Kaggle Top1% 是如何炼成的！

不知道你有没有这样的感受，在刚刚入门机器学习的时候，我们一般都是从 MNIST、CIFAR-10 这一类知名公开数据集开始快速上手 ,复现别人的结果，但总觉得过于简单，给人的感觉太不真实。因为这些数据太"完美"了（干净的输入，均衡的类别，分布基本一致的测试集，还有大量现成的参考模型），要成为真正的数据科学家，光在这些数据集上跑模型却是远远不够的。而现实中你几乎不可能遇到这样的数据（现实数据往往有着残缺的输入，类别严重不均衡，分布不一致甚至随时变动的测试集，几乎没有可以参考的论文），这往往让刚进入工作的同学手忙脚乱，无所适从。

Kaggle 则提供了一个介于"完美"与真实之间的过渡，问题的定义基本良好，却夹着或多或少的难点，一般没有完全成熟的解决方案。在参赛过程中与论坛上的其他参赛者互动，能不断地获得启发，受益良多。即使对于一些学有所成的高手乃至大牛，参加 Kaggle 也常常会获得很多启发，与来着世界各地的队伍进行厮杀的刺激更让人欲罢不能。更重要的是，Kaggle 是业界普遍承认的竞赛平台，能从 Kaggle 上的一些高质量竞赛获取好名次，是对自己实力极好的证明，还能给自己的履历添上光辉的一笔。如果能获得金牌，杀入奖金池，那更是名利兼收，再好不过。

Kaggle 适用于以下人群：

- 我是小白，但是对数据科学充满求知欲。
- 我想要历练自己的数据挖掘和机器学习技能 ,成为一名真正的数据科(lao)学(si)家(ji)。
- 我想赢取奖金，成为人生赢家。

0 简介

Kaggle 创办于 2010 年，目前已经被 Google 收购，是全球顶级的数据科学竞赛平台，在数据科学领域中享有盛名。笔者参加了由 Quora 举办的 Quora Question Pairs 比赛，并且获得了前 1%的成绩(3307 支队伍)。这是笔者 Kaggle 首战，所以写下此文来系统化地梳理比赛的思路，并且和大家分享我们参赛的一些心得。

Quora Question Pairs 是一个自然语言(NLP)比赛，比赛的题目可以简单地概括为"预测两个问句的语义相似的概率"。其中的样本如下：

也许是作为 Kaggle 上为数不多的 NLP 比赛，这看似简单的比赛却吸引了众多的参赛队伍。由于这是 NLP 问题，所以接下来的介绍都会偏向于 NLP，本文会分为以下三个部分：

1. 打 Kaggle 比赛的大致套路。（比赛篇）
2. 我们队伍和其他出色队伍的参赛经验。（经验篇）
3. 完成 Kaggle 比赛需要学会哪些实用的工具。（工具篇）

1. 比赛篇

为了方便，我们先定义几个名词：

- **Feature** 特征变量，也叫自变量，是样本可以观测到的特征，通常是模型的输入。
- **Label** 标签，也叫目标变量，需要预测的变量，通常是模型的标签或者输出。
- **Train Data** 训练数据，有标签的数据，由举办方提供。
- **Test Data** 测试数据 ,标签未知 ,是比赛用来评估得分的数据 ,由举办方提供。

- **Train Set** 训练集，从 Train Data 中分割得到的，用于训练模型（常用于交叉验证）。
- **Valid Set** 验证集，从 Train Data 中分割得到的，用于验证模型（常用于交叉验证）。

1.1 分析题目

拿到赛题以后，第一步就是要破题，我们需要将问题转化为相应的机器学习问题。其中，Kaggle 最常见的机器学习问题类型有：

- 回归问题
- 分类问题(二分类、多分类、多标签) 多分类只需从多个类别中预测一个类别，而多标签则需要预测出多个类别。

  比如 Quora 的比赛就是二分类问题，因为只需要判断两个问句的语义是否相似。

1.2 数据分析(Data Exploration)

所谓数据挖掘，当然是要从数据中去挖掘我们想要的东西，我们需要通过人为地去分析数据，才可以发现数据中存在的问题和特征。我们需要在观察数据的过程中思考以下几个问题：

- 数据应该怎么清洗和处理才是合理的？
- 根据数据的类型可以挖掘怎样的特征？
- 数据中的哪些特征会对标签的预测有帮助？

1.2.1 统计分析

对于数值类变量(Numerical Variable) ,我们可以得到 min ,max ,mean，meduim，std 等统计量，用 pandas 可以方便地完成，结果如下：

