Simone e Marco – MLTOX

Pierpaolo Brutti

June 2020

Per iniziare... dal 24 aprile al 5 maggio – 2020

Lista di features che andrebbero incluse nel dataset:

- **KoW** (o logKoW): indica quanto facilmente il composto viene assorbito **già inclusa**
- **peso molecolare** (lo prendi direttamente dallo smiles): influenza l'assorbilità **TROVATO ed inclusa**;
- **numero di gruppi OH**: influenza la solubilità del composto **TROVATO**
- **BCF values** - includile nel dataset (x esempio da chemspider), ma verifica se sono misurate o predette, perché se sono predette le hanno inferite dal KoW ed è importante saperlo.
- **Henry's Law constant** (logHLC) è una misura della volatilià, per cui potrebbe influenzare i risultati per esposizioni static o dosed (ma non flow). Ci sarà bisogno quindi di tenere conto anche della variabile "exposure", che appare direttamente sui dati di Ecotox.
- **water solubility**
- **logD**: credo che si chiami lipofilicità. È una alternativa al KoW, ma è più efficace per gli agenti chimici neutri. **TROVATO non inclusa**

  https://www.cambridgemedchemconsulting.com/resources/physiochem/logD.html

Rappresentazione e controllo albero specie

Una volta che hai filtrato Fish, dovresti rimanere con le seguenti categorie:

- Class (ce ne sono 4)
- Order (35)
- Family (137)
- Genus (473)

- Species (966)

Farei queste 2 cose (puoi anche farle nella stessa funzione):

- verificare che è un albero, ovvero che se due specie condividono genus, condividono anche la famiglia. Per questo non è necessario che produci un grafico, basta che fai il loop di codice che esegue il test e dice che è tutto a posto.
- misurare le distanze sull'albero

Può essere carino anche rappresentare l'albero, come albero o come insiemi inglobati.

Distanze

Abbiamo due tipi di distanze. Distanze tra specie, e distanze tra agenti chimici. Le distanze tra specie le plottiamo facendo riferimento alla loro classificazione (regno, etc...). Per le distanze tra chemicals usiamo il Tanimoto index (più avanti forse useremo pubchem2D vectors e autoencoders).

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4456712/

Non è difficile, basta usare pybel. Da fare: produrre un grafico di tutte le distanze tra composti. Magari disegna il grafo e identifica comunità con qualche libreria

Modelli

- random forest: utilizzare `h2o` sia per categoriche che continue (interesse maggiore);
- linear regression;

Riassunto lavoro per 5 maggio 2020

Come abbiamo detto, per il 5 maggio sarebbe ottimale se riuscissi a:

- completare il dataset (features+distanze);
- implementare i modelli che abbiamo detto, per la classificazione binaria;
- mettere il codice e grafici su github

Eventuali sviluppi

Giorno 1

Ho provato ad utilizzare una libreria trovato per caso chiamata thermo, ma in primo luogo, gli SMILES trovati da questa libreria non coincidono con quelli trovati dalla libreria webchem su R. In secondo

luogo, molti dei composti non sono presenti nel dizionario generale di thermo, perciò si ottengono circa 1000 composti senza proprietà.

Nel pacchetto webchem, sempre utilizzando la funzione *cirquery* è possibile estrarre il peso molecolare.

Che gli smiles non coincidano è un pochino preoccupante. Riesci a capire perché? Che ne manchino molti non importa. Il peso molecolare, se c'è una funzione per prenderlo, usala pure. Altrimenti basta prendere la somma dei pesi molecolari degli atomi che compongono il composto.

Non coincidono per pochi dettagli: ad esempio, se tra due atomi c'è un doppio legame, la libreria *webchem* lo indica mettendo [atomo1][atomo2], mentre *thermo* lo indica con atomo1=atomo2. E anche l'ordine di rappresentazione ad esempio - vado a memoria - *webchem* individua un composto come [Cl][Cl][Cm+] mentre *thermo* lo individua al contrario [Cm+][Cl][Cl].

Ah ok, quindi è solo una questione di convenzione. Teniamo quelli di webchem allora, e poi se vediamo che è necessario integrare facciamo una funzione che porta da uno all'altro.

