



AI創薬のための ケモインフォマティクス 入門

@fmkz__, @iwatobipen

Version 0.40000(Draft) 2019/03/18

目次

1章: はじめに	1
RDKitとは	2
対象読者	3
本書のコードについて	3
おまけ	3
謝辞	4
License	4
2章: ケモインフォマティクスのための環境を整えよう	5
Anacondaとは	5
Anacondaのインストール方法	5
仮想環境の構築とパッケージのインストール	5
インストールしたパッケージの説明	6
Condaについてもう少し詳しく	6
3章: Pythonプログラミングの基礎	8
Pythonの基礎	8
Jupyter notebookで便利に使おう	9
Pythonで機械学習をするために	9
4章: ケモインフォマティクスのための公開データベース	10
ChEMBL	10
PubChem	10
ChEMBLで欲しい情報を検索する	10
その他有用なデータベース	15
5章: RDKitで構造情報を取り扱う	17
SMILESとは	17
構造を描画してみよう	17
複数の化合物を一度に取り扱うには?	18
ヘテロシャッフリングをしてみる	20
6章: 化合物の類似性を評価してみる	25
化合物が似ているとはどういうことか?	25
類似度を計算する	26
バーチャルスクリーニング	26
クラスタリング	28
Structure Based Drug Design(SBDD)	30
7章: グラフ構造を利用した類似性の評価	31
主要な骨格による分類(MCS)	31
Matched Molecular PairとMatched Molecular Series	34
Cytoscapeを使ってMMPネットワークを可視化する	38
8章: 沢山の化合物を一度にみたい	41
Chemical Spaceとは	41
ユークリッド距離を用いたマッピング	42
tSNEをつかったマッピング	44

9章: 構造活性相関（QSAR）の基礎.....	45
効果ありなしの原因を考えてみる（分類問題）.....	45
薬の効き目を予測しよう（回帰問題）.....	48
モデルの適用範囲(applicability domain).....	49
10章: ディープラーニング入門.....	50
ディープラーニングについて.....	50
TensorFlowとKerasについて.....	50
インストールしてみよう.....	51
Google colabとは.....	51
11章: ディープラーニングを利用した構造活性相関.....	55
DNNを利用した予測モデル構築.....	55
記述子を工夫してみる(neural fingerprint).....	59
12章: コンピューターに化学構造を考えさせる.....	62
準備.....	62
実例.....	63
13章: おわりに.....	65
さらに学ぶために.....	65

1章: はじめに

ケモインフォマティクス(Chemoinformatics)とは、主に化学に関連するデータをコンピュータを用いて解析し、様々な課題を解くために用いる方法論のことです。ケモインフォマティクスという言葉は1990年代終わりから2000年代はじめに定義され、現在では製薬業界や薬学系のアカデミアにおいて、薬剤の効果と化合物特性の関連性を解析したり、大量の化合物情報の可視化、化合物の類似性に基づいたクラスタリングなど、多岐にわたるプロセスで利用されています。

近年、ディープラーニング(Deep Learning)の創薬応用が模索されていますが、活性や物性を予測するQSAR（構造活性相関）だけではなく、新規デザイン提案や合成経路提案といった従来のケモインフォマティクスではあまり扱わなかった領域への応用研究も盛んに行われています。

化合物デザインはイノベーティブ

そもそもどのような化合物を作るべきか?、またそれをどのように合成するか?を考えるプロセスは背景知識と想像力が求められる領域であり、従来は人以外が担うのは難しい領域であると認識されていましたが、このような領域に対してもAIと呼ばれるものの進出がここ数年(2017-2019)で急速に進みました。

ケモインフォマティクスはすでに様々な場面で利用されていますが関連情報はありませんでした。この理由として考えられることはいくつかありますが、オープンソースのツールキットが存在しなかつた、公開データベースが存在しなかった、という二点が最も大きな原因なのは間違いないありません。しかし、RDKitというオープンソースのケモインフォマティクツールキットとChEMBLという公開データベースの登場により、この点は解消しました。

近年では、ケモインフォマティクスもバイオインフォマティクス同様、ウェブで検索すれば多くの情報がすぐに得られ、自己学習することは十分可能となっていますが、最初の一歩を踏み出すためのまとまった情報として、「ケモインフォマティクスに関する基礎を学び、それらを応用できるようになるコンテンツ」を用意することにしました。近年のAI創薬ブームを考慮して後半の章では「AI創薬」の文脈で用いられるディープラーニングを利用した化合物の活性予測と化合物提案の章を収録していますので、一通り学習することで、最近のトレンドについていくようになっているはずです。

RDKitとは

warning

ここは@iwatobipenがRDKitについて熱く語るサブセクションです。下書き段階での「申し上げる」とか「踏まえて」等の言い回しをそのまま生かし、自称が「拙者」で「ござる」調の@iwatobipen風文体です。

拙者、本書の一部を執筆する@iwatobipenと申す。ここではRDKitに関して熱く語ってみようと思うでござる。

RDKitのRDはなんなのか？実は**Rational Discovery**の略称であり現在のオープンソースの前身となるフレームワークは2000年に開発されたでござる。随分と古いでござるね。その後、2006年にコードがオープンソースになりsourceforgeから公開されたでござる。PythonのケモインフォマティクスツールキットはRDKit以外にOpenBabelもあるぞと思われる読者もござろう。OpenBabelは2005年に最初のリリースがなされてある。いずれも、もう10年以上の歴史があるツールキットでござる。拙者がこの辺りに興味を持ち始めた2012年ころはどちらかというとOpenBabelの方がメジャーだったように記憶してある。当時、日本語の記事はほぼ皆無であり、拙者は本書共著者であり業界のパイオニアでもある@fmkz__殿の[ケモインフォマティクスツールキット](#)などを参考にRDKitのコードを書いて試行錯誤していたでござるよ。なお、ケモインフォ関連のヒストリを追いたい御仁はこちらの[記事](#)を一読されるとよかろう。

おっと話が横道に逸れてしまった。本題に戻ろう。

開発者のGreg Landorum氏いわく

RDKitはケモインフォマティクスにおけるSwiss Army Knifeであり、様々な機能ピースの集合体である

— Greg Landorum

これはまさに目的を得た表現でござる。[公式ドキュメント](#)を見ればわかるでござるが、既に色々な機能が用意されてあるのだ。 化合物情報の読み込み、書き込みに始まり、構造の描画、3次元構造配座発生、Rグループ分解、記述子、フィンガープリント計算、ファーマコフォア算出などなど、挙げればきりがないほどの機能が実装されてある。解析から可視化まで幅広い範囲をカバーできるのだ。 さらにContributerらがRDKitを利用して開発したツール群がその熱い想いとともに[Contrib](#)フォルダーに詰められてあるのだ。どうじゃ使ってみたらなるんか？。拙者はもう書きながらも早くRDKitに触りたくなってきたでござる。

NOTE

@iwatobipenももちろんContributerの1人で、[Fastcluster](#)という大量の化合物ライブラリを高速にクラスタリングするコードを提供しています。(by @fmkz__)

RDKitは開発やユーザーコミュニティの活動も活発で、どんどん機能追加がされておる。世界中の有能な研究者が全体で盛り上げ開発していくスタイルはオープンソースの強みであり、魅力であろう。もしチャンスがあれば毎年開催されるRDKit User Group Meetingへの参加を検討するのもよかろう。Face2Faceでユーザー同士議論ができるのは何事にも代え難いものがあるのでござる。また、先ほど拙者が使い始めた当時は日本語の情報ほぼ皆無であったと申したが、近年は非常に良質な日本語記事もたくさん増えておる。下記に何個か例を挙げたでござる。Qiitaにも多くの記事が掲載されているでござるよ。

また、有志による[RDKit-users-jp](#)も立ち上がっておる。英語での質問がちょっと、、、と思われる御仁はこちらに質問を投げかけるとよかろう。また、最新版のRDKitのリポジトリには日本語のドキュメントも

マージされてある。こちらも参考になるであろう。 本書ではRDKitの一部の機能しか使わん。それでも非常に多くのことができると感じていただけるはずじゃ。興味持ちはじめの一歩を踏み出したら後はどんどん自分の興味、意欲のままに足を進めていけばよかろう。何かわからないことがあれば上記のコミュニティに問い合わせ、本書のリポジトリへIssueとして投稿してみるのもよかろう。さあそれでは始めよう！

主な日本語解説サイト

- [rdkit-users.jp](#)
- [RDKitドキュメンテーション非公式日本語版サイト](#)
- [化学の新しいカタチ](#)

対象読者

次のような方々を読者として想定しています。

- 医学薬学系の大学院生及び薬学系のデータ解析を行いたいポスドク
- 製薬企業の薬理研究者で自分のデータを自分で解析したい人
- 創薬化学者でケモインフォマティクスの必要性を感じている方や謎の力により突然アサインされた方
- ケモインフォマティクスを学んでみようと考えているバイオインフォマティシャン
- AI創薬に興味があるがなにからはじめたらいいかわからない人

本書のコードについて

本書で使用したプログラミングコードは全てMishima.sykのpy4chemoinformaticsリポジトリのnotebooksディレクトリに置いてありますので利用してください。またそれぞれの章の最初のにその章のJupyter notebookへのリンクを張っていますので適宜参照してください。

2章のインストールを行うとgitコマンドが使えるようになりますので、以下のコマンドでpdfを含む本書の全てのデータがダウンロードできます

```
$ git clone https://github.com/Mishima-syk/py4chemoinformatics.git
```

おまけ

Chemoinformatics or Cheminformatics?

もともとはBioに対してChemoと語感を合わせて登場してきたように記憶しているが、[Journal of Cheminformatics](#)の創刊により一時期Chemに大きく離されました。

最近の[Google trend](#)によるとどちらでもいいようですが個人的にはRhymeを重視したほうが良いと思うので本書ではChemoの方を使うことにします。

謝辞

本書を執筆するにあたり、バグフィックスや改善のための助言をしてくれた以下の方々に感謝いたします。

@antiplastics, @bonohu, @ReLU_Tropy, @ski_nanko, @torusengoku, @yamasaKit_

ここから先は(Nujabes - reflection eternalを聴きながら書きました by @fmkz__ 2019/03/03)

まず、本書を書くきっかけとなった@bonohuに感謝したいと思います。@bonohuのDr. Bonoの生命科学データ解析の出版後のMishima.sykのミーティングで「Bono本のChemoinformatics版あつたらいいよね」という話がどこからともなくでた際に、「書けばえんちゃう、むしろなんで書かんの?」と言ってくれたことが本書を執筆するきっかけであることは間違ひありません。また@souyakuchanの創薬Advent Calendar 2018も執筆のいい刺激になりました。というより、ここで章立てしなかったら具体的に動き出さなかっただと思います。

また、忘れてはいけないのはy-samaの存在です。Mishima.sykを初期から盛り上げてきたy-samaは2019/01/06に永眠しました。彼はデータサイエンティストを目指す人のpython環境構築 2016やDruglikenessについてのよもやま話といった素晴らしいエントリを残しました。彼が存命であればきっと3人で執筆していたし、内容ももっと充実していたことでしょう。この出来事も我々に執筆しようという強い動機を与えました。

最後にMishima.sykに参加して美味しいワインやビールを飲みながら毎度熱い議論を交わしていただいた参加者の方々にも感謝します。いくつかのコンテンツはMishima.sykでの発表をもとにしており、みなさんのフィードバックをもとに加筆訂正しております。

もし、本書を読んで、ケモインフォマティクスって面白いなと感じたり、創薬やってみたいなと感じる方がいたら、是非Mishima.sykに参加してみてください。きっと楽しいと思います。今後の創薬研究では所属を超えてお互いにプッシュしあって自身のスキルを高めていくことが重要になるでしょう。というより、既にそういう社会になっているのだと思います。本書が皆さんのおもしろい研究生活を送る役に立てば幸いです。

やりたいことをやって生きてきて 私自身は自分の人生に後悔はありません 人生は楽しんだもの勝ち 皆さんも嫌なことは嫌だと言って自分の喜びを最大限に追い求めて人生を満喫した方が楽しいと思いますよ 皆様の人生に幸多い事を願っています

— y_sama

License

This document is copyright © 2019 by @fmkz__ and @iwatobipen

This document is Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License.



2章: ケモインフォマティクスのための環境を整えよう

本書に必要な環境構築を行います。

Anacondaとは

Anacondaは機械学習を行うための準備を楽にするためのパッケージです。また、後ほど説明するRDKitを簡単にインストールできます。

Q&A

なぜAnacondaを利用するのか？

プログラミング言語Pythonは比較的多くの標準ライブラリが用意されていますが、ケモインフォマティクス用のライブラリは自分でインストールする必要があります。この作業は慣れれば大した問題ではないですが、初学者にとっては面倒くさいでしょう。この手間を軽減するのが理由です。

Pythonには大きく2.x系のバージョンと3.x系のバージョンとがありますが？

2.x系はサポートが2020年に終了するため、新しく学ぶ方は2.x系を使う必要はありません。

Anacondaのインストール方法

では早速Anacondaをインストールしましょう。[公式サイト](#)にアクセスして、ご自身の環境にあつたPython3系インストーラーをダウンロードします。OSがLinuxであればターミナルからインストーラーを実行します。

```
$ bash ~/Downloads/Anaconda3-4.1.0-Linux-x86_64.sh
```

インストーラーが起動しいくつかの質問をされますが、基本的にエンターまたはYesで進めてください。

Anacondaのインストールが完了すると、コマンドプロンプトまたはターミナルから'conda'コマンドが使えるようになります。

仮想環境の構築とパッケージのインストール

AnacondaでインストールされるPythonは3.7ですが、本書執筆時点で配布されている最新のRDKitはPython3.6を必要とします。そのためcondaで仮想環境を構築し、必要なバージョンのPythonをインストールします。コマンドの-nの後ろは"py4chemoinformatics"としていますが皆さんの好きな名前でも構いません。仮想環境構築後、本章以降で利用するパッケージをインストールします。

```
$ conda create -n py4chemoinformatics python3.6
$ source activate py4chemoinformatics # Mac/Linux
$ activate py4chemoinformatics # Windows

# install packages
$ conda install -c conda-forge rdkit
$ conda install -c conda-forge seaborn
$ conda install -c conda-forge ggplot
$ conda install -c conda-forge git
```

インストールしたパッケージの説明

RDKit

RDKitはケモインフォマティクスの分野で最近よく用いられるツールキットの一つです。オープンソースソフトウェア(OSS)と呼ばれるものの一つで、無償で利用することができます。詳しくは[はじめに](#)を参照してください。

seaborn

[統計データの視覚化のためのパッケージ](#)の一つです。

ggplot

グラフ描画パッケージの一つで一貫性のある文法で合理的に描けることが特徴です。もともとはRという統計解析言語のために開発されました、yhatという会社により[Pythonに移植](#)されました。

Git

バージョン管理システムです。本書ではGitについては説明しませんのでもしGitについて全然知らないという方は[サルでもわかるGit入門](#)でも読みましょう。

「はじめに」でも説明しましたが、以下のコマンドでpdfを含む全てのデータがダウンロードされますので必要に応じてダウンロードしてください。

```
$ git clone https://github.com/Mishima-syk/py4chemoinformatics.git
```

Condaについてもう少し詳しく

なぜ仮想環境を作るのでしょうか

いくつかのシステムでは様々な機能を提供するために内部的にPythonを利用しているため、特定のパッケージのためにPythonのバージョンを変更してしまうと問題が起こることがあります。仮想環境はこのような問題を解決します。もし、パッケージが異なるライブラリのバージョンを要求しても仮想的なPython環境を準備して試行錯誤できます。不要になれば仮想環境を簡単に削除でき、もとの環境にトラブルを持ちこむこともありません。このように、ひとつのシステム内にそれぞれ個別の開発環境を

作成できるようにすることで開発時によく起こるライブラリの依存問題やPythonのバージョンの違いに悩まされることがなくなります。

本書では本書用に一つだけ仮想環境を用意しますが、実際はいくつもの仮想環境をつくって開発することが多いです。そのため、よく利用するcondaのサブコマンドを挙げておきます。

```
$ conda install <package name> # install package  
$ conda create -n 仮想環境の名前 python=/バージョン # 仮想環境の作成。  
$ conda info -e # 作った仮想環境一覧の表示  
$ conda remove -n 仮想環境の名前 # 仮想環境の削除  
$ source activate 仮想環境の名前 # 仮想環境を使う(mac/linux)  
$ activate 仮想環境の名前 # 仮想環境を使う(Windows)  
$ source deactivate # 仮想環境から出る  
$ conda list # 今使っている仮想環境にインストールされているライブラリの一覧を表示
```

3章: Pythonプログラミングの基礎

Pythonの基礎

この章ではPythonに触れたことのない読者のために効率的に勉強するためのサイトや本などを紹介します。もしこれ以降の章でわからないことなどがあったら、この章のサイトや本を参考に学んでみてください。

Pythonを本で学びたい

Pythonスタートブック増補改訂版

プログラミング自体が初心者であればこの本が良いでしょう。

みんなのPython 第4版

JavascriptやJavaなどのなかでプログラミングを少しかじっていて、これからPythonを覚えたいのであればこちらの本をおすすめします。

Pythonを本以外で学びたい

Python Boot Camp(初心者向けPythonチュートリアル)