从上图中可以观察 Label 是否均衡，如果不均衡则需要进行 over sample 少数类，或者 down sample 多数类。我们还可以统计 Numerical Variable 之间的相关系数，用 pandas 就可以轻松获得**相关系数矩阵**：

观察相关系数矩阵可以让你找到高相关的特征，以及特征之间的冗余度。而对于文本变量，可以统计词频(TF)，TF-IDF，文本长度等等，更详细的内容可以参考这里：

https://www.kaggle.com/sudalairajkumar/simple-leaky-exploration-notebook-quora?scriptVersionId=1184830

1.2.2 可视化

人是视觉动物，更容易接受图形化的表示，因此可以将一些统计信息通过图表的形式展示出来，方便我们观察和发现。比如用直方图展示问句的频数：

或者绘制相关系数矩阵:

常用的可视化工具有 matplotlib 和 seaborn。当然，你也可以跳过这一步，因为可视化不是解决问题的重点。

1.3 数据预处理(Data Preprocessing)

刚拿到手的数据会出现噪声，缺失，脏乱等现象，我们需要对数据进行清洗与加工，从而方便进行后续的工作。针对不同类型的变量，会有不同的清洗和处理方法：

- 对于数值型变量(Numerical Variable)，需要处理离群点，缺失值，异常值等情况。
- 对于类别型变量(Categorical Variable)，可以转化为 one-hot 编码。

- 文本数据是较难处理的数据类型，文本中会有垃圾字符，错别字(词)，数学公式，不统一单位和日期格式等。我们还需要处理标点符号，分词，去停用词，对于英文文本可能还要词性还原(lemmatize)，抽取词干(stem)等等。

1.4 特征工程(Feature Engineering)

都说特征为王，特征是决定效果最关键的一环。我们需要通过探索数据，利用人为先验知识，从数据中总结出特征。

1.4.1 特征抽取(Feature Extraction)

我们应该尽可能多地抽取特征，只要你认为某个特征对解决问题有帮助，它就可以成为一个特征。特征抽取需要不断迭代，是最为烧脑的环节，它会在整个比赛周期折磨你，但这是比赛取胜的关键，它值得你耗费大量的时间。

那问题来了，怎么去发现特征呢？光盯着数据集肯定是不行的。如果你是新手，可以先耗费一些时间在 Forum 上，看看别人是怎么做 Feature Extraction 的，并且多思考。虽然 Feature Extraction 特别讲究经验，但其实还是有章可循的：

- 对于 Numerical Variable，可以通过**线性组合**、**多项式组合**来发现新的 Feature。
- 对于文本数据，有一些常规的 Feature。比如，文本长度，Embeddings，TF-IDF，LDA，LSI 等，你甚至可以用深度学习提取文本特征（隐藏层）。
- 如果你想对数据有更深入的了解，可以通过思考数据集的构造过程来发现一些 magic feature，这些特征有可能会大大提升效果。在 Quora 这次比赛中，就有人公布了一些 magic feature。
- 通过**错误分析**也可以发现新的特征（见 1.5.2 小节）。

1.4.2 特征选择(Feature Selection)

在做特征抽取的时候，我们是尽可能地抽取更多的 Feature，但过多的 Feature 会造成冗余，噪声，容易过拟合等问题，因此我们需要进行特征筛选。特征选择可以加快模型的训练速度，甚至还可以提升效果。

特征选择的方法多种多样，最简单的是相关度系数(Correlation coefficient)，它主要是衡量两个变量之间的线性关系，数值在[-1.0, 1.0]区间中。**数值越是接近 0，两个变量越是线性不相关。但是数值为 0，并不能说明两个变量不相关，只是线性不相关而已。**

我们通过一个例子来学习一下怎么分析相关系数矩阵：

相关系数矩阵是一个对称矩阵，所以只需要关注矩阵的左下角或者右上角。我们可以拆成两点来看：

- Feature 和 Label 的相关度可以看作是该 Feature 的重要度，越接近 1 或-1 就越好。
- Feature 和 Feature 之间的相关度要低，如果两个 Feature 的相关度很高，就有可能存在冗余。

