机器学习理论：代码报告

17 级化学系　物理化学　祝震予　17110220038

2018 年 2 月 16 日

这是一个关于 Gilmer *et al.* (ICML 2017)[1] 与 Faber *et al.* (JCTC 2017)[2] 的代码报告。

到目前为止，Faber *et al.* (JCTC 2017)[2] 的数据相信是可重复的，只是现在尚未测试所有的特征向量构造方法、以及代码的运行效率不是很高。而关于 Gilmer *et al.* (ICML 2017)[1]，现在仍然没有找到重复的方法。关于这两份工作，后文将会详细叙述情况。

1　Faber 文章重复

Faber *et al.* (JCTC 2017)[2] 文章并没有给出源代码，但给出计算所需的特征向量。因此，我尝试自己写代码验证结果。我只成功重复了 Faber *et al.* (JCTC 2017)[2] 文章中的 Table 3 中的 CM (Coulomb Matrix) 特征方法下的 BR(Bayesian Ridge Regression)、EN (Elastic Net)、RF (Random Forest) 方法。尽管重复出的数据很少，但相信重复其它数据都是只要愿花时间就能解决的问题（除去深度学习的 GC、GG 方法，以及各种特征向量的获取也并非简单）。除此之外，BoB (Bag of Bonds) 特征向量也成功得到。详细的重复结果列举如下。

1.1　QM9 数据集信息提取、CM 与 BoB 特征向量获得

1.1.1　文件列表

这里以 QM9 数据集中第 106080 号原子的特征向量提取作为示例；其它原子类同。关于这部分工作，放置在目录 `Faber_Code/01-Feature_Vector` 下。关于其中的文件，列举如下：

- `ReadXYZ.ipynb`：主要代码示例。可执行的 Jupyter Notebook 文档。
- `ReadXYZ.html`：由 `ReadXYZ.ipynb` 保存后的网页文档。
- `dsgdb9nsd_106080.xyz`：QM9 数据集中，第 106080 号分子的源文件。这是 `ReadXYZ.ipynb` 执行时所需要的文件之一。
- `mol_106080.gjf`：手工修改的 QM9 集的 106080 号分子的结构文件。这份文件应当可以被现在最为流行的量子化学计算软件之一 Gaussian 所读取并进行 HF/STO-3G 计算；同时也应可以被 GaussView 软件所读取。
- `qm9_106080.png`：QM9 集的 106080 号分子结构。由 `mol_106080.gjf` 放入 GaussView 产生。
  - 其中红色代表氧原子、白色氢原子、灰色碳原子。
  - 尽管图中的双键确是 `dsgdb9nsd_106080.xyz` 中 SMILES 码给出的双键，但由于 GaussView 对何时分子成双键的定义未必与其它软件相同，所以由 GaussView 给出的键级信息只能作为直观参考，不能作为代入实际计算的特征。
- `qm9_106080`：从 Faber *et al.* (JCTC 2017)[2] 文章 SI (Supporting Information) 附件中截取出的 106080 号原子的不同特征表示下的特征向量。我们希望能从 `dsgdb9nsd_106080.xyz` 文件的信息给出特征向量，并与这份文件夹下的文件作比较，确定我们给出的特征向量正确。
- `BOB_Example`：部分分子的，从 Faber *et al.* (JCTC 2017)[2] 文章 SI 中提取处的 BoB 特征向量。关于为何要选取这些文件，已经在 `ReadXYZ.ipynb` 有所说明。

1.1.2　补充的说明

关于如何从 QM9 数据集读取 CM、BoB 特征向量的方法，已经在 `ReadXYZ.ipynb` 中有比较详实的说明了。这里只是一些补充的说明或 `ReadXYZ.ipynb` 说得不详细的补充。

之所以会选择 CM、BoB 特征向量作为被重复的示例特征向量，是因为这些特征向量比较容易得到。CM 所需要的信息不过原子间的相对位置、以及它们的核电荷数；BoB 相对于 CM 额外多出了对于构成分子键的原子对

