CRAFTML
An Efficient Clustering-based Random Forest for Extreme Multi-label Learning

Motivation

- In normal classification, we have a model defined, which classifies or tags a data instance with only one class label.
- If there are multiple class labels, the classifier will choose only one(best) among those.

  Questions that arise -

- What if there are multiple possible tags (labels) associated with the data?
- Can a data instance be classified/tagged with multiple possible class labels from the set?
- How the model should be designed and how can we calculate accuracy for that model?

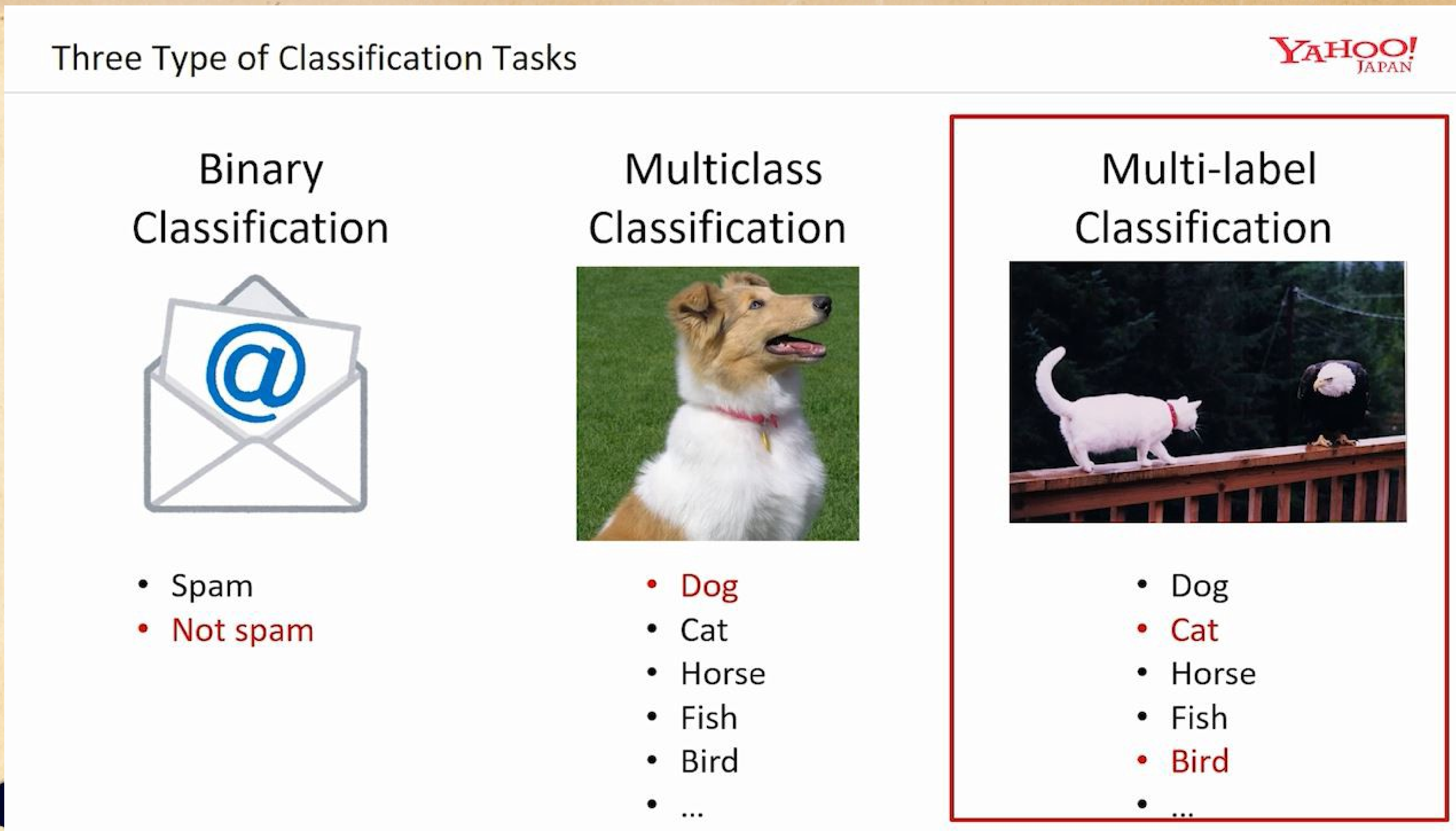
Examples

Single Label Classification : Is there a house ? Yes / No

Multi Label Classification :