除此之外，还可以训练模型来筛选特征，比如带 L1 或 L2 惩罚项的 Linear Model、Random Forest、GDBT 等，它们都可以输出特征的重要度。在这次比赛中，我们对上述方法都进行了尝试，将不同方法的平均重要度作为最终参考指标，筛选掉得分低的特征。

1.5 建模(Modeling)

终于来到机器学习了，在这一章，我们需要开始炼丹了。

1.5.1 模型

机器学习模型有很多，建议均作尝试，不仅可以测试效果，还可以学习各种模型的使用技巧。其实，几乎每一种模型都有回归和分类两种版本，常用模型有：

- KNN
- SVM
- Linear Model（带惩罚项）
- ExtraTree
- RandomForest
- Gradient Boost Tree
- Neural Network

幸运的是，这些模型都已经有现成的工具（如 scikit-learn、XGBoost、LightGBM 等）可以使用，不用自己重复造轮子。但是我们应该要知道各个模型的原理，这样在调参的时候才会游刃有余。当然，你也使用 PyTorch / Tensorflow / Keras 等深度学习工具来定制自己的 Deep Learning 模型，玩出自己的花样。

1.5.2 错误分析

人无完人，每个模型不可能都是完美的，它总会犯一些错误。为了解某个模型在犯什么错误，我们可以观察被模型误判的样本，总结它们的共同特征，我们就可以再训练一个效果更好的模型。这种做法有点像后面 Ensemble 时提到的 Boosting，但是我们是人为地观察错误样本，而 Boosting 是交给了机器。通过**错误分析->发现新特征->训练新模型->错误分析**，可以不断地迭代出更好的效果，并且这种方式还可以培养我们对数据的嗅觉。

举个例子，这次比赛中，我们在错误分析时发现，某些样本的两个问句表面上很相似，但是句子最后提到的地点不一样，所以其实它们是语义不相似的，但我们的模型却把它误判为相似的。比如这个样本：

- Question1: Which is the best digital marketing institution in banglore?
- Question2: Which is the best digital marketing institute in Pune?

为了让模型可以处理这种样本，我们将两个问句的最长公共子串(Longest Common Sequence)去掉，用剩余部分训练一个新的深度学习模型，相当于告诉模型看到这种情况的时候就不要判断为相似的了。因此，在加入这个特征后，我们的效果得到了一些提升。

1.5.3 调参

在训练模型前，我们需要预设一些参数来确定**模型结构**（比如树的深度）和**优化过程**（比如学习率），这种参数被称为**超参**（Hyper-parameter），不同的参数会得到的模型效果也会不同。总是说调参就像是在“炼丹”，像一门“玄学”，但是根据经验，还是可以找到一些章法的：

- 根据经验，选出对模型效果**影响较大的超参**。
- 按照经验设置超参的**搜索空间**，比如学习率的搜索空间为[0.001，0.1]。
- 选择**搜索算法**，比如 Random Search、Grid Search 和一些启发式搜索的方法。
- **验证模型**的泛化能力（详见下一小节）。

1.5.4 模型验证(Validation)

在 Test Data 的标签未知的情况下，我们需要自己构造测试数据来验证模型的泛化能力，因此把 Train Data 分割成 Train Set 和 Valid Set 两部分，Train Set 用于训练，Valid Set 用于验证。

- **简单分割**

  将 Train Data 按一定方法分成两份，比如随机取其中 70%的数据作为 Train Set，剩下 30%作为 Valid Set，每次都固定地用这两份数据分别训练模型和验证模型。这种做法的缺点很明显，它没有用到整个训练数据，所以验证效果会有偏差。通常只会在训练数据很多，模型训练速度较慢的时候使用。

- **交叉验证**

  交叉验证是将整个训练数据随机分成 K 份，训练 K 个模型，每次取其中的 K-1 份作为 Train Set，留出 1 份作为 Valid Set，因此也叫做 **K-fold**。至于这个 K，你想取多少都可以，但一般选在 3～10 之间。我们可以用 K 个模型得分的 mean 和 std，来评判模型得好坏（mean 体现模型的能力，std 体现模型是否容易过拟合），并且用 K-fold 的验证结果通常会比较可靠。

  如果数据出现 Label 不均衡情况，可以使用 Stratified K-fold，这样得到的 Train Set 和 Test Set 的 Label 比例是大致相同。

