INTRODUCTION

Data Science: Data + human insight + algorithms + iteration → information → knowledge

CHALLENGE IN DATA SCIENCE

- Distributed,
- Neither clean nor structured
- Cannot be directly used

DATA STRUCTURES

- Linked data / graph
- Dynamic (sequential) data
- Tabular (relational data)

"BIG" DATA

- Too complex for a human to understand directly
- Doesn't fit into a single uniform memory space; Need to think carefully about I/O and/or communication
- Need more than brute force algorithms to analyze
- May require multiple computers to work together to process
- Possibly high dimensionality, requiring feature selection and dimensionality reduction

GOALS

Pattern detection: raw data → patterns → partial understanding

Given observation: hypothesis → experiment over sample → significance

General: data → models → predictions → actions

STEPS

integration & representation → cleaning → model & analyze → understanding + ethical obligations

DATA REPRESENTATION

(PYTHON) DATA STRUCTURES

- Scalars
- set and dictionary
- indexed items: list, vector, array, tuple
- others: JSON object, relational database table

"BIG" DATA STRUCTURE

Physical data independence – independent of memory location, in-memory data structure, whether pointers and indices are available, implicit properties like order of access

This lead to a broadly applicable relational database which:

- move from memory to disk to network
- split ("shard") across a cluster
- Allow "bulk" computation efficiently

INFORMATION EXTRACTION

UNSTRUCTURED DATA

- Words give insights
- Words as features for models
- Convert to structured data

IR vs IE

Information retrieval gives documents; Information extraction gives "facts"

IE: GIVE DATA STRUCTURE

- Data extraction
- Data cleaning
- Entity recognition
- Data transformation
- ***Standard IE****:* predefined schema
- ***Open IE*****:** entity types not known in advance

IE PIPELINE

- Extract and clean raw data form
- Tokenize
- Detect term boundaries
- Detect sentence boundaries
- Tag parts of speech (POS) (e.g. verb, noun etc.)
- Parse
- Identify named entities (e.g. Person, place, organization, gene, chemical, book, movie)
- Determine co-reference
- Extract entities and relations

BIG DATA

CHALLENGES

- Huge DataFrame or expensive processing – out of memory
- How to efficiently retrieve data
- Data that changes on the fly

Issues in different stages:

- (Possibly updatable) data that must be stored on disk
  - Disk is slow
  - Concurrency vis isolation
- Data that needs to be stored in a compute cluster

CONSTRAINTS VS FLEXIBILITY

One to one, One to many, Many to many

DataFrame / JSON -

Relational database - tables related by keys/foreign keys

DBMS

A software layer designed to manage:

- Data storage, where we can enforce certain constraints
- Data relationships
- Data durability, i.e. the data is retrievable even with crashes and hardware failures
- Data consistency in the presence of updates
- (probably) query capabilities with some sort of optimization

DATABASE

Relational database, pretty similar to DataFrame

Non-relational database (NoSQL): MongoDB, Cassandra, Hbase …

- Store keys/values, and don't have much in terms of constraint enforcement
- Don't expressly capture relationships
- Don't support transactions

QUERY OPTIMIZATION AND TRANSACTIONS

Relational data model

- Separate physical implementation from logical
- Model the data independently from how it will be used
- Describe the data minimally and mathematically
- Use standard mathematical operations over the data – these are the relational calculus (~ SQL) or relational algebra (~ sequence of DataFrame operations)

Indexing

- Indices reduce the number of rows in a table that must be examined and improve performance
- B+-tree or Hash index
- work between disk and main memory

when is an index used:

The index can be used if the *initial* columns of the index (columns a, b, and so forth) appear in WHERE clause terms as

- *column = expression*
- *column* ***IS*** *expression*
- *column* ***IN*** *(subquery)*
- *column* ***IS NULL***

The right-most column of an index that is used can also use inequality, restricting the value for that column within a range; There can be up to two inequalities that must sandwich the allowed values of the column between two extremes.

- *column >/< expression* ***…***

Optimization based on query rewriting

Transactions

Managing updates to the data, which is especially complicated in distributed / sharded environments

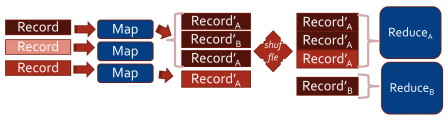
PERFORMANCE

PC – Server – Cluster

Principles to improve efficiency:

- Making computation *parallel*
- Minimizing communication and coordination

MAP-REDUCE



SHARDING VS HASHING

"Sharding" systematically distribute the data across the nodes using a key; Do work locally over the buckets, exchange (shuffle) data, go on to aggregate.

