sim(s_1, s_j) r(x_5, s_j)}{\sum_{s_j \in N} sim(s_1, s_j)}$

13) TREDICESIMA LEZIONE

WORD SEMANTICS, abbiamo fin qui parlato di parole come simboli che non hanno significato. Per noi invece le parole hanno significato e quindi possono anche non significare niente a livello "macchina" ma possono avere un significato profondo come per esempio le frasi "i believe that the heart does go on" e "All you need is love" per la macchina sono due frasi diverse, mentre noi siamo in grado di capire che hanno l'argomento dell'amore in comune. Per fare quest'associazione dobbiamo essere in grado di dare una semantica che ci dia similarità tra le parole. Tutto ciò si riduce a misurare la distanza tra parole: parole con simili semantica avranno distanza minore.

Per misurare la distanza tra le parole prima le mappiamo tramite una funzione di rappresentazione f e poi calcoliamo la distanza tramite una **FUNZIONE DISTANZA d** che del tipo d(love, Heart)=d(f(love),f(Heart)).

Come sempre dobbiamo trovare una funzione f di rappresentazione nello spazio in cui poi possiamo andare a misurare la distanza tra le parole.
Come facciamo a capire se due parole sono simili? Presa una parola possiamo guardare alle parole vicine: questi sets di parole li chiamiamo **context** per una data parola (sit, comfortable, napping oppure sit,comfortable,sleeping). Parole simili hanno context simili e quindi rappresentazioni di piccolo distanza.
Queste rappresentazioni vengono chiamate **WORD EMBEDDINGS**.

Per capire se una parola appartiene al contesto di un'altra parola costruiamo una **CO-OCCURENCE MATRIX**. Dato un insieme di documenti, un vocabolario V, la matrice di co-occorrenze X sarà composta da:

- X_{ij} il numero di volte che la parola j si ripete nel contesto della parola i
- $Xi = \sum_k X_{ik}$ il numero di volte che ogni parola appare nel contesto di i
- Definiamo $P_i=P(j|i) = x_{ij}/x_i$ la probabilità che la parola j appare nel contesto della parola i.

Un algoritmo di **APPRENDIMENTO DI PAROLE EMBEDDINGS** chiamato **GLOVE** in cui data una parola i noi vogliamo imparare un embedding di essa ovvero una rappresentazione di lunghezza fissa w con dimensione d tale che $w_i \in \Re^d$. Una coppia di parole i e j ha caratteristica di probabilità P_{ij}. Dato che abbiamo detto che simili parole hanno simile contesto ci aspettiamo che due parole i e k sono simili se $P_{ij} \approx P_{kj}, \forall j$ e se noi imponiamo che $w_i^T w_j = P_{ij}$ allora la nostra embedding ha la proprietà desiderata.
Date due parole simili i e k osserviamo $P_{ij} \approx P_{kj} \Rightarrow w_i^T w_j \approx w_k^T w_j \Rightarrow w_i \approx w_k$

Se oltre la prima imposizione imponiamo anche che $w_i^T w_j = \log P_{ij}$ e quindi dato che $P_{ij}=x_{ij}/x_i$ abbiamo che $w_i^T w_j = logX_{ij} - logX_i$ e quindi dato che il termine X_i non dipende

dalla coppia (i,j) possiamo dire che $w_i^T w_j = logX_{ij}$. Nel modello vengono usate due diversi embeddings $w_i \in \Re^d$ and $\widetilde{w_i} \in \Re^d$.
Per avere l'embeddings adatto dobbiamo definire una funzione costo da minimizzare

$$J(W,\widetilde{W}) = \sum_i^{|V|} \sum_j^{|V|} \left(w_i^T \widetilde{w}_j - logX_{ij}\right)^2$$

in cui W è la matrice che contiene nelle righe la parole embeddings wi, Wsegnato contiene nelle righe le parole embeddings di wi segnato. Le due matrici sono i parametri del modello.
La somma sopra non è definita nel caso di Xij=0 e quindi non puo essere calcolata in quei casi. Il problema lo risolviamo aggiungendo una funzione pesp

$$J(W,\widetilde{W}) = \sum_i^{|V|} \sum_j^{|V|} f(X_{ij})\left(w_i^T \widetilde{w}_j - logX_{ij}\right)^2$$

che assume valore

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha} & if\ x < x_{max} \\ 1 & otherwise \end{cases}$$

in cui alpha=3/4 e xmax=100. Quando Xij=0 allora la funzione =0. Il suo valore invece aumenta quando il peso delle coppie si alza verso 1.

GLOVE OPTIMIZATION - Per trovare la parola embeddings dobbiamo fare in modo di ottimizzare il costo della funzione J con rispetto verso i parametri W e W segnato. Possiamo farlo o tramite least square o tramite gradiente discendente. Per l'ottimizzazione troviamo $W^*, \widetilde{W}^* = arg_{W,\widetilde{W}} \min\left(J(W,\widetilde{W})\right)$ che contiene TUTTE le parole embeddings. Ogni parola è rappresentata da 2 embeddings wi e wi segnato quindi possiamo dire che $\overline{w}_i = w_i + \widetilde{w}_i$→ [Fin qui tutte le volte che abbiamo trovato Wsegnato si indicava il 3° w dell'equazione].

