CS 194 - Midterm Notes

Lecture 1: Introduction

Data Science: Hacking Skill + Math & Statistics + Expertise
Research: Math & Statistics + Expertise
Machine Learning: Hacking Skill + Math & Statistics

Database Engineer
- Experience with No-SQL systems: SQL systems, i.e. column-family stores, text and XML DBs,
- Data cleaning: resolving entities, fuzzy matching, bounds checking etc.
- Schema-on-read (or never) – late-commitment semantics of data
- Data validation, using checksums, range checks statistical checks.

Scientific Computing Expert
- We were looking for the recent trend in scientific computing toward integration of machine learning as an adjunct or alternative to precise physical modeling (examples in lecture and one of the project topics). So skill in modeling with ML is important.
- Another answer was skill in using cluster tools (Spark, Hadoop, GraphLab etc.).

Machine Learning Expert
- Appropriateness of particular algorithms for various types of data
- Also how to featurize data, BOW, n-grams, skip-grams, dependency parsing.
- Knowledge of ML toolkits: tuning for performance and accuracy

Jeff Hammerbacher's Model
1. Identify problem
2. Instrument data sources
3. Collect data
4. Prepare data
5. Build model
6. Evaluate model
7. Communicate results

What's hard about Data Science?
1. Overcoming assumptions
2. Making ad-hoc explanations of data patterns
3. Overgeneralizing
4. Communication
5. Not verifying enough (model validation, data pipeline integrity, etc.)

Lecture 2: Data Preparation

ETL: Extract, Transform, Load
1. Extract data from the **source**
2. Load data into the **sink**
3. Transform data at source into a staging area

Some examples of sinks are: Python, R, SQLite, NoSQL stores
Source can be anything, from structured to unstructured data

Schema

Schema specify the structure and types of a data repository, and specify constraints within or between data fields. They are traditionally **schema-on-write**, and you can't write data into a table without a schema.

However, new data stores are **schema-on-read** or schemaless.

```
CREATE SCHEMA Sprockets
CREATE TABLE NineProngs (source int, cost int, partnumber int) GO
INSERT INTO NineProngs (source, cost, partnumber) VALUES (5, 100, 45312453)
```

XML
- Generalizes HTML and helps to encode data, e.g. DOM.
- Data can be represented and stored without schema
- More verbose (not true after compression or in DB)
- Standard query/transformation languages like XSLT and Xquery
- Use Mark Logic Server as tooling
- Example of defining an XML Schema to the right.

XML Schema

```
<location>
  <latitude>37.78333</latitude>
  <longitude>122.4167</longitude>
</location>
```

An XML schema for this element:

```
...
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
<xsd:complexType name="location">
  <xsd:sequence>
    <xsd:element name="latitude" type="xsd:decimal"/>
    <xsd:element name="longitude" type="xsd:decimal"/>
  </xsd:sequence>
</xsd:complexType name="location">

...
```

JSON
- Schemaless data description language.
- Typically used to represent hierarchical data structures. Procedural in the target language.
- More succinct in ASCII form.
- Transformation/ingestion rely on code.
- Use MongoDB as modeling

Tabular Data
- Table is a collection of rows and columns
- Each row has an index, and each column has a name
- Cell is specified by an (index, name) pair
- Cell may or may not have a value
- Schema = minimally column types
- Can be stored as CSV or TSV

Syslog – A standard for System Messages (Splunking)
- Grab data from many machines
- Check for unusual events: disk problems, network congestion, security attacks
- Monitor resources: network, memory usage, disk use, latency, threadss
- Dashboard for cloud administration

Event-Driven Parsing: SAX
Finds all the open-close tag events in XML documents and does callbacks to user code.
- User code can respond to subset of events it's interested in
- User code can correctly compute aggregates from the data instead of implementing for each tag
- User code has to implement statement to keep track of where it is in the DOM tree
- User code can implement recover strategies for ill-formed XML
- Jackson and JSON-simple work the same for JSON.

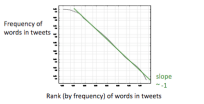
Web Services
A web service is a software system designed to support interoperable machine to machine interaction over a network. There are two kinds of interactions: XML-based (SOAP), or RESTful interactions.

SOAP: Simple Object Access Protocol
SOAP RPC messages typically encode arguments that are presented to calling program as parameters and return values. This is longer because it requires a request-response cycle and often longer conversations. Both endpoints need to agree on the stata. Design emphasis is on verbs, e.g. getUser(), addUser().

REST: Representation State Transfer
Stateless Client/Server Protocol. Each message in the protocol contains all the information needed by the receiver to understand or process it. Keeps things simple, with a set of well-defined list of operations (POST, GET, ...). Resources are typically stored in a structured data format that supports hypermedia links, such as XHTML or XML. Design emphasis is on nouns, e.g. user location.

Data Cleaning
OpenRefine is a spreadsheet-like tool checking data quality and reformats, substitutes, constraint checks, etc.

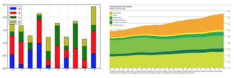
Data Exploration
Histograms can show:
- Skewed distributions, long-tailed data
  - To deal with long-tailed data, you can sort histogram counts by descending magnitude, and plot count vs bucket number on a log-log plot.



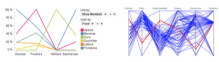
- Characteristic of social-influence processes
- Also called preferential attachment models.
- Multimodal data
  - Arise from gender/political views, other binary factors
  - Explore further by using a color and histogram of multiple populations
- Weird data
  - Compare with previous pictures of OK data

Scatter plots can quickly expose relationships between two variables.