House	Tree	Beach	Cloud	Mountain	Animal
Yes	Yes	no	Yes	no	no

Examples


Three Type of Classification Tasks
YAHOO! JAPAN
Binary Classification
Multiclass Classification
Multi-label Classification
• Spam
• Not spam
• Dog
• Cat
• Horse
• Fish
• Bird
• ...
• Dog
• Cat
• Horse
• Fish
• Bird
• ...

eXtreme Multi-label Learning (XML)

- eXtreme Multi-label Learning (XML) considers large sets of items described by a number of labels that can exceed one million.
- We can do this classification using many existing machine learning algorithms, but there are some disadvantages.

  Problems with existing algorithms when large number of labels are present -
- Scalability issues
- Performance degradation.
-

How to overcome these problems ?

3 common ways :

- Using optimization tricks like sparsification and parallelization.
- Reducing the data dimensionality for solving a smaller size problem.
- Tree based approach : Hierarchically partitioning the initial problem into small scale sub-problem.

Previous works

- Optimization tricks and parallelization :

  PDSparse, PPDSparse, DISMEC

- Dimensionality reduction :

  WSABIE, LEML, SLEEC, AnnexML

- Tree based approach :

  LPSR, FastXML, PFastReXML

CraftML

- CRAFTML is a random forest based algorithm with a very fast partitioning approach.
- The splitting conditions are based on all the features.
- CRAFTML randomly reduces both the feature and the label spaces to obtain diversity.
- It replaces random selections with random projections to preserve more information.

CraftML : Building a tree

- Node structure of the decision tree

```
struct Node{
    int number_of_children;
    int branch_value;
    int split_attribute;
    int leaf_value;
    struct Node *children[10];
};
```

number_of_children : number of children in each node

branch_value : make branch decision based on this value

split_attribute : splitting attribute (-1 for leaf node)

leaf_value : class value at leaf node (-1 for decision node)

CraftML : Building a tree

- Random projection of the dataset:

  We randomly project the label and feature vectors into lower dimensional spaces.

```
void chooseRandomFeatures(){
    vector<vector<double> > trainFileRandom( N , vector<double> (M, 0));
    int number_of_features = 50;
    for(int i=0; i<number_of_features; i++){
        int guess = rand() % (M-1);
        trainFileRandom[i]=train_file[guess];
    }
    train_file=trainFileRandom;
}
```

Note : In contrary to classical random forests which use bootstraps, each tree of CRAFTML is trained on the full initial dataset.

We only select (project) a subset of the feature and labels space.

CraftML : Building a tree

- K Means algorithm

  We build a k-means based partitioning of the instances into k temporary subsets from their projected labels.

```
double** k_means(){
    int minima[features]={INT_MAX};
    int maxima[features]={INT_MIN};
    int cluster[N];
    int t=20, k;
    double mean_arr[K][features];
    for(int i=0; i<K; i++){
        for(int j=0; j<features; j++){
            int num = (rand() % (maxima[j] - minima[j] + 1)) + minima[j];
            mean_arr[i][j]=num;
        }
    }

    for (int i = 0; i < t; i++) {
        for (int j = 0; j < N; j++) {
            double* dists = new double[k];
            for (int p = 0; p < k; p++) {
                dists[p] = cosine_distance( trainFile1[j], mean_arr[p], M);
            }
            cluster[j] = std::min_element(dists, dists + k) - dists;
            delete[] dists;
        }
```

CraftML : Building a tree

When to stop?

(i) the cardinality of the node’s instance subset is lower than a given threshold.

