Reference: https://www.kaggle.com/alexisbcook/xgboost

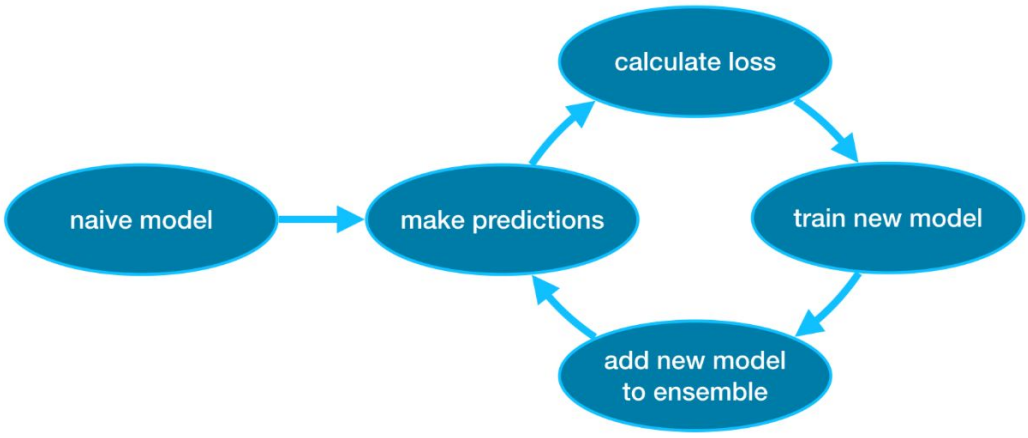
**XGBoost (eXtreme Gradient BOOSTing)** is regarded as the most accurate modeling technique for structured data; one of the ensemble methods

What we have learnt so far: learn about ensemble method

- Make prediction with single decision tree
- Improve performance of prediction with with random forest method

This time we are going to learn about another ensemble method: gradient boosting

- **Gradient Boosting**: a method that goes through cycles to iteratively add models into an ensemble.



- **Start with naive model**: errors and inaccurate tends to happen but this will be addressed by next steps
- **Make predictions**: use the current ensemble to generate predictions for each observation in the dataset. To make a prediction, we add the predictions from all models in the ensemble
- **Calculate loss**: use predictions to calculate loss
- **Train new model:** use the loss function to fit a new model that will be added to the ensemble. The goal is to reduce the loss; so we determine model parameters so that adding this new model to the ensemble will achieve this
- **Add new model to ensemble**:
- **Repeat…**...

- Implementation
  - From Scikit-learn: XGBoost library vs Gradient Boosting
  - XGBoost has more technical advantages, xgboost.XGBRegressor…..
  - Parameter Tuning
    - **n_estimators**: specifies how many times to go through the modeling cycle described above. It is equal to the number of models that we include in the ensemble. Range [100-1000]; Its value depends on learning_rate.

    - Too *low* a value causes *underfitting*, which leads to inaccurate predictions on both training data and test data.
    - Too *high* a value causes *overfitting*, which causes accurate predictions on the training data, but inaccurate predictions on test data (*which is what we care about*).
- **early_stopping_rounds:** offers a way to automatically find the ideal value for **n_estimators**. Early stopping causes the model to stop iterating when the validation score stops improving, even if we aren't at the hard stop for **n_estimators**. It's smart to set a high value for **n_estimators** and then use **early_stopping_rounds** to find the optimal time to stop iterating.
    - early_stopping_rounds=; is a reasonable choice. In this case, we stop after 5 straight rounds of deteriorating validation scores.
    - When using early_stopping_rounds, you also need to set aside some data for calculating the validation scores; this is done by setting the eval_set parameter.
- **learning_rate:** each tree we add to the ensemble helps us less; we can then set higher **n_estimators** without overfitting.

    - Use **learning_rate** multiply with the predictions from each model before adding them in ensemble.
    - Default value = 0.1
- **n_job:** use on larger dataset where runtime is critical, it makes use of parallelism to build model faster.
    - Set n_job equal number for cores on your machine