的分类整理。其它的特征向量多少会涉及到 UFF (Universal Force Field) 势函数、L-J (Lennard-Jones) 势函数、Morse 势函数，以及成键的判断、键级的判断等。这些判断可能是不简单，或者是需要依靠其它化学信息学软件支持的，因为单纯的分子原子位置信息也许不足以下判断。

关于 CM 特征方法，需要补充的一句是，尽管文章中提及，如果将特征向量作为 30×30 的矩阵，那么其向量的行列排序将是其 L_1 范数的大小逆序；但实际上更可能是 L_2 范数的大小逆序：因为对于 106080 号分子，L_1 范数大小逆序无法给出能与 `qm9_106080/CM_106080.txt` 相同的特征向量。

另外需要说明的是，CM 特征向量的定义在 Rupp *et al.* (PRL 2012)[3] 与在 Faber *et al.* (JCTC 2017)[2] 的定义不同。Faber *et al.* (JCTC 2017)[2] 的向量中包含了矩阵的所有信息，而 Rupp *et al.* (PRL 2012)[3] 中则只包含了矩阵的本征值向量信息。

关于 BoB 特征方法，需要作补充的是：其一，BoB 特征向量确实应该比 CM 大（是指向量长度，若论非零值应当是 CM 更多一些），但我想也不至于到 2209 个数（相比之下，CM 为 900）。我认为 BoB 方法下，有效的（对于任一分子而言必不为零的）数组大小应不大于 1150。尽管没有做过测试，但我相信如果代入 Faber *et al.* (JCTC 2017)[2] 文章 SI 的 BoB 特征向量，恐怕一半内存是被浪费的。

第二，尽管原则上 BoB 特征向量中所有值应都能与 CM 特征向量的某些元素相等；但实际上在 Faber *et al.* (JCTC 2017)[2] 的处理中，关于不同原子之间的库伦排斥势 $M_{AB} = Z_A Z_B / |\boldsymbol{R}_A - \boldsymbol{R}_B|$ 的计算上，两者对于距离采用的单位不同，BoB 采用 a.u.（Bohr 半径 0.529 Å 为单位），而 CM 采用 Å（Angstrom，10^{-10} m）为单位。同时，相同原子间的向量元 $M_{AA} = 0.5 Z_A^{0.24}$ 则使用了相同的数值。换言之，对于 CM 与 BoB 方法，它们的 M_{AA} 与 M_{AB} 量纲其实是不一样的。这对于 BR、EN 等线性模型而言可能不重要，但对于 KRR 等需要计算特征向量间相似度（譬如 RBF 核下，$d(\boldsymbol{M} - \boldsymbol{M'}) = \|M - M'\|_2$）的方法，不同的单位换算可能会导致不同的相似标度。因此，我认为这意味着 CM 与 BoB 不算是性质上良好的特征向量。

1.2 Table 3 部分数据重复

这部分工作的代码放置在 `Faber/02-Repeat_Table3` 中。这里重复了所有 13 个性质的 CM 特征向量下 BR、EN、RF 数据。KRR (Kernel Ridge Regression) 数据由于本身计算量偏大、以及超参数筛选过程较为漫长，所以现在没有给出结果。训练过程使用 Python 包 sklearn。程序环境为 Red Hat 6，Python 3.6, Anaconda 1 月 20 日左右的最新版本（以 `conda update all` 更新）。计算使用 12 核、24 线程的 Intel Xeon E5 系列处理器。