"Hashing" spread out the data into predictable locations

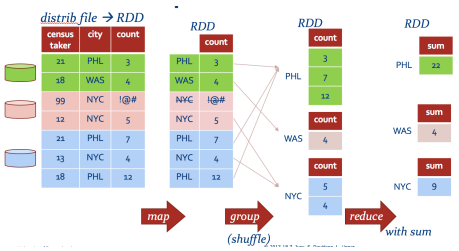
SPARK

"RDD" – resilient distributed dataset
a map from (sharded) index field→value + incorporates the MapReduce model (seldom directly used)

Map and Reduce on RDD

map($[x_1, x_2, x_3, ...]$) = $[x_{11}, x_{12}, ...]$. $[x_{21}, x_{22}, ...]$. $[x_{31}, x_{32}, ...]$

reduce (k_i, $[v_{i1}, v_{i2}, v_{i3}, ...]$) →($k_i$, v_i')



Spark's DataFrame

A distributed table with a subset of its attributes used as the index (built over an RDD)

- can't iterate over all of its elements
- accept bulk operations
- simultaneously emulate both Pandas DataFrames and SQL relations

GRAPH

TERMINOLOGY

Directed and *undirected graph*

Adjacency: u, v are adjacent if there's an edge between u and v

$degree(u)$: # of adjacent vertices

Centrality: Indegree vs outdegree

Path: sequence of adjacent vertices

Connectivity: u, v are connected if path between u and v

Connected component: set of vertices connected to each other, that is not part of a larger connected set.

Triangle: 3 vertices that are pairwise adjacent.

Clique: Any set of vertices that are all pairwise adjacent.

REPRESENTATION

Adjacency list (list of edges)

Adjacency matrix

GRAPH TRAVERSAL

n nodes m edges

Random walk $E = 2m(n-1)$ stateless, useful as a conceptual building block but not directly used

BFS use queue for centralized implementation, recall HW2 for parallel implementation

PAGERANK

$$pr^{(i)}(x) = \alpha \sum_{j \in B(x)} \frac{1}{N_j} pr^{(i-1)}(j) + \beta$$

Intuition: random walks on graphs

MATRICES

USE OF MATRIX

- Graph adjacency matrices
- Rows of readings of sets of numeric values
- Machine learning features
- Images, volumes, numeric state values in a multidimensional space

BULK OPERATIONS

- Addition and multiplication with scalars
- Simple arithmetic over arrays / matrices (add, multiply)
- Linear algebra operations: multiply, transpose, …
- "Slicing"
- Many of these are key building blocks for scientific computing

HYPOTHESIS TESTING

P-VALUES AND NULL HYPOTHESES

Null hypothesis is a general statement or default position that there is no relationship between two measured phenomena, or no association among groups

P-value the probability that you see some outcome by chance

Often p < 0.05 (or 0.01) is used

A small p-value (≤ 0.05) indicates strong evidence against the null hypothesis, while a large p-value (> 0.05) indicates weak evidence against the null hypothesis.

Parametric and non-parametric tests

PROCEDURE OF HYPOTHESIS TEST

- **Identify a *hypothesis* (and *null hypothesis*)**
  - Hypothesis: x_1 is on average bigger than x_2
  - Null hypothesis: x_1 and x_2 have same mean
- **Model any assumptions**
  - x_1 and x_2 are both Gaussian with the same variance
- **Select and compute an appropriate *test statistic***
  - t-statistic: scipy.stats.ttest_rel(a, b, axis=0, equal_var=True)
- **Assess statistically significance**
  - Find p-value for t-statistic, compare to 0.05

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_p\sqrt{\frac{2}{n}}} \qquad s_p = \sqrt{\frac{s_{X_1}^2 + s_{X_2}^2}{2}}$$


```
import numpy as np
import scipy.stats as st

m = 10000

x = np.random.randn(m) / 1000.0

# compute t statistic and p value
xbar = np.mean(x)
s2 = np.sum((x - xbar)**2)/(m-1)
std_err = np.sqrt(s2/m)
t = xbar/std_err
t_dist = st.t(m-1)
p = 2*t_dist.cdf(-np.abs(t))

print (p, t)
```

