有名ライブラリと比較した LightGBM の現在

PyData.Tokyo Meetup #21

株式会社AlphaImpact

大元 司

自己紹介

- 大元（おおもと） 司（つかさ） (@henry0312)
- 株式会社ドワンゴ (2015.04-2019.06)
  - Dwango Media Village
- 株式会社AlphaImpact (2019.02-)
  - 競馬を予想して飯を食ってる
- LightGBMのコミッターの１人
  - ということで今日呼ばれたが、最近コントリビュート出来ていない

LightGBMとの出会い

- 2016年10月頃にGBDT + ランク学習に対応したライブラリとして偶然見つけた
- 当時、Pythonインターフェースは存在しなかったが、12月頃に公開された
- ここから本格的に自分でも利用するようになり、数々のPRを送り、OSSとしての質を上げる助言などを行っている内にコミッターとなった

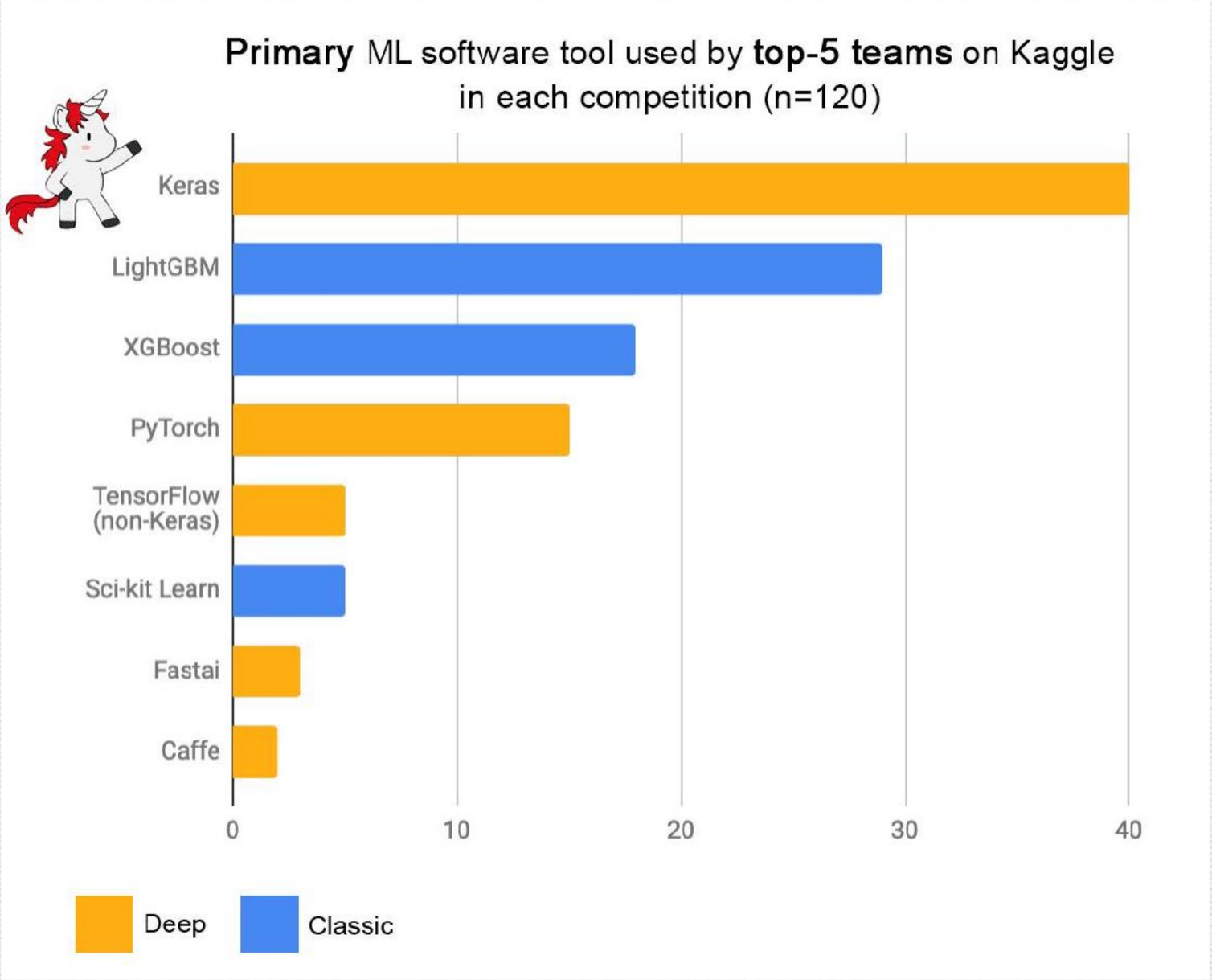
本日の発表内容

- LightGBMの現在
  - LightGBMとは
  - XGBoost，CatBoostとの比較
- LightGBMを使う上で気をつけたいこと
  - パラメータチューニング