For multiple variables, we can use:
1. Stacked Plots, where stack variable is discrete.



2. Parallel Coordinate Plot: One discrete variable, and an arbitrary number of other variables.



3. Radar Chart: One discrete variable with an arbitrary number of other variables.

Principal Component Analysis
Allows visualization of high-dimensional continuous data in 2D using principal components, and will from the orthogonal dimensions in the dataset. Great for Dimensionality Reduction.

Conclusion
Beware of Martian Canals (false conclusions) - expect the unexpected!

Lecture 3: Manipulating Tabular Data

Structured Data
Data model is a collection of concepts for describing data.
Schema is a description of a particular collection of data.

Relational database is a set of relations (true for certain arguments).
A relation is made of 2 parts:
- Schema: specifies name of relation, plus name and type of each column
- Instance: the actual data at a given time.

SQL is a Data Definition Language and Data Manipulation Language.

SQL Calls

Creating Tables
`CREATE TABLE Students (sid CHAR(20), name CHAR(20), login CHAR(10), age INTEGER, gpa FLOAT)`

Adding and Deleting Tuples
`INSERT INTO Students (sid, name, login, age, gpa) VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2)`

Basic SQL Query
Select *target-list* from *relation-list* where qualification

Joins and Inferences
`SELECT S.name, M.mortality FROM Students S, Mortality M WHERE S.Race=M.Race`

Inner Joins
`SELECT S.name, E.classid FROM Students S (INNER) JOIN Enrolled E ON S.sid=E.sid`

For unmatched keys on both sides, they are not input into the final table.

Left Outer Join and Right Outer Join – the unmatched on the left and right respectively get put into the final table. Full Outer Join – unmatched on both sides get put into the final table.

Left Semi Join – only return the rows on the left if there's a match on both tables.

Cross Join – combines all possibilities from both tables

Theta Join – allows for arbitrary comparison relationships.

Conceptual strategy when evaluating SQL calls:
- Do FROM clause (cross-product), WHERE clause (conditions), SELECT clause (projection), DISTINCT clause (duplicate elimination)

Normalization

Process of minimizing data redundancy. Includes a foreign key to the information in another table for repeated data. The original table is the result of inner joins between tables.

SQLite
Table has fixed number of named columns of specified type.

5 storage classes for columns:
- Null
- Integer
- Real
- Text
- Blob

Reductions and Group By

Use GroupBy to select which columns to average, count, sum, average, min, max.

```
SELECT SID, Name, AVG(GPA)
 FROM Students
 GROUP BY SID
```

Pandas / Python

- Positives: Pandas is lightweight and fast, full SQL expressiveness plus expressiveness of Python, integration with plotting functions like Matplotlib.
- Negatives: Tables must fit into memory. No post-load indexing functionality, indices are built when a table is created. No transactions or journalings. Large and complex joins are slow.

Series
- Keys of dictionary are indexes, built on NumPy's `ndarray`

DataFrame
- Tables with named columns represented as a dictionary
- Each Series object represents a column

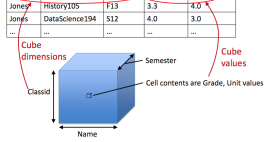
Operations
- Map, filter, sort, group by, aggregate, pivot or reshape
- Relational functions, e.g. union, joins, renames

Online Analytical Processing - OLAP

- Like a n-dimensional spreadsheet (cube)
- Columns become dimensions, where some of the variables are used as qualifiers, whereas those that we want to measure are normally numeric

Constructing OLAP cubes

name	classid	Semester	Grade	Units
Jones	History105	F13	3.3	4.0
Jones	DataScience194	S12	4.0	3.0
...	...	...	...	...



Aggregation
- We can project a field on another axis, and then this is precomputed and maintained automatically in an OLAP cube, so queries are instantaneous. This helps to support real-time querying.
- However, aggregates increase space and cost of updates, but storage overhead is still small
- Good integration with clients
- Numpy and Matlab use dense nd-arrays which are OLAP

What's Wrong with RDBMS Tables?
- Indices: Typical RDBMS tables mostly use indices, but we can't afford this for large datastores
- Transactions: Safe state changes require journals and are slow
- Relations: Checking relations adds further overhead to updates
- Sparse Data Support: Very wasteful, however there is both row-based and column-based

NoSQL Storage Systems
- Cassandra, CouchDB (JSON), Pig (sequences of MapReduce Jobs), Hive (relational database built on Hadoop, developed at Facebook)
- Could be dynamic columns, or no schema (like CouchDB)

Pig

- SQL functionality built on top of Hadoop
- Example Problem

An Example Problem

In Pig Latin

```
Users  = load 'users' as (name, age);
Filtered = filter Users by
        age >= 18 and age <= 25;
Pages  = load 'pages' as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed  = foreach Grouped generate group,
        count(Joined) as clicks;
Sorted  = order Summed by clicks desc;
Top5    = limit Sorted 5;

store Top5 into 'top5sites';
```

- Sample measurements have variance, and bias.
- Unbiased includes sample mean and median. Biased includes min and max. Variance:

$$\sigma^2 = \sum_{i=1}^{n}(x-\bar{x})^2 / n$$

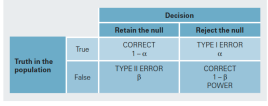
Variance
Central Limit Theorem
- The distribution of the sum of a set of n identically-distributed random variables X_i approaches a normal distribution as $n \to$ infinity
- Many statistical tools including mean and variance, t-test, ANOVA assume data are normally distributed