When we got n times more observations, the standard error becomes the $1/\sqrt{k}$ of the original one

Central limit theorem: regardless of the population's distribution, larger sample distributions' means will converge towards the population mean

TEST METHODS

Parametric method

E.g. T-test Under Gaussian assumption, a method to compare the mean of two groups of samples, to evaluate whether the means of the two sets of data are statistically significantly different from each other

Different t-test:

- Null hypothesis is known mean (e.g. 0)
- A/B testing vs. a control group
- Paired t-test

Non-parametric method

E.g. Wilcoxon–Mann–Whitney Test the null hypothesis that it is equally likely that a randomly selected value from one sample will be less than or greater than a randomly selected value from a second sample

Type of Test:	Use:
Correlational	These tests look for an association between variables
Pearson correlation	Tests for the strength of the association between two continuous variables
Spearman correlation	Tests for the strength of the association between two ordinal variables (does not rely on the assumption of normal distributed data)
Chi-square	Tests for the strength of the association between two categorical variables
Comparison of Means:	***look for the difference between the means of variables***
Paired T-test	Tests for difference between two related variables
Independent T-test	Tests for difference between two independent variables
ANOVA	Tests the difference between group means after any other variance in the outcome variable is accounted for
Regression:	***assess if change in one variable predicts change in another variable***
Simple regression	Tests how change in the predictor variable predicts the level of change in the outcome variable
Multiple regression	Tests how change in the combination of two or more predictor variables predict the level of change in the outcome variable
Non-parametric:	***are used when the data does not meet assumptions required for parametric tests***
Wilcoxon rank-sum test	Tests for difference between two independent variables - takes into account magnitude and direction of difference
Wilcoxon sign-rank test	Tests for difference between two related variables - takes into account magnitude and direction of difference
Sign test	Tests if two related variables are different – ignores magnitude of change, only takes into account direction

TYPES OF ERROR

false discovery rate FDR: E[FP/(FP+TP)]

Bonferroni: a p-value of 0.05/n

Simes:

Find p-values for all n hypotheses

Sort them from smallest (p_1) to largest (p_n)

For i=1:n

If p_i < 0.05 i /n

accept hypothesis

Else halt

- Goal: reject the null hypothesis
  Failure to reject could be due no difference or due to small sample size
- If distribution is known: use appropriate test statistic
  Otherwise, use nonparametric test or permutation test
- Adjust p-values to control False Discovery Rate

CORRELATION VS CAUSALITY

It could happen that there are relationship but correlation is 0

Correlation is not causality

ALGORITHMIC ANALYSIS

CACHING & MEMORIZATION

Caching e.g. spark.cache()

- A cache is a key/value store with a maximum **capacity**
  It lets us **trade off space for computation time**
- Items may need to be **evicted** to stay within capacity bounds (LFU, LRU)
- We can place a cache on disk, in memory, on the network, …
- A cache is "lazy" – nothing is added until it's asked for
- It's sometimes called "soft state" – nothing is vital to correctness, but it can speed things up

Dynamic Programming

- Requires "principle of optimality" – optimal solution can be efficiently constructed from optimal results of sub-problems
- Longest common substring, 0-1 knapsack, word break across pages, query optimization in Spark, …

An example: string similarity and Levenshtein Distance

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j)+1 \\ \mathrm{lev}_{a,b}(i,j-1)+1 \\ \mathrm{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

CONTEX SWITCHING

Problem Formulation: n Tasks to Do "Concurrently", each has m steps, when n is larger than # of CPU cores

- Use queues e.g. queue.Queue()
  "round-robin" scheduling
  Requires us to break down the computation into steps
  "data" and "op"
- Threads – an automatic approach
  e.g. threading.Thread
  can't share data unless very careful

Clusters

Spark roughly does central coordination and context switching, tasks are re-added to the queue if they don't finish within a timeout

SEARCH

Exhaustive, random, pruning, memorization…

Forward vs Backward Chaining

F: start states → programs → goal state

B: goal state → possible ways of building towards goal → possible ways of building towards those

"branch and bound" pruning

heuristic, never prune viable path but may still be low efficient

GENETIC ALGORITHM

randomization + parallelism

Most useful if we don't understand the structure of our problem, and there are many "good enough" solutions

- Selection
- Crossover
- Mutation

e.g. DEAP toolkit for Python

- Effectively, parallel randomized search across a space of parameters or a space of potential "plans": search + memoization + context switching
  - Each individual in the population: a point in the search space
  - Key ideas – (1) score items by fitness, prune bad ones; (2) mutate and crossover
- Randomized search! **may** find a decent solution in a huge space
  - Risk: **no guarantee of convergence on a solution!**
- GAs are most useful when we don't understand the topology of the space, so we can't optimize for it. Otherwise we are better off with a solution that exploits the "structure of the problem" and probably guarantees something about quality!