一般社団法人PyCon JPが開催している初心者向けPythonチュートリアルイベントです。全国各地で行われているので近くで開催される場合には参加するとよいでしょう

その他ローカルコミュニティなど

あちこちで入門者向けからガチのヒト向けまでの勉強会やコミュニティなどもあるので、そういうのに参加してモチベーションを高めるのもよい方法です。

udemy/python

オンライン学習サービスを利用するのも効果的な手段のひとつですが、筆者は試したことがないのでわかりません。周りの評判を聞いてみても良いでしょう。YouTubeを探すのもあります。

本書でわからないことがあったら

py4chemoinformaticsのissues

py4chemoinformaticsのissuesに質問していただければお答えします。わかりにくい場合だったら修正しますので、よりよくなつてみんなハッピー。

Qiita

Qiitaで探せば大抵答えが見つかるはずです。

stackoverflow

それでも答えが見つからなかったらsofで探すか質問しましょう

Mishima.syk

本書を書いている人たちが集まるコミュニティです。特に話題をPythonに限定していませんが、Pythonを使ったネタが多めです。かなりガチですが、初心者対応も万全でハンズオンに定評があります。質問されれば大体答えられます。

Jupyter notebookで便利に使おう

Jupyter notebookを利用すると、コードを書いて結果を確認するということがとても簡単にできるようになります。

Jupyter notebookはWebブラウザベースのツールで、コードだけではなくリッチテキスト、数式、なども同時にノートブックに埋め込みます。また結果を非常に綺麗な図として可視化することも容易にできます。つまり、化学構造やグラフも描画できるため、ケモインフォマティクスのためのプラットフォームとして使いやすいです。さらに、プログラミングの生産性を上げるような、ブラウザ上でコードを書くとシンタックスハイライトや、インデント挿入を自動で行ってくれたりという便利な機能もついているので、特に初学者は積極的に使うべきでしょう。

使い方

terminal(Windowsではanaconda prompt)から

```
$ jupyter notebook
```

と打てばJupyter Notebookが立ち上がります。本書ではこれ以降特に断らない限りJupyter Notebook上でのコードを実行することとします。

Pythonで機械学習をするために

ケモインフォマティクスに限らず、インフォマティクスを学ぶにあたり、機械学習は外せません。本書でもある程度の機械学習の知識があることを前提に進めていきます。Pythonで機械学習をするにはScikit-learnというライブラリを利用するのが定番であり、本書でも特に説明せずに利用していきますが、初学者のために参考となる書籍などをすすめておきます。

Pythonではじめる機械学習 —scikit-learnで学ぶ特徴量エンジニアリングと機械学習の基礎

Pythonで機械学習をやるために基礎を学べます。数学的な表現があまりないので読みやすいです。

sklearn-tutorial

y-samaによるsklearnのチュートリアルハンズオンのjupyter notebookです。

4章: ケモインフォマティクスのための公開データベース

この章ではケモインフォマティクスでよく使うデータベースを紹介します。

ChEMBL

ChEMBLはEBIのChEMBLチームにより維持管理されている医薬品及び開発化合物の結合データ、薬物動態、薬理活性を収録したデータベースです。データは主にメディシナルケミストリ関連のジャーナルから手動で抽出されており、大体3,4ヶ月に一度データの更新があります。

メディシナルケミストリ関連のジャーナルからデータを収集しているため、QSARに関連する情報や背景知識を論文そのものに求めることができます。創薬研究をする際には有用です。

NOTE

ChEMBLはもともとはStARliteという商用データベースでした。詳しくは慶應大学池田先生のChEMBLに関する資料を参照してください。

PubChem

PubChemはNCBIにより維持管理されている低分子化合物とその生物学的活性データを収録している公開リポジトリです。5000万件以上の化合物情報と、100万件を超えるアッセイデータを含みそのデータ量の多さが特徴とも言えます。もうひとつの特徴はデータをアカデミアからの化合物登録やアッセイ結果の登録により成長することであり、ここが先のChEMBLとの大きな違いです。

特にPubChemは初期スクリーニングのデータが多いため、そのようなデータに対しなんらかのマイニングや分析を行いたい場合は有用だと考えられます。

どちらを使うべき?

QSARをやりたい場合にはやはりChEMBLのデータを利用することが多いです。IC50のようなデータが得られていることが多いですし、モデルの解釈に元論文をあたることができるというのが大きな理由です。

ChEMBLで欲しい情報を検索する

NOTE

ChEMBLはユーザーインターフェースを刷新中で現在beta版のテストを行っていますが、いずれこちらに置き換わると思うので新バージョンのインターフェースでの検索方法を紹介します。

まずはChEMBLにアクセスし、画面上部のCheck out our New Interface (Beta). というリンクをクリックして新しいインターフェース画面に移行します。

ChEMBLのデータは主に4つのカテゴリに分かれています。一意なIDが振られており相互に関連付けされています。それぞれのカテゴリについて簡単に説明すると

Targets

ターゲット分子についてその分子を対象としてアッセイされた論文に関してまとめられており、どういったジャーナルに投稿されているかや、どの年に投稿されたのかといった情報がまとめられています。また、アッセイに関しても同様にまとめられています。

Compounds

化合物に関する基本的な物理量（分子量など）のほか、Rule of 5を満たしているかといった分子の特性情報や、臨床情報などの創薬関連情報のほか、ChEMBLでの関連アッセイ、関連論文のサマリがまとめられています。

Assays

アッセイに関する情報と元論文との関連付けがされているほか、アッセイに供された化合物データへのリンクが貼られています。

Documents

論文のタイトル、ジャーナル名、アブストラクトの他に関連論文データへのリンクと、その論文中で行なわれたアッセイへのリンクと使われた化合物データへのリンクが貼られています。

あるターゲットの関連化合物を探したい場合

ある創薬ターゲット分子がどのくらい研究開発されているかを知るために、それをターゲットとしてどのくらいの化合物が合成されたのか？さらに骨格のバリエーションはどのくらい存在するのかを調べたい場合がよくあります。ChEMBLを利用するとターゲット名で探索して関連化合物をダウンロードすることができます。

ここでは抗がん剤のターゲットとして知られているTopoisomerase2を検索します。画面上部のフォームにtopoisomeraseと入力して検索するとスクリーンショットのように表示されるはずです。

EMBL-EBI Services Research Training About us EMBL-EBI

ChEMBL

UniChem ChEMBL-NTD SureChEMBL Downloads Web Services More

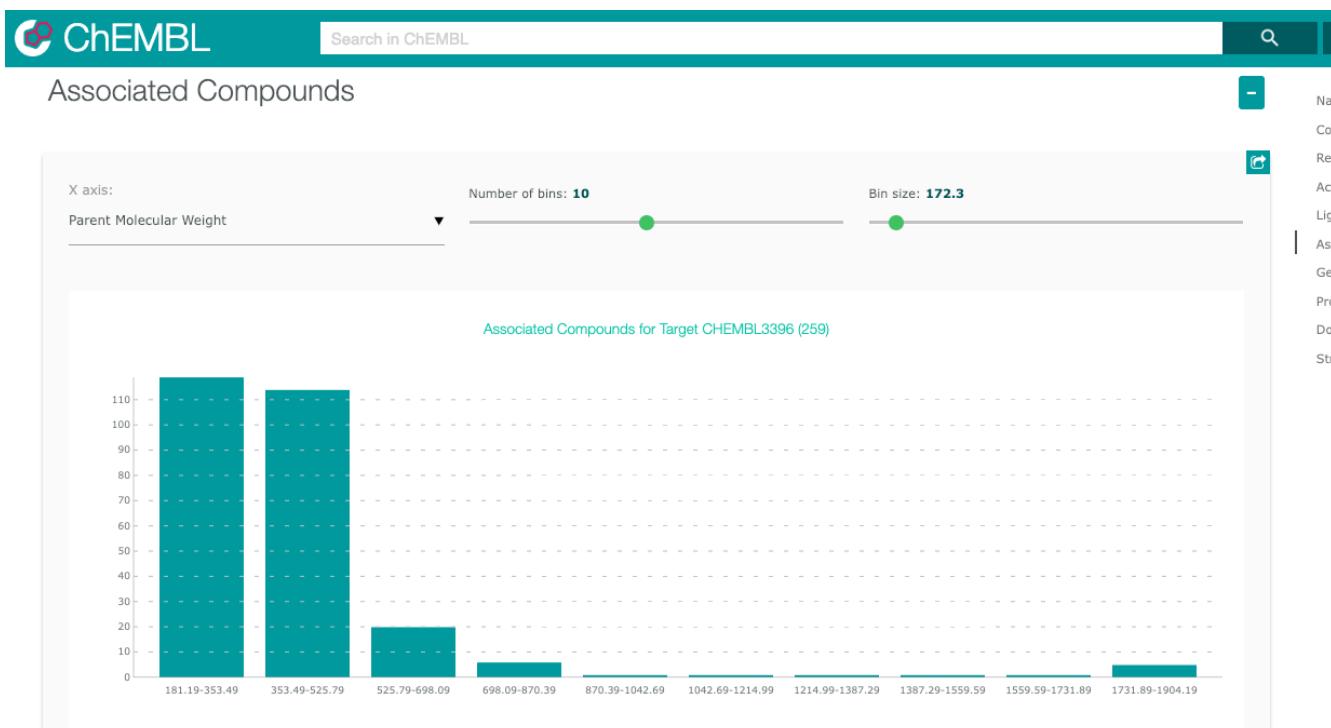
topoisomerase Examples: Dopamine HepG2 NAFRONYL NCC(=O)Oc... Draw a Structure

ChEMBL is a manually curated database of bioactive molecules with genomic data to aid the translation of geno



Topoisomerase I in the absence of DNA was evaluate	Multiple Assays
Topoisomerase I mediated DNA cleavage	Go to Assay CHEMBL815518
Topoisomerase I-mediated cleavage value on the pla	Go to Assay CHEMBL838220
Search for "topoisomerase" in all Documents	
Topoisomerase I (topo I) is an essential enzyme fo	Go to Document CHEMBL1641517
Topoisomerase I and II inhibitory activity, cytoto	Go to Document CHEMBL3585276
Topoisomerase I gene mutations at F270 in the larg	Go to Document CHEMBL1687733
Topoisomerase I inhibitors from Ruta graveolens ar	Go to Document CHEMBL1152593
Topoisomerase I-mediated antiproliferative activit	Go to Document CHEMBL1133737
Search for "topoisomerase" in all Targets	
Topoisomerase (DNA) II binding protein 1	Go to Target CHEMBL3175
Topoisomerase I	Go to Target CHEMBL6023
Topoisomerase I subunit B	Go to Target CHEMBL2150834
Topoisomerase IV	Multiple Targets
Topoisomerase IV subunit A	Multiple Targets

サジェスト機能による絞り込みでいくつか候補をリスト表示してくるのでTOP2Bを選んでください。画面をスクロールするとAssociated Compoundsセクションがありますのでグラフのタイトル(Associated Compounds for Target CHEMBL3396)をクリックすると関連化合物一覧画面が開きます。



259化合物存在することがわかります。スクロールすると全体をみることができます。画面右のアイコンをクリックするとそれぞれCSV(カンマ区切りテキスト), TSV(タブ区切りテキスト), SDF(5章で説明しています)の形式でダウンロードできます。

EMBL-EBI Services Research Training About us

ChEMBL Search in ChEMBL Examples: Dopamine HepG2 NAFRONYL NCC(=O)Oc1... Draw a Structure

UniChem ChEMBL-NTD SureChEMBL Downloads Web Services More

EBI > Databases > Chemical Biology > ChEMBL Database > Compounds > Query

Browse Compounds

[Edit Querystring](#)

[Show Full Query](#)

259 Compounds 0 Selected - Select All [Browse Activities](#)

Table Cards Graph Heatmap [Feedback](#)

Showing 1-24 out of 259 records

Records per page: 24 Select All

Filters

- Type
 - Small molecule 258
 - Unknown 1
- Max Phase
 - 0 243
 - 1 0

Chiral

あるアッセイの活性値と化合物が欲しい場合

QSARモデルを作る場合、アッセイの活性値と対応する化合物の構造情報が必要です。ChEMBLの場合アッセイのページからダウンロードすることでQSARモデル作成のためのデータを得ることができます。

大体次のような手順を辿ることがおおいです。

- 論文データを検索してからそれに関連付けられているアッセイデータを辿る
- ターゲットを検索してそれに紐付いているアッセイデータからQSARに使えそうなものを選ぶ

ここでは後者のターゲットから検索してQSARモデルに使えそうなアッセイデータを探します。心毒性関連ターゲットとしてよく知られているhERGのQSARモデルを作りたいという状況を想定しています。

検索フォームにhERGと入力して、Search hERG for all in Assaysを選びます。361件ヒットしました。

ChEMBL Search in ChEMBL Examples: Dopamine HepG2 NAFRONYL NCC(=O)Oc1... Draw a Structure

UniChem ChEMBL-NTD SureChEMBL Downloads Web Services More

ChEMBL is a manually curated database of bioactive molecules with genomic data to aid the translation of genomic information into therapeutic applications.

Drugs by Usan Year (4015)

hERG

HERGPUROLVFERR-UHFFFAOYSA-N [Go to Compound CHEMBL121245](#)

Search for "hERG" in all [Assays](#)

hERG Inhibition Assay: The effect of compounds of [Go to Assay CHEMBL3887159](#)

hERG Patch Clamp Assay: All testing was carried out on [Go to Assay CHEMBL3887163](#)

Search for "hERG" in all [Targets](#)

hERG [Multiple Targets](#)

Search for "hERG" in all [Documents](#)

hERG attracts attention as a risk factor for arrhythmia [Go to Document CHEMBL1139598](#)

hERG Channel Inhibitory Daphnane Diterpenoid Ortho States Adopted Name). Note: only shows compounds with a known USAN registration year. [Go to Document CHEMBL3588755](#)

Instructions: Click on a bar to explore the drugs' details.

モデル構築のためのデータが欲しいのでデータ数が多い順に並べ替えます。ヘッダーのCompoundsをクリックして降順に並べ替えます。

ChEMBL hERG 361 Assays 0 Selected - Select All Browse Activities

Table CSV TSV

Filters

Records per page: 20 Showing 1-20 out of 361 records

Organism Taxonomy L1: Eukaryotes (288), - N/A - (72), Unclassified (1)

Organism Taxonomy L2: Mammalia (288), - N/A - (73)

Organism Taxonomy L3: Primates (228), - N/A - (73), Rodentia (60)

Organism: - N/A - (50), Cricetus griseus (60), Homo sapiens (251)

CHEMBL ID	Search Hit	Description	Organism	Compounds	Document	BAO Format	Source
CHEMBL1909190		DRUGMATRIX: Potassium Channel HERG radioligand binding (ligand: [3H] Astemizole)	No Data	871	CHEMBL1909046	cell membrane format	DrugMatrix
CHEMBL1794573		PUBCHEM_BIOASSAY: qHTS Assay for Small Molecule Inhibitors of the Human hERG Channel Activity. (Class of assay: confirmatory)	No Data	661	CHEMBL1201862	assay format	PubChem BioAssays
CHEMBL3301459		MMV: Malaria Box compounds were tested for inhibition of the human ether a go-go related gene (hERG), Kv11.1 channel, using IonWorks 384-well patch clamp electrophysiology at 1.1uM (3 independent assay plates up to 12 cells per concentration).	Homo sapiens	400	CHEMBL3301458	assay format	MMV Malaria Box
CHEMBL3301460		MMV: Malaria Box compounds were tested for inhibition of the human ether a go-go related gene (hERG), Kv11.1 channel, using IonWorks 384-well patch clamp electrophysiology at 1.1uM (3 independent assay plates up to 12 cells per concentration).	Homo sapiens	400	CHEMBL3301458	assay format	MMV Malaria Box

論文由来で最もアッセイ数の多いCHEMBL829152を選んでクリックしてアッセイページを開きます。Activity chartの円グラフをクリックすると詳細画面が開くのでSelect allで全選択してTSV形式でダウンロードします。



NOTE

ダウンロードしたファイルをエディタで開くと^@C^@h^@E^@M^@B^@L^@と文字化けがあります。これはutf-16-leでエンコードしているためです(こうしないとExcelで問題が発生するようです)。

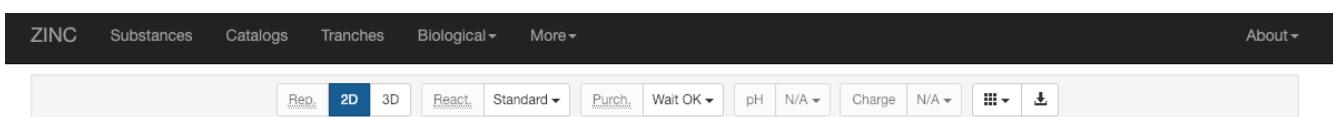
viの場合':e ++enc=utf16le'と打てばきちんと表示されるようになります。

その他有用なデータベース

ZINC

ZINCは購入可能な試薬をコレクションしたデータベースです。現在のバージョンは15で約7億5000万の構造が収載されています。もともとがドッキングシミュレーションでの利用を想定して開発されているため、三次元化したデータをダウンロードすることも可能です。ZINCのデータでバーチャルスクリーニング(6章で説明します)を行い、ヒットした化合物を購入し実際のアッセイに供するというのが主な使い方だと思います。