Distributions
- **Poisson:** distribution of counts that occur at a certain rate, e.g. web site clicks
- **Exponential**: Interval between two such events
- **Zipf/Pareto/Yule**: Govern the frequencies of different terms in a document, or web site visits
- **Binomial / Multinomial**: The number of counts of events (e.g. die tosses) out of n trials

Rhine Paradox: Experiment where the act of telling psychics that they have psychic abilities causes them to lose it

Hypothesis Testing:
- A test statistic is some measurement we can make on the data which is likely to be **big under the hypothesis** we want to prove (H_A) but small under the null hypothesis (H_O).
- E.g. we are trying to prove a biased coin. We set the test statistic to be that the coin flips heads many times.
  - H_A = Coin is biased
  - H_0 = Coin is fair (to be disproved!)
- Instead of proving the hypothesis, we attempt to **disprove** the null hypothesis (H_O)
- One-tailed or two-tailed. Two-tailed test is convenient because it checks either very large or very small counts of heads.
- H_0: null hypothesis that mean(A) = mean(B). We Reject if **Pr(x > s | H_0) < p** is small, e.g. less than 0.05.

Types of Error



Three Important Tests
- T-test: compare two groups to see if they have the same distribution
  - test statistic = **t = sample mean / sample s.d.**
  - This is known as the t-distribution
  - You could also have a single-sample test for one group of individuals in two conditions.
  - This is called a within-subjects design
- Chi-squared and Fisher's test: Compare counts in a contingency table
  - Test whether an observation is consistent with the data
  -

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

  - Fisher's exact test works when counts are under different conditions
- ANOVA (ANalysis Of VAriance)
  - Compare outcomes under several discrete interventions
- All the above tests are parametric:
  - Assume the data are
    - **normally distributed**
    - the samples are **independent of each other and all have the same distribution** (IID).
- Non-parametric tests (assume nothing about distribution)
  - K-S test
    - useful test for checking whether two (continuous or discrete) distributions are the same.
      - Such as testing if a distribution is normal

Lecture 4: Statistics

Measurements
- Properties like min, max, mean, std. dev.
- Relationships: scatter plots, regression, correlation
- For models: accuracy, performance

A **Null hypothesis** is essentially the position a devil's advocate would take when evaluating the results of an experiment (there is no difference in data samples or that difference is a certain value).
The **p-value** is the probability of a statistic value at least as extreme as the observation given that the null hypothesis is true.
- High **p-value** your data are likely with a true null.
- Low **p-value**: your data are unlikely with a true null.

Samples
- We usually only have samples from an infinite population
  - Or testing to make sure your data is similar to data produced by your pipeline.
- one-sided test is compared against a reference distribution
- two observed distributions are compared
- K-S statistic is just the max distance between the CDFs of the two distribution
- Permutation Tests
- Bootstrap sampling
  - Sampling your sample
  - Boostrap Confidence Interval tests to see if the 95% CI contains what we predict.

Train Test Validation
- Split data into training, test and validation sets
  - Validation set is used for parameter tuning
  - 60-20-20 split

Feature Selection
- Method 1: **Ablation**
  - Train a model on features {f_1,... f_n}, measure performance Q_0
  - Remove a feature and remeasure performance.
  - If new performance is **significantly (by some t-test)** worst than Q_0, then don't remove the feature.
- Method 2: **Mutual Information**
  - Mutual information measures the extent to which **knowledge of one feature influences the distribution of another** (the classifier output).
- Method 3: **Chi-Squared Test** ?
  - CHI-squared is an important statistic to know for comparing count data.
- Feature Hashing
  - Allows us to make sense of very rare features (such as URLs) by hashing them into similar categories and just keeping a count of features that fill a hash.
  -

Lecture 5: NLP

Basic Terms:
- **Syntax**: the allowable structures in the language: sentences, phrases, affixes (-ing, -ed, -ment, etc.).
- **Semantics**: the meaning(s) of texts in the language.
- **Part-of-Speech (POS)**: the category of a word (noun, verb, preposition etc.).
- **Bag-of-words (BoW)**: a featurization that uses a vector of word counts (or binary) ignoring order.
- **N-gram**: for a fixed, small N (2-5 is common), an n-gram is a consecutive sequence of words in a text.

Bag of Words Featurization
- Map words to a unique integer and store as a sparse vector (id, count):
  - (1,2),(3,1),(5,1),(12,1),(14,1)
- original word order is lost, replaced by the order of id's.
  - common to discard infrequent words
  - follow power law distribution
- tf-idf weighting
  - tf- term frequency - within the document
  - idf - inverse domain frequency - more rare words weighted more
  - tf-idf weighting can make BoW featurization more powerful
- Use cosine distance with some sort of weightings to optimize Bag of Words

N-grams
- modeling tuples of consecutive words
  - Sentence: The cat sat on the mat
  - 3-grams: the-cat-sat, cat-sat-on, sat-on-the, on-the-mat
- Using n-grams+BoW improves accuracy over BoW alone for classifying and other text problems. A typical performance (e.g. classifier accuracy) looks like this:
  - 3-grams < 2-grams < 1-grams < 1+2-grams < 1+2+3-grams
- N-grams also follow a power law distribution
  - Because of this you may see values like this:
    - Unigram dictionary size: 40,000
    - Bigram dictionary size: 100,000
    - Trigram dictionary size: 300,000
  - With coverage of > 80% of the features occurring in the text.
- An n-gram language model associates a probability with each n-gram, such that the sum over all n-grams (for fixed n) is 1.