(ii) all the instances have the same features

(iii) all the instances have the same labels

```
void decision(int *h_attr, int *h_data, node *root, int h_dataSize) {
  int threshold = 10;
  // stopping conditions

  // checking whether the cardinality of the node’s instance subset lower than a given threshold
  if(h_dataSize<=threshold)
    return;

  // checking whether every instances have the same labels
  flag=1;
  for(int i=1;i<h_dataSize;i++){
    if(trainFile[h_data[i]][M-1]!=trainFile[h_data[i-1]][M-1]){
      flag=0;
      break;
    }
  }
  if(flag==1){
    root->val=trainFile[h_data[0]][M-1];
    return;
  }
```

CraftML : Predictions

- For each tree, the input instance follows a root-to-leaf path.
- The path is determined by the successive decisions of the classifier.
- The prediction is the average label vector stored in the leaf reached.
- The forest aggregates the tree predictions with the average operator.

CraftML : Algorithm

Algorithm 1 trainTree

Input: Training set with a feature matrix X and a label matrix Y.

Initialize node v

v.isLeaf $\leftarrow$testStopCondition(X, Y)

if v.isLeaf $=$ false **then**

  v.classif $\leftarrow$ trainNodeClassifier(X, Y)

  $(X_{child_i}, Y_{child_i})_{i=0,\ldots,k-1} \leftarrow$ split$(v$.classif$, X, Y)$

  for i **from** 0 **to** $k-1$ **do**

    $v.child_i \leftarrow$trainTree$(X_{child_i}, Y_{child_i})$

  end for

else

  $v.\widehat{y} \leftarrow$computeMeanLabelVector(Y)

end if

Output: node v

CraftML : Algorithm

Algorithm 2 trainNodeClassifier

Input: feature matrix (X_v) and label matrix (Y_v) of the instance set of the node v.

$X_s, Y_s \leftarrow \text{sampleRows}(X_v, Y_v, n_s)$

$X'_s \leftarrow X_s P_x$ # random feature projection

$Y'_s \leftarrow Y_s P_y$ # random label projection

$\mathbf{c} \leftarrow k\text{-means}(Y'_s, k)$ # $\mathbf{c} \in \{0, ..., k-1\}^{\min(n_v, n_s)}$

for i **from** 0 **to** $k-1$ **do**

  $(\text{classif})_{i,.} \leftarrow \text{computeCentroid}(\{(X'_s)_{j,.} | c_j = i\})$

end for

Output: Classifier classif ($\in \mathbb{R}^{k \times d'_x}$).

$\mathbf{c}$ is a vector where the j^{th} component c_j denotes the cluster index of the j^{th} instance associated to $(X'_s)_{j,.}$ and $(Y'_s)_{j,.}$.

CraftML : How to parallelize?

There are 2 parts where we can parallelize our code :

1. Buliding trees : While building the individual decision trees.

2. Predictions : While making the predictions from the different decision trees.

CraftML : Building Trees in parallel

- The trees are independant of one another.
- We keep the number of blocks equal to the number of trees.
- We build each of the trees parallely in a thread.
- The code snippet for creation of trees in 50 blocks is shown below :

```
#define NUMBER_OF_TREES 50

buildDecisionTree<<<NUMBER_OF_TREES, 1>>>(device_data, number_of_features, number_of_samples)
```

```
_global_ void buildDecisionTree(int trainData[BLOCK_SIZE][BLOCK_SIZE], int number_of_features,int number_of_samples){
    int bid = blockIdx.x;
    _shared_ int randomFeatures[50];
    // Choose Random Features
    for(int i=0; i<number_of_features; i++){
        randomFeatures[i] = rand() % number_of_features;
    }
    // Build Tree
```

CraftML : Predictions in parallel

- The trees built are independant of one another.
- We send the input instances to each of the threads which run in parallel.
- Each of the trees outputs the predicted set of labels.
- We take the majority of the individual outputs to make our predictions.

CraftML : Alternative approach

- The trees built are independant of one another
- We send the input instances to each of the threads which run in parallel.
- Each of the trees outputs the predicted set of labels.
- We take the majority of the individual outputs to make our predictions.

CraftML : Alternative approach

Advantages :

- When we have large amounts of data, like in eXtreme Multi Label Learning (XML), deciding the split attributes for each tree sequentially is tedious.
- Thus, we make the construction of each tree in parallel.