データのダウンロード方法は上部のTranchesタブをクリックすると次の画面に縦軸にLogP横軸に分子量の大きさで分類されそれぞれの区画にいくつの化合物が収載されているかの表が表示されます。



Molecular Weight (up to, Daltons)															Totals, by LogP
LogP (up to)	200	250	300	325	350	375	400	425	450	500	>500				
-1	33,645	277,440	1,319,186	1,755,290	3,488,430	1,056,971	304,133	69,032	44,520	25,046	5,404				8,379,097
0	177,239	1,477,795	6,369,140	8,225,850	16,370,489	4,968,586	2,044,360	602,272	403,338	225,317	3,886				40,868,272
1	495,774	4,653,245	20,366,571	25,555,666	51,168,031	17,552,850	9,214,747	3,450,662	2,404,247	1,356,981	8,149				136,226,923
2	674,318	7,756,413	38,776,127	49,277,930	101,275,509	40,958,235	25,980,328	12,010,348	8,842,022	5,320,833	21,672				290,893,735
2.5	261,046	3,795,823	22,243,430	29,105,598	60,668,177	29,200,473	21,065,482	11,499,706	7,429,727	5,530,131	23,317				190,822,910
3	151,146	2,936,586	19,642,735	26,970,502	54,928,502	32,051,899	25,218,751	15,562,600	12,423,224	7,960,152	38,718				197,884,815
3.5	66,617	1,860,573	14,766,258	20,877,818	42,592,339	30,725,871	26,784,205	18,773,130	15,474,257	10,310,674	63,485				182,295,227
4	19,842	829,788	8,715,669	11,864,102	18,295,414	22,714,919	24,444,248	19,666,611	16,950,454	11,814,568	94,819				135,410,434
4.5	2,548	231,547	4,090,246	6,802,021	11,729,007	16,330,218	18,996,424	17,747,581	16,118,050	11,798,591	131,397				103,977,630
5	96	34,593	1,271,824	2,896,815	6,042,096	9,754,599	12,748,462	13,348,637	12,900,902	10,054,998	160,191				69,213,213
>5	29	893	45,703	179,001	557,757	1,238,236	2,075,695	2,666,976	2,968,489	2,481,367	817,621				13,031,767
Totals, by Weight		1,882,300	23,854,696	137,606,889	183,510,593	367,115,751	206,552,857	168,876,835	115,397,555	95,959,230	66,878,658	1,368,659			1369M Substances

ここから必要なデータセットを選んでダウンロードボタンを押すと、実際にデータセットのURLが列挙されたテキストファイルが得られますのでそれにアクセスしてデータをダウンロードします。

統合TV

統合TVは生命科学分野の有用なデータベースやツールの使い方を動画で紹介するサイトで、[ライフサイエンス統合データベースセンター\(DBCLS\)](#)により管理、運用されています。その名の通りバイオインフォマティクス関連の動画が多いですが、ケモインフォマティクスを紹介した動画もいくつかありますので参考にしてみてください。[文献・辞書・プログラミング](#)のカテゴリも役に立つはずです。

- PubChemを利用して化学物質やアッセイの結果を調べる 2017
- ChEMBLを使って医薬品候補となる化合物について調べる

NOTE

生命科学データベース・ウェブツール 図解と動画で使い方がわかる! 研究がはかどる定番18選という書籍も出版されています。

NOTE

これ以外にもケモインフォマティクスに有用なデータベースがあればお知らせください。IssueやPRでも受け付けてます。

5章: RDKitで構造情報を取り扱う



この章ではRDKitを使って分子の読み込みの基本を覚えます。

SMILESとは

Simplified molecular input line entry system(SMILES)とは化学構造を文字列で表現するための表記方法です。 詳しくは[SMILES Tutorial](#)で説明されていますが、例えばc1ccccc1は6つの芳香族炭素が最初と最後をつないでループになっている構造、つまりベンゼンを表現していることになります。

構造を描画してみよう

SMILESで分子を表現することがわかったので、SMILESを読み込んで分子を描画させてみましょう。まずはRDKitのライブラリからChemクラスを読み込みます。二行目はJupyter Notebook上で構造を描画するための設定です。

```
from rdkit import Chem  
from rdkit.Chem.Draw import IPythonConsole  
from rdkit.Chem import Draw
```

RDKitにはSMILES文字列を読み込むためにMolFromSmilesというメソッドが用意されていますので、これを使い分子を読み込みます。

```
mol = Chem.MolFromSmiles("c1ccccc1")
```

続いて構造を描画しますが、単純にmolを評価するだけで構造が表示されます。

```
mol
```

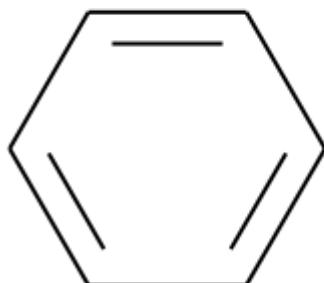
図のように構造が表示されているはずです。

```
In [31]: from rdkit import Chem  
from rdkit.Chem.Draw import IPythonConsole
```

```
In [32]: mol = Chem.MolFromSmiles("c1ccccc1")
```

```
In [35]: mol
```

Out[35]:



上のように原子を線でつなぎ構造を表現する方法（構造式）と、SMILES表記はどちらも同じものを表現しています。構造式は人が見てわかりやすいですが、SMILESはASCII文字列で表現されるのでより少ないデータ量で表現できるというメリットがあります。

NOTE 文字列で表現できるということは、文字列生成アルゴリズムを応用することで新規な化学構造を生成することも可能ということです。この内容に関しては12章で詳しく説明します。

複数の化合物を一度に取り扱うには？

複数の化合物を一つのファイルに格納する方法にはいくつかありますが、sdfというファイル形式を利用するのが一般的です。

sdfフォーマットとは？

MDL社で開発された分子表現のためのフォーマットにMOL形式というものがあります。このMOL形式を拡張したものがSDF形式です。具体的にはMOL形式で表現されたものを""という行で区切ることにより、複数の分子を取り扱えるようにしてあります。

MOL形式は分子の三次元座標を格納することができ二次元だけでなく立体構造を表現できる点はSMILESとの大きな違いです。

sdfファイルをChEMBLからダウンロードする

4章を参考にChEMBLのトポイソメラーゼII阻害試験(CHEMBL669726)の構造データをsdfファイル形式でダウンロードします。

NOTE

具体的な手順はリンクのページを開いて、検索フォームにCHEMBL669726を入力すると検索結果が表示されるので、Compoundsタブをクリックします。その後、全選択してSDFでダウンロードするとgzip圧縮されたsdfがダウンロードされるので、gunzipコマンドまたは適当な解凍ソフトで解凍してください。それをch05_compounds.sdfという名前で保存します。

RDKitでsdfを取り扱う

RDKitでsdfファイルを読み込むにはSDMolSupplierというメソッドを利用します。複数の化合物を取り扱うことになるのでmolではなくmolsという変数に格納していることに注意してください。どういう変数を使うかの決まりはありませんが、見てわかりやすい変数名をつけることで余計なミスを減らすことは心がけるとよいでしょう。

```
mols = Chem.SDMolSupplier("ch05_compounds.sdf")
```

何件の分子が読み込まれたのか確認します。数を数えるにはlenを使います。

```
len(mols)
```

34件でした。

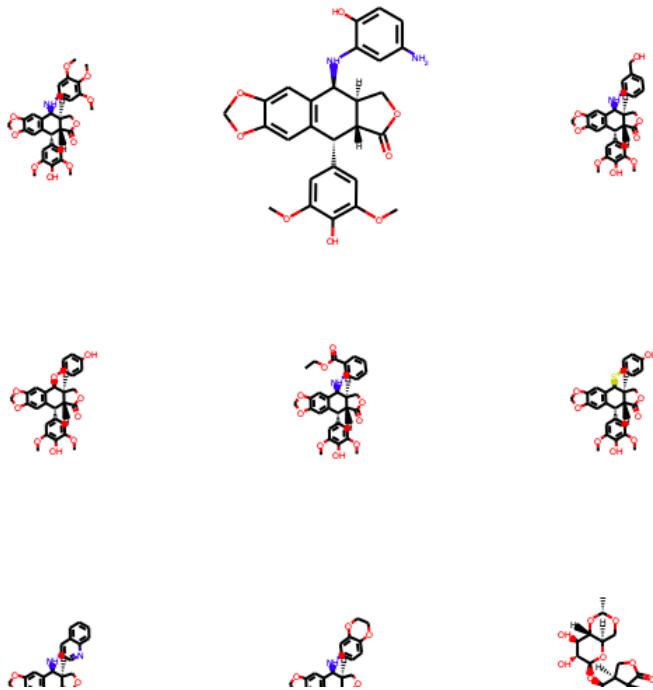
分子の構造を描画する

forループを使って、ひとつずつ分子を描画してもいいですが、RDKitには複数の分子を一度に並べて描画するメソッドが用意されているので、今回はそちらのMolsToGridImageメソッドを使います。なお一行に並べる分子の数を変更するにはmolsPerRowオプションで指定します

```
Draw.MolsToGridImage(mols)
```

In [49]: Draw.MolsToGridImage(mols)

Out[49]:



(おまけ)

参考までにループを回すやりかたも載せておきます。

```
from IPython.core.display import display
for mol in mols:
    display(mol)
```

ヘテロシャッフリングをしてみる

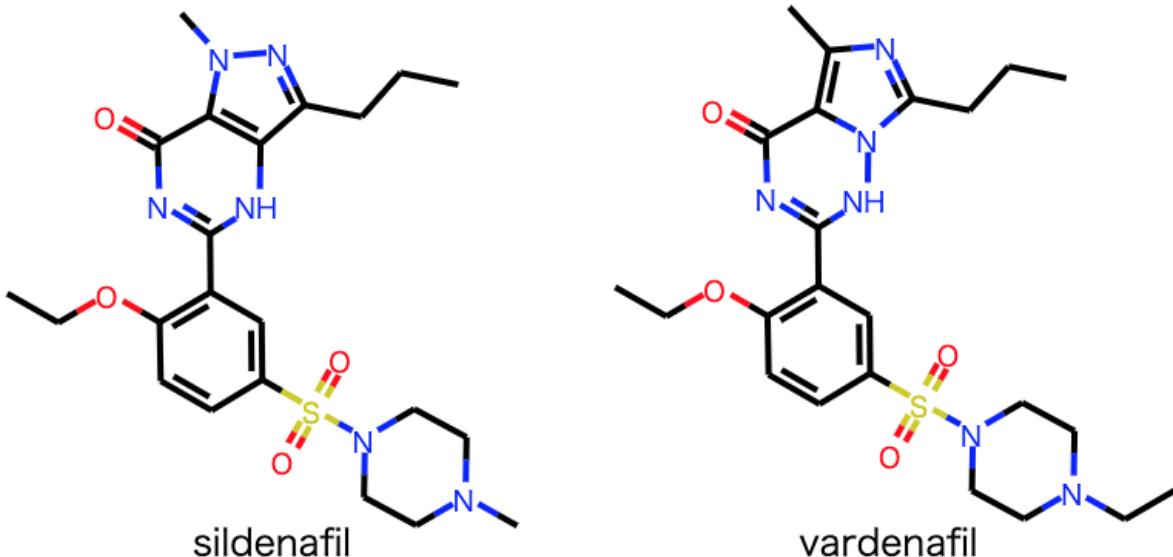


創薬の化合物最適化プロジェクトで、分子の形を変更しないで化合物の特性を変えたいことがあります。このような場合、芳香環を形成する炭素、窒素、硫黄、酸素などの原子種を入れ替えることでより良い特性の化合物が得られることがありますがこのようにヘテロ原子(水素以外の原子)を入れ替えるアプローチをヘテロシャッフリングといいます。

ヘテロシャッフリングを行うことで、活性を維持したまま物性を変化させて動態を良くする、活性そのものを向上させる、特許クレームの回避といった効果が期待できます。

少しの構造の違いが選択性や薬物動態が影響を与える有名な例として、Pfizer社のSildenafilとGSK社のVardenafilが挙げられます。

二つの構造を比較すると中心の環構造部分の窒素原子の並びが異なっているだけで極めて似ています。両分子は同じ標的蛋白質を阻害しますが、その活性や薬物動態は異なります。



上記の画像を生成するコードを示します。単にDraw.MolsToGridImageを適用するのではなく Core構造をベースにアライメントしていることとDraw.MolToGridImageのオプションにlegendsを与え、分子名を表示していることに注意してください。

```

from rdkit import Chem
from rdkit.Chem import AllChem
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Draw
from rdkit.Chem import rdDepictor
from rdkit.Chem import rdMCS
from rdkit.Chem import TemplateAlign
IPythonConsole.ipython_useSVG = True
rdDepictor.SetPreferCoordGen(True)

sildenafil = Chem.MolFromSmiles(
    'CCCC1=NN(C)C2=C1NC(=NC2=O)C1=C(OCC)C=CC(=C1)S(=O)(=O)N1CCN(C)CC1')
vardenafil = Chem.MolFromSmiles(
    'CCCC1=NC(C)=C2N1NC(=NC2=O)C1=C(OCC)C=CC(=C1)S(=O)(=O)N1CCN(CC)CC1')
rdDepictor.Compute2DCoords(sildenafil)
rdDepictor.Compute2DCoords(vardenafil)
res = rdMCS.FindMCS([sildenafil, vardenafil], completeRingsOnly=True, atomCompare
=rdMCS.AtomCompare.CompareAny)
MCS = Chem.MolFromSmarts(res.smartsString)
rdDepictor.Compute2DCoords(MCS)

TemplateAlign.AlignMolToTemplate2D(sildenafil, MCS)
TemplateAlign.AlignMolToTemplate2D(vardenafil, MCS)
Draw.MolsToGridImage([sildenafil, vardenafil], legends=['sildenafil', 'vardenafil'])

```

RDKITで再帰的にヘテロシャッフルした分子を生成するコードが以下です。 replace_atom関数で芳香環を形成する原子のインデックスを取得し、総当たりでC, N, S, Oに変更します。変更後の分子が芳香環を形成している正しい分子になっているかどうかをSanitizeMolによりチェックします。

```

def replace_atom(mol):
    res = []
    aro_idxs = [atom.GetIdx() for atom in mol.GetAromaticAtoms() if atom.GetDegree() <
3]
    for atm_num in [6, 7, 8, 16]:
        for idx in aro_idxs:
            cp_mol = copy.deepcopy(mol)
            cp_mol.GetAtomWithIdx(idx).SetAtomicNum(atm_num)
            p = Chem.MolFromSmarts("nnn")
            if cp_mol.HasSubstructMatch(p) != True:
                try:
                    smi = Chem.MolToSmiles(cp_mol)
                    smi.replace("[SH]", "s")
                    cp_mol = Chem.MolFromSmiles(smi)
                    Chem.SanitizeMol(cp_mol)
                    res.append(cp_mol)
                except:
                    pass
    return res

```

recursive_replaceという関数では再帰的に原子を入れ替えます。thresは閾値で、入力分子内の芳香環を形成する原子に対する芳香環を形成する窒素原子の数の割合です。閾値を設定することで芳香環の原子が全て炭素以外の原子になってしまふことを防いでいます。

```

def recursive_replace(mols, check=set([]), thres=0.4):
    before_n = len(check)
    print(before_n)
    for mol in mols:
        replaced_mols = replace_atom(mol)
        for mol_conv in replaced_mols:
            aro_n = Fragments.fr_Ar_N(mol)
            aro_a = len(mol.GetAromaticAtoms())
            ratio = float(aro_n) / float(aro_a)
            if ratio < thres:
                smi = Chem.MolToSmiles(mol_conv)
                check.add(smi)
    after_n = len(check)
    print(before_n, after_n)
    if before_n < after_n:
        mols = [Chem.MolFromSmiles(mol) for mol in check]
        recursive_replace(mols, check=check)
    return [Chem.MolFromSmiles(smi) for smi in check]

```

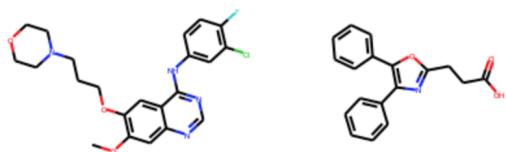
実際に使ってみます。

```
# Gefitinib
mol1 = Chem.MolFromSmiles('COC1=C(C=C2C(=C1)N=CN=C2NC3=CC(=C(C=C3)F)C1)OCCCN4CCOCC4')
# Oxaprozin
mol2 = Chem.MolFromSmiles('C1=CC=C(C=C1)C2=C(OC(=N2)CCC(=O)O)C3=CC=CC=C3')
Draw.MolsToGridImage([mol1, mol2])
```