Nello spazio embedding la similitudine semantica è riflessa nella similitudine geometrica.

Questa proprietà geometrica della parola embeddings permette di ottenere un word vectors math e quindi si possono compiere delle operazioni matematiche come per esempio king-man+woman = queen oppure Paris-france+Italy=rome.

WORD2VEC ci permette di ottenere una rappresentazione delle parole. Un modo facile per rappresentare una frase o un documento è fare la media di parole embeddings dit tutte le parole della frase. Data una frase e trovate le parole embeddings, facendo la media di queste ultime andiamo a trovare una frase rappresentativa a lunghezza fissa che poi puo essere utilizzata per algoritmi di ML come ad esempio la text classification.

Dall'estrapolazione dei testi, come ad esempio i commenti nei social media oppure i commenti in un prodotto di amazon, possiamo andare a compiere quella che è **L'ANALISI SENTIMENTALE** ovvero capire se quel testo è positivo, negativo o neutrale.
Le situazioni piu difficili da andare ad analizzare sono le <u>frasi sarcastiche</u> (bello come un gelato sciolto), le <u>comparazioni</u> (la camera canon è meglio di quella fisher price), <u>gli slang</u> (imma that peng thing).

L'analisi puo essere svolta andando ad assegnare alle parole un peso

positivo/negativo. Partiamo da un Training Example che utilizziamo come set con parole + valori positivl/negativi rappresentandole con un sacco di parole e dopo utilizzare una regressione logistica. I pesi ci diranno quanto una parola puo essere positiva/negativa; pesi che possono essere assegnati pure a parole doppie tipo better than o the worst. Il training della regressione logisitca però non è sempre fattibile in quanto un sistema di regressione logistica utilizzato per “revisioni sui film” potrebbe non essere buono per “post dei social media”.

Per risolvere il problema che la regressione logistica non è sempre utilizzabile adottiamo **l’algoritmo VADER** Valency Aware Dictionary and sEntiment Reasoner che è un’analisi sentimentale basata sul **lexicons** legato ai sentimenti in cui si vuole assegnare uno score a tutta la frase. Ovvero parole che sono in grado di dare polarita = positive, negative, neutrali. Vengono anche assegnati alla parola 5 semplici euristici per i casi speciali come punteggiatura, capitalizzation (maiuscola) , modificatori, but, sentence e negazioni.

Il lexicons include parole, emoticons (“:)”), slang (“LOL”). Ogni lexicon è valutato tramite un numero che sta nell’intervallo [-4;+4]. Lo score finale per ogni lexicon è ottenuto facendo la media dei risultati ottenuti dalle differenti annotazioni (pag 17 tabella con i pesi).
Vader ha lo scopo di assegnare ad una frase completa un sentiment score.

Si fa una somma di tutti i lexicons contenuti nel testo $z = \sum_i s(w_i)$ dove wi è la parola nel testo e s(wi) è il sentimento della parola wi, z è la somma dei valori sentimentali associati ad ogni parola.

Valore di z poi utilizzato nella formula $x = \frac{z}{\sqrt{z^2 + \alpha}}$ per ottenere il sentimento x finale in cui a=15. La normalizzazione porta poi il valore assoluto di x ad essere tra [-1,+1].

Dopo aver computato il totale sentimento della frase, VADER considera la punteggiatura. Vader modifica poi nuovamente il sentimento della frase in questo modo:
- Se il sentimento è positivo viene aggiunto 0.292 per ogni ! e 0.18 per ogni ? ; se negativo sottratti i valori
- Per ogni parola scritta in maiuscola (capitalizzation) abbiamo 0.733 al sentimento della parola
- I modificatori come <<very>> e <<sort of>> che aggiungono 0.293 allo score sentimentale della frase. Un secondo modificatore il 95% di 0.293 mentre un terzo il 90%
- La polarità shifta da positiva a negativa o vicerversa quando vengono utilizzate le parole tipo <<but>> perciò la parte sentimentale di tutte le parole prima di but è decrementato del 50%, di quelle dopo invece incrementato del 150%
-Ci sono un set di espressioni in grado di svoltare il sentimento di una parola come <<isn’t really that>> great, in quando great da solo sarebbe positivo ma con quelle parole cambia polarità. Vader controlla questi casi analizzando i trigrams che precedono la parola a cui è associato sentimento (great). Quando il trigram è trovato il sentimeto della parola è decrementato dello 0,74.
Vader ritorna una lista di valori per ogni testo:
-Positive: la percentuale di parole nel testo con sentimento positivo
-Neutral: la percentuale di parole non associate a sentimento

-Negative: la percentuale di parole nel testo con sentimento negativo
-Compound: il sentimento generale associato al testo.