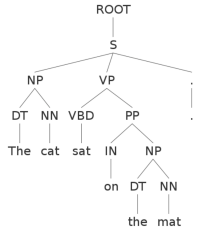
Skip Grams
- BOSG (Bag of Skip Gram Model) - reveals an algebraic structure to word meaning.
- A skip-gram is a set of non-consecutive words (with specified offset), that occur in some sentence.

POS (Parts of Speech) Taggers
- Taggers typically use HMMs (Hidden-Markov Models) or MaxEnt (Maximum Entropy) sequence models, with the latter giving state-of-the art performance.
- Tagging is fast, 300k words/second typical

Named Entity Recognition
- People, places, (specific)things
- The learning methods use sequence models: HMMs or CRFs = Conditional Random Fields, with the latter giving state-of-the-art performance.

Grammars
- Grammars comprise rules that specify acceptable sentences in the language
- English Grammars are context-free: the productions do not depend on any words before or after the production.

Parsing
- The reconstruction of a sequence of grammar productions from a sentence is called "parsing" the sentence.



- Probabilistic Context-Free Grammars (PCFGs) associate and learn probabilities for each rule
- The parser then tries to find the most likely sequence of productions that generate the given sentence. This adds more realistic "world knowledge" and generally gives much better results.
- CKY parsing uses dynamic programming - most parsers are some version of this.

Dependency Parsing
Breaks sentences down into their word relationships. Each vertex in the tree represents a word and it children are words that are dependent on it's parent (predicate as parent, subject and object as children). Thus it is helpful in understanding how words are related. It may be non-binary and the structure uses links (generally triples). We can create a dependency tree using Stanford's lexicalized constituency parser.

Constituency Parsing
Breaks sentences down in accordance to the grammatical standing (e.g. Sentence → Verb Phrase → Verb). This is helpful for understanding sub phrases in a sentence. This default output from Stanford's lexicalized constituency parser.

Constituency vs. Dependency Grammars
- Dependency and constituency parses capture attachment information (e.g. sentiments directed at particular targets).
- Constituency parses are probably better for abstracting semantic features, i.e. constituency units often correspond to semantic units.

Lecture 6: Machine Learning 1

Supervised:
- We are given input/output samples (X, y) which we relate with a function y = f(X). We would like to "learn" f, and evaluate it on new data. Types:
  - **Classification**: y is **discrete** (class labels).
  - **Regression**: y is **continuous**, e.g. linear regression.
  - Is this image a cat, dog, car, house?
  - How would this user score that restaurant?
  - Examples
    - knn
    - Naive Bayes
    - Linear and Logistic Regression
    - Random Forests
    - Neural Networks

Unsupervised:
- Given only samples X of the data, we compute a function f such that y = f(X) is "simpler".
  - **Clustering**: y is **discrete**
  - Y is **continuous**: Matrix factorization, Kalman filtering, unsupervised neural networks.
  - Cluster some hand-written digit data into 10 classes.
  - What are the top 20 topics in Twitter right now?
  - Examples
    - Clustering
    - Topic Models
    - HMM (Hidden Markov Models)
    - Neural Networks

Samples

- We are most interested in models of the population, but we have access only to a sample of it.
- We train on a training sample *D* and we denote the model as f_d(X)
- Bias: if we train models f_d(X) on many training sets *D*, bias is the expected difference between their predictions and the true *y*'s.
- Variance: if we train models $f_D(X)$ on many training sets *D*, variance is the variance of the estimates:
- There is usually a bias-variance tradeoff caused by model complexity.
  - **Complex models** (many parameters) usually have **lower bias, but higher variance.**
  - **Simple models** (few parameters) have **higher bias, but lower variance.**
- The total expected error is *Bias^2 + Variance*
- If *Variance* strongly dominates, it means there is too much variation between models. This is called over-fitting.
- If *Bias* strongly dominates, then the models are not fitting the data well enough. This is called under-fitting.

K-Nearest Neighbors (kNN)

- Given a query item: Find k closest matches in a labeled dataset
- The Data is the Model: No training needed.
- Classification:
  - Model is y = f(X), y is from a discrete set (labels).
  - Given X, compute y = majority vote of the k nearest neighbors.
  - Can also use a weighted vote* of the neighbors.
- Regression:
  - Model is y = f(X), y is a real value.
  - Given X, compute y = average value of the k nearest neighbors.
  - Can also use a weighted average* of the neighbors.
- * Weight function is usually the inverse distance.
- Small k → low bias, high variance
- Large k → high bias, low variance
- ***The curse of dimensionality*** refers to phenomena that occur in high dimensions (100s to millions) that do not occur in low-dimensional (e.g. 3-dimensional) space.
  - Data in high dimensions tend to be very sparse
  - means there are few points that are close to the point we want to predict.

Naive Bayes

D = Data, e = some event

$$P(e|D) = \frac{P(D|e)P(e)}{P(D)}$$

P(e) is called the **prior probability** of e. Its what we know (or think we know) about e with no other evidence.

P(D|e) is the **conditional probability** of D given that e happened, or just the likelihood of D. This can often be measured or computed precisely – it follows from your model assumptions.

P(e|D) is the **posterior probability** of e given D. It's the answer we want, or the way we choose a best answer.