元の分子

```
: Draw.MolsToGridImage([mol1, mol2])
```

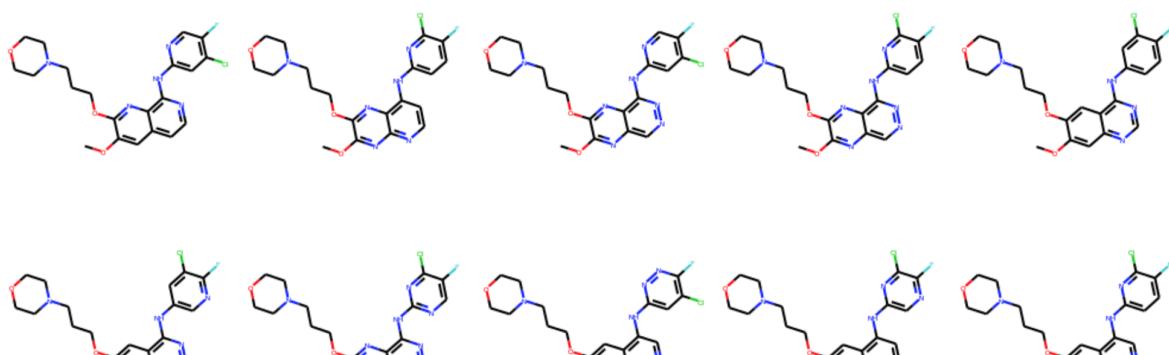
:



```
res = recursive_replace([mol1])
Draw.MolsToGridImage(res, molsPerRow=5)
```

変換後 1

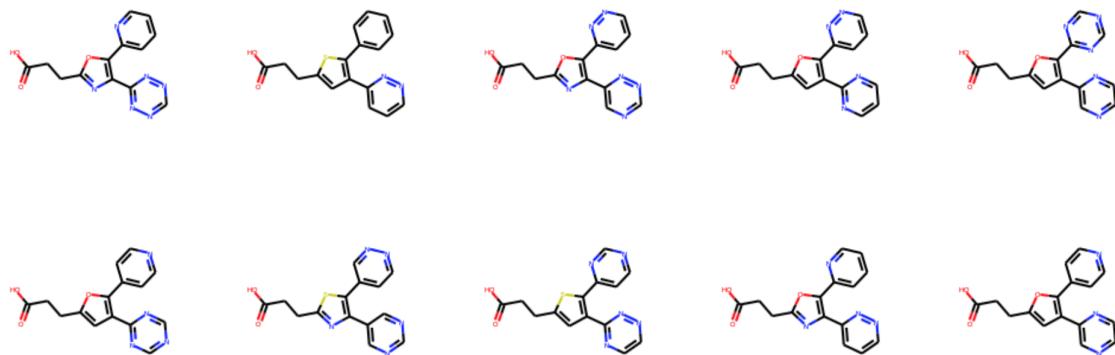
```
In [9]: Draw.MolsToGridImage(res, molsPerRow=5)
```



```
res = recursive_replace([mol2])
Draw.MolsToGridImage(res, molsPerRow=5)
```

変換後 2

```
In [10]: res = recursive_replace([mol2])
Draw.MolsToGridImage(res, molsPerRow=5)
```



どうでしょうか。二つの分子の例を示しました。一つ目は66の芳香環であり、それを形成できる原子は炭素と、窒素のみの場合です。二つ目は5員環で炭素、窒素、硫黄、酸素が原子の候補として入る場合の例です。いずれのケースでも上記のコードでヘテロ原子がシャッフルされたものが生成されています

ヘテロシャッフリングについてもう少し詳しく

[J. Med. Chem. 2012, 55, 11, 5151-5164](#)ではPIM-1キナーゼ阻害剤におけるNシャッフリングの効果をFragment Molecular Orbital法という量子化学的なアプローチを使って検証しています。さらに[J. Chem. Inf. Model. 2019, 59, 1, 149-158](#)ではAsp-Arg塩橋とヘテロ環のスタッキングのメカニズムを量子化学計算により探っており、置換デザインの指標になりそうです。

また、バイオアベイラビリティ改善のためにヘテロシャッフリングを行った例としては[J. Med. Chem. 2011, 54, 8, 3076-3080](#)があります。

6章: 化合物の類似性を評価してみる



化合物が似ているとはどういうことか?

2つの化合物が似ているとはどういうことでしょうか?なんとなく形が似ている?という表現は科学的ではありません。ケモインフォマティクスでは類似度(一般的に0-100の値を取ります)や非類似度(距離)といった定量的な尺度により似ているかどうかを評価します。

ここでは主に2つの代表的な尺度を紹介します。

記述子

分子の全体的な特徴を数値で表現するものを記述子と呼びます。分子量や極性表面性(PSA)、分配係数(logP)などがあり、現在までに多くの記述子が提案されています。これらの記述子の類似性を評価することで2つの分子がどのくらい似ているかを表現することが可能です。また分子全体の特徴を1つの数字で表現しており局所的な特徴ではないということに注意してください。

NOTE いくつかの記述子に関しては市販ソフトでないと計算できない場合があります。

フィンガープリント

もう一つがフィンガープリントです。フィンガープリントとは分子の部分構造を0,1のバイナリーで表現したもので部分構造の有無とビットのon(1),off(0)を対応させたものになり、部分構造の集合を表現することで分子の特徴を表現しています。フィンガープリントには固定長FPと可変長FPの二種類が存在し、古くはMACSKeyという固定長FP(予め部分構造とインデックスが決められているFP)が使われていましたが、現在ではECFP4(Morgan2)という可変長FPが利用されるのが普通です。

RDKitのフィンガープリントに関しては[開発者のGregさんのスライド](#)が詳しいので熟読してください。

今回はこのECFP4(Morgan2)を利用した類似性評価をしてみましょう。

SMILESとフィンガープリントの違い

SMILESは構造をASCII文字列で表現したものでフィンガープリントは部分構造の有無をバイナリで表現したものです。違いは前者は構造表現の一つであるのに対し、後者は特徴表現の一つだということです。部分構造の有無だけしか表現していないため、部分構造間の関係性(どう位置関係でつながっているのか)といった情報が失われ、もとの構造に戻ることはできません。

テキストマイニングでよく用いられるBag-of-Wordsに対応するのでBag-of-Fragmentsと呼ぶ人もいます。

類似度を計算する

簡単な分子としてトルエンとクロロベンゼンの類似性を評価してみましょう。

```
from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem, Draw
from rdkit.Chem.Draw import IPythonConsole
```

SMILESで分子を読み込みます。

```
mol1 = Chem.MolFromSmiles("Cc1ccccc1")
mol2 = Chem.MolFromSmiles("Clc1ccccc1")
```

一応目視で確認しておきます。

```
Draw.MolsToGridImage([mol1, mol2])
```

ECFP4に相当する半径2のモルガンフィンガープリントを生成します。

```
fp1 = AllChem.GetMorganFingerprint(mol1, 2)
fp2 = AllChem.GetMorganFingerprint(mol2, 2)
```

類似度の評価にはタニモト係数を使います。

```
DataStructs.TanimotoSimilarity(fp1, fp2)
# 0.5384615384615384
```

バーチャルスクリーニング

ここまでで化合物の類似性の評価方法について説明しました。この類似性の指標を用い多くの化合物の中から特定の化合物群を選び出すことをバーチャルスクリーニングと呼びます。

例えば薬になりそうな化合物が特許や論文で発表されたり、自社のアッセイ系で有望そうな化合物が見つかった場合、自社の化合物ライブラリデータベースや市販化合物のデータベースの中に類似の化合物で、より有望そうなものがあるかどうかを探したいことがあります。ここではノイラミニダーゼ阻害薬として知られるインフルエンザ治療薬である*イナビル*の類似体が購入可能であるかをZINCを利用して調べます。

イナビルの分子量が約350,LogPが 約-3だったのZINCの分子量350-375,LogP=-1の340万化合物の区画を選択しました。この区画は16のファイルに分かれていますが、最初の1セットだけダウンロードして使ってみます。

NOTE データのダウンロード方法は4章で説明しています。

jupyter notebookでは!で始めるとShellコマンドを実行できます。以下はjupyter notebook上でwgetコマンドでZINCのデータセットをダウンロードする例です

```
!wget http://files.docking.org/2D/EA/EAED.smi
```

ファイルからSMILESを読み込んでmolオブジェクトにしますが最初の行はヘッダーなので読み飛ばします。また、各行の最終文字は改行文字なのでl[:-1]として除いています。最後に何化合物あるか調べます。

```
mols = []
with open("EAED.smi") as f:
    f.readline()
    for l in f:
        mol = Chem.MolFromSmiles(l[:-1])
        mols.append(mol)
print(len(mols))
# 195493
```

続いてイナビル(LANIMAMIBIR)との類似度を調べるための関数を用意します。

```
laninamivir = Chem.MolFromSmiles(
    "CO[C@H]([C@H](O)CO)[C@@H]1OC(=C[C@H](NC(=N)N)[C@H]1NC(=O)C)C(=O)O")
laninamivir_fp = AllChem.GetMorganFingerprint(laninamivir, 2)

def calc_laninamivir_similarity(mol):
    fp = AllChem.GetMorganFingerprint(mol, 2)
    sim = DataStructs.TanimotoSimilarity(laninamivir_fp, fp)
    return sim
```

調べてみます。

```
similar_mols = []
for mol in mols:
    sim = calc_laninamivir_similarity(mol)
    if sim > 0.2:
        similar_mols.append((mol, sim))
```

結果を類似度の高い順に並べ替えて最初の10件だけ取り出します。

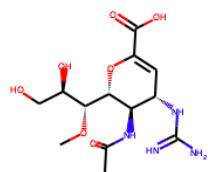
```
similar_mols.sort(key=lambda x: x[1], reverse=True)
mols = [l[0] for l in similar_mols[:10]]
```

描画してみます。

```
Draw.MolsToGridImage(mols, molsPerRow=5)
```

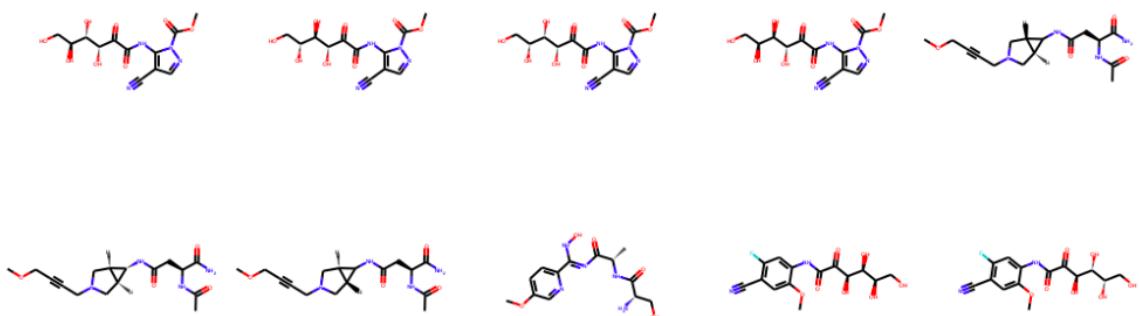
In [32]: Iainamivir

Out[32]:



In [30]: Draw.MolsToGridImage(mols, molsPerRow=5)

Out[30]:



類似度を確認すればわかりますが、今回調べた約20万件の化合物は最高でも23%の類似度の化合物しか見いだせませんでした。しかしZINCは7億5000万件のデータを収録してあるのでその中にはもっと似ている化合物はたくさんあるはずです。

クラスタリング

例えば市販化合物を購入してライブラリを作る場合にはできるだけ多様性をもたせたいので、似ている化合物ばかりが偏らないように類似化合物どうしをまとめ、その中の代表を選びます。このように化合物を構造の類似性でまとめたい場合、クラスタリングという手法を使います。

Novrtisの抗マラリアアッセイの5614件のヒット化合物をクラスタリングします。

クラスタリング用のライブラリをインポートし、データを読み込みます。

```
from rdkit.ML.Cluster import Butina
mols = Chem.SDMolSupplier("ch06_nov_hts.sdf")
```

何らかの理由でRDKitで分子の読み込みができない場合、molオブジェクトではなくNoneが生成されます。このNoneをGetMorganFingerprintAsBitVectメソッドにわたすとエラーになるので、Noneを除きながらフィンガープリントを生成します。

```

fps = []
valid_mols = []

for mol in mols:
    if mol is not None:
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2)
        fps.append(fp)
        valid_mols.append(mol)

```

フィンガープリントから距離行列（下三角の距離行列）を生成します。

```

distance_matrix = []
for i, fp in enumerate(fps):
    similarities = DataStructs.BulkTanimotoSimilarity(fp, fps[:i+1])
    distance_matrix.extend([1-sim for sim in similarities])

```

距離行列を用いて化合物をクラスタリングします。3番目の引数は距離の閾値です。この例では距離0.2つまり80%以上の類似度の化合物でクラスタリングしています。

```
clusters = Butina.ClusterData(distance_matrix, len(fps), 0.2, isDistData=True)
```

クラスタ数を確認します。

```

len(clusters)
#2492

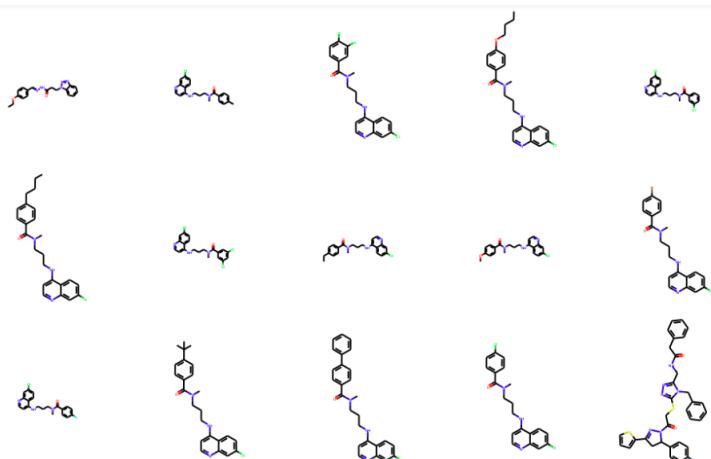
```

最初のクラスタの構造を表示してみます

```

mols_ =[valid_mols[i] for i in clusters[0]]
Draw.MolsToGridImage(mols_, molsPerRow=5)

```



今回はRDKitに用意されているライブラリでクラスタリングを行いましたが、Scikit-learnでも幾つかの手

法が利用できますし、実際にはこちらの方を使うことが多いです。

Structure Based Drug Design(SBDD)

(LBVSに触れたのでSBVS,SBDDにも言及するかどうか検討中のサブセクション)

ここでは抗凝固薬として上市されているapixaban, rivaroxabanの類似性を評価します。構造を見るとわかりますが、なんとなく似ていますが、どの部分とどの部分が対応するか想像つくでしょうか？実はこの2つの化合物は両方共FXaというセリンプロテアーゼの同じポケットに同じような結合モードで結合することでプロテアーゼの働きを阻害することが知られています。興味があれば実際にPDBから複合体の結晶構造を探して眺めてみるといいかもしれません。（pymol入門まで拡張するか？要検討）

```
apx = Chem.MolFromSmiles("C0c1ccc(cc1)n2nc(C(=O)N)c3CCN(C(=O)c23)c4ccc(cc4)N5CCCCC5=O")
rvx = Chem.MolFromSmiles("Clc1ccc(s1)C(=O)NC[C@H]2CN(C(=O)O2)c3ccc(cc3)N4CCOCC4=O")
```

構造を眺めてみます。メトキシフェニルとクロロチオールは同じような結合様式をとるんでしょうか？このような結合の成分をきちんと評価する方法もあるのですが、本書の内容を超えるので説明はしません。もし興味があればFragment Molecular Orbital Methodで調べてみてください

```
Draw.MolsToGridImage([apx, rvx], legends=["apixaban", "rivaroxaban"])
```

```
apx_fp = AllChem.GetMorganFingerprint(apx, 2)
rvx_fp = AllChem.GetMorganFingerprint(rvx, 2)
```

```
DataStructs.TanimotoSimilarity(apx_fp, rvx_fp)
```

```
# 0.40625
```

40%くらいの類似度ということになりました。

7章: グラフ構造を利用した類似性の評価



グラフとはノード（頂点）群とノード間の連結関係を示すエッジ（枝）群で構成されるデータのことを指します。化学構造はこのグラフで表現できます。つまり原子をノード、結合をエッジとしたグラフ構造で表せます。

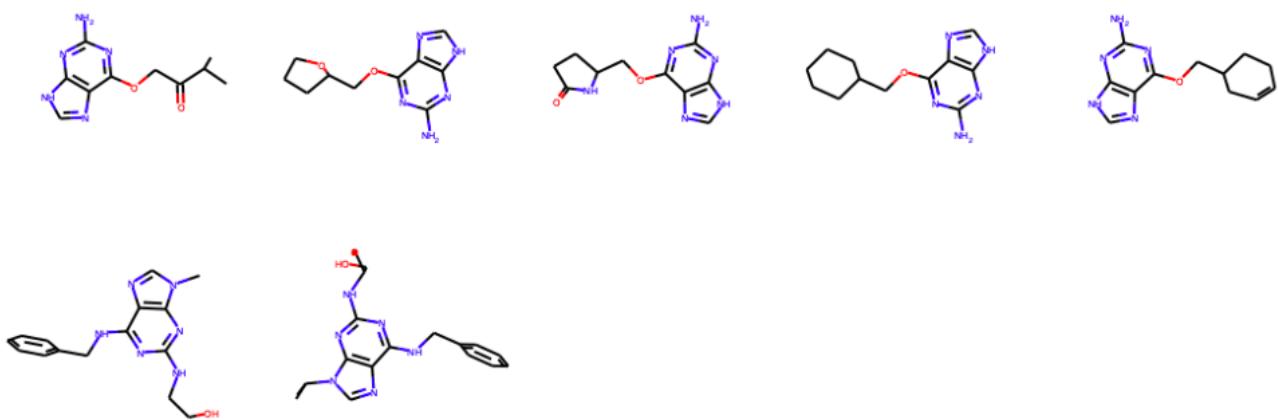
通常、6章で紹介したようなフィンガープリントを使い分子同士の類似性を評価することが多いですが、グラフ構造を利用して類似性を評価する手法もあります。次に紹介するMCS（Maximum Common Substructure）は対象となる分子集合の共通部分構造のことを指します。共通部分構造が多いほどそれらの分子はより似ていると考えます。

