IntermsPrediction交接文件

環境說明:

* Windows 10, java 64bit 1.8.0\_131
* IDE: intellij idea 2017.1
* Library manager: maven

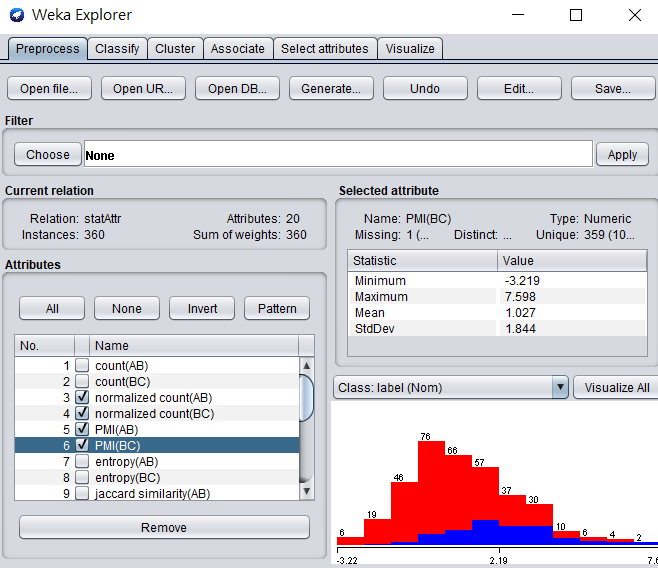
程式執行說明:

1. 處理SemMedDB資料

* Download from <https://skr3.nlm.nih.gov/SemMed/> (DB schema 說明請看官網介紹)
* 執行ken.prepare.SemMedPreparser 內function產生DB biomedical\_literature\_by\_year內table (之後用到的features都從幾張表而來，我已經處理好放在James上了，可以直接沿用):
  + create\_mesh\_predication\_aggregate() => mesh\_predication\_aggregate
  + processMedlineGroupByYear() => mesh\_concept\_by\_year
  + create\_neighbor\_by\_predication() => neighbor\_by\_predication
  + processCooccurNeighborGroupByYear() => neighbor\_cooccur

1. 評估Path Importance Classification

* 執行ken.label.AttrTester.main()，詳細資訊請看程式註解
  1. 執行createStatArffFile()、createContentArffFile()、mergeInst()產生3個arff檔案 (stat.arff, content.arff, combined.arff)
  2. 可以使用weka保留不同feature type( predication-based, concept-based)，並存成新檔案，如下圖。之後使用可以evaluateModel()評估新data set



* 1. 利用前面步驟產生的檔案(input)執行evaluateModel()，選擇model (請看TODO範例)進行評估
* 檔案說明
  1. output/attr/內的檔案: weka定義的檔案，請使用文字編輯器打開or使用weka軟體打開
  2. vocabulary/61predicate.txt: SemMedDB內全部的predicate，下SQL而來: SELECT distinct(predicate) FROM semmed\_ver26.predication\_aggregate
  3. vocabulary/contentWord.txt: 篩選後的predicate，濾掉出現頻率不多的predicate，如何過濾請看paper
  4. vocabulary/trainingContentWord.txt: 與contentWord.txt內容相同但順序不同，因weka處理後會變這個順序，之後評估Drug Repurposing Discovery 所需要的content arff要和training cases一致，不然會有問題

1. 評估Drug Repurposing Discovery

* 前處理: 執行ken.prepare.InstancePrepare.main()，詳細資訊請看程式註解 (若使用我留下的檔案，則不需要重跑，有些function要跑蠻久的)
  1. 執行createGoldenFile()產生所有drug標準答案，default time cut-off year = 2005，若要修改請自行修改程式，輸出檔案dat\_file/eval/golden\_pairRank.dat
  2. 執行createTreatRankSet()，input為前一步驟產生的golden\_pairRank.dat，保留標準答案有3(TREATS)的藥物與初步過濾(有分數的/全部>=0.01)，輸出檔案dat\_file/eval/golden\_treatRank.dat
  3. 執行create500Evalrank()，input為golden\_treatRank.dat，將分數包成物件，僅保留focal drug(隨機挑選500個drug)的標準答案，輸出檔案dat\_file/eval/goldenRank.dat與vocabulary/500drugs\_seed.txt (藥名)
  4. 執行createInstFile()計算focal drug-disease pair的變數，input為dat\_file/eval/goldenRank.dat與vocabulary/500drugs\_seed.txt ，輸出檔案dat\_file/instaMap/xxxx (focal drug).dat
  5. Used for benchmark 2: 執行createCooccurFile()，input為dat\_file/eval/goldenRank.dat"，輸出檔案為dat\_file/cooccur/xxxx (focal drug).dat
* 計算NDCG: 執行ken.model.LearningModel.main()即可，詳細資訊請看程式註解。執行程式前須確定前處理都已經完成

1. 執行runModel()、baseLine()、baseLine2()產生ndcg檔案，input為dat\_file/instaMap/xxxx (focal drug).dat (runModel() & baseLine())與dat\_file/cooccur/xxxx (focal drug).dat (baseLine2())，需自行設置feature type, classifier, ranking method, 輸出檔名(請看TODO)，輸出檔案為ndcg object
2. 執行ken.evaluation. Evaluator.main()得到比較結果(ex: 提升3.4%)，input有兩個，為前一步驟得到的ndcg object (benchmark and 自己的model)，輸出檔案為所有disease平均ndcg score，輸出檔案output/ndcg/xxxx.txt(xxxx與輸入檔案相同)