UNSUPERVISED LEARNING

PCA

- reduce dimensionality of data
- linear transformation
- For visualization, increase computational speed and data noise reduction

Core: Express a vector x in terms of coefficients on an (orthogonal) basis vector (eigenvectors)

$$x_i = \Sigma_k \, t_{ik} \, w_k$$

The goal is to minimizing distortion $\|x_i - \Sigma_k \, t_{ik} \, w_k\|^2$ or to maximize the data variance in the new coordinates

NOT *scale invariant*

In Python+Pandas+Numpy+Scikit:

- Use Scikit StandardScalar for standardization
- Use np.linalg.eig to decompose covariance to get eigenvectors
- But in practice, DO NOT use eigenvectors, use SVD instead: $X_{centered} = USV^T$, take k rows of V^T with largest eigenvalues = loadings
- Select top k eigenvectors and construct a projection matrix W
- Transform X by getting the projection on W
- Or we can use Scikit PCA
- For non-linear feature, consider transform them ahead (e.g. log), using kernels, or use other nonlinear model such as auto-encoder

K-MEANS CLUSTERING

Scaling out K-means to data sitting on disk (using SQL) / Apache Spark:

```
INSERT INTO km_clusters (lat, lng, iter)
  SELECT lat, lng, 0 FROM km_data LIMIT v_K;
REPEAT -- assign clusters to data points
  UPDATE km_data d SET cluster_id = (SELECT id FROM km_clusters c
  ORDER BY POW(d.lat-c.lat,2)+POW(d.lng-c.lng,2) ASC LIMIT 1);
  -- calculate new cluster center
  UPDATE km_clusters C, (SELECT cluster_id, AVG(lat) AS lat, AVG(lng)
  AS lng FROM km_data GROUP BY cluster_id) D SET C.lat=D.lat,
  C.lng=D.lng WHERE C.id=D.cluster_id;
UNTIL ROW_COUNT() = 0 END REPEAT;
```

- MLib

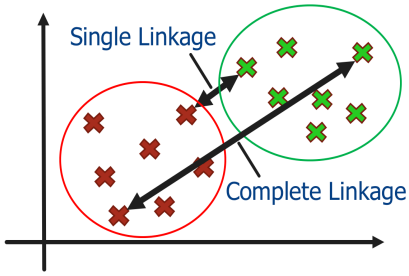
Better initialization: Repeated random initialization; K-means++

Determining number of clusters: the elbow method

HIERARCHICAL CLUSTERING

Agglomerative (AHC/HAC) (merge) and Divisive (split)

Distance between clusters: single linkage, complete linkage

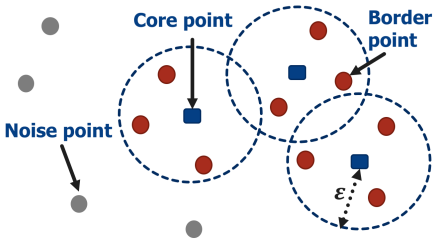



```
from scipy.cluster.hierarchy import linkage
row_clusters = linkage(pdist(df, metric = 'euclidean'),
                       method = 'complete')
```

- Don't need to specify the number of clusters upfront
- Doesn't scale well to big problems

DBSCAN

- a density-based clustering technique



DBSCAN ALGORITHM

• Form a separate cluster for each core point or connected set of core points (connected cores are those that are no farther away than ε)
• Assign each border point to the cluster of its corresponding core point

HASHING

Local sensitive hashing: Map points that are "similar" to the same bucket, then for clustering one can quickly find candidate pairs of points

MinHash: Hashing for Jaccard Distance

- Map (hash) each element of A and B to an integer
- $P(h_{min}(A)=h_{min}(B)) = J(A,B) = |A \cap B| / |A \cup B|$
- Repeat multiple times for a better estimate

