# eXtreme Gradient Boost: A Scalable Tree Boosting System

# Introduction to XGBoost

- **What is XGBoost?**
  - An open-source **gradient boosting** library optimized for speed and performance.
- **Features and Advantages:**
  - Efficiency: Optimized internal parallel computing and tree structure.
  - Powerful Regularization: Prevents overfitting.
  - Flexibility: Supports custom loss functions and evaluation metrics.
- **Applications:**
  - Widely used in competitions, financial forecasting, classification, and regression problems.

# Core Principles of XGBoost

## Overview of Boosting Methods

- Boosting is an ensemble learning method that combines multiple weak learners (usually decision trees) to form a strong learner.
- The basic idea of boosting is to train a new weak learner to correct the errors of the previous weak learner.
- Common boosting methods include AdaBoost and Gradient Boosting.

## Gradient Boosting Decision Tree (GBDT)

- GBDT is an iterative decision tree algorithm that combines multiple weak learners (decision trees) sequentially.
- Each tree is built based on the residuals (errors) of the previous tree to minimize the loss function.
- Gradient information of the loss function is used to **guide the construction of the next tree**, hence the name gradient boosting.

# Improvements in XGBoost

- **Column and Row Subsampling:**
  - Randomly selects a subset of features and samples to increase the model's generalization ability.
- **Sparsity-aware Algorithm:**
  - Effectively handles sparse data using sparse matrix computations, improving computation speed.
- **Regularization:**
  - Adds penalty terms (like L1 and L2 regularization) to **prevent overfitting** and enhance generalization.
- **Speedup in Classification and Ranking:**
  - Introduces heuristic search methods during node splitting to speed up model training.

# Detailed Explanation of XGBoost Parameters

## General Parameters

- `booster` : Choose the type of booster ('gbtree', 'gblinear', 'dart')
- `nthread` : Set the number of parallel threads
- `verbosity` : Set the level of messages printed

## Booster Parameters

- `eta` : Learning rate, range [0,1]
- `max_depth` : Maximum depth of the tree
- `min_child_weight` : Minimum sum of instance weight (hessian) needed in a child
- `subsample` : Subsample ratio of the training instance

## Learning Objective Parameters

- `objective`: Define the learning task and corresponding learning objective (e.g., regression, classification)
- `eval_metric`: Set the evaluation metrics

## Training Parameters

- `num_boost_round`: Number of boosting iterations
- `early_stopping_rounds`: Stops training if the validation score doesn't improve

# Model Tuning and Optimization

## Grid Search Tuning

1. Set up a parameter grid
2. Create an XGBoost model
3. Perform Grid Search with cross-validation
4. Evaluate the best parameters and accuracy

## Random Search Tuning

1. Set up a parameter distribution
2. Create an XGBoost model
3. Perform Random Search with cross-validation
4. Evaluate the best parameters and accuracy

## Cross-Validation

1. Set parameters
2. Perform cross-validation with a specified number of folds
3. Evaluate results

# Implementation in our case

# dataset

## Mnist test dataset (10000, 28, 28, 1) -> (10000, 32, 32, 1)

```
(mnist_train_images, mnist_train_labels), (mnist_test_images, mnist_test_labels) = keras.datasets.mnist.load_data()
#(cifar_train_images, cifar_train_labels), (cifar_test_images, cifar_test_labels) = keras.datasets.cifar10.load_data()

def resize_batch(imgs):
    from skimage import transform
    # A function to resize a batch of MNIST images to (32, 32)
    imgs = imgs.reshape((-1, 28, 28, 1))
    resized_imgs = np.zeros((imgs.shape[0], 32, 32, 1))
    for i in range(imgs.shape[0]):
        resized_imgs[i, ..., 0] = transform.resize(imgs[i, ..., 0], (32, 32))
    return resized_imgs

mnist_train_images = resize_batch(mnist_train_images)
mnist_test_images = resize_batch(mnist_test_images)
```

# cwSaab

depth=2, energyTH=0.5, splitMode=0, cwHop1=True

```
# set args
SaabArgs = [{'num_AC_kernels':-1, 'needBias':False, 'useDC':False,'batch':None},
            {'num_AC_kernels':2, 'needBias':True, 'useDC':False, 'batch':None}]
shrinkArgs = [{'func':Shrink, 'win':2},
              {'func': Shrink, 'win':2},
              {'func': Shrink, 'win':2}]

cwsaab = cwSaab(depth=2, energyTH=0.5,
                SaabArgs=SaabArgs, shrinkArgs=shrinkArgs, concatArg=concatArg,
                splitMode=0, cwHop1=True)
output = cwsaab.fit(X)
output = cwsaab.transform(X)
```

```
output[0].shape, output[1].shape = (60000, 16, 16,4), (60000, 8, 8, 2)
```

# DFT

```
# Feature selection
features = output[1].reshape(len(X), -1)
labels = mnist_train_labels
selected, dft_loss = feature_selection(features, labels, FStype='DFT_entropy', thrs=0.8, B=16)
print("Selected features:", selected)
```

```
100%|██████████████████████████████████████████████████| 128/128 [00:13<00:00,  9.22it/s]
Selected features: [ 52  68  84  90  24  71  70  55 100  73  72  74  75  86  89  53  42  57
  76  58  83  56  22  37  59  54  82  60  93  77 118  38 102   9  92   8
  85  67 101  36  44  23  69  40  66  98 107  61  21 119 121  20 120  25
  26 103  10  45 104  87  91   7  51  41  27  28 106   6  47  39 116 109
  50  46  34  29  99 105  88  43  35 122  11 123 108  94  63 117  95  31
  30   5  19  79  78  18  62  81  97  33  80   4]
X_train.shape: (60000, 102)
```

# XGBoost

```
# Prepare data for XGBoost
X_train = mnist_train_images.reshape(len(mnist_train_images), -1)[:, selected]
y_train = mnist_train_labels
X_test = mnist_test_images.reshape(len(mnist_test_images), -1)[:, selected]
y_test = mnist_test_labels
print(f"X_train.shape: {X_train.shape}")
```

```
X_train.shape: (60000, 102)
```

```python
# Create and train the XGBClassifier
model = xgb.XGBClassifier(
    booster='gbtree',
    objective='multi:softprob',  # multi-class classification
    um_class=10,  # number of classes
    eta=0.1,  # learning rate
    max_depth=6,  # maximum depth of the trees
    eval_metric='mlogloss',  # evaluation metric
    use_label_encoder=False  # to suppress a warning
)
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print(f"MNIST Accuracy: {accuracy * 100:.2f}%")
```

```
MNIST Accuracy: 21.50%
```

# Future Work

- better understanding of three moduls (**cwSaab**, **DFT**, **XGBoost**)
- adjust the parameters to improve performance