Lesson 24 – Data Science at Scale

**Questions for Mentor:**

**Hadoop ecosystem:**

* Can be used with any OS
* 4 modules
  + Distributed file system
    - Allows data to be stored in an easily accessible format
  + MapReduce
    - Reading data from DB
    - Putting it into format suitable for analysis and performing mathematical operations
  + Hadoop Common
    - Provides tools (in java) needed for users computer system to read data stored under Hadoop File system
* Developed because of need for people to be able to store and analyze datasets far larger than can be practically stored on hard drive

**Learning Hadoop:**

* Understanding relational database (RDBMS) limits
  + Scalability
    - Most projects moving into terabytes and petabytes
  + Speed
    - Real time?
  + Others
    - Queryability
* Database choices
  + File systems
    - Other fields
    - HDFS (Hadoop Distributed file systems)
  + Databases
* Hadoop is meant to work alongside DB system, not replace it
* Hadoop and HBase
  + Hadoop uses alternative file system (HDFS)
  + HBase is a NoSQL database (wide columnstore)
    - Wide column implementation
    - Width of column varies based on the information entered
* CAP theory
  + Consistency
    - Transactions
  + Availability
    - Up-Time
  + Partitioning
    - Scalability
  + CAP theory says RDBMS is only good at 2 of the above 3
  + Where Hadoop fits
    - Scalability (partitioning)
      * Designed to be run on commodity hardware for data storage
    - Flexibility (availability)
      * Commodity hardware for distributed processing
* What kind of data for Hadoop?
  + LOB data
    - Usually transactional and not a good fit
    - Should stay in RDBMS
  + Behavioral data
    - Batch processed
    - Great fit for Hadoop
    - Example: healthcare – from devices like fitbits
  + Type of data
    - Biz critical should be in RDBMS
* Changing data landscape
  + RDBMS – HADOOP – NoSQL
* What is Hadoop?
  + Two components plus projects
    - Open source data storage: HDFS
    - MapReduce
    - Projects – Hbase etc
  + Hadoop distributions
    - Open source
      * Apache Hadoop
      * Can be considered immature due to open source nature
    - Commercial
      * Cloudera
      * Hortonworks
      * MapR
      * More reliable
    - Cloud
      * AWS
      * Windows Azure HDInsight
    - Can use open source or commercial on cloud solutions
      * Not all commercial solutions work on all clouds
  + Why use Hadoop?
    - Cheaper
      * Scales to petabytes or more
    - Faster
      * Parallel data processing
    - Better
      * Suited for particular types of ‘big data’
  + Hadoop business problems
    - Risk modeling
    - Customer churn analysis
    - Recommendation engine
    - Ad targeting
    - Transactional analysis
    - Threat analysis
    - Search quality
  + Organizations using Hadoop
    - Facebook
    - Yahoo!
    - Amazon
    - eBay
    - American Airlines
  + Hadoop vs HBase
    - Hadoop
      * File system: HDFS
      * Processing API: MapReduce
      * Many people don’t want to use data in Hadoop form
    - HBase
      * Widecolumn store
      * Holds ID column and delimited data in data column
      * Could have all different types of data
        + Name, location
        + Name, car
        + Location, car, color
  + Understanding Java Virtual Machines
    - Hadooop processes run in separate JVMs
    - JVMs don’t share state
  + Hadoop File systems
    - HDFS
      * Distributed (3 copies) or pseudo-distributed (implemented on single node on single machine)
    - Regular file system
      * Standalone
    - Cloud file system
    - Files and JVMS
      * Single node
        + Local file system
        + Single JVM
      * Pseudo distributed
        + Uses HDFS
        + JVM daemons run processes
    - Different distributions for Hadoop use different libraries
  + Vendor distributions are easier to work with
    - Look at products
    - Free and premium subscriptions
      * Cloudera live allows you to try some of querying in a browser
    - Hive query language (HQL) is very similar syntax to SQL
      * Works alongside mapreduce jobs but takes a longer time
  + Cloud based Hadoop distributions
    - Virtual machine clusters
    - Optimized, partially managed distribution
      * AWS – Elastic MapReduce
      * Microsoft – HDInsight