1.2.1 文件列表

- 主程序文件
  - `test_BR.py`：测试 BR、EN 的主程序。尽管 RF 部分的代码也在其中，但程序将因为 `FOut` 文件关闭而在 101 行报错。该程序的输出文件为 `CM-BR-EN-RF.txt`。其文件名中的“`RF`”不代表数据中由 RF 的结果。
  - `test_RF.py`：测试 RF 的主程序。该程序的输出文件为 `CM-RF.txt`。
  - `test_KRR.py`：测试 KRR 的主程序。尽管程序能运行，但由于耗时太长，现在没有结果。
- 辅助程序文件
  - `util.py`：一些通用的小函数。
  - `feature.py`：从 Faber *et al.* (JCTC 2017)[2] 提供的 SI 中读取 CM 特征向量。这里我没有用到上一小节中提到的直接从 QM9 分子集中获得 CM 特征向量的方法。
  - `prop.py`：读入 Faber *et al.* (JCTC 2017)[2] 的 SI 中提供的作过处理的数据作为训练过程中的目标值。
  - `fold.py`：读入 Faber *et al.* (JCTC 2017)[2] 提供的 10-fold 中每一个 fold 的训练、验证集，并返回由对象 `sklearn.model_selection.PredefinedSplit` 组成的列表。
  - `fit.py`：确定训练方法的及其参数。同时也返回训练与测试误差、耗时。
- 其它输出与结果文件
  - `CM-BR-EN-RF.txt`：BR、EN 训练误差与耗时。
  - `CM-RF.txt`：RF 训练误差与耗时。
  - `Table_3_Repeat.xlsx`：Faber *et al.* (JCTC 2017)[2]Table 3 数据重复的总结。

1.2.2 数据重复情况

表 1 列举了文中的数值、我所计算得到的数值，以及两者之间的比例。可以看出绝大部分数据的误差不超过 3%。这应当表明我所执行的程序是正确的。

性质		BR			EN			RF		
物理量	单位	原文	计算	比例	原文	计算	比例	原文	计算	比例
μ	Debye	0.844	0.8441	0.9998	0.844	0.8443	0.9997	0.608	0.6007	1.0121
α	Bohr^3	1.33	1.3332	0.9976	1.33	1.3337	0.9972	1.04	1.0139	1.0258
$\varepsilon_{\text{HOMO}}$	eV	0.338	0.3375	1.0014	0.338	0.3376	1.0011	0.208	0.2039	1.0199
$\varepsilon_{\text{LUMO}}$	eV	0.632	0.6315	1.0009	0.631	0.6311	0.9999	0.302	0.2959	1.0206
$\Delta\varepsilon$	eV	0.723	0.7228	1.0003	0.722	0.7221	0.9998	0.373	0.3652	1.0214
$\langle R^2 \rangle$	Bohr^2		55.512			55.515			44.293	
ZPVE	eV	0.0265	0.02647	1.0011	0.0265	0.02647	1.0011	0.0199	0.01948	1.0218
U_0	eV	0.911	0.9107	1.0003	0.911	0.9106	1.0004	0.431	0.4201	1.0259
U	eV		0.9119			0.9118			0.4228	
H	eV		0.9120			0.9119			0.4226	
G	eV		0.9093			0.9093			0.4140	
C_v	$\text{kCal} \cdot \text{Mol}^{-1}$	0.907	0.9062	1.0008	0.906	0.9060	1.0000	0.777	0.7603	1.0219
ω_1	cm^{-1}	131	131.44	0.9967	131	131.37	0.9972	13.2	12.71	1.0385

表 1: Faber *et al.* (JCTC 2017)[2] Table 3 部分数据重复情况

1.2.3 补充的说明

由于每个程序都不大，程序的文档都有所注释，所以这里也不作太多叙述。

这个程序中，使用了许多绝对路径指定了文件的位置，包括至少下述的几行代码：

- `fold.py` Line 30：10-fold 的每个 fold 训练、验证、测试集分子序号；
- `test_*.py` Line 42：CM 特征向量的集合；
- `test_*.py` Line 48：目标值（处理后的分子性质）集合。

如果要实现这段代码，需要手动将这些文件路径改变。同时，这些文件也是不包含在附件中发送过来的代码中。这些文件可以从 Faber *et al.* (JCTC 2017)[2] 的 SI 提供的 Google 网盘中得到，需要翻墙下载、解压以及整理。

对于每个性质，其目标值都经过平均值归零与标准差归一的处理。需要注意的是，这种处理是对所有有效的训练、验证、测试集中的 131029 个分子进行的，并没有排除测试集。这意味着实际的训练、验证集的目标值分布可能仍然不是 $\mathcal{N}(0,1)$，尽管多半也比较接近了。

