1.

缺失值（missing value）的自動處理類別特徵(Categorical Feature) 的進一步優化，不再使用類似one-hot coding的分割方式。對於類別數量很多的類別特徵，使用one-vs-other的切分方式會長出很不平衡的樹，不能實現較好的精度。這是樹模型在支持類別特徵的一個痛點。 LightGBM可以找出類別特徵的最優切割，即many-vs-many的切分方式。並且最優分割的查找的時間複雜度可以在線性時間完成，和原來的one-vs-other的複雜度幾乎一致。

2.

完成了python-package，欢迎使用。

直接支持类别特征(Categorical Feature)，不需要进行0/1展开。相对0/1展开的解决方案，速度快非常多，且精度一致。大多数机器学习工具都无法直接支持类别特征作为输入，一般需要转换成多维0/1特征，带来计算和内存上的额外消耗。LightGBM增加了针对于类别特征的决策规则，这在决策树上也很好实现。主要的思想是，在对类别特征计算分割增益的时候，不是按照数值特征那样由一个阈值进行切分，而是直接把其中一个类别当成一类，其他的类别当成另一类。这实际上与0/1展开的效果是一样的。

3.

GBDT 雖然是個強力的模型，但卻有著一個致命的缺陷，不能用類似 mini batch 的方式來訓練，需要對數據進行無數次的遍歷。如果想要速度，就需要把數據都預加載在內存中，但這樣數據就會受限於內存的大小；如果想要訓練更多的數據，就要使用外存版本的決策樹算法。雖然外存算法也有較多優化，SSD 也在普及，但在頻繁的 IO 下，速度還是比較慢的。為了能讓 GBDT 高效地用上更多的數據，我們把思路轉向了分佈式 GBDT， 然後就有了 LightGBM。設計的思路主要是兩點，1. 單個機器在不犧牲速度的情況下，盡可能多地用上更多的數據；2.多機並行的時候，通信的代價盡可能地低，並且在計算上可以做到線性加速。基於這兩個需求，LightGBM 選擇了基於 histogram 的決策樹算法。相比於另一個主流的算法 pre-sorted（如 xgboost 中的 exact 算法），histogram 在內存消耗和計算代價上都有不少優勢。 Pre-sorted 算法需要的內存約是訓練數據的兩倍(2 \* #data \* #features\* 4Bytes)，它需要用32位浮點來保存feature value，並且對每一列特徵，都需要一個額外的排好序的索引，這也需要32位的存儲空間。對於 histogram 算法，則只需要(#data\* #features \* 1Bytes)的內存消耗，僅為 pre-sorted算法的1/8。因為 histogram 算法僅需要存儲 featurebin value (離散化後的數值)，不需要原始的 feature value，也不用排序，而 binvalue 用 uint8\_t (256bins) 的類型一般也就足夠了。在計算上的優勢則主要體現在“數據分割”。決策樹算法有兩個主要操作組成，一個是“尋找分割點”，另一個是“數據分割”。從算法時間複雜度來看，Histogram 算法和 pre-sorted 算法在“尋找分割點”的代價是一樣的，都是O(#feature\*#data)。而在“數據分割”時，pre-sorted 算法需要O(#feature\*#data)，而 histogram 算法是O(#data)。因為 pre-sorted 算法的每一列特徵的順序都不一樣，分割的時候需要對每個特徵單獨進行一次分割。 Histogram算法不需要排序，所有特徵共享同一個索引表，分割的時候僅需對這個索引表操作一次就可以。 （更新: 這一點不完全正確，pre-sorted 與level-wise 結合的時候，其實可以共用一個索引表(row\_idx\_to\_tree\_node\_idx)。然後在尋找分割點的時候，同時操作同一層的節點，省去分割的步驟。但這樣做的問題是會有非常多隨機訪問，有很大的chche miss，速度依然很慢。）。另一個計算上的優勢則是大幅減少了計算分割點增益的次數。對於一個特徵，pre-sorted 需要對每一個不同特徵值都計算一次分割增益，而 histogram 只需要計算 #bin (histogram 的橫軸的數量) 次。最後，在數據並行的時候，用 histgoram 可以大幅降低通信代價。用 pre-sorted 算法的話，通信代價是非常大的（幾乎是沒辦法用的）。所以 xgoobst 在並行的時候也使用 histogram 進行通信。當然， histogram 算法也有缺點，它不能找到很精確的分割點，訓練誤差沒有 pre-sorted 好。但從實驗結果來看， histogram 算法在測試集的誤差和 pre-sorted 算法差異並不是很大，甚至有時候效果更好。實際上可能決策樹對於分割點的精確程度並不太敏感，而且較“粗”的分割點也自帶正則化的效果。在 histogram 算法之上， LightGBM 進行進一步的優化。首先它拋棄了大多數 GBDT 工具使用的按層生長(level-wise) 的決策樹生長策略，而使用了帶有深度限制的按葉子生長 (leaf-wise) 算法。 level-wise 過一次數據可以同時分裂同一層的葉子，容易進行多線程優化，不容易過擬合。但實際上level-wise是一種低效的算法，因為它不加區分的對待同一層的葉子，帶來了很多沒必要的開銷。因為實際上很多葉子的分裂增益較低，沒必要進行搜索和分裂。 leaf-wise則是一種更為高效的策略，每次從當前所有葉子中，找到分裂增益最大(一般也是數據量最大)的一個葉子，然後分裂，如此循環。因此同 level-wise 相比，在分裂次數相同的情況下，leaf-wise 可以降低更多的誤差，得到更好的精度。 leaf-wise 的缺點是可能會長出比較深的決策樹，產生過擬合。因此 LightGBM 在leaf-wise 之上增加了一個最大深度的限制，在保證高效率的同時防止過擬合。另一個比較巧妙的優化是 histogram 做差加速。一個容易觀察到的現象：一個葉子的直方圖可以由它的父親節點的直方圖與它兄弟的直方圖做差得到。通常構造直方圖，需要遍歷該葉子上的所有數據，但直方圖做差僅需遍歷直方圖的 k 個桶。利用這個方法，LightGBM 可以在構造一個葉子的直方圖後，可以用非常微小的代價得到它兄弟葉子的直方圖，在速度上可以提升一倍。