* Using AWS and Microsoft cloud-hosted Hadoop
  + Need to understand the type of activity and what type of library you want to work with
* Setting up Hadoop Development Environment
  + Hadoop binaries or vendor distribution
  + Hadoop version
  + Location
    - Local install
      * Free but takes a long time
    - Local VM
      * Must install virtualization software
    - Cloud
      * Costs money to test
  + Setting up Hadoop Data Storage
    - Local
      * File System (single)
      * HDFS (pseudo or distributed)
    - Cloud
      * Cloud files (S3, BLOB)
      * HDFS
  + Setting up Hadoop Libraries
    - MapReduce
      * Version 1.0 or 2.0
    - Other libraries
      * i.e. Hive or Pig
    - Developer tools
  + Cloudera Hadoop Virtual Machine
    - OS + Virtualization software
    - Hadoop virtual machine for that type of VM
    - Cloudera tools and samples
    - Install developer tools
* Adding core libraries
  + Vendor solution will be included
  + With open source you need to add
    - Hive – SQL like query, create batch (MapReduce) jobs
    - Pig – ELT-like scripting language
    - Impala -SQL-like query, interactive process
* Adding other libraries
  + Mahout
    - ML algorithms
  + Spark
    - Resilient distributed data sets
  + Storm
    - Complex event processing
* MapReduce Programming Languages
  + Java (JDK)
  + Python
  + R
* Hadoop IDEs
  + Eclipse (Java)
  + Sublime (python)
  + Rstudio (R)
* Exploring Cloudera VM IDE
* What is MapReduce?
  + Programming paradigm
  + Designed to solve one problem
  + How to index or get meaning out of all info on internet
  + Two parts: Map and Reduce
    - Map
      * Execute Map() function on data
      * Execute on each node
      * Bringing compute to data instead of vice versa
      * Output <key, value> pairs on each node
    - Reduce
      * Execute Reduce() function on data
      * Only executes on some nodes of mapper
      * Aggregates key, value pairs on some nodes
      * Output is single combined list
  + Shuffle and sort phase of MapReduce
  + MapReduce 1.0
    - Distributed, scalable and cheap
    - Storage is resilient
      * HDFS triple replicated
      * Commodity hardware
    - Processing
      * Parallel via map and reduce
  + Coding Steps
    - Create a class
    - Create static (global) map class
    - Create reduce class
    - Create main job function
      * Create job
      * Job calls the map and reduce classes
  + Key aspects of MapReduce
    - It’s an API, or set of libraries
      * Job – unit of MR work/instance
      * Map task – runs on each node
      * Redeuce task – runs on some nodes
      * Source data – HDFS or other file system
    - MayReduce Daemons and Services
      * JVMs or services – isolated processes
        + Job tracker – one
        + Task trackers – one per cluster
      * Job configurations
        + Specify input/output locations for job instances
        + Job clients submit jobs for execution
    - MR coding patterns
      * Standard – usually written in java
      * Hadoop streaming – java base
        + Other language for mapper/reducer logic
      * Hadoop pipes – uses C++
  + Running first MR job
    - Word count == ‘hello world!’
    - IDEs and runners
    - Eclipse and Hadoop SDKs
    - File system or HDFS
  + MR job output
    - SUCCESS
    - Series of text files
    - Immutable if stored on HDFS
    - Each run needs new file name
  + MR job status and logs
    - Monitor job run status
    - Local websites to monitor
    - Read logs
    - Troubleshoot failed job runs
    - Error logs
* Linux shell commands
  + Can use Hadoop and shell commands
    - Syntax: hadoop fs -cat <file:///file2>
  + Cloudera has a browser – hue
    - Can browse files like you would be able to in command line
* Key components
  + Input/output (data)
  + Mapper
  + Reducer
  + Partitioner
  + Reporter
  + Output Collector
* Ways to run MR jobs
  + Configure JobConf options
  + From the development environment
  + From a GUI (Hue / HDInsight console)
* Job execution optimizations
  + Speculative execution
    - Can kill if taking too long
* Methods to write MR Jobs
  + Standard – usually written in Java
  + Streaming
  + Pipes
  + Abstraction libraries
    - Hive, Pig, etc (high-level language)
* Figure out at what level of abstraction are you comfortable using MR