主要な骨格による分類(MCS)

最大共通部分構造Maximum Common Substructure(MCS)とは与えられた化学構造群において共通する最大の部分構造のことです。RDKitではMCS探索のためにrdFMCSというモジュールが用意されています。

今回はMCS探索のサンプルデータとしてrdkitに用意されているcdk2.sdfというファイルを利用します。RDConfig.RDDocsDirが、サンプルデータのディレクトリを表す変数で、そのディレクトリ以下のBooks/data/にcdk2.sdfというファイルが存在するので、os.path.joinメソッドでファイルパスを設定します。尚、os.path.joinはosのパスの違いを吸収するためのpythonの組み込みモジュールです。

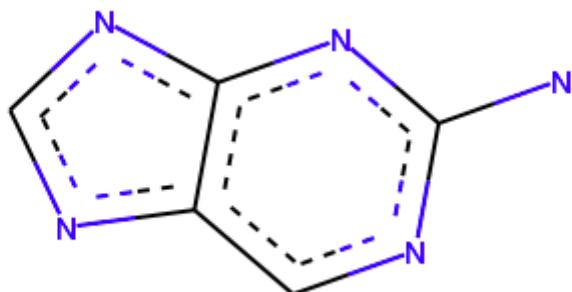
```
import os
from rdkit import Chem
from rdkit.Chem import RDConfig
from rdkit.Chem import rdFMCS
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Draw
filepath = os.path.join(RDConfig.RDDocsDir, 'Book', 'data', 'cdk2.sdf')
mols = [mol for mol in Chem.SDMolSupplier(filepath)]
# 構造を確認します
Draw.MolsToGridImage(mols[:7], molsPerRow=5)
```



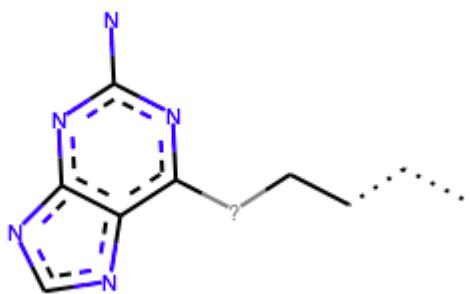
読み込んだ分子を使ってMCSを取得します。RDKitではMCSの取得方法に複数のオプションが指定できます。以下にそれぞれのオプションでの例を示します。

1. デフォルト
2. 原子がなんであっても良い（構造とボンドの次数があつていれば良い）
3. 結合次数がなんでも良い（例えば、ベンゼンとシクロヘキサンは同じMCSとなる）

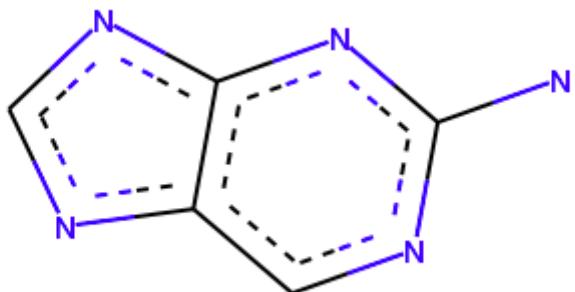
```
result1 = rdFMCS.FindMCS(mols[:7])
mcs1 = Chem.MolFromSmarts(result1.smartsString)
mcs1
print(result1.smartsString)
#[#6]1:[#7]:[#6](:[#7]:[#6]2:[#6]:1:[#7]:[#6]:[#7]:2)-[#7]
```



```
result2 = rdFMCS.FindMCS(mols[:7], atomCompare=rdFMCS.AtomCompare.CompareAny)
mcs2 = Chem.MolFromSmarts(result2.smartsString)
mcs2
print(result2.smartsString)
#[#6]-,:[#6]-,:[#6]-[#6]-[#8,#7]-[#6]1:[#7]:[#6](:[#7]:[#6]2:[#6]:1:[#7]:[#6]:[#7]:2)-[#7]
```

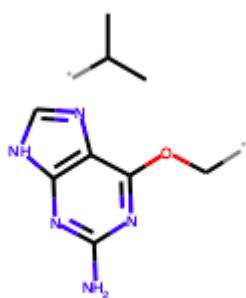


```
result3 = rdFMCS.FindMCS(mols[:7], bondCompare=rdFMCS.BondCompare.CompareAny)
mcs3 = Chem.MolFromSmarts(result3.smartsString)
mcs3
print(result3.smartsString)
#[#6]1:[#7]:[#6](:[#7]:[#6]2:[#6]:1:[#7]:[#6]:[#7]:2)-[#7]
```



RDKitではMCSに基づく類似性を数値化するアルゴリズムのひとつにFraggle Similarityが実装されています。これを利用することでクラスタリングや、類似性に基づいた解析が行なえます。

```
from rdkit.Chem.Fraggle import FraggleSim
sim, match = FraggleSim.GetFraggleSimilarity(mols[0], mols[1])
print(sim, match)
#0.925764192139738 *C(C)*C0c1nc(N)nc2[nH]cnc12
match_st = Chem.MolFromSmiles(match)
match_st
```



このようにFraggleSimilarityは類似性及びマッチした部分構造を返します。ECFPを利用した類似性よりもケミストの感覚に近いことが多いです。詳しくは参考リンクを参照してください。

参考リンク

- [Efficient Heuristics for Maximum Common Substructure Search](#)

- Fraggle – A new similarity searching algorithm

Matched Molecular PairとMatched Molecular Series



創薬研究の構造最適化ステージにおいて、起点となる化合物（リード化合物）をどのように構造変換していくかは非常に重要ですが、ステージが進んだ場合どの構造変換が活性や物性に影響を及ぼしたかというレトロスペクティブな解析することもまた非常に大切です。

TIP

興味があればhttps://sar.pharm.or.jp/wp-content/uploads/2018/09/SARNews_19.pdfを読むといいです。

Matched Molecular Pair(MMP)は、二つの分子のうち一部の部分構造だけが異なりそれ以外は同一な分子のペアのことです。例として、クロロベンゼンとフルオロベンゼンはCl基とF基のみが異なるのでMMPです。このようなペアの特性の変化を大量に解析することで、置換基変換のトレンドを掴むことができます。これをMatched Molecular Pair Analysis (MMPA)と呼びます。大規模なデータでMMPAを行うことにより、置換基の変化がもたらす特性変化の普遍的なルールを抽出できます。このようなルールを理解していれば構造最適化を効率的に進められることになります。

ここではRDKitのContribに提供されている[RDKit/Contrib/MMPA\[mmpa\]](#)を使ってMMP解析を行います。

RDKitインストール先の下にあるContrib/mmpaに移動し、pythonスクリプトを順次実行します。

```
python rfrag.py <MMPAを実施したいFileの名前>フラグメント化したデータの保存ファイル名  
# 例えば  
# python rfrag.py <data/sample.smi>data/sample_fragmented.txt
```

```
python indexing.py <先のコマンドでできたフラグメントのファイル>MMP_アウトプットファイル.CSV  
# 例えば  
# python index.py <data/sample_fragmented.txt>data/mmp.csv
```

以上のコマンドを実行すると分子A, 分子B, 分子AのID, 分子BのID, 変換された構造のSMIRKS, 共通部分構造 (context) がcsvファイルが生成されます。このデータに基づき活性や物性などを紐つけることでMMPAが行えます。

NOTE

SMIRKSは分子の変換をSMILESのように文字列表記によって表現する手法です。

MMPの拡張としてMatched Molecular Series(MMS)という手法も提案されています。MMPは分子のペアですが、このペアを共通構造をもった3つ以上の集団としてリスト化したものがMMSです。

実際にMMSを作ってみます。以下の例ではChEMBLのFactor Xaのデータを使いました。MMSの実装に関してはNoel O'Boyle氏のRDKit UGMでの[プレゼンテーション](#)のコードを利用しています。

実際にMMSを作ってみましょう。以下の例ではChEMBLよりFactor Xaのデータを[ダウンロード](#)し、一例

として使いました。MMSの実装に関してはNoel O'Boyle氏のRDKit UGMでの[プレゼンテーション](#)のコードを利用しています。

まず利用するライブラリの読み込みと、データの読み込みを行い、SaltRemoverを使い脱塩しています。

```
import sys
import os
import pandas as pd
from rdkit import Chem
from rdkit.Chem import rdMMPA
from rdkit.Chem import RDConfig
from rdkit.Chem import rdBase
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import Draw
from rdkit.Chem import SaltRemover
mmpopath = os.path.join(RDConfig.RDContribDir, 'mmpa')
sys.path.append(mmpopath)
df = pd.read_csv('ChEMBL_FXa.txt', sep='\t')
remover = SaltRemover.SaltRemover()
mols = []
for i, smi in enumerate(df.CANONICAL_SMILES):
    try:
        mol = Chem.MolFromSmiles(smi)
        mol.SetProp('CMPD_CHEMBLID', df.CMPD_CHEMBLID[i])
        mol = remover.StripMol(mol)
        mols.append(mol)
    except:
        print(smi)
```

続いてRDKit contribに登録されているmmpaのrfragをインポートして、分子をフラグメントに分割します。

```
import rfrag
rfragdata = []
for i, smi in enumerate(df.CANONICAL_SMILES):
    try:
        out = rfrag.fragment_mol(smi, df.CMPD_CHEMBLID[i])
        rfragdata.append(out)
    except:
        print(smi, df.CMPD_CHEMBLID[i])
```

MMSを作成する関数を定義します。コードはUGMの資料に記載されているものをほぼそのまま利用しますが、Jupyter上で全ての処理をおこなうため読み込み先をファイルからリストに変更しました。

以下MMSの作成プロセスの概要です。

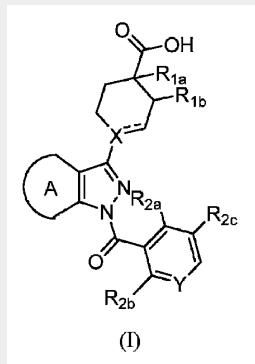
1. 各分子を一定のルール（回転可能結合で切断など）でカット
2. カットしたフラグメントがkeyの辞書を作成、同じキーを持つ分子のフラグメントを辞書のvalueに格納

上記の作業を繰り返すことで共通のスキヤフォールドを持つ分子をまとめられます。共通のスキヤフォールドでまとまった分子は、スキヤフォールド以外の置換基が異なる分子となります。

スキヤフォールドとは？

創薬において、前臨床試験の前のステージに構造最適化というステージがあり、そこでは化合物の主要骨格以外の部分をちょこまかと変換して薬にふさわしいバランスの取れたプロパティにします。

この主要骨格のことをスキヤフォールドと呼びます。例えば[この特許](#)ではRを除いた部分は固定されておりこの主要骨格をスキヤフォールドと呼びます。



```

from collections import namedtuple

Frag = namedtuple( 'Frag', [ 'id', 'scaffold', 'rgroup' ] )

class Series():
    def __init__( self ):
        self.rgroups = []
        self.scaffold = ""

    def getFrags(rfrags):
        frags = []
        for lines in rfrags:
            for line in lines:
                broken = line.rstrip().split(",")
                if broken[2]: # single cut
                    continue
                smiles = broken[-1].split(".")
                mols = [Chem.MolFromSmiles( smi ) for smi in smiles]
                numAtoms = [mol.GetNumAtoms() for mol in mols]
                if len(numAtoms) < 2:
                    continue
                if numAtoms[0] > 5 and numAtoms[1] < 12:
                    frags.append(Frag(broken[1], smiles[0], smiles[1]))
                if numAtoms[1] > 5 and numAtoms[0] < 12:
                    frags.append(Frag(broken[1], smiles[1], smiles[0]))
        frags.sort(key=lambda x:(x.scaffold, x.rgroup))
        return frags

    def getSeries(frags):
        oldfrag = Frag(None, None, None)
        series = Series()
        for frag in frags:
            if frag.scaffold != oldfrag.scaffold:
                if len(series.rgroups) >= 2:
                    series.scaffold = oldfrag.scaffold
                    yield series
                series = Series()
            series.rgroups.append((frag.rgroup, frag.id))
            oldfrag = frag
        if len(series.rgroups) >= 2:
            series.scaffold = oldfrag.scaffold
            yield series

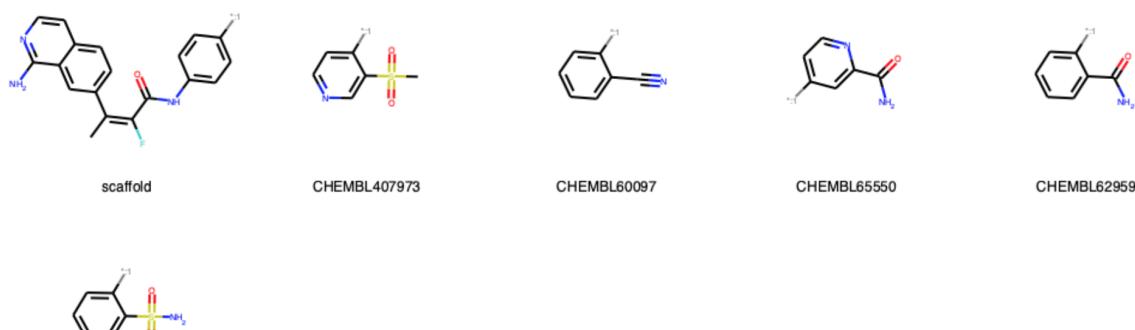
```

MMSを作る準備ができたので実行します。同じスキヤフォールドに対して4つ以上置換基の変換があったデータのみを可視化します。

```

frags = getFrags(rfragdata)
series = getSeries(frags)
series =[i for i in series]
from IPython.display import display
for s in series[:50]:
    mols = [Chem.MolFromSmiles(s.scaffold)]
    ids = ['scaffold']
    for r in s.rgroups:
        rg = Chem.MolFromSmiles(r[0])
        mols.append(rg)
        ids.append(r[1])
    if len(mols) > 5:
        display(Draw.MolsToGridImage(mols, molsPerRow=5, legends=ids))
    print("#####")

```



スキヤフォールドに対して5つの置換基のMMSが表示されました。

NOTE このMMSを利用して[活性予測](#)を行うこともできます。

Cytoscapeを使ってMMPネットワークを可視化する

WARNING この内容は入門の内容を超えるので興味がなければ飛ばしてください

MMPは変換前、変換後の情報をノード、変換ルールをエッジとするグラフ構造と考えることができます。Cytoscapeなどのネットワーク可視化ツールを利用するとこのグラフ構造を直感的に把握できます。

RDKitには先に紹介したMMPAの他に[mmpdb](#)という別プロジェクトがあります。こちらはコマンドラインのツール群とデータベースシステムとして提供されているため、長期的な管理がしやすいという特徴があります。本セクションではこの[mmpdb](#)と[Cytoscape](#)を利用したMMPの可視化を紹介します。

NOTE [mmpdb: An Open Source Matched Molecular Pair Platform for Large Multi-Property Datasets](#)

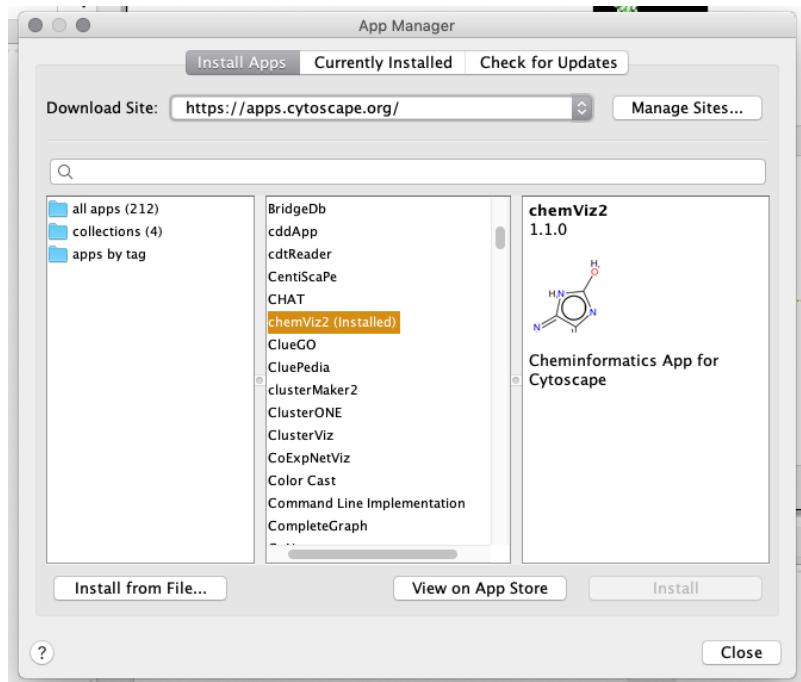
Cytoscapeのインストール

[Cytoscape](#)はオープンソースのネットワーク可視化ソフトで色々なシーンで広く使われています。化合物

の構造表示用プラグインを使うことで構造のネットワークを表示することができます。

インストールは簡単で[ダウンロードサイト](#)から対応するOSのインストーラをダウンロードして指示のとおりにインストールするだけです。

インストールが完了したらCytoscapeを起動して化合物構造描画用のChemviz2プラグインをインストールします。手順は簡単でApps→App Managerからchemviz2を選択してインストールします。



mmpdbからgmlファイルを作成する

今回利用するデータは[Inhibition of recombinant GSK3-beta](#) J. Med. Chem. (2008) 51:2062-2077 の151化合物です。MMPAを行うにはHTSのような探索データではなくて構造最適化のようにスキヤフオールドが決まっているものを使うのが原則です。

