Gradient boosting with XGBoost

→ State of the art data science library for performing classification and regression.

→ XGBoost makes use of gradient boosted decision trees, which provide better performance

→ The problem with regular decision is that they are often not complex enough to capture intricacies of many large datasets. We could increase the max depth ~~but~~ it causes overfitting of data.

Instead, we use gradient boosting to combine many decision trees into a single model for classification or regression. Gradient boosting starts with a single decision tree, then iteratively adds more decision trees to overall model

→ The basic structure for XGBoost is DMatrix, which represents a data matrix. DMatrix can be constructed from numpy array.

→
```
import xgboost as xgb
dmat1 = xgb.DMatrix(data)
dmat2 = xgb.DMatrix(data, label=labels)
```
↑ numpy array.

The DMatrix object can be used to train a Booster object, which represents the gradient boosted decision tree.

```
→ dtrain = xgb.DMatrix(data, labels = labels)
params = {
        'max_depth': 0,
        'objective': 'binary:logistic'
}
bst = xgb.train(params, dtrain) # booster

print (bst.eval(deval)) #evaluation
dpred = xgb.DMatrix(new_data) #new data
predictions = bst.predict(dpred)
print (predictions)
```

Note - predictions from predict function are probabilities and <u>not</u> class labels

<u>Cross-validation</u>

We can tune the parameters using cross validation

```
→ dtrain = xgb.DMatrix(data, label = labels)
params = {
      'max_depth': 2,
      'lambda': 1.5,
      'objective': 'binary:logistic'}
cv_results = xgb.cv(params, dtrain)
print (cv_results)
```

The output of cv is a pandas dataframe
It contains the training and testing ~~s~~results
(mean & SD) of a k-fold cross validation.
The value of k is set with the nfold argument.
The keyword num_boost_round specifies the
number of boosting iterations.

→ cv_results = xgb.cv(params, dtrain,
                       num_boost_round=5)

<u>Storing Boosters</u>

→ ~~bst~~ Storing into a binary file called model.bin
  bst = xgb.train(params, dtrain)
  bst.save_model('model.bin')

→ loading a booster
  This requires us to create an empty booster
  then load the file's data into it.
    new_bst = xgb.booster()
    new_bst = load_model('model.bin')

<u>XGBoost Classifier</u>

XGBoost classifier takes NumPy arrays as
input arguments

→ model = xgb.XGBClassifier()
  model.fit(data, labels)
  predictions = model.predict(new_data)

Note:- predict function returns classes
and <u>not</u> probabilities

→ We can set classification type to XGBClassifier
Object like in the Original Booster Object.

<u>XGBoost Regressor</u>:

• XGBoost also provides a scikit-learn style
linear regression model.

```
→ model = xgb. XGB Regressor (max_depth = 2)
model. fit (data)
predictions = model. predict (data_new)
print (predictions)
```

<u>Feature importance</u>

→ Not every feature in a dataset is used equally
for helping a boosted decision tree make a
decision.

After training an XGBoost model, we can
view the relative (proportional) importance of
each dataset feature.

```
→ print (model. feature_importances_)
```

→ We can also plot this

```
xgb. plot_importance (model)
plt. show ()  #matplotlib plot
```

The resulting plot is a bar graph of the F-scores (F, -scores) for each feature.

We can ~~oo~~ pass importance_type = "gain" to xgb.plot_importance. By default it uses weight. Gain → information gain
Information gain is a commonly used metric for determining how good a feature is at differentiating the dataset.

Hyperparameter Tuning.

Apply grid search cross validation to XGBoost model

```
→ model = xgb.XGB~~oo~~B Classifier()
params = {'max_depth': range (2,5)}

from sklearn.model_selection import GridSearchCV
cv_model = GridSearchCV (model, params,
                         cv=4, iid=False)
cv_model.fit (data, labels)
print (cv_model.best_params_)
```

Note - A couple of commonly tuned params are max_depth and eta (learning rate of boosting algorithm)

Model Persistence
Same as scikit-learn using the joblib API
Use the dump and load functions.