**Spark and Pyspark**

* Spark
  + Distributed processing framework
  + Big data processing
  + Resilient distributed datasets (RDD)
    - Values computed only when needed
    - Keep info on calculation of each slice so we can regenerate if node fails
  + Functional programming in python
    - Function tools in python
      * Map applies function to each element of a list
      * Filter
      * Reduce
      * Itertools
      * Lambda
      * Functional tools can be combined (reduce and map to take sum of the second elements in the list of tuples)
    - Itertools.chain maps list of variables into single flat list
  + Transformations vs actions
    - Transformations
      * Returns another RDD
      * Not really performed until an action is called (lazy)
      * .map(f) – returns new RDD applying f to each element
      * .filter(f) – returns new RDD containing elements that satisfy f
      * .flatmap(f) – returns flattened list
    - Actions
      * Return a value other than an RDD
      * Performed immediately
      * .reduce(f) – returns value reducing RDD elements with f
      * .take(n) – returns n elements from RDD
      * .collect(f) – returns all elements from list
      * .sum(f) – sum numeric elements of RDD
* What is Apache spark?
  + Difference b/w spark and MR
    - RDD is used as its on a single node
    - Driver node and worker nodes
    - Spark gives you functions to use that would otherwise have to be written in MR
  + Spark used for:
    - Large scale jobs on data
    - Analysis
    - Processing data and putting into storage
  + To scale more, add more nodes to cluster
    - Don’t need to replace old hardware
  + Spark becomes chain of RDDs, keep transforming
* Text

  Description automatically generated
* Spark Linkedin Learning
  + First step is creating a Spark session
  + Now acts as single unified point of entry into spark
  + Partitions
    - Collections of rows on one server
    - Partitions are how they are split within cluster
    - Don’t usually need to manage partitions
  + Transformations
    - Modify DF
    - Doesn’t act on transformations until we perform actions
      * Lazy evaluation
* Install spark
  + !apt-get update
  + !apt-get install openjdk-8-jdk-headless -qq > /dev/null
  + !wget -q http://archive.apache.org/dist/spark/spark-2.3.1/spark-2.3.1-bin-hadoop2.7.tgz
  + !tar xf spark-2.3.1-bin-hadoop2.7.tgz
  + !pip install -q findspark
* Setup environment
  + import os
  + os.environ["JAVA\_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
  + os.environ["SPARK\_HOME"] = "/content/spark-2.3.1-bin-hadoop2.7"
  + import findspark
  + findspark.init()
  + from pyspark import SparkContext
  + sc = SparkContext.getOrCreate()
  + import pyspark
  + from pyspark.sql import SparkSession
  + spark = SparkSession.builder.getOrCreate()
  + spark
* Download csv from web
  + !wget \*url\*
* DataFrame API
  + DataFrames
    - High level API
  + Resilient Distributed Datasets (RDDs)
    - Low level API
  + Python doesn’t support Dataset API
* Spark Scheme
  + Explicitly defining the schema in spark is recommended
* Challenges with RDD
  + Spark doesn’t understand the inner structure of records
  + You will need to manually recreate any optimizations
* When to use RDDs?
  + For control over physical distribution and partitioning of data
  + When maintaining a legacy codebase using RDDs