Giorno 2

Sono riuscito a trovare il peso molecolare di molti composti, ora passo ad un'altra possibile feature da implementare e poi unisco tutto. Mentre cerco altre feature, faccio anche alcuni modelli. Metterò tutto al più presto su GitHub... appena capisco come si fa... :)

Per il numero di gruppi OH, posso contare direttamente dentro gli SMILES? Facendo così, risulta non molto buona in quanto la maggior parte dei composti (2727 su 3201) hanno 0 gruppi OH, 6 su 3201 hanno 1 gruppo OH, 2 su 3201 hanno 2 gruppi OH, 1 su 3201 ha 3 gruppi OH e 0 su 3201 hanno > 4 gruppi OH. Si, prendili direttamente dagli smiles. Però non mi torna la somma: $2727 + 6 + 2 + 1 < 3201$. Mancano 465 composti all'appello.

I 465 composti sono quelli Not Available.

E' probabile che i miei predecessori abbiano già trovato il KoW index. Cercando in giro lo chiamano anche $\log P$ e quindi è già stato incluso nell'analisi. Va bene. Puoi verificare la fonte da cui lo prendono, per essere sicuri? Lo hanno ricavato da rdkit nel modulo Chem.Crippen c'è la funzione 'MolLogP'.

Inoltre sempre in rdkit, ho trovato la lipophilicity, ma tutti i risultati coincidono con LogP, quindi non la includo. Se differiscono è per millesimi (es. 1,777 a fronte di 1,7771). Ok.

Giorno 3

Penso di aver fatto il logarithmic binning dei dati come mi chiedevi. Sta nel file *analisi 1-logbinning*. Ho modificato la funzione che mi

avevi dato così che restituisse solo i bins. Non so se è giusto, in quanto non l'ho mai fatto.

**** Ti ho aggiunto un paio di celle per fare il binning logaritmico nel plot delle concentrazioni (basta che vedi l'ultimo commit - è il primo notebook). Ora dovresti essere capace di farlo molto facilmente ovunque sia necessario. Dimmi pure se c'è qualcosa che non è chiaro.

Giorno 4

Mancano da trovare: water solubility, logHLC e BCF values. Rimangono da processare le due nuove variabili trovate: Molecular Weight and Number of oxidrilic groups (OH)

Ok per ora va bene cosi. Più avanti è possibile che torneremo a cercare le features trascurate. Ho trovato anche questa libreria python in cui ci sono un sacco di features da usare, includendo le pubchem fingerprints, che sono un vettore di 881 features relative a ogni composto. `http://projects.scbdd.com/pybiomed.html`

PubChem2D

Per trovare i PubChem fingerprint delle molecole, occorre scaricare i seguenti pacchetti:

- rdkit: installato tramite conda
- pybel: pip install pybel L'hai provato? Dovrebbe essere facilissimo, no?
- PyBioMed: procedura letta su

  `http://projects.scbdd.com/pybiomed/User_guide.html#installing-the-pybiomed-package`

C'è un grosso problema con la libreria PyBioMed. In sostanza, per ogni print (che sono tanti) non mette le le parentesi tonde. Inoltre in alcuni passaggi utilizza un linguaggio di Python che Jupyter non riesce a leggere. Non starà usando python2? Puoi eseguirlo con python2, fare un replace generale di tutti i print incriminati, per esempio usando `sed` da linea di comando, o vedere come eseguire python2 da python3 (per esempio https://stackoverflow.com/questions/27863832/calling-python-2-script-from-python-3). Ad ogni modo mi sembra che hai risolto :), quindi puoi ignorare i suggerimenti.

Provo a vedere con il pacchetto PubChemPy: installato con *pip install pubchempy*. Tramite cactvs_fingerprint si ricavano i pubchem (881).

Ho estratto i pubchem, domani li analizzo. Ottimo.

Li ho salvati in pubchem.csv Ok, puoi metterli nel dataframe con il resto dei dati.