CLUSTERING SUMMARY

Method name	K-Means	Agglomerative clustering	DBSCAN
Parameters	number of clusters	number of clusters, linkage type, distance	Eps, MinPts
Scalability	Very large # samples, medium # clusters	Medium # samples And # clusters	Very large # samples, medium # clusters
Use case	General-purpose, even cluster size, flat geometry, not too many clusters	Many clusters, possibly connectivity constraints, non Euclidean distances	Non-flat geometry, uneven cluster sizes
Geometry (metric used)	Distances between points	Distances between points	Distances between nearest points

Complexity

- **K-means** — n points, d dimensions, k clusters
  - d n k (# iterations) (# restarts)
- **Agglomerative**
  - > d n^2
- **DBSCAN**
  - Depends on the region size
  - But suffers from the curse of dimensionality

SUPERVISED LEARNING

DECISION TREES

Entropy $H(X) = -\sum p_i \log_2 p_i$

Conditional Entropy

$H(Y|X) = \sum_i P(X = x_i) H(Y|X = x_i)$

Information Gain

IG(Y|X) = H(Y) - H(Y|X)

Idea of building decision tree: split on the predicate with "maximum information gain":

$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$

where $I(D_x)$ is the impurity measure

e.g.

Entropy $I_H(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$

Gini impurity $I_G(t) = 1 - \sum_{i=1}^{c} p(i|t)^2$

Avoid overfitting

- Perform PCA to reduce dimensions If features are highly correlated
- Balance the positive and negative examples
- Limit maximum depth
- Keep minimum number of samples for a split from being too low
- Prune
- Use random forest or boosting(?)

Random Forest

1. Bagging: draw a random **bootstrap** data sample of size *n*, with replacement
2. Build a decision tree
   At each node, randomly select *d* candidate features w/o replacement
   Split the node using the feature with best split according to objective function (e.g. info gain)
3. Repeat to produce *k* decision trees
4. For prediction, use **majority vote** to predict a class for new data

Both decision tree and random forest are **scale invariant**

LINEAR REGRESSION

Loss function: RMSE/MSE

Naïve sol: $\hat{w} = (\boldsymbol{X}^T \cdot \boldsymbol{X})^{-1} \cdot \boldsymbol{X}^T \cdot \boldsymbol{y}$

Time cost: $O(np^2) + O(n^3)$

Works fine if # of features <100,000

If we have more features, use SVD+ gradient descent

Scale invariant without regularization

Regularization: avoid overfitting, feature selection

Ridge regression (L2)

Elastic Net (L1)

Principal Component Regression

PCA + linear regression

LOGISTIC REGRESSION

Also, ***Scale invariant*** without regularization

Loss function: log-loss

$-\frac{1}{N}\sum_i (y^i \log p^i + (1 - y^i) \log(1 - p^i))$

plus regularization term (e.g. $\lambda w^T w$)

BOOSTING

Focus on training samples that are misclassified

Scale invariant

SVM

Goal: maximize the margin

NOT scale invariant

Soft margin SVM Trading margin width against violations – tuning the big C (large C: smaller margin less violations; small C: larger margin more violations)

KERNELS

Idea: map to a higher-dimensional feature space

K(x1, x2) – reflex the "similarity" of two samples

Kernels: linear $K(w, x) = w^T \cdot x$,

polynomial $K(w, x) = (\gamma w^T \cdot x + r)^d$,

radial basis $K(w, x) = e^{-\gamma ||w-x||^2}$

CLASSIFIER EVALUATION

Prob: overfitting, computational cost, bias

Normalization

e.g. sklearn.preprocessing MinMaxScalar (.fit(), .fit_transform)

Regularization

e.g. StandardScalar (…)

Missing data

- Drop
- Fill in average if missing is random e.g. Imputer
- Treat as a categorical feature if not random

Categorical feature

To interger: e.g. LabelEncoder

To one-hot: e.g. OneHotEncoder

Model Complexity

e.g. maximum depth for decision tree; number of features, penalty weight for regression; number of terms in the series for boosing

Train, Validation, Test

e.g. sklearn.cross_validation train_test_split

To evaluate a hypothesis with respect to a population, we need independent sample.

Cross validation: k-fold, LOOCV

e.g. cross_val_score

How to evaluate a model?