Disadvantages :

- We incur a loss in computational time by building the different trees sequentially.

Implementation

Dataset :

- We take a popular multi label dataset : Yeast dataset
- It has 1500 samples, 103 features and 14 labels.
- A snapshot of a part of the dataset is shown below :

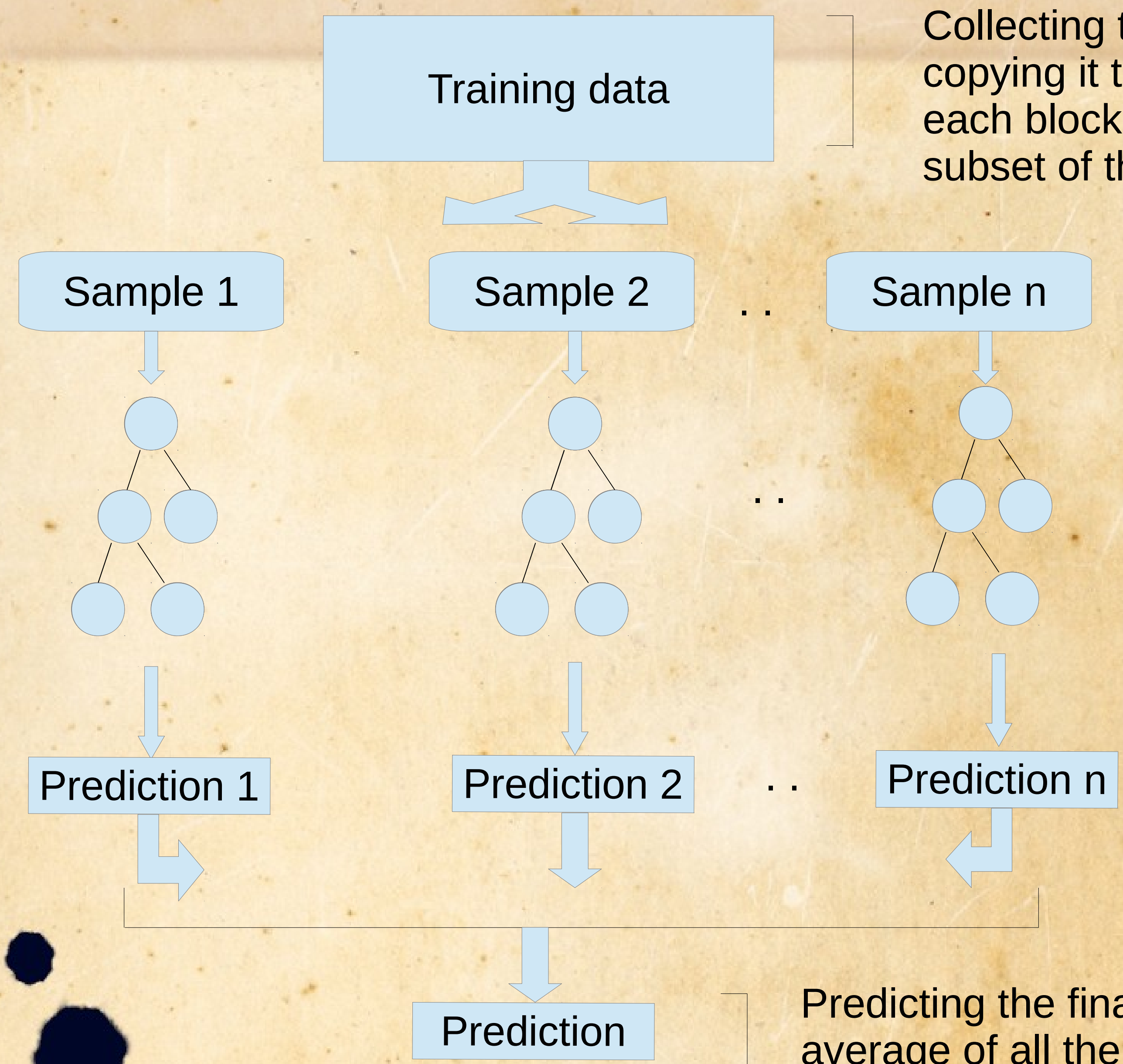
	Att1	Att2	Att3	Att4	Att5	Att6	Att7	Att8	Att9	Att10	...	Class5	Class6	Class7	Class8
0	0.093700	0.139771	0.062774	0.007698	0.083873	-0.119156	0.073305	0.005510	0.027523	0.043477	...	0	0	0	0
1	-0.022711	-0.050504	-0.035691	-0.065434	-0.084316	-0.378560	0.038212	0.085770	0.182613	-0.055544	...	0	0	1	1
2	-0.090407	0.021198	0.208712	0.102752	0.119315	0.041729	-0.021728	0.019603	-0.063853	-0.053756	...	0	0	0	0
3	-0.085235	0.009540	-0.013228	0.094063	-0.013592	-0.030719	-0.116062	-0.131674	-0.165448	-0.123053	...	0	0	0	0
4	-0.088765	-0.026743	0.002075	-0.043819	-0.005465	0.004306	-0.055865	-0.071484	-0.159025	-0.111348	...	0	0	0	0

