工作6年，谈谈我对“算法岗”的理解

原创 Severus 夕小瑶的卖萌屋 2021-10-21 12:05

文 | Severus
编 | 小轶

> 写在前面：本文完全基于我个人的工作经验，没有经过任何形式的行业调研，所以我的理解也有相当浓厚的个人印记，可以认作一家之言。如果能对读者朋友们起到任何帮助，都是我的荣幸。如果不赞同我的看法，则还请一笑了之。

大家好，我是Severus，一个在某厂做中文文本理解的程序员。

今天我想要分享的是，在算法岗工作6年之后，**我心中对“算法岗”的理解，以及我在这个职业中，是如何生存的**。

我时常提到，“AI算法工程师”分为两个部分，“AI算法”和“工程师”。二者的关系是：**算法工程师，首先得是个工程师**。有了 idea，还得有能力高效、优美、完整地实现出来，还要去解决一系列工程上的问题。

同时，**除了有产生想法的能力，还要有及时放弃想法的能力**。因为我做出来的东西，**要面对的从来不是那几个数据集，几个指标，而是实打实的用户**。我向来认为，指标可以控制，gold 集合只能定性，最终的作用才是最重要的。所以，在工业中做算法，又必须要**严谨**，那么除了扎实的背景知识，**还需要分析能力，以及下这些笨功夫的耐心**。

扎实的分析能力

我认为合格的算法工程师，应当具备相当的**分析能力**。**遇到问题的时候，能一步一步地找到问题的根源，并提出扎实的方法去解决它**。扎实的方法指的是，每一步都是在切实地解决问题，有可靠的逻辑、理论支撑。而不是一拍脑门，就觉得某一个方法可以用，唯 SOTA 论，看了论文就觉得这个方法肯定 work。这种分析能力和提出扎实方法的能力，其实就是一个算法工程师的方法论。

例如，我一直说，**模型预测时的表现通常反映了训练样本中的现象**。所以，通常我们解决 case 的流程为：

1. 观察 case，假设 case 的发生原因
2. 按照假设，到大规模数据中过滤复现 case，验证假设是否正确。如果不正确，就根据复现结果更正假设；如果一直假设不出来，就记录 case，看能否找到更多的相似 case，继续更新假设。
3. 验证原因正确后，估算规模，看是否需要特化解决，如需要，就特殊解决一下。
4. 看相同或相似的原因还有多少，设计新的通用优化方案。

所谓做算法，尤其是想要落地或者推广，大多数时候的确更像是**做数据**，下这些笨功夫。相对来讲，**很多“灵光一现”，或者仅仅以指标为基准，看到的论文中的方法，多数在脑暴会议中就会被 fail 掉**。那些方法要么在理论上不靠谱，要么徒增算法的复杂度，却仅仅解决了一小部分问题。

毕竟，部分学术打榜任务，更像是**先有了指标，后有了任务或数据**，而不是以**实际需求为基准，去定义相关的任务及数据**。例如，充斥着大量远监督数据的各路关系抽取数据集（不仅仅是在 train 中，也在 test 中），但又仅仅给出了文本数据；中文需要自然语言推断任务，则机翻了一个数据集，得到了某中文推断数据集；中文需要自己的理解榜单，就有了高 bias 的某文本相似判定数据集；有着大量不可推断关系的某知识图谱补全数据集（知识图谱补全本身也是个指标大于需求的任务）等等。

> 例如，ACL2021 某方法在某数据集上的实验结果中，随机初始化参数的模型和 BERT 模型对比，召回相差10个点左右，而在 tuning 好的模型上，将头尾实体 span 替换成对应的 type （如 China 替换成 country），抹去三元组本身可能过拟合的信息后，召回下降40个点有余。case 分析发现，替换后的正确召回的确是通过文本理解得到的，而远监督得来的三元组，的确大多被判定成了 unrelated。使用先验知识的增益可能仅仅是10个点（所有方法用 BERT 和不用 BERT 的区别都是10个点左右），那么其余的30多个点，大概就是三元组名字过拟合了吧。