LightGBMの現在

LightGBMとは

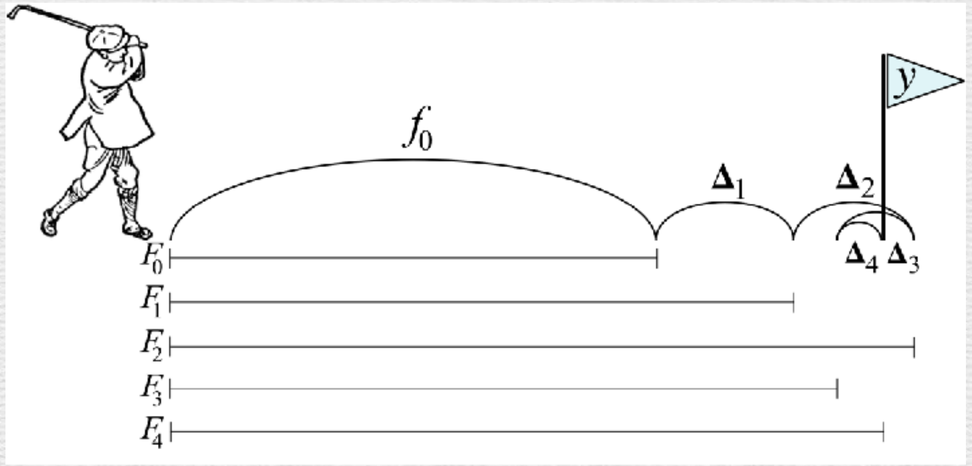
- Microsoft社製の勾配ブースティングライブラリ
  - 構造化データを対象としたとき、極めて高い性能を達成することが期待出来るアルゴリズムとして認知されている（はず）
- Kaggleの上位入賞チームの中で最も利用されている（古典的）機械学習ライブラリ (2019/4/4)



https://twitter.com/fchollet/status/1113476428249464833 より引用

そもそもGBMとは

- Gradient Boosting Machines
- 弱識別器を1個ずつ追加していくアンサンブル学習で、N番目までの部分で説明できない部分をN+1番目で説明できるように学習していく手法



How to explain gradient boosting より

そして、GBDTへ

- Gradient Boosting Decision Trees
- GBMの弱識別器に決定木 (Decision Tree) を利用する
- 決定木の様々な利点とアンサンブル学習の利点が噛み合った強い手法

3つの有名なGBDTライブラリ

- 上から順に登場時期が新しくなる
  - dmlc/xgboost (KDD 2016)
  - microsoft/LightGBM (NIPS 2017)
  - catboost/catboost (NIPS 2018)
- *a new one to be appeared in 2019? :)*

XGBoostの貢献

- GBDTの性能の高さを世に知らしめるという大きな貢献を残した
- 多くの人々の注目を集めた結果、
  - 様々なプログラミング言語のインターフェースを用意できた
    - それがまたユーザーを集めた
  - 勾配ブースティング決定木に関する情報が増えた

XGBoostの長所

- 多くの開発者が長年開発してきたので、ライブラリが安定している
  - 多くのユーザーが開発に貢献してくれるのはやはり大きい
- 様々なプログラミング言語インターフェースを持つ
  - HadoopやSparkで動くのはやはりそれなりに嬉しい人達が居るのでは？

XGBoostの短所

- 他の後発ライブラリと比較して、
  - プログラムの実行速度が遅い（学習に時間がかかる）
    - 結果、同程度の性能を得るためのコストが高くつく
    - 現代的なタスクに適していないのでは？

LightGBMの誕生

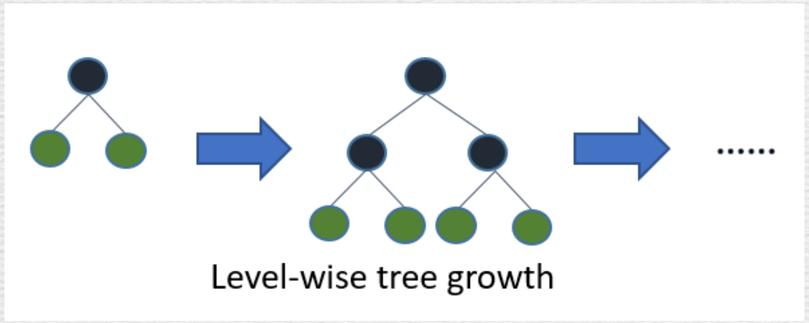
- XGBoostをもっと高パフォーマンスで動かせると嬉しいよね
- LightGBMが特に力を入れる2つ
  - 効率の良い並列化による学習の高速化
  - 大規模なデータに適用できるメモリパフォーマンス