在上述表格中的误差均为 MAD (Mean Absolute Deviation)；而在 `CM-*.txt` 文件中，所有表示时间的单位为秒。EN 方法之所以较慢，可能还与其进行了超参数调整有关。

KRR 方法的耗时与误差没能打出。但在监视程序运行的过程中，发现对于 BR、EN、RF 过程中，大多数时候 CPU 占用为 2300%，但 KRR（`sklearn.kernel_ridge.KernelRidge`）大多数时候为 100%，较少时间实际有并行。在我写的代码中，没有进行过任何并行优化；所有并行优化由 `sklearn` 包提供。

2 Gilmer 文章重复

这篇文章的作者也是 Faber *et al.* (JCTC 2017)[2] 的共同作者。我认为 Gilmer *et al.* (ICML 2017)[1]（Google）迄今为止没有提供过任何代码、以及验证与测试集的信息。

在 1 月 23 日左右，我在网上找到 Microsoft 对于这份工作的代码 gated-graph-neural-network-samples（GG-NN 方法）。这部分代码看起来在他们的 Repository 现在仍然在更新中。

关于这段代码，我现在尚未能理解。这主要是因为我本身对 Python 的熟悉程度远远不够，以及对 Tensorflow 也不很了解。由于我平时使用的编程语言是 `FORTRAN77`，所以在写上一篇文章的工作重复时，使用的也是带有 `FORTRAN77` 风格的面向过程的编程习惯。

该程序可以在我自己的电脑上运行，环境为 Ubuntu 16.04 LTS，Anaconda 最新（Python 3.5, Tensorflow 1.5，以及 `docopt`、`RDKit` 包）。使用 CUDA 8.0 进行 GPU 计算，GPU 为 GeForce GTX 1050 Ti。

2.1 文件目录

附件并不包含所有的 Microsoft 提供的代码，部分代码也有改变。其中的文件目录为：

- `get_data.py`：数据集预处理。
  - 直接读入 QM9 数据集。Microsoft 提供代码中，是从提供数据集的网站上直接下载。由于该网站在国内的下载速度偏慢，于是我稍改了这段代码，使得程序可以直接读入本地文档。若要使用这段代码，需要在该目录下新建 `data` 文件夹，并将下载好的 `dsgdb9nsd.xyz.tar.bz2` 文件放置于其中。
  - 其输出将是文件 `molecule_train.json` 与 `molecule_valid.json`。这两个文件将是后续训练过程的输入文件。
- `get_data.ipynb`：我对 `get_data.py` 的注解。这是一份可执行的 Jupyter Notebook 文档，执行后的效果应当与 `get_data.py` 相同。它同时被另存为网页版的 `get_data.html`。
- `valid_idx.json`：验证集的分子序号指标集。
- `chem_tensorflow_dense.py`：可执行的训练程序。其训练文档输出为 `2018-01-25-12-21-43_9839_*`；其屏幕输出流到 `chem_tensorflow_dense.out`，屏幕错误流输出到 `chem_tensorflow_dense.err` 中。其中第 48、49 行是我补上的，可能是原来程序的一个 bug。
  - `*=log.json`：里面存有每次训练过程的结果与耗时。
  - `*=model_best.pickle`：存储了学习结果，该结果应可以被再利用。这是二进制文件。
  - `*=params.json`：训练过程所选用的与优化出的超参数。
  - `chem_tensorflow_dense.err`：包含了比 `log.json` 更多的训练结果信息。
- `chem_tensorflow.py`：底层的程序。`chem_tensorflow_dense.py` 的执行需要调用这个程序。这个程序的主要目标是完成数据读取、预处理、屏幕输出、文件输出等，并且搭出虚函数，即构成一个完整的程序框架，包括了训练过程与构建网络。除此之外的个性化的方法都是由 `chem_tensorflow_dense.py` 完成的。
- `utils.py`：底层的小函数。
- `chem_tensorflow_dense.ipynb`：我对 `chem_tensorflow_dense.py` 所作的笔记。这份 Jupyter Notebook 是不可执行的。它也被另存为网页版的 `chem_tensorflow_dense.html`。但这份笔记非常不完全，没有关于算法的描述，大多数是对于其中 Python 语法的注解。
- `README.md`：这是 Microsoft 在 Github 上的文件。