这些任务，以及发论文**唯指标论**的风气，也就造成了算法相关工作从学生到工作最大的 gap。

我在工作中见到的刷分团队，其成果很多都是两只脚皆踩在虚空之上。他们在试图将自己研发的所谓“算法”落地时，做的事情往往就是：管应用方要一份数据集，把分数刷上去，超过某些方法，就算是交付了，却完全不分析问题。刷分的手段包括但不限于搜参数（提几百个任务爆搜，连训几个 epoch 都要搜），堆大模型，搞集成。不会优先考虑工程上是否能接受，是否具备应对其他情况的泛化能力，或者这个“算法”是否还有未来成长空间。于是就会导致——一波又一波地赶着热点，或许能发不少 paper，但很难做出来一个能够落地应用的算法。

说了这么多，看上去我描述了一种极端——就是算法工程师就是在闷闷地做分析，做工程优化等，似乎与前沿研究毫不相关。不是的，**算法工程师也要花费相当的时间去刷论文，了解相关的前沿领域发生了什么**。只不过，拥有了这些分析能力及经验后，在遇到前沿论文时，就会具备能力去**拆解一篇论文提出的方法中的每一个组件，吸收其中有价值的 idea 用到自己的项目当中**（而非一定要全盘复用论文，或者纠结论文中方法之外的细枝末节）；或者受到论文中某一个 idea 的启发，提出适用于自己的方案。毕竟，AI 领域，大家的方法都是明牌，数据也是海量。真正理解了“为什么”，才能让它们发挥作用。

> 举个例子。在过去一年的工作中，我根据 Cross-Thought 的论文，提出了一种简化版的优化方案，应用到了我的预训练语言模型中，在序列标注任务上取得了一定的成绩，用一种相对朴素的方法就能得到十分有效的句子的表示。
> 再比如，我们还将 prompt 方法直接应用到了细粒度分类任务中，并根据模型测试时的一些表现，得出了用 prompt 去清洗训练数据，优化其他模型的方法。这个方法已经在我们的一个落地项目，以及某一个学术任务上有了初步成效。同时也孵化了一个文本挖掘框架，今年完成了开源，填补了某一个空白，在相关领域的开发者中颇受好评。小弟不才，在接下来的 WAVE SUMMIT 峰会上，会上场介绍我们最新的成果。

列位可能要问了：有些 idea 可能就是灵光一现，完全凭借灵感和直觉想到的，而非基于繁琐的数据/任务分析。以我说的这种工作风格，这类想法就应该直接放弃？那不就相当于只能做些细枝末节的改进工作了么？

不，扎实的工作风格不代表会错过变革。近些年的经典论文，如 Transformer 、BERT、RoBERTa 等工作，都有相当的理论依据做支撑，其核心思想的立足点，都是相当扎实的，它们自然他们也就成为了经典。而我们看后面涌现了很多不知所谓的改进，哪怕不断地刷新SOTA，却往往又掀不起什么其他的浪花（更有一些直接被**其他研究者用理论去推翻或证明某种改进是有相当的限制的，或过于繁琐的**）。

> 这里我也分享一下我在看待一篇工作时常用的视角。
> 我在观察一个模型的时候，观察的重点一般是这**模型计算某一答案时，利用的信息是什么**。
> 例如，我曾看到一篇用图神经网络建模依存关系树的工作，任务是关系抽取。作者的假设是：语法依存关系是对关系抽取能起到作用的。
> 当我看到这一模型后，心路历程是：首先，一些依存关系的确可以直接表明实体间关系。使用部分依存关系去强化文本 token 之间的关系看上去也是合理的。但是，做过关系抽取的也都知道，并不是所有的依存关系都可以去抽取实体间关系的。有些关系引入了、强调了，则是在引入噪音。所以我当时的第一个疑虑就是：在建模的时候是否对边的关系有所区分（比如用门控结构去控制强调，或者用不同的矩阵往不同的空间上投影）。另外，依存关系的不可传递性与统计模型的连续空间上存在的一种天然冲突，是否会造成问题。