Distanze

- Plot distanze composti tramite Tanimoto e pubchem2d (Hamming distance) – Cluster
- Controllare pubchem2d uguali tra due composti
- X–Grafi con distanze.
- Distanze tra specie (numero di segmenti dall'albero) + istogramma distanze.

Ho trovato le due distanze e ho fatto gli istogrammi. Per quanto riguarda il clustering ho fatto semplicemente un dendrogramma con un metodo scissorio usando il metodo di ward. Ho fatto anche una heatmap che penso possa essere utile.

Ho anche controllato i composti che hanno uguale pubchem. Ovviamente risultano essere quelli con una struttura molto simile: ad esempio stessi atomi ma diversa composizione. Ne ho rappresentati un paio nello script "pubchem2d analisi".

Ricominciamo... (15-06-2020)

Descrivo il lavoro fatto finora.

Partiamo dal lavoro precedente: dai dati raw (tests.csv, species.csv, results.csv), filtrando i dati sulla categoria su cui siamo interessati (*Fish*) e facendo un po' di preprocessing (vedere quali informazioni sono utilizzabili, regolarizzare le distribuzioni tramite trasformazioni) e inoltre, tramite il CAS number, trovare la rappresentazione SMILES da cui trarre informazioni sul composto usato nei test, si è ottenuto il dataset ***final.db.processed.csv***, che contiene:

variabili relative al test:

- **test.cas**: il composto testato
- **conc1.type**: file concentration.type.code.txt
- **conc1.mean**: concentrazione media
- **exposure.type**: file exposure.type.code.txt
- **obs.duration.mean**: tempo medio di durata dell'esperimento?

variabili relative al composto: Proprietà estratte dal CAS Number:

- **atom.number**: numero di atomi totali
- **alone.atom.number**: numero di atomi "soli"
- **bonds.number**: numero di legami singoli

- **doubleBond**:legami doppi
- **tripleBond**: tripli legami
- **ring.number**: numero di anelli
- **Mol**: peso molecolare
- **MorganDensity**: ????
- **LogP**: lipofilia

Proprietà dell'animale su cui è stato testato il composto (in ordine di classificazione scientifica del regno animale):

- **class**
- **tax.order**
- **family**
- **genus**
- **species**

Lavoro mio

Dalle linee guida di Marco, ho iniziato la ricerca delle nuove feature: tra quelle che mi avevi dato da trovare, ho verificato la presenza di alcune di esse e la loro provenienza (funzione usata per trovarle): tra queste ci sono:

- **peso molecolare**: già incluso, si usa la funzione MolWt nel modulo Descriptors di rdkit (serve prima MolFromSmiles del composto).
- **KoW e LogD**: già incluse nel lavoro precedente, la variabile è *LogP*. Il controllo sulla loro sovrapposizione (tutti i modi di trovare KoW, LogD e LogP riconducono allo stesso valore, che cambia per millesimi) l'ho fatto partendo sempre da rdkit nel modulo Crippen.
- **numero di gruppi OH**: nuova feature, trovata ed inclusa, usando sempre rdkit e andando a cercare il pattern "*[OX2H]*" che identifica i gruppi OH nello SMART (altra rappresentazione dei composti).

Tra le variabili che mi avevi proposto rimangono da trovare, *BCF values, logHLC, water solubility*.

A questo punto, abbiamo spostato l'attenzione sui pubchem2d, ossia una sequenza di 881 caratteri ciascuno che assume il valore 0 se il composto non possiede quel carattere e 1 se invece lo possiede : The PubChem System generates a binary substructure fingerprint for chemical structures. These fingerprints are used by PubChem for similarity neighboring and similarity searching.

Ed è proprio sulla similarità che mi sono concentrato: gli indici usati sono due:

- Hamming distance: si basa sui caratteri. Confronta due stringhe di uguale lunghezza e più sono diverse più sono lontane.
- Tanimoto Index: è un tipo di distanza tra i composti leggermente più difficile da ottenere. Ho usato GetTanimotoMatrix di rdkit.

Alcuni composti sono risultati avere lo stesso pubchem seppur con una composizione diversa, alcuni esempi possono essere fatti (vedere pubchem2 analisi.ipynb).