- Learning curve (t vs train/test acc.)
  e.g. sklearn.model_selection learning_curve
- Validation curve (C vs performance)
  e.g. validation_curve
- Confusion matrix
  e.g. sklearn.metrics confusion_matrix
- Performance measurements
  e.g. precision_score, recall_score, f1_score

$$Error = \frac{FP+FN}{FP+FN+TP+TN},\ Accuracy = 1 - Error$$

$$False\ Pos.Rate = \frac{FP}{FP+TN},\ True\ Pos.Rate = \frac{TP}{FN+TP}$$

$$Precision = \frac{TP}{TP+FP},\ Recall = \frac{TP}{FN+TP},\ F1 = 2\frac{Precision \cdot Recall}{Precision+Recall}$$

- ROC curve (precision vs recall)
  area under the curve (AUC) (random guessing 0.5, optimal 1)
  e.g. roc_curve, auc

GRADIENT DESCENT

$\boldsymbol{a} := \boldsymbol{a} + \eta \nabla f_{loss}(\boldsymbol{a})$

Compute gradient: numerically or analytically

Selecting η: adaptive step size

E.g. each time increase step size by 10%, if error ever decreases cut it in half

Local minima: use simulated annealing, or randomness

Stochastic gradient descent

update the model after observing each single observation or a mini-batch

NEURAL NETWORKS

Activation function

Sigmoid: $1/(1 + e^{-z})$

Heavyside: 1 if z > 0 -1 if z < 0

Tanh: $(1 - e^{-2z})/(1 + e^{-2z})$

ReLU: max(z, 0)

Perceptron

No hidden layer, learn linear decision boundaries, guaranteed to converge if data is linear separable

$w_{i,j}^{next\ step} = w_{i,j} + \eta(y - \hat{y}_j)x_i$

Multi-Layer Networks

Forward propagation

Back propagation

Convolutional Neural Networks

Regularization: norm, early stopping, data augmentation, dropout

Gradient descent:

- Gradient descent
  - stochastic
  - gradient clipping
- Minibatch

- Momentum
- Learning rate adaptation

Learning rate:

- Adjust over time
- Adagrad: depend on previous changes in *each* weight

TIME VARYING DATA

VISUALIZATION

- Plot (group and plot)
- Histogram
- Boxplot
- Heat map

MODELING TIME SERIES DATA

Moving average/mean (rolling average/mean)

- creating series of averages of different subsets of the full data set.
- finite impulse response filter.
- commonly used with time series data to smooth out short-term fluctuations and highlight longer-term trends or cycles.
- a type of convolution and a low-pass filter
- smoothing the data.

Prediction vs translation

Prediction: temperature, stock…

Translation: text, speech, video…

STATIONARITY

Do not have trend or seasonal effects

Check stationarity

visualization

summary statistics (histogram; split, take mean and variance; Box-and-Whiskers)

statistical test: Augmented Dickey-Fuller (ADF)

- null hypothesis: non-stationary
- If the Test Statistic is less than the Critical Value (e.g. 1, 5, 10%), we can reject the null hypothesis
- The more negative the Test Statistic is, the stronger the rejection of the hypothesis.

e.g. statesmodels.tsa.stattools: adfuller

Make data stationary

- Estimate and eliminate trend: transformation (e.g. log), aggregation, smoothing (esp. weighted moving average), polynomial fitting
- Remove trend and seasonality: differencing, decomposition
- De-seasonalizing: compute average over each period e.g. month of the year, divide by the this average, fit a model and scaled back

PREDICTION

Auto-Regressive model AR(p)

$$x(t) = c_t + c_{t-1}1\, x_{t-1} + \cdots + c_{t-p}\, x_{t-p} + \epsilon_t$$

Moving Average model MA(q)

$$x(t) = c_t + c_{t-1}1\, e_{t-1} + \cdots + c_{t-p}\, e_{t-p} + \epsilon_t$$

where $e(t)$ is the error of prediction at t

Differencing

$$y(t) = x(t) - x(t-1)$$

fit model on $y(t)$ (more stationary)

ARIMA (AR+MA+Differencing)

p for AR, q for MR, d for diff

Determine p and q: plot autocorrelation functions and partial autocorrelation functions and see when they cross an upper confidence interval

RNN

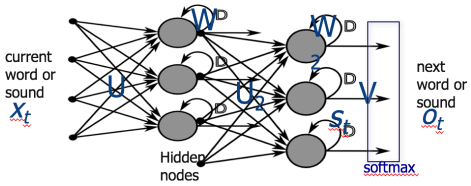
$p(x_{t+1}|x_t, x_{t-1}, x_{t-2}, \ldots) = f(h(x_t, h_{t-1}))$

Generalize HMMs or Linear Dynamical Systems, nonlinear

Take input of varying length

$s_t = \tanh(Ux_t + Ws_{t-1})$ Can use multiple layers

$o_t = \mathrm{softmax}(Vs_t)$



HMM

Training: estimate transition and emission matrix using EM (or spectral methods)

History is forgotten with an exponential decay

LSTM

Use gated RNN to prevent from forgetting too fast: store hidden state

LSTM is a kind of gated RNN, all of these gates have weights which are sigmoidal functions of weighted inputs

STREAMING

ONLINE LEARNING

Why?