1.6 模型集成(Ensemble)

曾经听过一句话，"**Feature 为主，Ensemble 为后**"。Feature 决定了模型效果的上限，而 Ensemble 就是让你更接近这个上限。Ensemble 讲究"好而不同"，不同是指模型的学习到的侧重面不一样。举个直观的例子，比如数学考试，A 的函数题做的比 B 好，B 的几何题做的比 A 好，那么他们合作完成的分数通常比他们各自单独完成的要高。

常见的 Ensemble 方法有 Bagging、Boosting、Stacking、Blending。

1.6.1 Bagging

Bagging 是将多个模型(基学习器)的预测结果简单地加权平均或者投票。Bagging 的好处在于可以并行地训练基学习器，其中 Random Forest 就用到了 Bagging 的思想。举个通俗的例子，如下图：

老师出了两道加法题，A 同学和 B 同学答案的加权要比 A 和 B 各自回答的要精确。

Bagging 通常是没有一个明确的优化目标的，但是有一种叫 Bagging Ensemble Selection 的方法，它通过贪婪算法来 Bagging 多个模型来优化目标值。在这次比赛中，我们也使用了这种方法。

1.6.2 Boosting

Boosting 的思想有点像**知错能改**，每训练一个基学习器，是为了弥补上一个基学习器所犯的错误。其中著名的算法有 AdaBoost，Gradient Boost。Gradient Boost Tree 就用到了这种思想。

我在 1.2.3 节(错误分析)中提到 Boosting ,错误分析->抽取特征->训练模型->错误分析，这个过程就跟 Boosting 很相似。

1.6.3 Stacking

Stacking 是用新的模型(**次学习器**)去学习怎么组合那些基学习器，它的思想源自于 Stacked Generalization 这篇论文。如果把 Bagging 看作是多个基分类器的线性组合，那么 Stacking 就是多个基分类器的非线性组合。

Stacking 可以很灵活，它可以将学习器一层一层地堆砌起来，形成一个网状的结构，如下图：

举个更直观的例子，还是那两道加法题：

这里 A 和 B 可以看作是基学习器，C、D、E 都是次学习器。

- Stage1: A 和 B 各自写出了答案。
- Stage2: C 和 D 偷看了 A 和 B 的答案，C 认为 A 和 B 一样聪明，D 认为 A 比 B 聪明一点。他们各自结合了 A 和 B 的答案后，给出了自己的答案。
- Stage3: E 偷看了 C 和 D 的答案，E 认为 D 比 C 聪明，随后 E 也给出自己的答案作为最终答案。

在实现 Stacking 时，要注意的一点是，避免**标签泄漏**(Label Leak)。在训练次学习器时，需要上一层学习器对 Train Data 的测试结果作为特征。如果我们在 Train Data 上训练，然后在 Train Data 上预测，就会造成 Label Leak。为了避免 Label Leak，需要对每个学习器使用 K-fold，将 K 个模型对 Valid Set 的预测结果拼起来，作为下一层学习器的输入。如下图：

由图可知，我们还需要对 Test Data 做预测。这里有两种选择，**可以将 K 个模型对 Test Data 的预测结果求平均，也可以用所有的 Train Data 重新训练一个新模型来预测 Test Data**。所以在实现过程中，我们最好把每个学习器对 Train Data 和对 Test Data 的测试结果都保存下来，方便训练和预测。

对于 Stacking 还要注意一点，固定 K-fold 可以尽量避免 Valid Set 过拟合，也就是全局共用一份 K-fold，如果是团队合作，组员之间也是共用一份 K-fold。

如果想具体了解为什么需要固定 K-fold，请看这里：

https://www.zhihu.com/question/61467937/answer/188191424

1.6.4 Blending

Blending 与 Stacking 很类似，它们的区别可以参考这里:

https://mlwave.com/kaggle-ensembling-guide/

1.7 后处理

有些时候在确认没有过拟合的情况下，验证集上做校验时效果挺好，但是将测试结果提交后的分数却不如人意，这时候就有可能是训练集的分布与测试集的分布不一样而导致的。这时候为了提高 LeaderBoard 的分数，还需要对测试结果进行分布调整。