Dopo aver trovato questi due indici di distanza, le strade da portare avanti sono due: continuare il clustering usando le due matrici di distanze e pensare ad un modo di metterle insieme, e la seconda è controllare (e cercare di visualizzare) l'albero delle specie di animali su cui sono stati fatti i test).

Per quanto riguarda la seconda strada, qualche cosa ho fatto in Species analysis.ipynb: ho verificato che le specie testate (eliminando quelle che hanno dei missing values, e i duplicati, sono rimaste 901 specie) costituiscano un albero.

Al contrario di quanto mi avevi detto sono rimasto con:

- Class (ce ne sono 4) **OK**
- Order (35) **OK**
- Family (137) **OK**
- Genus (473) **NO ne ho 471**
- Species (966) **NO ne ho 747**

Ai fini dei test che il lavoro precedente considera, non è importante questo gap, in quanto ne vengono utilizzati molti di meno. É vero però che per una generalizzazione migliore un nuovo merge con il dataset raw *results.txt* alla luce di queste nuove informazioni va fatto.

Per quanto riguarda la prima strada il clustering, domani lo continuo.

16/06/2020

Ho finito le elaborazioni con il clustering. Ho fatto anche alcune cose veloci utilizzando il biclustering, ma non so quanto senso abbiamo farlo, io comunque l'ho fatto: per l'hamming distance i risultati del biclustering coincidono con quelli del clustering agglomerativo, mentre per il tanimoto index il biclustering non ottiene gli stessi risultati.

Hamming distance

Le distanze oscillano tra 0 e 0.3 mentre l'indice tra $[0, 1]$, questo vuol dire che i composti sono molto vicini.

Il clustering agglomerativo identifica 3 gruppi, ma abbassando la soglia si può arrivare anche a 5 gruppi, pur mantenendo una buona separazione.

Tanimoto index

Le distanze oscillano tra 0 e 1 così come l'indice. Molti valori sono alti e quesot significa che i composti sono molto lontani, o comunque meglio "distribuiti".

Il clustering agglomerativo identifica 4 gruppi "puliti" e il dendrogramma si taglia alla stessa altezza più o meno.

.

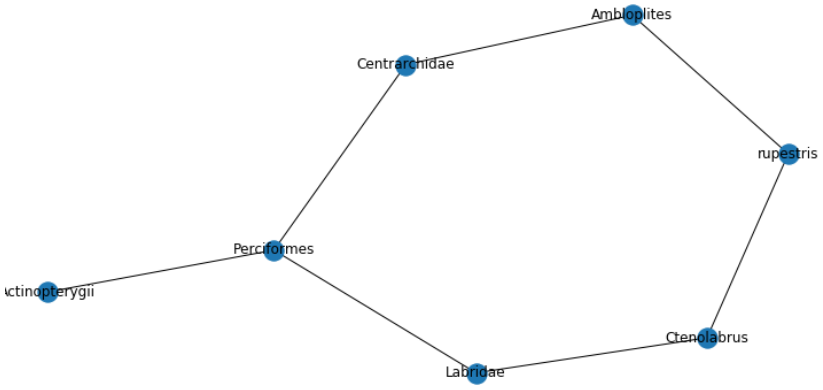
Per fare un confronto tra i due metodi di clustering, ho preso 3 cluster sia per hamming sia per tanimoto, e i due partizionamenteìi non coincidono per nulla, ossia i risultati ottenuti con Hamming sono incorrelati con quelli ottenuti tramite Tanimoto.

.

Prossimo step, unire la variabile trovate (OH count al dataset e procedere coi modelli).

20/06/2020

Nell'analisi delle specie di pesci testate, mi sono accorto che l'albero filogenetico si riunisce. Mi spiego meglio: due pesci diversi hanno una nomenclatura diversa perciò non deve essere possibile che partendo dalla stessa classe i due pesci hanno la stessa specie. Invece, quello che accade è:



ossia, accade che:
CLASS: *Actinopterygii*
TAX.ORDER: *Perciformes*
Ramificazione in due FAMIGLIE: *Centrarchidae* e *Labridae*
dalla FAMIGLIA *Centrarchidae* si passa al GENERE: *Ambloplites*
dalla FAMIGLIA *Labridae* si passa al GENERE: *Ctenolabrus*
Ricongiunzione nella SPECIES: *rupestris*

Questa ricongiunzione non è ammissibile. Perciò la nostra strategia è quella di unire la specie con il genere in un unico nome **genus+species**, mantenendo comunque valido il *genus* e quindi trascurando la species presa singolarmente.