- Batch learning can be expensive for big datasets
- Online methods are easy in a map-reduce environment

Least mean squares (LMS)

Online regression (L2), solve by gradient descent

PERCEPTRON

Guaranteed to converged if data is linearly separable, elsewise will bounce

Voted Perceptron

keep track of all the intermediate models you created, take the majority vote

better generalization performance, can use kernels, slower

Averaged Perceptron

Average every model, good performance and almost as fast as regular perceptron

Streaming K-Means

Use the new centers to repeat the procedure on the next batch

Guaranteed to converge if the pattern is not dynamic; For dynamic patterns, add a parameter to balance the importance between new data and history, eliminate dying clusters

STREAMING

Streaming: handling data as it arrives

- Views: computation over time windows
- Stream processing systems
- Size of memory is crutial

Data Streams: prefixes of infinite DataFrames

```
stream_df[stream_df['TS'] <= datetime.now()]
```

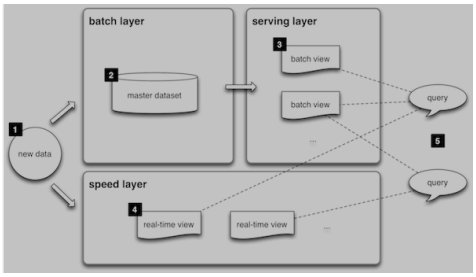
Stream (Continuous) Queries

Views: the result of stream queries (which is also a stream)

Rolling windowed operations can be used for goals such as finding trend in data

The Lambda Architecture

- put static data and streams together



Real Platforms: Apache Spark Streaming(SQL-like) Apache Storm (lower level)

STREAM PROCESSING SYSTEM

- Ensures that messages sent along the streams **do get delivered**
- May ensure **in order**
- Processes data as it arrives
- Sometimes shard the data
- Recover from crashes, failures

WINDOW OVER STREAMS

- Tumbling (non-overlapping)
- Sliding (can overlap)
- Sliding + Partition (group by into different windows)
- Partition by time, or by number of rows
- **Punctuation: determining event for opening or closing a window**

Aggregate over windows

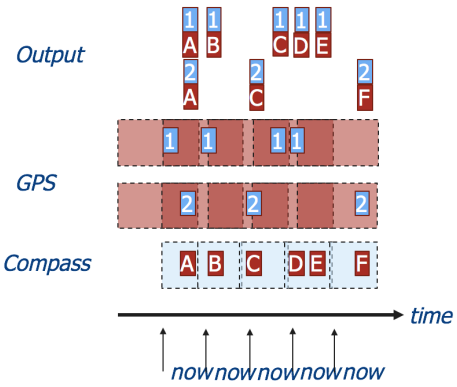
```
SELECT STREAM ticker, sum(amount) OVER lastHour,
 count(*) OVER lastHour, sum(amount) OVER lastThree
FROM Trades WINDOW
 lastHour AS (RANGE INTERVAL '1' HOUR PRECEDING),
 lastThree AS (ROWS 3 PRECEDING),
 lastZeroRows AS (ROWS CURRENT ROW),
 lastZeroSeconds AS (RANGE CURRENT
 lastTwoSameTicker AS (PARTITION BY ticker ROWS 2
PRECEDING),
 lastHourSameTicker AS (PARTITION BY ticker RANGE
INTERVAL '1' HOUR PRECEDING)
```

In some other diale
LAST 3 ROWS

Joints across windows

- Joint "parallel" windows

- Output a result for each pair that satisfies the condition
- Inherits the later timestamp of the two inputs