比如这次比赛，训练数据中正类的占比为 0.37，那么预测结果中正类的比例也在 0.37 左右，然后 Kernel 上有人通过测试知道了测试数据中正类的占比为 0.165，所以我们也对预测结果进行了调整，得到了更好的分数。具体可以看这里：https://www.kaggle.com/davidthaler/how-many-1-s-are-in-the-public-lb

2. 经验篇

2.1 我们的方案（33th）

深度学习具有很好的模型拟合能力，使用深度学习可以较快得获取一个不错的 Baseline，对这个问题整体的难度有一个初始的认识。虽然使用深度学习可以免去繁琐的手工特征，但是它也有能力上限，所以提取传统手工特征还是很有必要的。我们尝试 Forum 上别人提供的方法，也尝试自己思考去抽取特征。

总结一下，我们抽取的手工特征可以分为以下 4 种：

- Text Mining Feature，比如句子长度；两个句子的文本相似度，如 N-gram 的编辑距离，Jaccard 距离等；两个句子共同的名词，动词，疑问词等。

- Embedding Feature，预训练好的词向量相加求出句子向量，然后求两个句子向量的距离，比如余弦相似度、欧式距离等等。
- Vector Space Feature，用 TF-IDF 矩阵来表示句子，求相似度。
- Magic Feature，是 Forum 上一些选手通过思考数据集构造过程而发现的 Feature，这种 Feature 往往与 Label 有强相关性，可以大大提高预测效果。

我们的系统整体上使用了 Stacking 的框架，如下图：

- Stage1: 将两个问句与 Magic Feature 输入 Deep Learning 中，将其输出作为下一层的特征（这里的 Deep Learning 相当于特征抽取器）。我们一共训练了几十个 Deep Learning Model。
- Stage2: 将 Deep Learning 特征与手工抽取的几百个传统特征拼在一起，作为输入。在这一层，我们训练各种模型，有成百上千个。
- Stage3: 上一层的输出进行 Ensemble Selection。

比赛中发现的一些深度学习的局限：

通过对深度学习产生的结果进行错误分析，并且参考论坛上别人的想法，我们发现深度学习没办法学到的特征大概可以分为两类：

- 对于一些数据的 Pattern，在 Train Data 中出现的频数不足以让深度学习学到对应的特征，所以我们需要通过手工提取这些特征。
- 由于 Deep Learning 对样本做了独立同分布假设（iid），一般只能学习到每个样本的特征，而学习到数据的全局特征，比如 TF-IDF 这一类需要统计全局词频才能获取的特征，因此也需要手工提取这些特征。

传统的机器学习模型和深度学习模型之间也存在表达形式上的不同。虽然传统模型的表现未必比深度学习好，但它们学到的 Pattern 可能不同，通过

Ensemble 来取长补短，也能带来性能上的提升。因此，同时使用传统模型也是很有必要的。

2.2 第一名的解决方案

比赛结束不久，第一名也放出了他们的解决方案，我们来看看他们的做法。

https://www.kaggle.com/c/quora-question-pairs/discussion/34355

他们的特征总结为三个类别：

- Embedding Feature
- Text Mining Feature
- Structural Feature（他们自己挖掘的 Magic Feature）

并且他们也使用了 Stacking 的框架，并且使用固定的 k-fold：

- Stage1: 使用了 Deep Learning ,XGBoost ,LightGBM ,ExtraTree ,Random Forest，KNN 等 300 个模型。
- Stage2: 用了手工特征和第一层的预测和深度学习模型的隐藏层，并且训练了 150 个模型。
- Stage3: 使用了分别是带有 L1 和 L2 的两种线性模型。
- Stage4: 将第三层的结果加权平均。

对比以后发现我们没有做 LDA、LSI 等特征，并且 N-gram 的粒度没有那么细（他们用了 8-gram），还有他们对 Magic Feature 的挖掘更加深入。还有一点是他们的 Deep Learning 模型设计更加合理，他们将筛选出来的手工特征也输入到深度学习模型当中，我觉得这也是他们取得好效果的关键。因为显式地将手工特征输入给深度学习模型，相当于告诉"它你不用再学这些特征了，你

去学其他的特征吧”，这样模型就能学到更多的语义信息。所以，我们跟他们的差距还是存在的。

3. 工具篇

工欲善其事，必先利其器。

Kaggle 的上常工具除了大家耳熟能详的 XGBoost 之外，这里要着重推荐的是一款由微软推出的 LightGBM，这次比赛中我们就用到了。LightGBM 的用法与 XGBoost 相似，两者使用的区别是 XGBoost 调整的一个重要参数是树的高度，而 LightGBM 调整的则是叶子的数目。与 XGBoost 相比，在模型训练时速度快，单模型的效果也略胜一筹。