Un'altra idea è quella di cercare tra i file raw se ci sono caratteristiche delle specie di pesci che possono essere utili per la classificazione.

25/06/20

Un'altra distanza è quella tra le specie. Per farla ho contato il numero di rami che dividono due specie e preso l'inverso. Perciò se due pesci non condividono la stessa classe allora il numero di rami che li dividono sarà 4 e la distanza 1/4, se non condividono l'ordine tassonomico il nujmero di rami è 3 e la distanza 1/3 e così via.

Si è risolto anche il problema con l'albero filogenetico che si riuniva nelle specie. Semplicemente, mantenendo l'informazione sul *genus* indipendente, si è unito *genus + species* in maniera tale da avere l'informazione sia sul genere che sulla specie. Inoltre, è stato plottato un grafo che rappresenta l'albero filogenetico delle specie.

Tra i compiti che ho per questa settimana ci sono:

- Abbiamo visto che circa 5000 composti hanno distanza di Tanimoto uguale a 0, perciò, i composti sembrano essere gli stessi. Allo stesso tempo, circa 300 hanno distanza di Hamming uguale a 0, quindi bisogna vedere quali coppie di composti hanno hamming = 0 e controllare se anche in tanimoto sono uguali a 0.
- Fare modelli (Random Forest, Boosting e altro) con tutte le variabili continue e solo le variabili categoriche relative ai composti in one-hot encoding.
- Pacchetto ***h2o***:
  - Rifare i modelli usando il pacchetto ***h2o*** per capire come funziona solo con variabili continue
  - rifare i modelli con one-hot encoding
  - sia con continue e con one-hot.
  - ⋆ **Capire come funziona lo splitting delle variabili categoriche in h2o diversamente da sklearn**

Possibili problemi:

- Alta dimensionalità: togliere le variabili sui pesci riducendo l'albero filogenetico (soprattutto genus+species perchè hanno molte modalità).
- Cercare approcci **non one-hot** per riduzione dimensionalità (Pensarci)
- Modo di implementare una distanza tra specie e tra composti nei modelli: in sostanza quando in un modello ad albero si usa una variabile categorica, lo split viene eseguito dividendo il set delle possibili modalità in 2. In un approccio labelEncoding, si da un ordine alle modalità della variabile categorica. Perciò si deve pensare ad un modo che considera anche la distanza tra le specie e i composti.

01/07/2020

In questa settimana mi sono concentrato sulla costruzione di modelli tramite il pacchetto **h2o**. Questo ha anche una piattaforma online *H2OFlow* che ne facilita l'utilizzo.

Comunque, la logica di **h2o** è leggermente diversa da quella di **sklearn** e in più prevede alcuni parametri all'interno dei modelli che, ad esempio, permettono di gestire variabili categoriche in maniera molto semplice, oppure permettono di fare cross-validation direttamente dentro al modello, per la stima dell'accuratezza o AUC.

Ho provato 3 diversi dataset:

- Solo variabili continue: per la riproducibilità dei modelli usando h2o piuttosto che sklearn
- Solo variabili categoriche: per capire come esse vengono utilizzate dentro al modello. Il primo enconding è *one-hot*, ci sono alcune altre soluzioni che possono essere interessanti da vedere.
- Sia categoriche che continue

In questi dataset **non** ho ancora riunito species e genus.

L'encoding fatto sulla concentrazione media è: se la concentrazione è inferiore ad una determinata soglia (1 mg/L) , quella concentrazione è considerata tossica, se è superiore non lo è.

la logica di h2o è in https://docs.h2o.ai/h2o/latest-stable/h2o-docs/welcome.html.