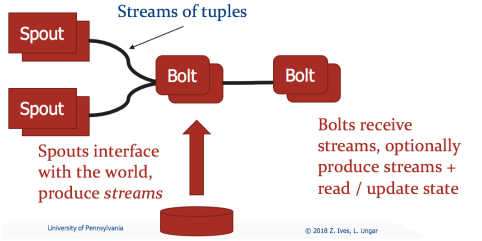
Summary

- Streams are conceptually infinite, but typically look "locally" based on current time
  Stream processing typically goes directly over the data – no storage
  LAST n ELEMENTS, LAST n SECONDS
  Tumbling vs sliding windows
  PARTITIONing windows
- Aggregates are across elements in a (partitioned) window
- Joins are across pairs of windows

SPARK STREAMING

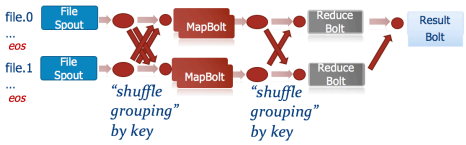
- Based on "micro batching" – periodic invocations of Spark Engine on batches of tuples
- To process:
  - **StreamingContext** gets **start**ed
  - **awaitTermination**() is run
- Can process whatever is in the Dstream as a DataFrame
- Can run countByWindow() etc. to get time-based or tuple-based windows

STORM



Storm Promises Robust Execution

- Guaranteed **at least once** processing of every data item; With **Trident** layer above, can guarantee **exactly once**
- Fail-over if a bolt or a spout dies or "master" node dies



Summary

- Creates a multi-node, multi-worker topology with **reliable delivery**
  - Spouts create data
  - Bolts process data
  - Each can access a database tec.
- Time windows can be created internally, or we can use Windowed Bolts
- Lower level, more distributed stream engine than Spark Streaming

APPROXIMATION

Handling Unlimited Data with Limited Resources

- Takes sub-linear space (vs the input size)
- We can produce **bounds** on how accurate we will be (with high probability)!

e.g. Set-Intersection Problem
Hashing → Bloom Filter
Probability of false positive

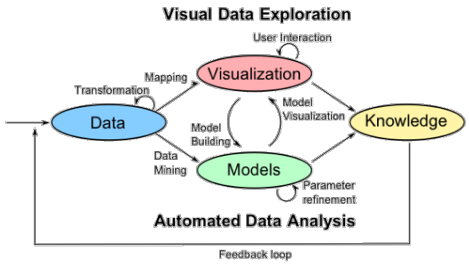
$$\left(1 - e^{-kn/m}\right)^k$$

Optimal parameter:

$$k = \frac{m}{n} \ln 2, \quad m = -\frac{n \ln p}{(\ln 2)^2}$$

VISUALIZATION

Visual analytics is "the science of analytical reasoning supported by ***interactive*** visual interfaces."



Grammar of graphics

- layers
  data; aesthetic mappings; geometric objects; statistical transformations; position adjustments
- scales
- coordinate system
- facets (groups)

VIS TOOLS IN PYTHON

- Matplotlib + Pyplot
- ggplot2
- Seaborn (for statistics)
- Builds upon matplotlib with a focus on statistical plots
  - Confidence intervals
  - Attribute-to-attribute correlations
  - Heatmaps
  - Box-and-whiskers and variants (e.g., violin plots)
- Lightning visualization server + D3.js (for interactive)

Lightning includes a library of visualization types

- Based on Javascript code from D3.js
- Options for adding other code

Lightning can

- Create the visualization using a Python data structure
- Update the data
- Append the data (potentially streaming)

Lightning can return the selected items from JavaScript → Python

- Allows the programmer to operate on what was selected

ETHICS

ETHICAL PRINCIPLES

Autonomy The right to control your data, possibly via *surrogates*
Informed Consent You should explicitly approve use of your data based on understanding

- must understand what is being done
- must voluntarily consent to the experiment
- must have the right to withdraw consent at any time

Beneficence People using your data should do it for your benefit
Non-maleficence Do no harm

ETHICS SURROUNDING DATA

Data Collection
Data and Informed Consent
Intellectual Property
Privacy
Anonymity; Differential privacy

ALGORITHM ETHICS

Biased algorithms
Training data isn't representative
Past population is not representative of the future population
Correlated attributes; Misleading results; P-hacking
Reproducibility
Fairness

SUMMARY

Codes of conduct for research are fairly well understood

- Get IRB approval
- obtain informed consent
- protect the privacy of subjects
- maintain the confidentiality of data collected
- minimize harm

Fairness is more subtle

- What is fair treatment of a group: equal accuracy? FP rate?