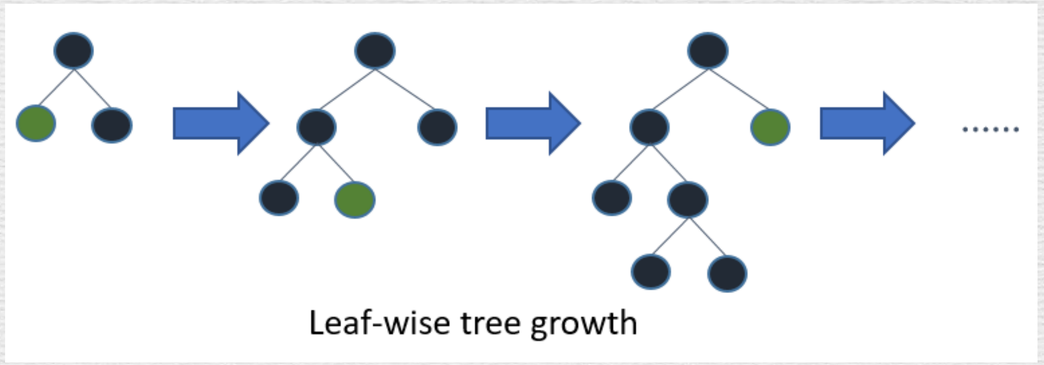
LightGBMの特徴

- ヒストグラムに基づく決定木アルゴリズムを採用
  - 連続値を bin にまとめ、離散値として扱う
  - これにより時間計算量、空間計算量ともに削減
    - ノードの分割に O(#data) 必要だったものが O(#bin) に

LightGBMの特徴

- 深さベースではなく、葉ベースで決定木を学習する
  - より小さなロスを達成できる
    - ただし、データが少ないとき過学習を引き起こしやすい





LightGBMの長所

- 先に説明した特徴はそのまま長所になる
- 様々なレベルの多くのユーザーが利用することで生まれた情報の豊富さ
  - Kaggleのカーネルが宝の山となる
  - Optuna に integration がある
    - 何やらLightGBM用の新しい integration の開発が進んでいる模様 :D

LightGBMの短所

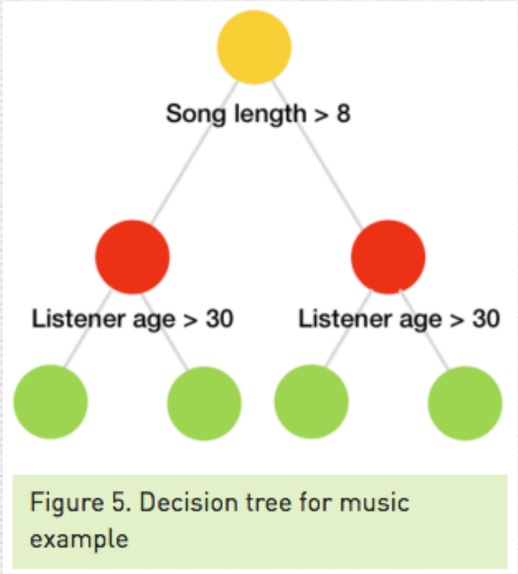
- 過学習が厳しい（訓練データに過適合しやすい）
- デフォルトパラメータが使い物にならない（ことが多い）
  - パラメータチューニングについては後述する
- カテゴリカル変数が多すぎると厳しい

CatBoostの登場

- 学習データという本来あるべきデータの一部を用いてモデルの学習を行うことで生じる”真のモデル”との”ずれ”を抑えようと努力したGBDTライブラリ
  - カテゴリカル変数の効率的な前処理 (Ordered TS)
  - 訓練データの並びをランダムに変えながら行う学習 (Orderd Boosting)
    - これを高速に計算する実装工夫

CatBoostの登場

- 弱識別器（1つの決定木）に binary symmetric decision tree を利用する
  - 過学習を抑える
  - ハイパーパラメータの変化に安定する
  - 並列化に向いた構造のため、推論が早い



NVIDA Developer Blog より

CatBoostの長所

- カテゴリカル変数の前処理が不要
  - ライブラリに様々なアプローチが実装されている
- binary symmetric decision tree のお陰で
  - デフォルトパラメータが悪くない
  - 推論が早い
- ドキュメントがとても充実している

CatBoostの短所

- 学習が遅い
  - XGBoostよりは早い
  - ただし、GPUを使うと LightGBM (GPU) より早くなるという報告がある
- 自社製のよく分からないビルドツール
  - ユーザーにはあまり関係ないが、開発に貢献したい人は少し大変そう