Tornando ai modelli, h2o prende il dataset intero e richiede di specificare quali variabili sono da considerare come predittori e quale è da considerare come target. Inoltre, le variabili categoriche devono essere trasformate in fattori, un po' come succede in R. Un fattore è un oggetto che, avendo una "lista" di modalità che può assumere, un'unità statistica assumerà il valore della posizione di quella modalità all'interno della lista, ma ciò non vuol dire che le modalità siano ordinate.

Infatti, l'encoding nei modelli per le variabili categoriche è esplicitato dal parametro **categorical.encoding**. Questo può assumere i seguenti valori:

- *one.hot.explicit* fa l'encoding one-hot
- *label.encoder* che trasforma il livello in un numero: livello 0 diventa 0, livello 1 diventa 1, e così via.
- *binary* che mantiene un massimo di 32 colonne per variabile
- *eigen* k columns per categorical feature, keeping projections of one-hot-encoded matrix onto k-dim eigen space only

- *sort.by.response* che ordina in base alla frequenza di risposta. La modalità meno frequente avrà il livello 0, la seconda meno frequente avrà il livello 1 e così via.

i modelli che ho costruito sono tutti di default e vanno molto bene, lho provato a fare anche un grid search, ma ci ha messo 6 ore senza finire. lo faccio più in là con clama.

03/07/2020

Fino ad ora ho utilizzato modelli standard per capire le potenzialità dei modelli, e tutto sommato sia il Gradient Boosting che le Random Forest ottengono risultati molto soddisfacenti. Al posto dell'encoding one-hot per le variabili categoriche ho preferito l'alternativa *eigen* perchè ci mette meno tempo.

Il risultato che i miei predecessori hanno ottenuto è il seguente: 0.90 di accuratezza sia con K-NN che con matrix factorization, 0.312 RMSE per k-NN e 0.272 per MF.

Per iniziare a cross-validare gli iperparametri della Random Forest, inizio con la profondità massima degli alberi, e cerco un range idoneo, perchè è quello che è più time-consuming.

Ho lanciato una griglia abbastanza fina da 5 a 40 ogni 2 (5,7,9,11,...,40) e basandomi sull'AUC ho ristretto tra 30 e 40. I risultati mostrano i modelli ordinati in base all'auc sul training set decrescente:

	max_depth	model_ids	auc
0	30	depth_grid1_model_11	0.9530832832767969
1	32	depth_grid1_model_12	0.9530678582923222
2	40	depth_grid1_model_15	0.9530663892461819
3	37	depth_grid1_model_14	0.9530661067373087
4	35	depth_grid1_model_13	0.9530651300637759
5	27	depth_grid1_model_10	0.9530614413050604
6	25	depth_grid1_model_9	0.9529849944039835
7	22	depth_grid1_model_8	0.9529710546090135
8	20	depth_grid1_model_7	0.9525636041687785
9	17	depth_grid1_model_6	0.9507487510242358
10	15	depth_grid1_model_5	0.9468935865110052
11	12	depth_grid1_model_4	0.9387574843259638
12	10	depth_grid1_model_3	0.9255323399449611
13	7	depth_grid1_model_2	0.8919196279367826
14	5	depth_grid1_model_1	0.8576160525868152

La stessa griglia di valori risulta basandoci sull'accuratezza. Massima accuratezza sul training ottenuta si aggira attorno a 0.899

	max_depth	model_ids	accuracy
0	27	depth_grid1_model_10	0.899583384389168
1	32	depth_grid1_model_12	0.899522117387575
2	30	depth_grid1_model_11	0.8994608503859821
3	35	depth_grid1_model_13	0.8993995833843892
4	37	depth_grid1_model_14	0.8993995833843892
5	40	depth_grid1_model_15	0.8993995833843892

Per quanto riguarda il test set, il miglior modello secondo l'AUC (massima profondità 30) ottiene un'accuratezza intorno a 0.85 (migliorabile) e un'AUC

circa pari a 0.92.

Ora che ho una griglia più ristretta per la profondità massima passo a fare il tuning degli altri parametri.

Un'idea è quella di vedere, una volta trovati i parametri ottimali per i dati, qual è l'encoding migliore sia dal punto dei risultati che dal punto di tempo.