You can see that the posterior is heavily colored by the prior, so Bayes' has a GIGO liability. e.g. its not used to test hypotheses

- **Key Assumption:** (Naive) the features are generated independently given *cj*.
- \+ Simple and fast. Depend only on term frequency data for the classes. One pass over the training dataset, no iteration.
- \+ Compact model. Size is proportional to numfeats x numclasses.
- \+ Very well-behaved numerically. Term weight depends only on frequency of that term. Decoupled from other terms.
- \+ Can work very well with sparse data, where combinations of dependent terms are rare.
- − Subject to error and bias when term probabilities are not independent (e.g. URL prefixes).
- − Can't model patterns in the data.
- − Typically not as accurate as other methods for the above reasons.

If you care, here is the formula we are trying to maximize with Naive Bayes (The classifier is trying to pick a category cj that maximizes):

We want the category c_j that maximizes:

$$\Pr(X_1, \ldots, X_k \mid c_j)\Pr(c_j) = \Pr(c_j)\prod_{i=1}^{k}\Pr(X_i|c_j)$$

Lecture 7: Machine Learning 2

Linear Regression

- We want to find the "best" line (linear function y=f(X)) to explain the data.
- The most common measure of fit between the line and the data is the least-squares fit.
- R^2 → A way of measuring how linear it really is. Want R^2 to be as close to 1 as possible

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

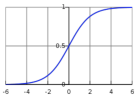
- Statistic: From R-squared we can derive another statistic (using degrees of freedom) that has a standard distribution called an F-distribution.
  - The P-value is, as usual, the probability of observing the data under the null hypothesis of no linear relationship.
  - If p is small, say less than 0.05, we conclude that there is a linear relationship.

Logistic Regression

- Logistic regression is designed as a binary classifier (output say {0,1}) but actually outputs the probability that the input instance is in the "1" class.
- **Logistic regression is probably the most widely used general-purpose classifier.**

Logistic regression maps the "regression" value $-X\beta$ in $(-\infty,\infty)$ to the range [0,1] using a "logistic" function:

$$p(X) = \frac{1}{1+\exp(-X\beta)}$$



i.e. the logistic function maps any value on the real line to a probability in the range [0,1]

- A very efficient way to train logistic models is with Stochastic Gradient Descent (SGD) - we keep updating the model with gradients from small blocks (mini-batches) of input data.
- Cons: It can overfit sparse data. (often used with regularization)

Support Vector Machines (SVMs)

- Classifier that tries to maximize the margin between training data and the classification boundary. Maximizing the margin maximizes the chance that classification will be correct on new data.
- SVMs can be trained using SGD.

Newtons Method

- Classic approach to iterative optimization

$$X^1 = X^0 - H_f^{-1}(X^0)\,G_f(X^0)$$

Where $G_f = \frac{df}{dX_i}$ is the gradient and

$H_f = \frac{d^2 f}{dX_i dX_j}$ (a matrix) is the Hessian.

- Converges very fast when its feasible
- Challenges:
  a. Both the gradient and Hessian are normally computed on the entire dataset. This means many passes over the data to complete a series of Newton steps.
  b. The Hessian has size O(M2) if there are M features. This is impractical for large feature spaces, e.g. text or event data

Stochastic Gradient

- A very important set of iterative algorithms use stochastic gradient updates.
- They use a small subset or mini-batch X of the data, and use it to compute a gradient which is added to the model
- Stochastic gradient has some serious limitations however, especially if the gradients very widely in magnitude. Some coefficients change very fast, others very slowly.
- For harder (non-convex) learning problems, its common to use learning rate schedules that decay more slowly, or not at all.

Decision Trees

- Walk from root to a class-labeled leaf.
- At each node, branch based on the value of some feature.
- At each node, we choose the feature f which maximizes the information gain. which is the minimal entropy.
- Information Gain:
  - Entropy is defined at each node based on the class breakdown as a function over the probability distribution
  - Equation for entropy by feature split (keep in mind we want to maximize information gain, while avoiding overfitting.

Before the split by f, entropy is

$$E = -\sum_{i=1}^{m} p_i \log p_i$$

After split by f, the entropy is

$$E_f = -p^f \sum_{i=1}^{m} p_i^f \log p_i^f - p^{\neg f} \sum_{i=1}^{m} p_i^{\neg f} \log p_i^{\neg f}$$

The information gain = $E - E_f$ (information = -entropy)

Ensemble Methods

- Crowdsourced machine learning. take a bunch of weak learning algorithms and combine to create a good one.
- Types:
  - Bagging: train learners in parallel on different samples of the data, then combine by voting (discrete output) or by averaging (continuous output).
  - Stacking: combine model outputs using a second-stage learner like linear regression.
  - Boosting: train learners on the filtered output of other learners.

Random Forests

The basic premise is that we don't want the models to be too similar so we split them across different learners, train them on that data, and decided on the processing of our actual data based on the consensus.

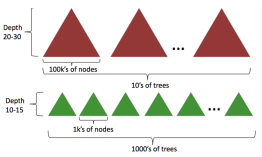
- Grow K trees on datasets sampled from the original dataset with replacement (bootstrap samples) (p = number of features)
- Draw K bootstrap samples of size N (each tree is trained on different data).
- Grow each Decision Tree, by selecting a random set of *m out of p features* at each node, and choosing the best feature to split on (correspoinsfond notes in a nearby tree can't split on the same feature.
- Aggregate the predictions of the trees (most popular vote) to produce the final class.
- Typically m will be sqrt(p) but can be smaller.