LightGBMをいつ使うか

- データが大規模なとき
  - AlphaImpactでは、200GB程度のデータを対象に利用している
- データの前処理を頑張るとき
- Sparkで利用したいとき
  - 一昔前はXGBoost一択だったが、今はまぁまぁ安定していると StrikerRUS 氏が言っていた

逆にLightGBMを使わないとき

- カテゴリカル変数の前処理が大変なとき => CatBoost
- プロダクションサービスで推論速度が重要なとき => CatBoost
- タスクの初手でベースラインを作りたいとき => CatBoost

CatBoostの宣伝か？

私がLightGBMを使う理由

- 扱うデータが大きいので、現実的な時間で使えそうなのがLightGBMしか残っていない
  - GPUのVRAMにも限度がある
- 以前は、自分に必要な機能を良い感じにねじ込んだりもしたが、最近はそういうのもほとんど無くなった
  - つまり、機能的に満足しており、特に大きなバグにも遭遇しなくなった（安定している）

LightGBMを使う上で気をつけたいこと

パラメータチューニング

- LightGBMのパラメータチューニングは必須
- ポイントは、いかにして過学習を抑えるか
- 公式ドキュメント Parameters Tuning は必読

パラメータ名	役割
boosting_type	ブースティング方法。色々あるが、多くの場合 gbdt を使う。
n_estimators	木の数。early stopping を使うので、実質無限に相当する値を指定しておけば良い。
num_leaves	ツリーの葉の数。2^(max_depth)より小さくする。
max_depth	ツリーの深さの最大値。7ぐらいにしとけば良いと思う。
min_child_samples (min_data_in_leaf)	葉を作るのに必要な最低サンプル数。 データが少なく厳しいときは、これを小さくする。
min_child_weight (min_sum_hessian_in_leaf)	葉を分割するのに必要な（ロスの）hessianの合計値。小さければ小さいほどロスを小さくしようと葉を分割するが、それはオーバーフィッティングを引き起こす。

パラメータ名	役割
subsample (bagging_fraction)	バギングの割合（訓練データの何パーセントを利用するか）
subsample_freq	バギングを行う間隔
colsample_bytree (feature_fraction)	特徴量サンプリングの割合（何パーセントの特徴量を利用するか）
min_split_gain	葉を分割する条件として設定するロス改善度の最小値。 この値以上の改善が無ければ葉を分割しない。
reg_alpha	L1正則化。過学習していそうなら調整する。
reg_lambda	L2正則化。過学習していそうなら調整する。

パラメータ名	役割
learning_rate	学習率。0.01の固定で良いと思う。
random_state	乱数シード。必ず固定すること。

探索空間の一例

```python
space = {
    'boosting_type': FixedSpace('boosting_type', 'gbdt'),
    'n_estimators': FixedSpace('n_estimators', 10000),
    'num_leaves': IntParamSpace('num_leaves', 2, 100),
    'max_depth': FixedSpace('max_depth', 7),
    'max_bin': FixedSpace('max_bin', 255),
    'subsample': UniformParamSpace('subsample', 0.5, 1.0),
    'subsample_freq': IntParamSpace('subsample_freq', 1, 20),
    'colsample_bytree': UniformParamSpace('colsample_bytree', 0.01, 1.0),
    'min_child_samples': IntParamSpace('min_child_samples', 1, 50),
    'min_child_weight': LogUniformParamSpac('min_child_weight', 1e-3, 1e+1),
    'reg_alpha': FixedSpace('reg_alpha', 0.),
    'reg_lambda': LogUniformParamSpace('reg_lambda', 1e-2, 1e+3),
    'min_split_gain': FixedSpace('min_split_gain', 0.),
    'learning_rate': FixedSpace('learning_rate', 0.01),
    'random_state': FixedSpace('random_state', 42),
}
```

FixedSpace: 固定値

IntParamSpace: 整数

UniformParamSpace: 一様分布

LogUniformParamSpace: 対数一様分布

その他、注意点

- LightGBMを使う時は必ずソースコードからビルドすること
  - PyPIにあるパッケージには、並列化周りに謎のバグがある
- https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html

まとめ

- LightGBMの現在をXGBoostとCatBoostの比較から見直してみた
- パラメータチューニングの一例を紹介した
  - さらに踏み込んだ話しはこの後のOptunaの発表に期待！
- LightGBMを使う時はソースコードからビルドすること

参考・引用文献

- https://lightgbm.readthedocs.io/
- https://xgboost.readthedocs.io/
- https://catboost.ai/
- 『CatBoost: unbiased boosting with categorical features』at NeurIPS2018 読み会
- How to Win a Data Science Competition: Learn from Top Kagglers