调参也是一项重要工作，调参的工具主要是 Hyperopt，它是一个使用搜索算法来优化目标的通用框架，目前实现了 Random Search 和 Tree of Parzen Estimators (TPE)两个算法。

对于 Stacking，Kaggle 的一位名为Μαριος Μιχαηλιδης的 GrandMaster 使用 Java 开发了一款集成了各种机器学习算法的工具包 StackNet，据说在使用了它以后你的效果一定会比原来有所提升，值得一试。

以下总结了一些常用的工具：

- Numpy | 必用的科学计算基础包，底层由 C 实现，计算速度快。
- Pandas | 提供了高性能、易用的数据结构及数据分析工具。
- NLTK | 自然语言工具包，集成了很多自然语言相关的算法和资源。
- Stanford CoreNLP | Stanford 的自然语言工具包，可以通过 NLTK 调用。
- Gensim | 主题模型工具包，可用于训练词向量，读取预训练好的词向量。
- scikit-learn | 机器学习 Python 包 ，包含了大部分的机器学习算法。

- XGBoost / LightGBM | Gradient Boosting 算法的两种实现框架。
- PyTorch / TensorFlow / Keras | 常用的深度学习框架。
- StackNet | 准备好特征之后，可以直接使用的 Stacking 工具包。
- Hyperopt | 通用的优化框架，可用于调参。

4. 总结与建议

在参加某个比赛前，要先衡量自己的机器资源能否足够支撑你完成比赛。比如一个有几万张图像的比赛，而你的显存只有 2G，那很明显你是不适合参加这个比赛的。当你选择了一个比赛后，可以先"热热身"，稍微熟悉一下数据，粗略地跑出一些简单的模型，看看自己在榜上的排名，然后再去慢慢迭代。

Kaggle 有许多大牛分享 Kernel，有许多 Kernel 有对于数据精辟的分析，以及一些 baseline 模型，对于初学者来说是很好的入门资料。在打比赛的过程中可以学习别人的分析方法，有利于培养自己数据嗅觉。甚至一些 Kernel 会给出一些 data leak，会对于比赛提高排名有极大的帮助。

其次是 Kaggle 已经举办了很多比赛，有些比赛有类似之处，比如这次的 Quora 比赛就与之前的 Home Depot Product Search Relevance 有相似之处，而之前的比赛前几名已经放出了比赛的 idea 甚至代码，这些都可以借鉴。

另外，要足够地重视 Ensemble，这次我们组的最终方案实现了 paper "Ensemble Selection from Libraries of Models" 的想法，所以有些比赛可能还需要读一些 paper，尤其对于深度学习相关的比赛，最新 paper，最新模型的作用就举足轻重了。

而且，将比赛代码的流程自动化，是提高比赛效率的一个关键，但是往往初学者并不能很好地实现自己的自动化系统。我的建议是初学者不要急于构建自动

化系统，当你基本完成整个比赛流程后，自然而然地就会在脑海中形成一个框架，这时候再去构建你的自动化系统会更加容易。

最后，也是最重要的因素之一就是时间的投入，对于这次比赛， 我们投入了差不多三个多月 ,涉及到了对于各种能够想到的方案的尝试。尤其最后一个月，基本上每天除了睡觉之外的时间都在做比赛。所以要想在比赛中拿到好名次，时间的投入必不可少。另外对于国外一些介绍 kaggle 比赛的博客(比如官方博客)也需要了解学习，至少可以少走弯路，本文的结尾列出了一些参考文献，都值得细细研读。

最后的最后，请做好心理准备，这是一场持久战。因为比赛会给你带来压力，也许过了一晚，你的排名就会一落千丈。还有可能造成出现失落感，焦虑感，甚至失眠等症状。但请你相信，它会给你带来意想不到的惊喜，认真去做，你会觉得这些都是值得的。

参考文献：

1.http://59.80.44.99/www.cs.cornell.edu/~alexn/papers/shotgun.icml04.revised.rev2.pdf

2. https://zhuanlan.zhihu.com/p/26820998

3. https://mlwave.com/kaggle-ensembling-guide/