**Spark in DataCamp**

* Can create RDD with parallelize() to create RDD from python lists
* Create from external datasets – textFile()
* Must understand partitioning in PySpark
  + Logical division of large distributed dataset
* Pair RDDs
  + RDDs in key value pairs
* Creating pair RDDs
  + From list of key-value tuple
  + From a regular RDD
    - Map function splitting items on a space or something
  + reduceByKey() transformation combines values with same key
  + sortByKey() orders pair RDD by key
* Advanced RDD actions
  + countByKey()
    - counts values by key
    - should only be used if data will be small. Stored in memory
  + collectAsMap()
    - changes key value pairs into dictionary
    - should only be used if data will be small. Stored in memory
* Abstracting data with DataFrames
  + Can be created using createDataFrame() method or using read method
  + Need to have a schema for dataframe to ensure right data types, names, whether error values can exist etc
    - Can use inferSchema=True parameter to use standard schema
* In Pyspark you can use DF API or SQL queries
  + Many DF operations can also be done with SQL queries
  + Sql() method takes SQL statement as argument and returns DF
  + Can extract data as well as grouping or summarizing data
* Data viz in pyspark
  + Pyspark\_dist\_explore library gives quick insight into dfs
    - Hist()
    - Distplot()
    - Pandas\_histogram()
  + Can convert pyspark DF into pandas DF
    - Single mission tool – size is limited by server memory
  + HandySpark
    - Package to improve PySpark UX
* PySpark MLlib
  + Scikit-learn algorithms only work for small datasets on single machine
  + Sparks MLlib algs are designed for parallel processing on a cluster
  + Only supports RDDs
  + Provides high-level API to build ML pipelines
  + 3 Cs of ML
    - Collaborative filtering (recommender engines)
    - Classification
    - Clustering
  + Collaborative filtering
    - Finding users that share common interests
    - Commonly used for recommender systems
    - User-user – finds users similar to target user
    - Item-item – finds similar items
    - Rating class
      * Wrapper around tuple (user, product and rating)
      * Syntax: r = rating(user, product, rating)
    - randomSplit() to split b/w train and test
      * returns multiple RDDs
    - Alternating least squares (ALS) provides collaborative filtering
    - predictAll() returns RDD of rating objects – predicted ratings for input user and product pair
    - evaluate using MSE (among others)
  + Classification
    - Logistic regression predicts binary response based on some variables
    - Vectors
      * Dense vector: store all entries in an array of floating point numbers
      * Sparse vectors: store only nonzero values and their indices
    - LabelledPoint
      * Wrapper for input features and predictor value
      * Label in LogReg is either 0 (negative) or 1 (positive)
    - HashingTF() is used to map feature value to indices labels in feature vector
    - LogisticRegressionWithLBFGS class
  + Clustering
    - Unsupervised method to organize collection of data into groups
    - KMeans.train() method to train with k and maxIterations paramters along with data
    - Model.clusterCenters to see centers
    - Evaluate with defined function
    - Visualize clusters showing points and centers

**Neural Networks:**

* Keras in python to implement models
* Perceptron – first neural network
  + Performs weighted sum of the inputs with bias
  + Binary classifier
* Gradient descent
  + Learn neuron’s parameters (weight and bias) from the data
  + Optimizing alg used to iterate through diff combinations of weights to find the best combination of w and b
  + Minimizing loss function
  + Learning rate is size of steps we take to get to minimal loss function
  + Use sigmoid function to smooth step function and differentiate b/w just 0 and 1
* XOR challenge
  + Can’t be separated by single line
    - Blue vs black dots example in 4 quadrants
  + Need to introduce nonlinarity into function
* Multilayer perceptron
  + Deep neural network is a NN with at least one hidden layer
* Activation functions
  + Step function – binary 0/1
  + Multiple classes
  + Can’t use linear – SOR problem
  + Sigmoid – output bw 0 and 1
    - Vanishing gradient at both ends of sigmoid
  + Tanh
    - Similar problem to sigmoid
  + ReLU (rectified linear unit)
    - 0 for x < 0
    - X for x>=0
    - Non-linear – less computational expensive than sigmoid
* Backpropogation
  + Hyperparameters considerations
    - What activation function
    - What learning rate
    - How many neurons
    - How many hidden layers

**Recurrent Neural networks (RNN) and Long Short-Term Memory (LSTM):**

* What’s for dinner?
  + Sushi
  + Waffles
  + pizza
  + use past to predict future
* vector
  + fancy word for list of numbers
  + vectors are computers native language
  + everything reduced to list of numbers
  + can use one-hot vector for whats for dinner
* neural network is connections between predictions, dinner today and dinner yesterday vectors
* LSTM – can look back 2, 3 etc time steps and use info to make good predictions to what happens next
  + Sequential patterns
    - Text
    - Speech
    - Audio
    - Video
    - Physical processes
    - Anything embedded in time

**Neural networks in Keras:**

* Model.compile(self, optimizer, loss, metrics=None)
  + Optimizer
    - SGD – stochastic gradient descent
    - RMSprop – good choice for RNNs
    - Adam – algorithm for first order gradient based optimization of stochastic objective functions
      * Optimizer of choice in examples
  + Loss functions
    - MSE
    - Categorical\_crossentropy – when target has multiple classes
    - Binary\_crossentropy