コマンドの流れを載せておきます。SMILESのtextと活性や物性値のデータは別々にデータベースに登録する必要があります。

```
$ mmpdb fragment smiles.txt -o CHEMBL930273.fragments      # fragmentation
$ mmpdb index CHEMBL930273.fragments -o CHEMBL930273.db      # make db
$ mmpdb loadprops -p act.txt CHEMBL930273.db                  # load properties
```

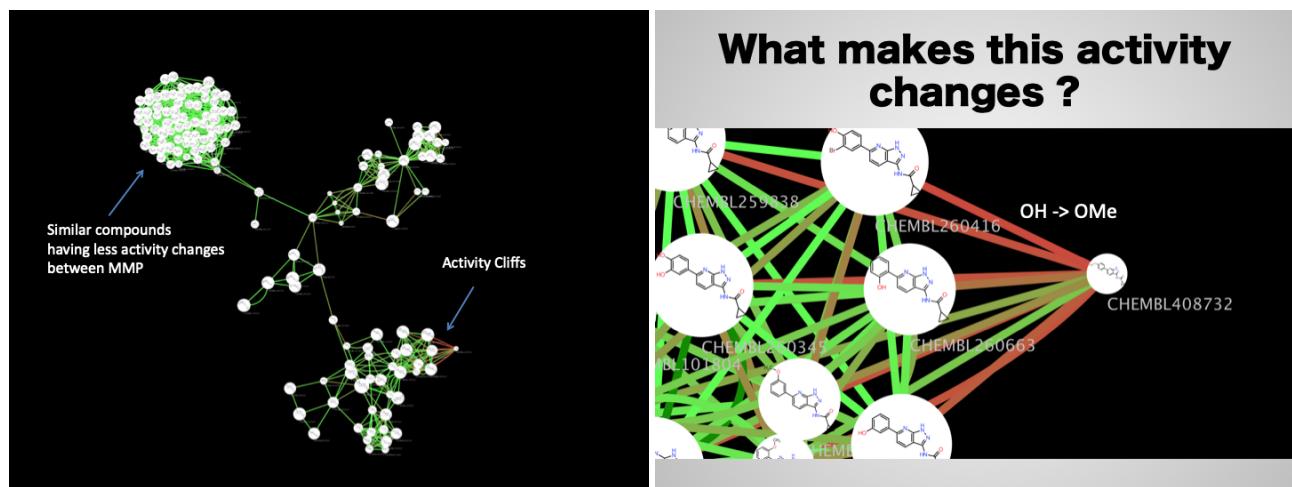
その後Cytoscapeで読み込むためのgmlファイルを作成しますが、これは本書の範囲を超えるので割愛します。もし興味があるのであれば[コード](#)を直接読んでもらうといいのですが流れは以下のとおりです。

1. [mmpdbからpython-igraphを使ってgmlファイルを作る](#)
2. [gmlファイルをCytoscapeで読み込む](#)
3. Cytoscapeで属性を各パラメータにアサインして視覚的に理解しやすくする
 - a. ノードの大きさを物性値に対応
 - b. エッジの色を活性差に対応

c. chemviz2 pluginで構造を描画してノードに貼り付ける

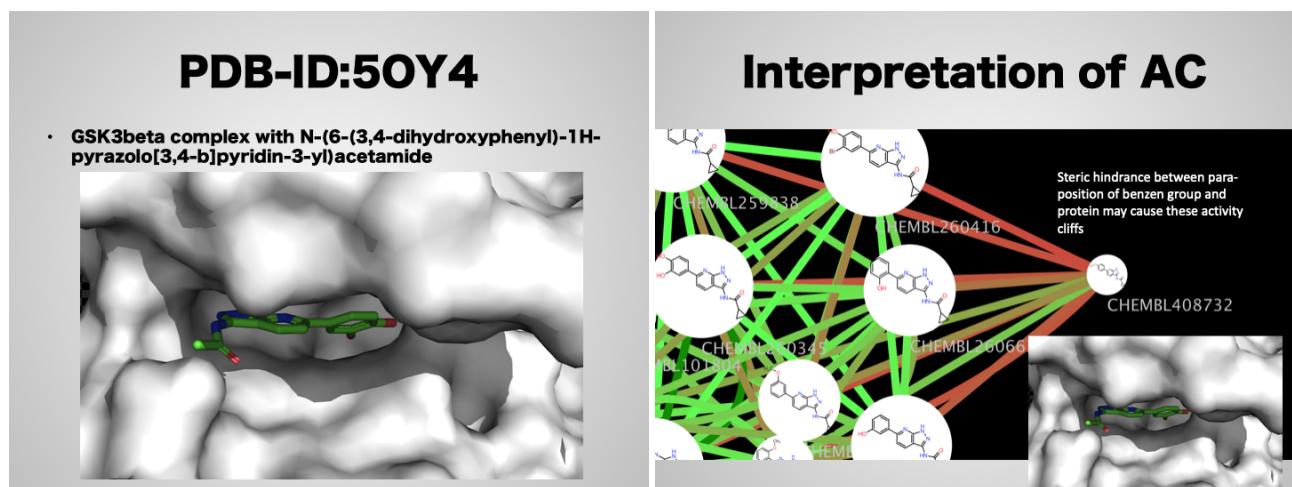
解釈する

MMPネットワークを見てみましょう。あまり活性差のないMMPが左上の方に固まっています。右下の方にはエッジが赤い（活性差が大きい）ものが観測されます。このような小さな置換基変化が大きな活性差を生むもMMPをActivity Cliffと呼びます。一般的にActivity Cliffは創薬プロジェクトにおいてブレークスルーとなることが多いため、このような活性変化を見逃さないことは大切です。



実際にどういう置換が行われたのかを確認すると、OH基がMeO基に置換されることで活性の消失が起こっています。

MMPだけではこのように単純に事実しかわからないので、もう少し深く考察するために類似体の複合体結晶構造を探してみました。するとPDBID:5OY4というGSK3 β と類似化合物の複合体が見つかりました。



OH基をMeO基に置換するとポケットの壁にぶつかりそうですね。つまりこのActivity Cliffはリガンドと蛋白質の立体障害により引き起こされたと考えられます。

8章: 沢山の化合物を一度にみたい



澤山のデータがどのように分布しているのかを見るには適当な空間にマッピングするのが一般的です。特にケモインフォマティクスではケミカルスペースという言葉が使われます。

Chemical Spaceとは

ケミカルスペースとは化合物を何らかの尺度でn次元の空間に配置したものを指します。一般に、2次元または3次元が使われることが多いです（人間の理解のため）。尺度つまり類似性に関しては色々な手法が提案されていますが、うまく化合物の特徴を表すような距離が定義されるように決められることが多いです。

今回は睡眠薬のターゲットとして知られているOrexin Receptorのアンタゴニストについて、どの製薬企業がどういった化合物を開発しているのかを視覚化してみます。データのダウンロード方法は4章を参照してください。今回は表の10個の論文のデータを利用しました。

今回知りたいことは主に以下の2つです。

- 似たような化合物を開発していた会社はあったのか？
- Merckは似たような骨格ばかり最適化していたのか、それとも複数の骨格を最適化したのか？

Table 1. Orexin Receptor Antagonist

Doc ID	Journal	Pharma
CHEMBL3098111	Bioorg. Med. Chem. Lett. (2013) 23:6620-6624	Merck
CHEMBL3867477	Bioorg Med Chem Lett (2016) 26:5809-5814	Merck
CHEMBL2380240	Bioorg. Med. Chem. Lett. (2013) 23:2653-2658	Rottapharm
CHEMBL3352684	Bioorg. Med. Chem. Lett. (2014) 24:4884-4890	Merck
CHEMBL3769367	J. Med. Chem. (2016) 59:504-530	Merck
CHEMBL3526050	Drug Metab. Dispos. (2013) 41:1046-1059	Actelion
CHEMBL3112474	Bioorg. Med. Chem. Lett. (2014) 24:1201-1208	Actelion
CHEMBL3739366	MedChemComm (2015) 6:947-955	Heptares
CHEMBL3739395	MedChemComm (2015) 6:1054-1064	Actelion
CHEMBL3351489	Bioorg. Med. Chem. (2014) 22:6071-6088	Eisai

ユークリッド距離を用いたマッピング

描画ライブラリにはggplotを使います。主成分分析(PCA)を利用して、化合物が似ているものは近くになるように分布させて可視化します。まずは必要なライブラリをインポートします

```
from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem, Draw
import numpy as np
import pandas as pd
from ggplot import *
from sklearn.decomposition import PCA
import os
```

ダウンロードしたsdfを読み込んで、製薬企業とドキュメントIDの対応が取れるようにしてそれぞれの化合物についてフィンガープリントを構築します。もし不明な点があれば6章を確認してください。

```
oxrs = [("CHEMBL3098111", "Merck"), ("CHEMBL3867477", "Merck"),
        ("CHEMBL2380240", "Rottapharm"), ("CHEMBL3352684", "Merck"),
        ("CHEMBL3769367", "Merck"), ("CHEMBL3526050", "Actelion"),
        ("CHEMBL3112474", "Actelion"), ("CHEMBL3739366", "Heptares"),
        ("CHEMBL3739395", "Actelion"), ("CHEMBL3351489", "Eisai")]

fps = []
docs = []
companies = []

for cid, company in oxrs:
    sdf_file = os.path.join("ch08", cid + ".sdf")
    mols = Chem.SDMolSupplier(sdf_file)
    for mol in mols:
        if mol is not None:
            fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2)
            arr = np.zeros((1,))
            DataStructs.ConvertToNumpyArray(fp, arr)
            docs.append(cid)
            companies.append(company)
            fps.append(arr)
fps = np.array(fps)
companies = np.array(companies)
docs = np.array(docs)
```

フィンガープリントの情報を確認すると10の論文から293化合物のデータが得られていることがわかります。

```
fps.shape
# (293, 2048)
```

これで主成分分析の準備完了です。主成分の数はn_componentsで指定できますが今回は二次元散布したいので2にします。

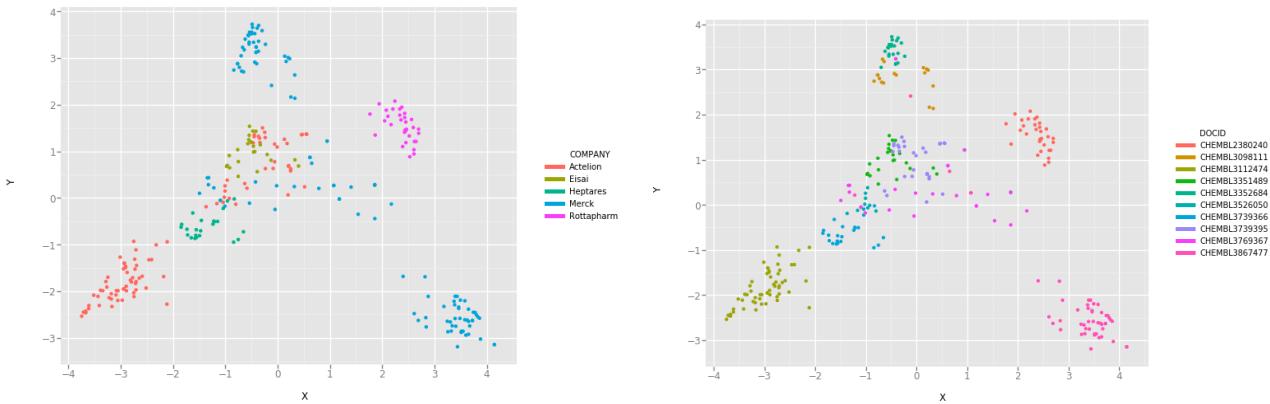
```
pca = PCA(n_components=2)
x = pca.fit_transform(fps)
```

描画します。colorオプションを変えると、それぞれのラベルに応じた色分けがされるので、COMPANYとDOCIDの2つの属性を選んでみました。

```
d = pd.DataFrame(x)
d.columns = ["PCA1", "PCA2"]
d["DOCID"] = docs
d["COMPANY"] = companies
g = ggplot(aes(x="PCA1", y="PCA2", color="COMPANY"), data=d) + geom_point() + xlab("X") + ylab("Y")
g
```

各製薬会社がどのような化合物を最適化したのかがわかるようになりました。ケミカルスペースの中心部に各社重なる領域があるので、Merck, Acterion, Eisai, Heptaressは似たような化合物を最適化していたと思われます。Acterionはうまく独自性のある方向(左下)に展開できたのか、展開できなくてレッドオーシャン気味の中心部に進出してきたのかは興味深いです。

またMerckは色々な骨格を最適化していたようです。同時に最適化したのか先行がこけてバックアップに走ったのかわかりませんが、多数の骨格の最適化が動いていたのは間違いないので、ターゲットとしての魅力が高かったということでしょう。実際SUVOREXANTは上市されましたしね。



patinformatics

本章では論文データを利用しましたが、実際の現場でこのような解析をする場合には論文データは使いません。なぜなら企業が論文化するときはそのプロジェクトが終わったこと（成功して臨床に進んだか、失敗して閉じたか）を意味するからです。実際の場面では特許データを利用して解析をします。

このような解析とメディシナルケミストの経験と洞察力をもとに他社状況を推測しながら自分たちの成功を信じてプロジェクトは進んでいきます。

tSNEをつかったマッピング

PCAよりもtSNEのほうが分離能がよく、メディシナルケミストの感覚により近いと言われています。sklearnではPCAをTSNEに変更するだけです。

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=0)
tx = tsne.fit_transform(fps)
```

描画するとわかりますが、PCAに比べてよく分離されています。

```
d = pd.DataFrame(tx)
d.columns = ["PCA1", "PCA2"]
d["DOCID"] = docs
d["COMPANY"] = companies
g = ggplot(aes(x="PCA1", y="PCA2", color="COMPANY"), data=d) + geom_point() + xlab("X") + ylab("Y")
g
```



今回紹介したPCA,tSNEの他にも色々な描画方法があるので調べてみるとよいでしょう。

9章: 構造活性相関（QSAR）の基礎



化学構造と生物学的活性における相関関係をStructure Activity Relationship(SAR)またはQuantitative SAR(QSAR)と呼びます。一般的には似たような化合物は似たような生物学的活性を示すことが知られており、この相関関係を理解しドラッグデザインに活かすことが創薬研究において大変重要です。

また、このような問題には細胞の生死、毒性の有無といった化合物がどのクラスに入るのかを推定する分類問題と阻害率（%inhibition）といった連続値を推定する回帰問題の2つがあります。

効果ありなしの原因を考えてみる（分類問題）

ChEMBLからhERG阻害アッセイの73データを用いてIC50が1uM未満のものをhERG阻害あり、それ以外をhERG阻害なしとラベルします。

まずは必要なライブラリをインポートします。

```
from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem, Draw
from rdkit.Chem.Draw import IPythonConsole
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.ensemble import RandomForestClassifier
```

ChEMBLでダウンロードしたタブ区切りテキストの処理は8章とほぼ同じですが、今回は活性データが欲しいのでSTANDARD_VALUEという列を探して数値を取り出します。この値が1000nM未満であればPOSというラベルを、そうでなければNEGというラベルを振ります。最後にラベルをnumpy arrayにしておきます。

```

mols = []
labels = []
with open("ch09_compounds.txt") as f:
    header = f.readline()
    smiles_index = -1
    for i, title in enumerate(header.split("\t")):
        if title == "CANONICAL_SMILES":
            smiles_index = i
        elif title == "STANDARD_VALUE":
            value_index = i
    for l in f:
        ls = l.split("\t")
        mol = Chem.MolFromSmiles(ls[smiles_index])
        mols.append(mol)
        val = float(ls[value_index])
        if val < 1000:
            labels.append("POS")
        else:
            labels.append("NEG")

labels = np.array(labels)

```

続いてmolオブジェクトをフィンガープリントに変換します。このフィンガープリントからhERG阻害の有無を予測するモデルを作成します。

```

fps = []
for mol in mols:
    fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2)
    arr = np.zeros((1,))
    DataStructs.ConvertToNumpyArray(fp, arr)
    fps.append(arr)
fps = np.array(fps)

```

データセットを訓練セットテストセットの2つに分けます。テストセットは作成した予測モデルの精度を評価するためにあとで使います。

```
x_train, x_test, y_train, y_test = train_test_split(fps, labels)
```

予測モデルを作成するにはインスタンスを作成してfitメソッドで訓練させるだけです

```

rf = RandomForestClassifier()
rf.fit(x_train, y_train)

```

先程分割しておいたテストセットを予測します。