2.2 结果说明

首先，从效率上讲，它确实速度不慢。整个学习过程从输出文件的生成时间上看，是约 40 分钟。这比 CM 方法下的 RF（120 分支树）快上一些（大约 1-2 小时）；并且需要注意到 GG-NN 方法使用的是微机的游戏版 GPU，而 CM/RF 方法用的是 24 线程的服务器级 CPU。

训练的对象是 Gilmer *et al.* (ICML 2017)[1] 文章中提及的 13 个性质中的其中一个（偶极 μ）。特征向量与邻接矩阵看起来不包含键长信息，但包含键级的信息，预期其结果将不会很好。特征向量与邻接矩阵的构成是直接通过 `RDKit` 包读取 SMILES 产生的，所以实际上对于 Microsoft 的程序，他们真正需要的输入并不是完整的 QM9 数据集，而只要分子的 SMILES 码以及分子的性质结果即可。

训练、验证、测试集上，看起来是 Microsoft 并没有使用测试集，而把验证集的结果作为测试集的结果输出；验证集大小为 13082 分子，是 Faber *et al.* (JCTC 2017)[2] 提及的总有效分子的 1/10，且分子固定；但训练集是 QM9 分子集 133885 个分子去除验证集的余下的分子。如果我没有理解错，那么 Microsoft 并没有排除 Faber *et al.* (JCTC 2017)[2] 推荐排除的 3056（3054 若包含两个线性振动分子）个无效分子、以及没有通过 SMILES 测试的 367 个分子。所以这个训练、验证集定义上仍然可能需要改进。

训练结果上，其最佳的结果为 4.85047 倍“Chemical Accuracy”（可从 `chem_tensorflow_dense.err` 的 Line 48633 找到）；而对 μ 物理量而言，其“Chemical Accuracy”根据 Faber *et al.* (JCTC 2017)[2] Table 4 中的定义为 0.1 Debye。这很显然不是一个很好的结果，至少不像 Gilmer *et al.* (ICML 2017)[1] Table 2 中 GG-NN 方法的结果 1.22 倍“Chemical Accuracy”那么好。因此，这个程序本身就不能与 Gilmer *et al.* (ICML 2017)[1] 文章的数据核准。原因可能各种各样，包括邻接矩阵没有键长信息，训练、验证、测试集不同（Gilmer *et al.* (ICML 2017)[1] 则是随机 10000 分子的验证集与 10000 分子的测试集），超参数设置不同，等等。

在 Microsoft 的 Github 上，除了上述的 `chem_tensorflow_dense.py` 外，还有其它三个程序也能进行计算；但我曾经执行过两个，`chem_tensorflow_sparse.py` 无法分配空间，`chem_tensorflow_async.py` 用尽内存后报

错（8 GB 内存，4 GB 显存与 4 GB Swap 空间）。

参考文献

1. Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; Dahl, G. E. In *Proceedings of the 34th International Conference on Machine Learning*; PMLR: International Convention Centre, Sydney, Australia; Proceedings of Machine Learning Research, Vol. 70; pp 1263–1272. http://proceedings.mlr.press/v70/gilmer17a.html.
2. Faber, F. A.; Hutchison, L.; Huang, B.; Gilmer, J.; Schoenholz, S. S.; Dahl, G. E.; Vinyals, O.; Kearnes, S.; Riley, P. F.; von Lilienfeld, O. A. *J. Chem. Theory Comput.* **2017**, *13*, 5255–5264.
3. Rupp, M.; Tkatchenko, A.; Müller, K.-R.; von Lilienfeld, O. A. *Phys. Rev. Lett.* **2012**, *108*, 058301.