**Introduction to TensorFlow in Python:**

* Tensor is a generalization of numbers
* Constants and variables
* Matrix multiplication – must have same number of columns
* Reduce\_sum = sum over dimensions of matrix
* Gradient() operation gives us the slope at a point
  + Try and find lowest loss value
* Reshaping a grayscale image – single vector
* External data can be imported using tensorflow
  + Simpler to import using pandas and converting to numpy array
* Importing mixed type datasets
  + Create numpy array with DF column as first argument and data type as second
  + Or use tensorflow cast() function
* Loss functions
  + Used to train model
  + Measure of model fit
  + Higher value = worse fit
    - i.e. want to minimize loss function
  + MAE
    - Scales linearly with size of error
    - Low sensitivity near minimum
  + MSE
    - Strongly penalizes outliers
    - High (gradient) sensitivity near minimum
  + Huber
    - Similar to MSE near minimum
    - Similar to MAE near minimum
  + Compute loss functions with targets and predictions
* Batch Training
  + When using large quantities of data and using GPU in memory
  + Chunksize parameter in pandas
  + Full sample VS Batch training
    - Full Sample
      * One update per epoch
      * Accepts dataset w/o modification
      * Limited by memory
    - Batch Training
      * Multiple updates per epoch
      * Requires division of dataset
      * No limit on dataset size
* Dense layers
  + Input layer has features
  + Output layer has predictions
  + Each hidden layer takes inputs from previous layer, applies numerical weights to them, sums them together, then applies an activation function
  + Tensorflow allows you to write code for several dense layers in less code than if you were to write out the low level linear algebra steps
* Activation functions
  + Dense layers has 2 components: linear (MatMul) and nonlinear (activation function)
  + Activation makes sure output is based on the features dependent on each other (default should depend on bill amount related to age)
  + Softmax activation function used in output layer and should be use if there are >= 2 outputs
  + Relu should be used typically in all layers other than output layer
* Optimization
  + How to find a minimum
  + Gradient descent to choose a point and see if the slope is going down
    - Susceptible to a local minimum (grand canyon plateau example) and not finding global minimum
  + Stochastic gradient descent
    - Less likely to get stuck in local minimum
    - Advantage is simple and easy to interpret
  + Adam (adaptive moment) and RMS (root mean squared)
    - Requires 10x more implementations to get same loss
    - RMS Advantage is that it applies diff learning rates to each feature
    - Adam performs better with default parameter values
* Training a network in Tensorflow
  + For complex problems, you can use initial values from distribution
  + Kernel\_initializer parameter in Dense function
  + Overfitting w/ neural networks
    - Especially problematic with problems using too many parameters
    - Can use dropout to randomly drop out some nodes
      * Improves out of sample performance
* Defining Neural Networks with Keras
  + Sign language classification A-D
  + Sequential API
    - Add hidden layers and output nodes using model.add(keras.layers.Dense(X, activation=’XX’)
    - Doesn’t require previous layer in current layer code
  + Functional API allows you to use multiple models to predict same thing
    - Requires previous layer in code
* Training and validation in Keras
  + Load and clean data
  + Define model
  + Train and validate
    - Batch size
      * # of examples in each batch (default 32)
    - Epochs
      * # of times you train on full set of batches
    - Validation split – like cross validation with other models I’ve used
    - Can change metrics with metrics= parameter in compile
  + Evaluate
    - Should split off test set and evaluate against that to ensure you’re not overfitting
* Estimators API
  + High level submodule
  + Less flexible
  + Enforces best practices
  + Upside is faster deployment – less code to train, evaluate and deploy
  + Model specification and training
    - Define feature columns
    - Load and transform data
    - Define estimators
      * Premade or custome w/ diff architectures
    - Apply train operation
* Other TF extensions worth checking out
  + Tensorflow Hub
    - Can import pretrained models
    - Transfer learning
  + Tensorflow Probability
    - More statistical distributions for random number generation
    - Extended set of optimizers that are commonly used in statistical research