```
y_pred = rf.predict(x_test)
```

Confusion matrixを作成します。

Confusion matrixとは

Confusion matrixとはクラス分類の結果をまとめた表です。クラスを正しく分類できているかをわかりやすく視覚化できTP,TNが多くFP,FNが少ないほどよく分類できているということになります。

		Actual class	
		Positive	Negative
Predicted class	Positive	True Positive(TP)	False Positive(FP)
	Negative	False Negative(FN)	True Negative(TN)

```
confusion_matrix(y_test, y_pred)  
#array([[11,  1],[ 5,  2]])
```

11	1
5	2

F1スコアを見てみましょう。

```
f1_score(y_test, y_pred, pos_label="POS")  
#0.4
```

あまりよくないですね。

NOTE

train_test_split関数がランダムに訓練セットとテストセット分割するので、Confusion matrix,F1スコアの値は実行するたびに変わります。

F1スコアとは

- 正しいと予測されたもののうち本当に正しいものの割合を適合率と呼ぶ $\text{precision} = \text{TP}/(\text{TP} + \text{FP})$
- 正しいものが正しいと予測された割合を再現率と呼ぶ $\text{recall} = \text{TP}/(\text{TP} + \text{FN})$

F1スコアは適合率と再現率の調和平均となり

$$\text{F1} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

で計算されます。

薬の効き目を予測しよう（回帰問題）

回帰モデルは最初に説明したとおり、連續値を予測するモデルとなります。今回はRandomForestの回帰モデルを作成して、その精度をR2で評価します。データは分類問題で使ったhERGのアッセイデータを利用することにしましょう。最初に必要なライブラリをインポートします。

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from math import log10
```

分類問題のときにはラベル化しましたが、今度は連續値を予測したいのでpIC50に変換します。(なぜpIC50にすると都合が良いのかはそのうち補足する)

```
pIC50s = []
with open("ch09_compounds.txt") as f:
    header = f.readline()
    for i, title in enumerate(header.split("\t")):
        if title == "STANDARD_VALUE":
            value_index = i
    for l in f:
        ls = l.split("\t")
        val = float(ls[value_index])
        pIC50 = 9 - log10(val)
        pIC50s.append(pIC50)

pIC50s = np.array(pIC50s)
```

データセットをトレーニングセットとテストセットの2つに分割します。フィンガープリントは分類モデルのときに作成したものを使います。

```
x_train, x_test, y_train, y_test = train_test_split(fps, pIC50s)
```

訓練します。Scikit-learnの場合はこの手順はどの手法でもほぼ同じメソッドでfitしてpredictです。

```
rf = RandomForestRegressor()
rf.fit(x_train, y_train)
```

予測しましょう。

```
y_pred = rf.predict(x_test)
```

予測精度をR2で出してみます。

```
r2_score(y_test, y_pred)  
#0.52
```

まあまあといったところでしょうか。

R2スコアとは

回帰の当てはまりの良さへの1つの評価指標としてよく利用されるもので[決定係数](#)とも呼ばれます。

モデルの適用範囲(applicability domain)

今回紹介した手法は似たような化合物は似たような生物学的活性を示すという仮設に基づいて生成されるモデルです。もしトレーニングセットに似ている化合物が含まれなかった場合の予測精度はどうなるのでしょうか？

当然その場合は予測された値は信頼できませんよね。つまり、予測値にはその予測が確からしいか？という信頼度が常にについてまわります。そのようなモデルが信頼できる、または適用できる範囲をapplicability domainと呼びます。これに関しては明治大学金子先生の[モデルの適用範囲・モデルの適用領域](#)が詳しいです。

(おまけコラム)applicability domainはどこまで信頼できるのか？

筆者は昔、Hugo Kubinyi先生の似ている化合物は果たして似た活性を示すのか？という疑問を、estradiolのOH基をMethoxy基に変換すると活性が消失する例を上げて説明されていたことに感銘を受けたのを覚えています。

applicability domainはトレーニングセットの類似性からその予測が信頼できるかという確度を測る手法です。ここで類似性が誰のための類似性なのかという問題が出てきます。我々がこの化合物とこの化合物は似ているよねと思うのは我々の勝手ですが、似ているか似ていないかは最終的には蛋白質が判断します。そのため必ずしも類似度から活性が予測できるわけではなく、極めて類似度が高いのに活性が消失してしまうことがあります。特にMMPの文脈で説明されたActivity Cliffはこのような事象にそれっぽい名前をつけたものです。

10章: ディープラーニング入門

本章からディープラーニングを利用して、QSARモデルや生成モデルを作成します。

ディープラーニングに関して

生物の脳には神経細胞が存在し、それらがネットワークを形成することで情報を伝達したり、記憶や学習しています。このネットワーク構造を数理モデル化したものがArtificial Neural Network(ANN)です。

一般的なANNは、学習のための情報を入れる入力層、入力情報のパターンを元に反応（神経シナプスの発火に対応）を学習する中間層（または隠れ層）、最後の出力層の三層から構成されていますが、ディープラーニングは隠れ層を多層にすることで高精度な予測を可能とします。

本書では特にこれ以上の説明はしませんが、自分でゼロからコードを書いて理解したい場合は[ゼロから作るDeep Learning](#)が助けになります。また、理論についてきちんと学びたい場合は[深層学習](#)をお薦めします。

TensorFlowとKerasについて

Tensorflowは Googleが開発してOSSとして公開している機械学習のためのフレームワークです。主にディープラーニングのフレームワークとして使われることが多いです。

NOTE: Tensorflowは最近1.xから2.xにメジャーアップデートをしましたが、2.x版はまだ登場したばかりで参考情報が少ないので、1.x系を利用します。また同じ1.xでもバージョンによってAPIが異なるので、動かしたいコードがあった場合、どのバージョンで書かれているかに気をつけてください。

KerasはTensorflowなどの低レベルフレームワークをバックエンドにした高レベルAPIで、より簡単にコードが書けます。KerasはTensorflowとは独立して開発されてきましたが、最近、TensorflowはKerasを同梱するようになりました。そのため別にインストールせずにKerasを利用できます。Tensorflow同梱バージョンのKerasが本家の最新バージョンになっていないことがあります。

どちらのKerasを使うのが良いかは悩ましいところですが、本書では利便性のためにTensorflow統合版Kerasを利用します。

KerasとTensorflowの関係性

KerasとTensorflowを[オフィシャルブログを参照](#)しつつ少し整理しておきます。元々KerasはTensorflowとは別のプロジェクトとして開発されており（もちろん今もです）Kerasを使うにはTensorflowとはインストールする必要がありました。しかし2017年Keras2.xのメジャーバージョンアップのタイミングのあたりでTensorflowプロジェクトがKerasを統合するようになりました。以下の英文は上記リンクの記事の抜粋です。現在はTensorflowからKerasを呼び出し利用することが可能になっています。

TensorFlow integration Although Keras has supported TensorFlow as a runtime backend since December 2015, the Keras API had so far been kept separate from the TensorFlow codebase. This is changing: the Keras API will now become available directly as part of TensorFlow, starting with TensorFlow 1.2. This is a big step towards making TensorFlow accessible to its next million users.

インストールしてみよう

Tensorflow とKerasをインストールしてみましょう。 anacondaでインストールする場合、GPU対応バージョンを使うか、CPUバージョンを使うかでインストールするパッケージが少し異なります。

```
# CPU版  
$ conda install -c conda-forge tensorflow  
# GPU版  
$ conda install -c anaconda tensorflow-gpu
```

NOTE pipコマンドを利用してTensorFlowをインストールすることもできます。その場合は[公式ドキュメント](#)を参照してください。しかし基本的にはCondaで環境を作ったらCondaでパッケージを入れることが望ましいでしょう。

参考リンク

- <https://keras.io/#installation>
- <https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html>

Google colabとは

Google colab

Google colaboratoryはクラウド上で実行できるJupyter notebook環境です。Theano, Tensorflow, Keras, Pytorchなどのディープラーニング用のフレームワークがインストール済みなのと時間の制限はありますがGPUが使えるため、手元にGPUマシンがなくてもディープラーニングを利用できる点が非常に魅力的です。

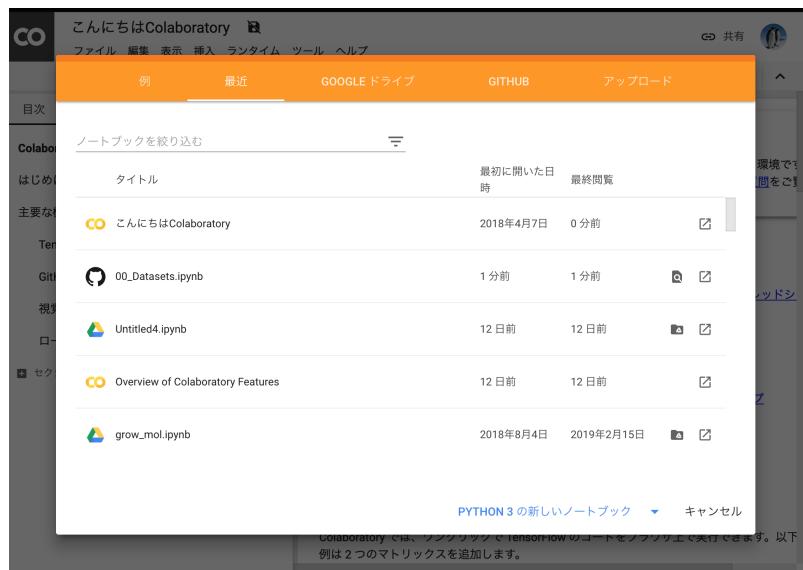
利用にはGoogleのアカウントが必要なので、もしGoogleアカウントを持っていなければこの機会にアカウントを取得し利用してみると良いでしょう。

Google のアカウントがある方はGitHub状のノートブックをそのままColab上で実行することもできます。以前Mishima.sykで使ったScikit-learnハンズオンのノートブックを開いてみます。

NOTE

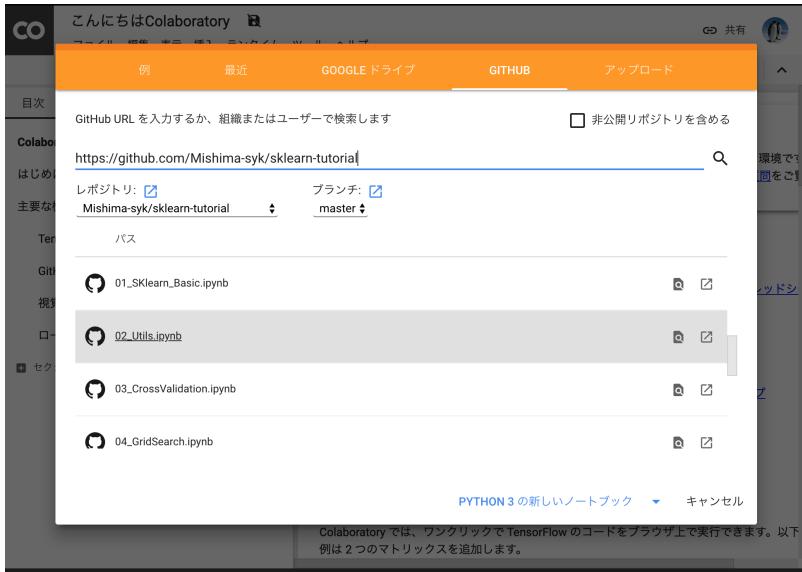
@y_samaが作成したノートブックですが、データの準備から、AutoSklearnまで一通り学習できるようになっています。

まずGoogle colaboratoryにアクセスします。もし下のような画面にならなければ左上の「ファイル」から「ノートブックを開く」を実行してください



次にGitHubというタブをクリックし、以下のURLをコピーペーストするとJupyter Notebookからコードを動かせます。

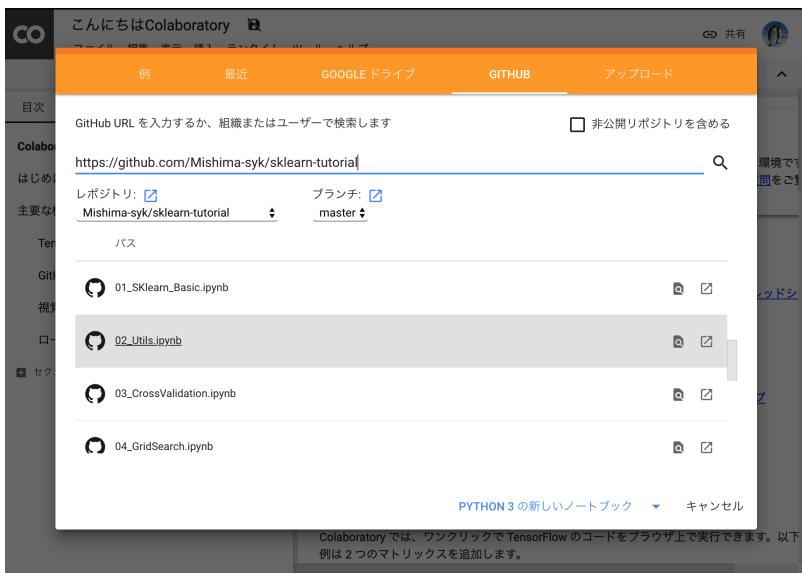
<https://github.com/Mishima-syk/sklearn-tutorial>



ノートブックを開くとJupyter

Notebookと同じような画面になります。Shift+リターンキーでセルの

コードを実行できます。



GoogleColabデフォルトで利用できるライブラリを確認するにはセルの中で '!pip freeze'と打つと列記されます。

- absl-py==0.7.0
- alabaster==0.7.12
- たくさん出てくる
- yellowbrick==0.9.1
- zict==0.1.3
- zmq==0.0.0

Pythonのディープラーニング用フレームワーク

Pythonのディープラーニング用フレームワークはいくつもあります。主に[Theano](#), [Tensorflow](#), [Keras](#), [MXNet](#), [Chainer](#), [PyTorch](#), などが挙げられます。

様々なディープラーニングの文献では実装に上記のフレームワークのいずれかが使われていることが多いです。色々試してみて自分が使いやすいフレームワークを選ぶのもよいでしょう。

11章: ディープラーニングを利用した構造活性相関



本章ではDNNを利用して構造活性相関解析をします。

DNNを利用した予測モデル構築

はじめにDNNを利用したシンプルな予測モデルを構築してみます。ここでは9章と同じデータを使います。最初に分類モデルを作成し、Positiveのラベルを[0, 1], Negativeのラベルを[1, 0]の二次元のOneHotベクトルで表します。KerasのModelオブジェクトを利用してモデルを作成した場合、上記の二次元のそれぞれの期待値が得られます。どちらのクラスに属する可能性が高いかを知るにはNumpyのArgmax関数を使えばよいです。

NOTE

OneHotベクトルとはある一つの値が1でそれ以外が0になるようなベクトルです。10クラスの分類問題を考えた場合[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]のようにどこかが1で残りの9個が0になるようなベクトルでクラスを表現できます。上の例ではPositive/Negativeの2クラスですので、OneHotベクトルは2次元になっています。

必要なライブラリをインポートします。

```
from rdkit import Chem, DataStructs
from rdkit.Chem import AllChem, Draw
from rdkit.Chem.Draw import IPythonConsole
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, f1_score
# DNN用のライブラリを読み込みます。 tensorflowに統合されたKeras
を使うように変えました(20190306)
# from keras.layers import Input
# from keras.layers import Dense
# from keras.layers import Dropout
# from keras.layers import Activation
# from keras.models import Model
from tensorflow.python.keras.layers import Input
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.layers import Dropout
from tensorflow.python.keras.layers import Activation
from tensorflow.python.keras.Model import Model
```

次にデータを読み込みます。9章では”POS”/”NEG”をlabelsというリストに入れたので一次元表現でしたが、今回はここが二次元になっています。

```

mols = []
labels = []
with open("ch09_compounds.txt") as f:
    header = f.readline()
    smiles_index = -1
    for i, title in enumerate(header.split("\t")):
        if title == "CANONICAL_SMILES":
            smiles_index = i
        elif title == "STANDARD_VALUE":
            value_index = i
    for l in f:
        ls = l.split("\t")
        mol = Chem.MolFromSmiles(ls[smiles_index])
        mols.append(mol)
        val = float(ls[value_index])
        if val < 1000:
            labels.append([0,1]) # Positive
        else:
            labels.append([1,0]) # Negative
labels = np.array(labels)

```

続いて分類モデルと回帰モデルを順次作成します。

まずは回帰モデルで、入力は9章と同じECFPを利用しています。DNNの構築には入力データの次元を明示的に指定する必要があるためnBitsという変数を定義しています。

TIP train_test_splitにrandom_stateで適当な整数を指定すると毎回同じデータが得られるので検証の際に有用です。

```

nBits = 2048
fps = []
for mol in mols:
    fp = AllChem.GetMorganFingerprintAsBitVect(mol, 2, nBits=nBits)
    arr = np.zeros((1,))
    DataStructs.ConvertToNumpyArray(fp, arr)
    fps.append(arr)
fps = np.array(fps)

x_train1, x_test1, y_train1, y_test1 = train_test_split(fps, labels, random_state=794)

```

入力が2048次元、300ニューロンの全結合層が三層、最後の出力層が2となるニューラルネットワークを作成します。活性化関数にはReLU、出力層には二次元の多クラス分類のためにSoftmaxを用いました。