Random forest is very popular on data with fewer features (dense). Easily parallelizable (fast), not the most accurate, needs at least the max depth of the tree passover and is easy to overfit.

Boosted Decision Trees

A modified random forest implementation where the trees are training independently and the trees are trained sequentially by boosting. Each tree is trained on weighted data which emphasis incorrectly labeled instances by the previous trees. Generally the models are very good.

Comparison (RF vs BDT)



Lecture 8: Unsupervised Learning

Clustering

- We don't claim that clusters model underlying structure, but that they give a reduced-variance "sample" of the data.
- Humans tend to see cluster structure whether it is there or not
- Type of clustering:
  - Hierarchical clustering: clusters form a hierarchy. Can be computed bottom-up or top-down.
  - Flatclustering:nointer-clusterstructure.
  - Hard clustering: items assigned to a unique cluster.
  - Soft clustering: cluster membership is a real-valued function, distributed across several clusters.

K-Means Clustering

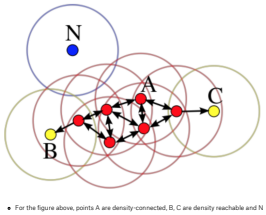
- The standard k-means algorithm is based on Euclidean distance.
- The cluster quality measure is an intra-cluster measure only, equivalent to the sum of item-to-centroid kernels.
- A simple greedy algorithm locally optimizes this measure (usually called Lloyd's algorithm):
  - Find the closest cluster center for each item, and assign it to that cluster.
  - Recompute the cluster centroid as the mean of items, for the newly-assigned items in the cluster.
  - Iterate for fixed number of iterations, or Until no/small changes in assignments
  - It's a greedy algorithm with random setup – solution isn't optimal and varies significantly with different initial points.
- Very simple convergence proofs.
- Performance is O(nk) per iteration, not bad and can be heuristically improved.
  - n = total features in the dataset, k = number clusters
- Many generalizations, e.g.
  - Fixed-size clusters
  - Simple generalization to m-best soft clustering
- As a "local" clustering method, it works well for data condensation/compression.
- Akaike Information Criterion

$$K = \arg\min_K [-2L(K) + 2q(K)]$$

- K=dimension, L(K) is the likelihood (could be RSS) and q(K) is a measure of model complexity (cluster description complexity).
- AIC favors more compact (fewer clusters) clusterings.

DBSCAN

- DBSCAN performs density-based clustering, and follows the shape of dense neighborhoods of points.
- Core points have at least minPts neighbors in a sphere of diameter *e* around them. and can directly reach neighbors in their -sphere.
- Non-core points cannot directly reach another point.
- Point q is density-reachable from p if there is a series of points p=p1,p2,...pn=q s.t. pi+1 is directly reachable from pi
- Points, p, q are density-connected if there is a point o such that both p and q are density-reachable from o.
- All points not density-reachable from any other points are outliers.



- For the figure above, points A are density-connected, B, C are density reachable and N is not.
- The algorithm overall can be made to run in O(n log n).

Matrix Factorization

- Motivated often by the curse of dimensonality when the model has more degrees of freedom (variables in the output) than we have in the data.
- When having to deal with thousands or millions of responses, as well as thousands or millions of features.
- High-dimensional data will often have simpler structure, e.g.
  - Topics in documents
  - General product characteristics (reliability, cost etc.)
- We can approximate these effects using a lower-dimensional model
- The subspace allows us to predict the values of other features from known features
- Usually, data vary more strongly in some dimensions than others. Fitting a linear model to the (k) strongest directions gives the best approximation to the data in a least-squares sense.
- Algebraically, we have a high-dimensional data matrix (typically sparse) S, which we approximate as a product of two dense matrices A and B with low "inner" dimension:
  - From the factorization, we can fill in missing values of S. This problem is often called "Matrix Completion".
- Some complex math and SGD, or Alternating Least Squaresa can be used to find the factorization (think we should focus more on high level and applications than the math)

Performance

- There are two different dimensions of performance:
  a. Offline: Model training. Plenty of resources. Typical goal is 1 to few days training time. Best possible model accuracy.
  b. Online: Model prediction. Limited resources (memory and machines). Goal is usually to minimize latency, and perhaps online model size.
- There are at least three approaches to improving performance:
  - Algorithmic improvements. e.g. adaptive SGD methods.
    - Freal Orders of magnitude improvements possible.
  - Computational improvements: Fully optimize code, leverage hardware (e.g. GPUs).
    - Freal Orders of magnitude improvements possible. Compounds algorithmic improvements.
  - Cluster scale-up: Distribute model or data across computers on a network.
    - Superlinear cost: a k-fold speedup requires, fn(k) > k additional cost. Usually does not compound algorithmic and computational improvements (network bottleneck).

Lecture 9: Deep Learning

Deep Learning is sub branch of machine learning specifically for modeling complex and high-level data by decomposing it into processing layers of learn build upon the previous layer. Commonly it is used for Computer Vision and Voice recognition.

Loss (cost) is a function that specifies the goal of a learning task and marks the values against some inadequacy metric called badness.

Risk is measuring loss over the input. Since we can't try all the values in a unbounded domain, we estimate it using a summation function and we call the resulting value **Empirical Risk**.

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

Empirical Risk

We can try to optimise against our loss by using a **Stochastic Gradient Decent (SGD)** where you descend into layers of values in order and update your estimation to the one that minimizes loss.

