CS 6240 Final Project Report

# Technology Stack:

Apache spark + Python => Distributed computing

MLLib and Scikit learn => Machine learning libraries

PySpark is very helpful in implementing the program in distributed way. The built-in functions help us to implement the MapReduce jobs easily. Also, Saving the Models as RDD and reading it again for prediction is easy in PySpark. Also, Scikit learn is the best machine learning library.

# Design Outline:

1. Data cleaning
2. Model Training
3. Prediction

### Data Cleaning overview:

We have 1657 columns in the data set. We can safely eliminate less important columns in order to improve the learning of the Machine Learning Algorithm. We represent it in sparse matrix representation which reduces the size of data that we deal with.

### Model Training Overview:

Cleaned data is replicated n times, so we get bigger RDD, now we Group the replicated data by key, and we apply ML algorithm on each RDD to get N different models based on the Key. This is saved as pickle file.

RDD 0

M1

Replicated RDD

Input File:

Training data

Cleaned Data



M2

RDD 1

M3

RDD 3

### Prediction Overview:

The cleaned data, is zipped with index for maintaining the order. This data is given to each model that we trained earlier (The models will be broadcasted after reading from pickle file). Each of these N models make some predictions. Finally, we average the prediction to get the final prediction.

P1

M0

Zipped with Index

Final Prediction

Input File:

Test data

Cleaned Data

M1

P2

M3

P3

# Data Engineering/Cleaning:

Processes involved in Data Engineering includes,

## Removing less important columns:

We are dropping the columns that are,

1. IDs -> Doesn’t have any mathematical properties so they won’t help in learning.
   1. Ex. SAMPLING\_EVENT\_ID, LOC\_ID, OBSERVER\_ID, GROUP\_ID, SUBNATIONAL2\_CODE
2. Categorical Columns -> Columns with multiple possible values.
   1. Ex. BCR, BAILEY\_ECOREGION, OMERNIK\_L3\_ECOREGION
   2. COUNTY, COUNTRY, STATE\_PROVINCE can be dropped since they are represented by Longitude, Latitude.
   3. YEAR contains only one value so its dropped as well.
3. Filtering Rows where PRIMARY\_CHECKLIST\_FLAG is not set to true.
   1. This is a way to get unique group id, we are interested only in rows with unique group ID
4. One-Hot Encoding -> Encode categorical columns with finite possible values So the ML algorithm learns better.
   1. COUNT\_TYPE: Only 20 possible protocols are present. So we encoded it. This is done by adding 20 Boolean columns Where only the particular protocol gets 1 others get 0.
   2. TIME: Column is Divided into four columns each represents 6-hour slot.
5. Manipulate certain values
   1. Longitude and latitude are converted into xyz plane. Because, WGS-84 representation doesn’t work well because of the earth’s spherical shape, So even if two places are too close, one half will gets positive value and another one gets negative value.
   2. ELEV\_GT, ELEV\_NED -> both are elevations so we have taken average and dropped it.
   3. Missing values in CAUS\_PREC, CAUS\_SNOW, CAUS\_TEMP\_MIN, CAUS\_TEMP\_MAX, CAUS\_TEMP\_AVG are replaced with respective values in extended covariates, So the columns in extended covariates are dropped.
   4. NLCD\_\* these columns are dropped.
6. Handle Missing values
   1. Any column that has ‘?’, its replaced with 0. ‘x’ in any of the bird column is replaced with a random number between (2,10) because the observer forgot the bird count. It can’t be 0.

Finally, every value is converted into float and target column in converted into binary (seen or not seen). Every step in the above process is a function. All these functions are called from a custom function. Using Spark’s map function, we apply this custom function to the record.

Also, we maintain a column name – ID mapping by removing the first row of the record and we parse it to get all the column name and we create a dictionary of Column Name – ID mapping. We don’t hardcode any value here. Repeating IDs like SAMPLING\_EVENT\_ID, LOC\_ID are maintained as a list of values. Ex. LOC\_ID = [1, 954, …]. To access a value, we just need a column name. This makes our program more generic and it works with any ordering of the columns.

Create\_header\_dictionary() //Happens only once in the program.

* header = RDD.first() //RDD.first() - built-in spark function, returns the first record of RDD
* header\_dict = {}
* Split header by Comma, and get a column list.
* For each ii from 0 to length (column list):
  + add the column name, ii to header\_dict. Update ID count
* Return header\_dict

Custom\_function(record): //we call RDD.map(x => custom\_function(x)) so it is applied to each record.