强悍的工程能力

> 援引我最近读的一本书中看到的一句话：**搞深度学习的人，都应该学习一下计算理论。**

工作以来，我越来越感受到，自己得以将各类方法拆解改造、化为己用，这份能力主要都仰赖于我接受过的**数据结构与算法训练**。

高效选数据

上面提到，**与其是做算法，更多时间则是在做数据**。而在工业界，我们最不缺的就是数据——**与其说是“做数据”，不如说是“选数据”**。从海量的数据获取合适的训练样本，则需要繁琐的策略、高效的方法作为支撑。用对了方法，运行效率可能就是几百倍的差异，迭代模型的周期亦是天壤之别。

顶层设计能力

除数据结构和基础算法外，**顶层设计能力同样重要**。无论是单兵作战，还是团队协同，扎实、漂亮的工程设计都会让我们在效率（甚至心情）上有质的变化。

具体的实施细则在大学 CS 课程中已经多有强调。不过，光关注那些说教型的细则还是远远不够，更多的其实还是**多写，多想，多看**。看一看别人的代码中，有什么设计或编码技巧可以吸收借鉴的。想一想**自己在某个工程中写出来的基础模块，之后在其他工程里还是否愿意复用？在团队协同的时候，别人调用你开发的模块是否会遇到困难**？

工程方案的选择

除此之外，工程能力还有一个重要方面，就是**工程方案的选择**。

工作过程中，总是不可避免会应用其他人开发的东西。而当我们拥有选择权时，就要能清楚哪个方案更加适合自己的任务。

基于个人的工作经验，我也总结出了一些原则：

- **明确的，而非黑盒的**。比如，有个分布式工具，它可能“看上去方便”——用简单的几行指令即可完成较为复杂的需求。但没有任何文档说明它中间如何生成各种复杂逻辑，也不提供任何代码可供研究。相比之下，我还是宁愿选择自己下一些笨功夫。

- **简洁的，而非复杂的**。还是以分布式工具为例。如果原本三两个节点就可以解决的问题，它却生成数倍的节点。当然，也就造成了数倍的调度和I/O，以及数倍的资源占用。即使它其他方面做得再好，我还是会尽量弃之不用。

- **可控的，而非限制的**。也举个例子。如果一个模型训练套件，定死了输入样本和输出的格式，或者测试步数、保存逻辑也定死了（不写文档四舍五入也差不多算定死了）。再有甚者，可能连给出的模型也是加密的。相比之下，我肯定愿意选择更可控的方案，或者干脆完全自己写。

- **鲁棒的，而非漏洞的**。例如，框架版本选择，一定选择有人维护且完善的版本，而非久远的旧版本。部署中台，也一定选择 log 明确，错误好查，容易定位，部署过程稳定的，而非每一个步骤都有崩溃的风险，更换一个开发版本直接 bomb 的。

> （上面列举出的都是一些虚构的极端例子...如有雷同，纯属巧合。）

小结

写这篇文章，是因为看到了知乎上关于算法工程师的“职业护城河”的一些讨论，也有感于工作过程中见过的一些事情。希望上面的内容能帮助一些小伙伴刚在入行的时候迅速跨过心理落差，更好地投入业界。

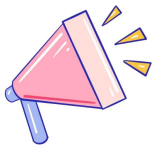
正如开篇所说，这些经验和想法，带有非常浓厚的个人印记。**全都是一家之言**。如果不赞同，还请一笑了之。

萌屋作者：Severus

Severus，在某厂工作的老程序员，主要从事自然语言理解方向，资深死宅，日常愤青，对个人觉得难以理解的同行工作都采取直接吐槽的态度。笔名取自哈利波特系列的斯内普教授，觉得自己也像他那么自闭、刻薄、阴阳怪气，也向往他为爱而伟大。

作品推荐

1. 深度学习，路在何方？
2. 数据还是模型？人类知识在深度学习里还有用武之地吗？
3. 在错误的数据上，刷到 SOTA 又有什么意义？



后台回复关键词【**入群**】

加入卖萌屋NLP/IR/Rec与求职讨论群

后台回复关键词【**顶会**】

获取ACL、CIKM等各大顶会论文集！

喜欢此内容的人还喜欢

预习、体验、理解，只需这三步，轻松学会Python

图灵教育

人工智能领域有哪些曾被拒稿的优秀工作?

深度学习算法与计算机视觉