Vision

We use the practices in SGD to enable Computers to structure the rich visual world around it. In order to recognize objects we separate the features of the object(s) we are trying to recognize into layers. The visual complexity increase as we decend into deeper and deeper layers. For example, our first layer focuses on color, the second on colors and patterns or edges, the third on colors, patterns, and shapes, and so on until the program has collected is tuned to effectively look at the wholistic object (face, animal, building, etc).

To better **tune** this process we train it on a high quality recognition network with a large set of image data (like ImageNet) and then freeze all the but the last couple layers of the network which we train on a small sample set of labeled data similar to the ones we want to classify.

When we take the gradient of the model layer by layer (using the chain rule to nest them together) to give us the total gradient.

Deep learning is **not** logistical regression because logistical regression is a linear classify that does not compose models; it's just one step. It is also not a decision tree because those decisions are made on the input, there is no transformation or intermediate representation. It also doesn't follow the machine learning pipeline which takes an input, designs features, builds learn clusters, then put a classifier on top, where each stage is separate and not end to end.

Neural Networks

We don't know how the brain works and thus these networks may not at all be what they sound like. Our fundamental unit is perceptron (a threshholded sum with an activation function at that threshold) that combines with other to form a network. They use both linear and non-linear units.

Convolutional Neural Networks (CNN) are models that are adapted for vision and help to create feature maps that determine spatial structure. We using pooling and subsampling to increase the range of this model. We use multiple filters to generate feature. Non-linearity of this network is needed to "deepen" the representation by composition. Otherwise a sequence of linear steps "collapses" to a single linear transformation. We then pool to take the max of the filter response to at different locations to build robustness. The features are equalised as a consequence to that. LeNet and AlexNet are examples of CNN.

Recurrent Neural Networks (RNN) are great for text because they allow the network to store context of words in a corpus (he, it, there etc) in the node memory (giving it a recurrent state). It is great for translations but it is often difficult to train.

Long Short-Term Memory (LSTM) is great for speech recognition, handwriting recognition, and parsing, and works by improving long term memory by randomly knocking out cell states (forgetting) to retrain and improve model. It is easier to train and is great at generating descriptions of images.

Lecture 10: Scaling

Q: How much data do you need?
A: As much as you can get.

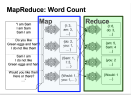
Budget hardware - the money goes into building distributed software instead.

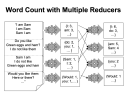
Problems with cheap hardware:

- Failures:
- Slower:
  - More latency
  - Slower throughput
- Uneven Performance

Map Reduce

- First introduced by Google, made available by Yahoo as Hadoop.





MapReduce for ML

- Two types of parallel ML algorithm:
  - Data Parallel -
    - Distributed data, shared model
    - batch algorithms can be implemented in one map-reduce step
  - Model Parallel -
    - Algorithms require two reduce steps for each iteration (reduce, and then redistribute to each mapper)
    - Data AND model are distributed
- Challenges:
  - Very Low Level programming interface
  - Very little reuse of MR code between projects
- Relies on Java reflection (?)

Spark

- Like MapReduce, but keeps intermediate data in memory (saves a TON of slow IO operations)
- Advantages:
  - High-level programming model
  - Interactive (Spark shell, Pyspark, etc)
  - Integrated User Defined Functions
  - Scala generics instead of reflection (spark code is generic over [key, value] pairs.
- Spark Programming Model:
  - RDD (Resilient Distributed Dataset)
    - Distributed array of data
    - Values are computed lazily
    - Values are cached.
- Spark Examples:

Spark programming model (Python)

```
lines = sc.textFile('...', 4)
comments = lines.filter(isComment)
print lines.count(), comments.count()
```

Getting data out of RDDs

```
rdd.reduce(lambda a, b: a * b)
# [1,2,3] * 6
rdd.take(1)
# RDD of [1,2,3] = [1,2] # as list
rdd.collect()
# RDD of [1,2,3] = [1,2,3] # as list
...
rdd.saveAsTextFile(...)
```

Shuffle transformations

```
rdd.groupByKey()
# [(1,'a'),(1,'b'),(2,'c')] =
#   [(1,['a','b']),(2,['c'])]
...
rdd.sortByKey()
# [(3,'a'),(1,'b'),(2,'c')] =
#   [(1,'b'),(2,'c'),(3,'a')]
```

Other transformations

```
rdd.filter(lambda x: x % 2 == 0)
# [1, 2, 3] => [2]

rdd.map(lambda x: x * 2)
# [1, 2, 3] => [2, 4, 6]

rdd.flatMap(lambda x: [x, x+5])
# [1, 2, 3] => [1, 6, 2, 7, 3, 8]
```

Example: Logistic Regression in PySpark

```
points = sc.textFile(...).map(parseVector).cache()
w = numpy.random.ranf(size = D) # number of dimensions
for i in range(ITERATIONS):
    gradient = points.map(
        lambda p: (1/(1+exp(-p.y*(w.dot(p.x))))-1)*p.y*p.x
    ).reduce(lambda a, b: a + b)
    w -= gradient
print "Final model: %s" % w
```

Spark ML Toolkit

- Classification
- Regression
- Unsupervised
- Optimizers (SGD, L-BGFS)

Spark Driver and Executors

- Driver runs user interaction, acts as master for batch jobs
- Driver hosts machine learning models
- Executors hold data partitions
- Tasks process data blocks

Spark Shuffle

- Used for GroupByKey, Sort, Join operations
- Map and Reduce tasks run on executors

Architectural Consequences

- Models must fit in a single machine memory
- simple programming: centralized master, broadcast to other nodes.