4.

非常讚的工作，實現了和XGBoost不一樣的搜索策略，所以在算法效果上並不是完全一樣。 - XGBoost在單機默認是exact greedy，搜索所有的可能分割點。分佈式是dynamic histogram，每一輪迭代重新estimate 潛在split candidate。 - LightGBM和最近的FastBDT都採取了提前histogram binning再在bin好的數據上面進行搜索。在限定好candidate splits,- 主要的速度提升似乎來自於兩點。一個是搜索的時候選取delta比較大的葉子擴展。第二個是pre-bin之後的histogram的求和用了一個非常巧妙的減法trick，省了一半的時間。在算法和效果上面最近比較多的工作都開始基於提前限定分割點的近似算法然後快速求histogram。這一類算法的潛在問題是限制了分割點只能是一開始的定下來的潛在這些。不知道這一點對於實際應用的影響會有多大。理論上數據越多，樹越深的時候，需要的潛在分割點越多，可能需要根據訓練來動態更新潛在的分割點。在算法上面採用delta比較大的擴展方向可以集中搜索提高比較重要的區域。一個潛在的問題是可能會忽略掉一些未來有潛力的節點。當然這些討論都和具體的應用場景有關。個人覺得exact greedy和histogram方法的還會共存一段時間。或許可以比較好的在系統上面對這兩個一起支持在系統上面因為集中做針對葉子的分割，似乎會對於數據集的random access有一定的要求。如果不shuffle data的情況下可能會有cache 的問題。這個數據結構是一個有趣的問題非常值得學習，機器學習系統優化是非常重要的方向。希望可以看到越來越多這樣實際的工作