Implementation

Steps :

- Collect the data in host memory.
- Copy data from host memory to device memory.
- Create number of blocks equal to the number of decision trees and call the kernel to build the trees.
- In each parallel block, create a projection into lower dimension space of the dataset.
- Build the trees in parallel.
- To make the predictions, take the test instance and make it traverse the root-to-leaf path of every tree.
- Take the average predictions from all the trees and output the final multi label prediction.
- Calculate the accuracy by comparing it to the ground truth labels.

Flowchart



Collecting the training data set, and copying it to device memory, where each block will randomly select a subset of the data

Constructing the decision trees. The splitting at each node is performed after performing k means clustering on the instance data

Collecting the predictions from each tree parallelely

Predicting the final set of labels, by taking average of all the predictions

Output

- We predict the labels for the different samples in the test data.
- The test data has 917 samples, with 103 features and also the ground truth values of the labels.
- We compare our predictions for each label (class) with it’s ground truth value, and calculate the accuracy of each class.

```
Class1  :  Accuracy: 0.688113
Class2  :  Accuracy: 0.571429
Class3  :  Accuracy: 0.580153
Class4  :  Accuracy: 0.640131
Class5  :  Accuracy: 0.693566
Class6  :  Accuracy: 0.761178
Class7  :  Accuracy: 0.817884
Class8  :  Accuracy: 0.791712
Class9  :  Accuracy: 0.912759
Class10  :  Accuracy: 0.899673
Class11  :  Accuracy: 0.900763
Class12  :  Accuracy: 0.750273
Class13  :  Accuracy: 0.74482
Class14  :  Accuracy: 0.985823
```

Similar calculations of accuracy of each label is tested using the Random Forest module of sklearn and shown later.

Comparing with sklearn

- We run the sklearn Random Forest module on our dataset, using train validation split.

Random forest classifier

[17]
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
clf = RandomForestClassifier(n_estimators=50, criterion='gini', max_depth=None, bootstrap=True)
```

Since number of samples is not very large, it gives us the output in comparable time.

Comparing with sklearn

- Output :

Accuracy of the classes

[18]

```
for category in y_test:
  clf.fit(X_train, y_train[category])
  y_pred = clf.predict(X_test)
  print(category, " : ", accuracy_score(y_test[category],y_pred))
```

```
Class1  :  0.7766666666666666
Class2  :  0.65
Class3  :  0.74
Class4  :  0.7166666666666667
Class5  :  0.7366666666666667
Class6  :  0.7666666666666667
Class7  :  0.85
Class8  :  0.84
Class9  :  0.9466666666666667
Class10  :  0.8933333333333333
Class11  :  0.8633333333333333
Class12  :  0.7433333333333333
Class13  :  0.7133333333333334
Class14  :  0.98
```

We see that we get comparable results.

Future scope

- Run CraftML algorithm on larger dimensional data, and more number of labels, and test the performance.
- Compute the multi label predictions parallely in each of the decision trees.
- Compare the performances and time gains with respect to sequential algorithm on multiple machines.

Thank You