Lecture 11: Visualizations

Visualizations are useful because they allow for Data Analysts to intuitively interpret the process and results of their data. It allows you to transform the symbolic to geometric.

It is useful for:

1. Finding Relationships
2. Understanding Structure
3. Quantifying values and dependencies
4. Engages with users or readers.
5. Allows for focus on the important things.

Always simplify to the point that it is intuitive. Only use "gratuitous" (revealing) features. Use interactivity to reveal structure.

Three Principles for Data Visualization

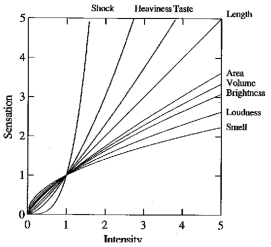
1. Simplify
2. Understand Magnitudes
3. Use Color
4. Use Structure

Weber's Law (Just Noticeable Difference)

$\Delta S = k(\Delta I/I)$
[*I* is the change in stimuli and *k* is experimental data statistics]

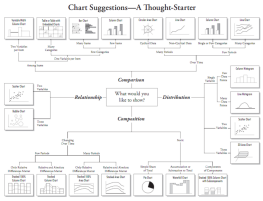
Steven's Power Law

$S = I^p$
[Where *S* is sensation, I is intensity, and p is the value determined by the type of stimulus]



Colors

- For **sequential** data use a color scheme ordered from low to high.
- For **divergent** data use two divergent sequential color emanating from a "critical midpoint" of neutral color.
- For **categorical** data use contrasting sequences of color such that no two adjacent pair are similar.



Interactive Web Toolkits

- D3: Framework designed as a low level "kernal" for visualizations to run on.
- Vega: a visualization syntax library for specifying graphics in JSON to be rendered by d3.
- Vincent: a Python to Vega compiler, write in Python, render in JS (via Vega on D3).

Lecture 12: Graphs

Six degrees of separation states that the typical distance from any random person to any other random person is about six edges in a social graph.

In the rank form of the power law for graphs, the degree (number of connections) for the person with rank k (i.e. the person with the kth-most connections) is proportional to $k^{-\sigma}$, where σ is usually quite close to 1.

Graph Structured Data

- Social Graph
- Social Media Graph
- Web Graph
- Semantics

- Transaction Networks
- Biological Networks
- Co-Authorship Networks
- etc.

Degree distribution follows a power law. (there are very few high degree nodes, but many low degree nodes).

Giant Connected Componenet
- Typically only one GIANT CC
- CC are components taht can reach each other.

Densification - Average distance between nodes decreases over time.

Things to look at when given a graph:
- Density (vertices / (edges ^ 2))
- Degree distribution
  - Skewed?
  - Properties of high degree vertices (who's the person at the middle of the social graph)
- Connected Component Sizes
- Community Profile Plots (does it fit some community pattern)
- Structure around large, important vertices (something like a bridge between two connected components)

Page Rank
- Recursive Relationship

Recursive Relationship:

$$R[i] = \alpha + (1-\alpha) \sum_{(j,i)\in E} \frac{1}{L[j]} R[j]$$

- α is the random reset probability (typically 0.15)
- L[j] is the number of links on page

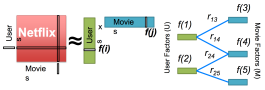
Predicting Behavior

We can use a similar recursive structure to predict behavior by analyzing behavior of connections.

Low Rank Matrix Factorization:
- Used for recommending products.
- Here's the function for it.

Recommending Products

Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg\min_{w\in\mathbb{R}^d} \sum_{j\in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

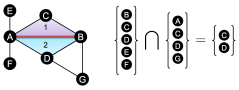
33

Clustering Coefficient
- How tightknit are the communities?
- Specifically, how tight knit is a community around a single node.

Measure "cohesiveness" of local community

$$\text{ClusterCoeff[i]} = \frac{2 * \#\text{Triangles[i]}}{\text{Deg[i]} * (\text{Deg[i]} - 1)}$$

- You need to be able to count triangles
  - Counting Triangles tells you how many shared connections a particular node's connections have. Take the intersection of two nodes out degrees to determine their shared connections.

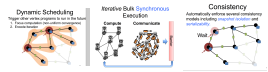


Connected Components:
- Describes nodes connected to each other by some sequence of connections.

Dynamic Working Sets
- Many entries (51%) only run once!
- But there's a long tail, a few entries run many times (change many times)

Summary:
- Graph algorithms address transitive structure
  - Iterative
  - Dynamic working sets
- Many graph algorithms can be specified locally
  - Defined at the level of a vertex and its neighbors

Graph Algorithm implementation
Vertex Programs
- user defined **vertex program** has access to the properties of neighboring edges and vertices. They interact by sending messages.
- When messages come in they might update their vertex's property.
- Then scatter that update to neighboring vertices with messages.
- Dynamic Scheduling→ Assigns jobs to be calculated in bulk. → Bulk Synchronous Excution ensures consistency?



Commutative Associative Vertex Programs
- Gather - Get messages
- Apply - Compute a new vertex property
- Scatter - Scatter that new information (in the form of messages)

Power law graphs are difficult to partition:
- They don't have low cost balanced cuts.
- Two options:
  - Cut Edges
    - Hella Edges end up getting cut
  - Cut Vertices
- Instead of partitioning by cutting edges, cut vertices!

You can treat Map Reduce (Spark) jobs as graph problems (joins can be solved more efficiently as a grpahing problem)