* add\_column(record) //one-hot encoding
* replace\_column(record) //CAUS features
* convert\_into\_numeric(record) //each value in record is converted into float, binary as stated above
* convert\_to\_sparse(record)

One-hot encoding takes the possible values list and for each value in the list, it adds 1/0 based on its match with corresponding value of the column ID in the record.

Add\_column(record, colID, Possible\_Values[]): //ex. add\_column(record, “TIME”, [0-6,6-12,12-20,20-24]

For value in Possible\_values[]: // iter 1: 2-8, iter 2: 8-14, iter 3: 14-20, iter 4: 20-2

* if record[colID] == value: //time is represented by values 0-24, check which slot it belongs
  + record.append(1)
* Else:
  + Record.append(0)
* Return record

Replace\_columns(record)

* If CAUS\_PREC or CAUS\_SNOW or CAUS\_TEMP\_MIN or CAUS\_TEMP\_MAX or CAUS\_TEMP\_AVG = ‘?’:
  + MM = record[“MONTH”]
  + Replace the Value of CAUS\_\* with CAUS\_\*\_MM

Convert\_to\_numeric(record):

* For val in record:
  + If val == ‘?’ : val = 0
  + If val == ‘x’ and it is one of the Bird Columns: val = rand(2,10)
  + Return float(val)
* Return record.

Drop\_column(record, drop\_list, drop\_multiple\_list):

//drop list is the list of column names that to be dropped. Ex. DAY, YEAR, COUNTY, COUNTRY, etc.

//drop\_multiple\_list is the list which wildcard match ex. NLCD\_\*, CAUS\_\*\_\*.

* For each name in drop\_list: drop(record[header\_dict(name)])
* For each name in drop\_multiple\_list: drop all records with the name matches the wild card search.

Finally, we are converting the record into sparse vector representation. Each record is represented as:

[#of total Index, Array[Index of non-zero elements], Array[corresponding non-zero]]

convert\_to\_sparse(record):

* Init Index\_array, Value\_array = [] , count = 0
* For each id in record (increment the count)

Add id to index\_array and record[id] to value\_array if record[id] != 0

* Return SparseVector(count, index\_array, value\_array) => built-in spark function

This improves the efficiency of the program. We have only around 60-70 columns with non-zero values in the 997 columns present in the preprocessed data.

**Data Distribution:** PySpark takes care of the Data Distribution. By default, the data is divided into number of partitions available. Each partition gets some portion of the input data. And, we call map function on the RDD. PySpark applies this custom function to each record in each partition in parallel.

# Model Training

Input data is read and cleaned. The cleaned RDD is replicated n times. We used flatMap on this RDD that duplicates the records and returns a list of records that are assigned keys from 1..n. Each record in the corresponding partition is replicated N times.

Rdd.flatMap( x => [(0,x), (1, x), (2, x), … (n, x)])

Now, we have a bigger RDD. This RDD is going to be grouped by Key so each group gets an RDD of whole input data. (Remember Key is 0 to N-1). Shuffling happens here.

Rdd.groupByKey()

Now, we have n groups of RDD, this RDD is given to different ML algorithm to build a model.

And these models(RDDs) are written into pickle file. This will be read in prediction program.

## Different Models

## Model Tuning

# Prediction

We read the test file and clean the data as stated early. So each partition gets its own share of input data.

We read the Models that were saved as RDD from the pickle file. These RDDs are converted into list and we broadcast the list using Spark’s broadcast function so each partition gets these models.

Sc.broadcast(model\_list)

Now, we apply Index to the input data. This is done by ZipWithIndex, where each records gets the unique index ordered ascendingly. **Shuffling** will happen here. All the records are assigned this unique IDs. The records are then put back into the same partitions with the unique Index. This index is very important for the final ordering of the data. We order these records based on the ID once we get the prediction.

cleanedRDD.zipWithUniqueIndex()

Now, we need to predict using the models. We apply map function in which we predict using the list of models and get the predictions. These predictions are aggregated into single value by taking mean. This is the final prediction. It returns record with the Unique Index, Sampling Event ID, prediction.

zippedRDD.map()

Now, we call sortByKey(). This re-orders the data based on the Unique Index. So we get the final output with Sampling Event ID and Prediction in same order as Input. We save them as final output file.

predictedRDD.sortByKey().map()

# Results