Dropout層はランダムにニューロンを欠損させることにより過学習を防ぐ役割を果たします。

Kerasではモデルを定義した後compile関数を呼ぶことでモデルを構築します。optimizer, lossは目的に応じて変更する必要がありますが、今回は'categorical_crossentropy'を使いましたがoptimizerはadam以外にも多くあるのでどれが適切かは実際は試行錯誤が必要となるでしょう。

TIP

ReLUはSigmoid関数の勾配消失の課題を克服できるためよく利用されます。

```
# Define DNN classifier model
epochs = 10
inputlayer1 = Input(shape=(nBits, ))
x1 = Dense(300, activation='relu')(inputlayer1)
x1 = Dropout(0.2)(x1)
x1 = Dense(300, activation='relu')(x1)
x1 = Dropout(0.2)(x1)
x1 = Dense(300, activation='relu')(x1)
output1 = Dense(2, activation='softmax')(x1)
model1 = Model(inputs=[inputlayer1], outputs=[output1])

model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

NOTE

KerasにはSequentialモデルが用意されており、これを使うことで上記の例（Functional API）よりもシンプルにネットワークを記述できます。今回Functional APIでモデルを定義したのは、こちらに慣れておくと入力が複数の場合やより複雑なモデルの構築にも対応しやすいからです。もしSequentialの書き方に興味がある方は公式サイトやQiitaを調べてください。

NOTE

DNNは初期のランダムに発生させた重みに基づいて予測した予測値と実際の値を比較し、その差（LOSS）を最小化するように重みを更新するBackpropagationという手順を繰り返しながらモデルを最適化します。この繰り返しの回数を指定するのがEpochsです。Epochsを増やすとどんどん賢くなるように思われるかもしれません、計算コストがかかることと過学習のリスクもあるので長ければ良いわけではないのでLoss/Accuracyなどを観測して適切なEpoch数を考えましょう。

Epochsを増やすとなぜ過学習のリスクがあるのか？

トレーニングデータを用いてEpoch毎に正解の値と予測値の誤差を小さくするように重みを調整していきます。十分な量のトレーニングデータを用い学習したとしても繰り返しすぎると、同じトレーニングデータで何度も何度も学習することになるのでモデルの汎化性能が落ちてしまいます。

過学習の判断にはEpoch毎、Training set/ Validation setの精度を評価しプロットするとTraining setの精度が向上する一方でValidation setの精度が変化しないまたは悪くなっているかどうかを調べればよいです。KerasにはEarly stoppingという機能があり、ある一定回数以上学習を進めてもモデルの性能が変化しなかったらそこで学習を打ち切るといったこともできます。

詳しくはEarly stoppingの紹介と参考文献をご覧ください。

モデルを構築したら後はScikit-learnと同じ感覚でfit/predictが行えます。

```
hist1 = model1.fit(x_train1, y_train1, epochs=epochs)
```

最後に結果を可視化してみます。

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(range(epochs), hist1.history['acc'], label='acc')
plt.legend()
plt.plot(range(epochs), hist1.history['loss'], label='loss')
plt.legend()
```

今回の例ではだいたい6Epochくらいでモデルが良い精度になりました。

次にテストデータで検証します。

```
y_pred1 = model1.predict(x_test1)
y_pred_cls1 = np.argmax(y_pred1, axis=1)
y_test_cls1 = np.argmax(y_test1, axis=1)
confusion_matrix(y_test_cls1, y_pred_cls1)
```

ちょっと微妙でしょうか、、、

回帰モデルも基本的には先ほどの分類問題と同じです。今度は回帰なので最後の出力層は値そのもの、つまり一次元になります。また活性化関数はSigmoidなどでは0-1になってしまってLinearとしています。学習データは9章のコードを流用しています。

```
from math import log10
from sklearn.metrics import r2_score
pIC50s = []
with open("ch09_compounds.txt") as f:
    header = f.readline()
    for i, title in enumerate(header.split("\t")):
        if title == "STANDARD_VALUE":
            value_index = i
    for l in f:
        ls = l.split("\t")
        val = float(ls[value_index])
        pIC50 = 9 - log10(val)
        pIC50s.append(pIC50)

pIC50s = np.array(pIC50s)
x_train2, x_test2, y_train2, y_test2 = train_test_split(fps, pIC50s, random_state=794)
```

次にモデルを定義します。Lossの部分が先ほどの分類モデルとは異なり、MSEになっていることに注意して下さい。

```

epochs = 50
inputlayer2 = Input(shape=(nBits, ))
x2 = Dense(300, activation='relu')(inputlayer2)
x2 = Dropout(0.2)(x2)
x2 = Dense(300, activation='relu')(x2)
x2 = Dropout(0.2)(x2)
x2 = Dense(300, activation='relu')(x2)
output2 = Dense(1, activation='linear')(x2)
model2 = Model(inputs=[inputlayer2], outputs=[output2])
model2.compile(optimizer='adam', loss='mean_squared_error')

```

ここまでできたら後は同じです。

```

hist = model2.fit(x_train2, y_train2, epochs=epochs)
y_pred2 = model2.predict(x_test2)
r2_score(y_test2, y_pred2)
plt.scatter(y_test2, y_pred2)
plt.xlabel('exp')
plt.ylabel('pred')
plt.plot(np.arange(np.min(y_test2)-0.5, np.max(y_test2)+0.5), np.arange(np.min(y_test2)-0.5, np.max(y_test2)+0.5))

```

いかがでしょうか。予測モデルはちょっとUnderEstimate気味ですかね。DNNは重ねるレイヤーの数、ドロップアウトの割合、隠れ層のニューロンの数、活性化関数の種類など数多くのパラメータをチューニングする必要があります。今回の例は決め打ちでしたが、色々パラメータを変えてモデルの性能を比較してみるのも面白いです。

記述子を工夫してみる(neural fingerprint)

さて、ここまで分子のフィンガープリントを入力としてRandomForestやDNNのモデルを作成してきました。DNNが大きく注目を浴びた理由の一つに人が特徴量を抽出しなくてもモデルが特徴量を認識してくれるということが挙げられます。

例えば画像の分類においては、からSIFTという特徴量を人が定義し、これを入力としたモデルが作られていましたが、現在のDNNにおいては基本的に画像のピクセル情報そのものを利用しています。

ケモインフォマティクスに置き換えてみると、SIFTは分子のフィンガープリントに相当します。ですのでここ(入力)をもっとPrimitiveな表現に変えることでDNNの性能が上がるのではないか?と考えるのは至極当然の流れです。2015年、Harvard大学の、Alan Aspuru-Guzikらのグループは一つのチャレンジとしてNeural Finger print/NFPというものを提唱しました。

今まで利用してきたECFPとNFPとの違いを、彼らの論文中の図を引用して示します。

Algorithm 1 Circular fingerprints	Algorithm 2 Neural graph fingerprints
<pre> 1: Input: molecule, radius R, fingerprint length S 2: Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$ 3: for each atom a in molecule 4: $\mathbf{r}_a \leftarrow g(a)$ \triangleright lookup atom features 5: for $L = 1$ to R \triangleright for each layer 6: for each atom a in molecule 7: $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$ 8: $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$ \triangleright concatenate 9: $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$ \triangleright hash function 10: $i \leftarrow \text{mod}(r_a, S)$ \triangleright convert to index 11: $\mathbf{f}_i \leftarrow 1$ \triangleright Write 1 at index 12: Return: binary vector \mathbf{f} </pre>	<pre> 1: Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$ 2: Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$ 3: for each atom a in molecule 4: $\mathbf{r}_a \leftarrow g(a)$ \triangleright lookup atom features 5: for $L = 1$ to R \triangleright for each layer 6: for each atom a in molecule 7: $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$ 8: $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^N \mathbf{r}_i$ \triangleright sum 9: $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ \triangleright smooth function 10: $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ \triangleright sparsify 11: $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ \triangleright add to fingerprint 12: Return: real-valued vector \mathbf{f} </pre>

Figure 2: Pseudocode of circular fingerprints (*left*) and neural graph fingerprints (*right*). Differences are highlighted in blue. Every non-differentiable operation is replaced with a differentiable analog.

ECFP(Circular Fingerprints)は入力の分子それぞれの原子からN近傍（Nは任意）までの原子までの情報をHash関数（この例ではMod）任意の値に変換、で固定長のベクトルに直すといったものでした。ざっくりいうと部分構造の有無を0/1のビット情報に直したものを利用するといったイメージです。一方、今回紹介するNFPはECFPにコンセプトは似ているのですが、Hash関数の部分がSigmoidに、Modで離散化する部分がSoftmaxになっています。従って入力されるデータセットによりECFPよりも柔軟に分子のフィンガープリントを生成することが期待されます。

この論文が発表されて以降、数多くの実装がGitHubに公開されていますが、各実装ごとにKerasでもBackendがTheanoであったり、Keras/Tensorflowであっても、Keras1.xじゃないと動作しなかったりと意外と環境依存のものが多く扱いにくい状況になっています。残念なことに今回構築した環境で動作するものが公開されていませんのでKeras2.x/Python3.6で動作するものをこちらのコードをベースに作成しました。

画像分類においてピクセルのまま古典的な手法を使うというアプローチは有効だったのか？

SIFTが提案されたのは1999年です。[原著論文](#)によると、物体（画像）認識においてピクセルそのものを扱う場合の難しさは、オブジェクトの位置、回転、大きさ（スケール）、光度などが異なるものを扱うところにあるようです。これら、変動する値を普遍的な特徴量に変える方法が種々研究されていたようです。ピクセルそのものを使う方法が全くないわけではなく、私が機械学習を勉強する際に購入した[pythonではじめる機械学習](#)には人の顔の画像データを学習して分類する例が載っています。ここではピクセルデータを入力に、主成分分析により顔の特徴を抽出し分類をしています。この問い合わせに関して明確に有効だったという文献を探せていませんが、タスクによっては有効であったと思います。もし詳しい方いたらぜひコメントください。

```
git clone https://github.com/iwatobipen/keras-neural-graph-fingerprint.git
```

example.pyというファイルのコードを眺めるとなんとなく雰囲気がつかめると思います。分子の表現は、これまでの例はフィンガープリントをRDKitを使い生成していましたが、今回はこのフィンガープリント

そのものをDNNが学習します。

ということで、分子をグラフとして表現したものが入力になります。Atom_matrixとして(max_atoms, num_atom_features)をEdge_matrixとして(max_atoms, max_degree)をbond_tensorとして(max_atoms, max_degree, num_bond_features)という三つの行列を使います。分子はそれぞれ原子数が異なるためmax_atomsで最大原子数を定義しています。こうすることで分子ごとに同一の行列サイズの入力となりバッチ学習が可能となります。

Exampleを実行するのであれば下記のコマンドを入力してください。

```
python example.py
```

参考リンク

- [NGF-paper](#)
- [DeepChem-paper](#)
- [keiserlab](#)
- [HIPS NFP](#)
- [Theano base](#)
- [for keras1.x](#)
- [ericmjl/graph_fp](#)
- [DeepChem](#)
- [Early stoppingについて](#)
- [SIFT原著論文](#)

12章: コンピューターに化学構造を考えさせる



Deep Learningがメディナルケミストリに大きなインパクトをもたらしたものの一つに生成モデルがあげられます。特にこの数年での生成モデルの進化は素晴らしいです。ここでは[Marcus Olivecrona](#)により開発された[REINVENT](#)を使って新規な合成案を提案させてみましょう。

生成モデルとは？

11章で構築した予測モデルを一般に識別モデルといいます。一方で入力の分布をモデル化することでモデルからのサンプリングつまり入力データを生成することができるようになります。これを生成モデルといいます。

詳しくは[PRMLの1.5.4](#)を読むのをおすすめします

準備

pytorchというディープラーニングのライブラリをcondaでインストールします。新しいバージョンでは動かないのでバージョンを指定してインストールします。

*pytorch*とは？

keras同様TensorFlowをより便利に使うためのライブラリです。

```
$ conda install pytorch=0.3.1 -c pytorch
```

続いてREINVENT本体をGitHubからクローンします。

```
$ cd <path to your working directory>
$ git clone https://github.com/MarcusOlivecrona/REINVENT.git
```

続いて、ChEMBLの110万件くらいのデータセットで予め訓練済みのモデルをダウンロードしてきて元のデータと置き換えます。このデータはGTX 1080TiGPUマシンを利用して5,6時間かかっていますのでもしトレーニングを自分で行うのであればGPUマシンは必須です。

```
$ wget https://github.com/Mishima-syk/13/raw/master/generator_handson/data.zip
$ unzip data.zip
$ mv data ./REINVENT/
```

これで準備が整いました。

実例

ここではJanuviaとして知られる抗糖尿病薬sitagliptinの類似体を生成するようなモデルを作成してみます。

まずはtanimoto係数をスコアとして類似度の高い構造を生成するようにモデルを訓練します。今回は3000ステップ訓練しますが、大体ちょっと前のMacbook Airで7,8時間かかるので気長に待ちましょう。待てない場合は[ここ](#)のデータを使ってください。

```
./main.py --scoring-function tanimoto --scoring-function-kwarg query_structure  
'N[C@H](CC(=O)N1CCn2c(C1)nnc2C(F)(F)F)Cc3cc(F)c(F)cc3F' --num-steps 3000 --sigma 80
```

ここからはjupyter notebookを立ち上げます。

必要なライブラリを読みこみます。sys.path.appendはREINVENTのディレクトリを指定してください。

```
%matplotlib inline  
import sys  
sys.path.append("[Your REINVENT DIR]")  
from rdkit import Chem  
from rdkit.Chem import AllChem, DataStructs, Draw  
import torch  
from model import RNN  
from data_structs import Vocabulary  
from utils import seq_to_smiles
```

続いて、トレーニングしたモデルから50化合物サンプリングします。

```
voc = Vocabulary(init_from_file="/Users/kzfm/mishima_syk/REINVENT/data/Voc")  
Agent = RNN(voc)  
Agent.rnn.load_state_dict(torch.load("sitagliptin_agent_3000/Agent.ckpt"))  
seqs, agent_likelihood, entropy = Agent.sample(50)  
smiles = seq_to_smiles(seqs, voc)
```

実際にどんな構造が生成されたのか見てみましょう。

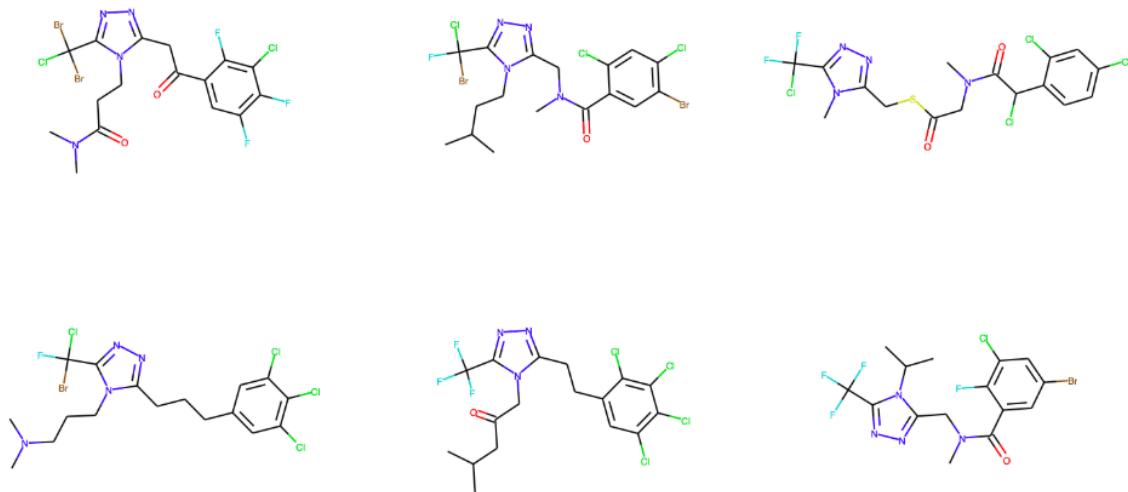
```
mols = []  
for smi in smiles:  
    mol = Chem.MolFromSmiles(smi)  
    if mol is not None:  
        mols.append(mol)  
  
Draw.MolsToGridImage(mols, molsPerRow=3, subImgSize=(500,400))
```

まずまずといったところでしょうか？

```
In [3]: mols = []
for smi in smiles:
    mol = Chem.MolFromSmiles(smi)
    if mol is not None:
        mols.append(mol)

Draw.MolsToGridImage(mols, molsPerRow=3, subImgSize=(500,400))
```

Out[3]:



REINVENTについて

是非とも[Molecular De Novo Design through Deep Reinforcement Learning](#)を読んでください。

13章: おわりに

さらに学ぶために

NOTE

どのあたりに興味があって、さらに知りたいというようなリクエストをissueに投げるかtwitterででもreplyしてもらえると。またおすすめの提案も助かります。

機械学習をさらに学びたい方

[Pattern Recognition and Machine Learning\(PRML\)](#)を一通り読めるようになることを目指すといいでしょ。英語のpdfは先のリンクからフリーでダウンロードできますが、日本語は[パターン認識と機械学習上, 下](#)で販売されています。

PRMLが手強いという方は「PRMLを読む前に」などで検索するともう少し易しい本が出てくるので自分に合ったものを選ぶとよいと思います。