* 其他檔案說明(基本上不會改動)
* vocabulary/500drugs\_seed.txt: focal drug檔案，create from ken.prepare.InstancePrepare.create500Evalrank()
* vocabulary/drug.txt, disease.txt, genes\_proteins\_enzyme.txt, anatomy.txt, pathology.txt: 全部的seed concept, 共17264個
* 需要手動設置的程式 (如何設定請看TODO)
  + ken.model.LearningModel.runModel()
  + ken.network.PredicateNetwork.setClassifier()
  + ken.evaluation.main()

DB說明:

* DB名稱: biomedical\_concept\_mapping (主要是儲存不同vocabulary terms之間的對應關係，請看學姊程式，我沿用並沒有修改)
  + mesh\_descriptor、mesh\_supplement: 原始mesh檔案只取出特定欄位(Created from: sandy.parser.MeshParser)
  + mesh\_umls: mesh term對應的umls concept

(Producedd from: sandy.parser.MeshMappingParser)

* + omim\_mesh: omim名稱和mesh term的對應

(Created from: sandy.parser.MeshMappingParser)

* + omim\_meshid: omim ID和mesh ID的對應

(Created from: sandy.parser.CtdParser)

* + omim\_name: 完整的omim名稱表，用於ID轉換

(Created from: sandy.parser.OmimParser)

* + umls\_mapped\_term: umls原始資料讀入

(Created from: sandy.parser.UmlsParser)

* + umls\_mesh: umls concept對應的mesh term

(Created from: sandy.parser.MeshMappingParser)

* DB名稱: biomedical\_relation (請看學姊程式，我沿用並沒有修改)
  + drugbank: DrugBank讀入的原始relation

(Created from: sandy.parser.DrugBankParser)

* + View: drugbank\_mesh\_target

SQL: CREATE VIEW `drugbank\_mesh\_target` AS select `biomedical\_concept\_mapping`.`drugbank\_mesh`.`ID` AS `DBID`, `biomedical\_concept\_mapping`.`drugbank\_mesh`.`Name` AS `DBName`, `biomedical\_concept\_mapping`.`drugbank\_mesh`.`Mesh` AS `Mesh`,`biomedical\_relation`.`drugbank`.`TargetName` AS `TargetName`, `biomedical\_relation`.`drugbank`.`TargetSwissProtID` AS `TargetUniprotID` from (`biomedical\_concept\_mapping`.`drugbank\_mesh` join `biomedical\_relation`.`drugbank`on(((`biomedical\_concept\_mapping`.`drugbank\_mesh`.`ID` = `biomedical\_relation`.`drugbank`.`ID`) and (`biomedical\_concept\_mapping`.`drugbank\_mesh`.`Name` = `biomedical\_relation`.`drugbank`.`Name`))))

* + omim\_relation: omim讀入的原始relation

(Created from: sandy.parser.OmimParser)

* + View: omim\_mesh\_relation

SQL: CREATE VIEW `omim\_mesh\_relation` AS select `biomedical\_concept\_mapping`.`omim\_mesh`.`MIM` AS `MIM`, `biomedical\_concept\_mapping`.`omim\_mesh`.`Name` AS `Name`, `biomedical\_concept\_mapping`.`omim\_mesh`.`Mesh` AS `Mesh`,`biomedical\_relation`.`omim\_relation`.`Related\_MIM` AS `Related\_MIM` from (`biomedical\_concept\_mapping`.`omim\_mesh` join `biomedical\_relation`.`omim\_relation` on(((`biomedical\_concept\_mapping`.`omim\_mesh`.`MIM` =`biomedical\_relation`.`omim\_relation`.`MIM`) and (`biomedical\_concept\_mapping`.`omim\_mesh`.`Name` = `biomedical\_relation`.`omim\_relation`.`Name`))))

* + ctd\_relation: ctd讀入的原始relation

(Created from: sandy.parser.CtdParser)

* + ctd\_neighbor: 轉換ctd\_relation的儲存格式

(Created from: sandy.parser.CtdParser)

* DB名稱: biomedical\_literature\_by\_year
  + mesh\_concept\_by\_year: 儲存concept根據年份共在predication出現幾次(freq)與document frequency

Created from: ken.prepare.SemMedPreparser.processMedlineGroupByYear()

* + mesh\_predication\_aggregate: 將semmed\_ver26.predicationaggregate裡的umls concept轉成mesh term，並只保留所需的欄位

Created from: ken.prepare.SemMedPreparser.create\_mesh\_predication\_aggregate()

* + mesh\_seeds: 17264個concept seeds, 與vocabulary/那五個檔案相同，儲存其他表所用的mesh\_id
  + neighbor\_by\_predication: 處理semmed\_ver26.predicationaggregate，並將共同出現的concept儲存起來，同時記錄共同出現的年份與次數

created from ken.prepare.SemMedPreparser.create\_neighbor\_by\_predication()

* + neighbor\_cooccur: 與neighbor\_by\_ predication相似，不過based on co-occurrence method (參考paper)

created from ken.prepare.SemMedPreparser.processCooccurNeighborGroupByYear()