Introduction

Data Science - Using computations plus statistics and applying to data to derive knowledge that we are collecting from a phenomenon that is being studied

Jeff Hammerbacher's Model:

1. Identify problem
2. Instrument data sources
3. Collect data
4. Prepare data (integrate transform, clean, filter, aggregate)
5. Build model
6. Evaluate model
7. Communicate results

What's Hard about DS

- Overcoming assumptions
- Making ad-hoc explanations of data patterns
- Overgeneralizing
- Communication
- Not checking enough (validate models, data pipeline, integrity, etc.)
- Using statistical tests correctly
- Prototypes → Production transitions
- Data pipeline complexity

Some Ethical Concerns

- Preserving privacy
- Avoiding bias
- Mitigating malicious attacks

Privacy is all about trust

- Some data cannot be anonymized
- Often, people's desires about their data involve questions of trust
- Privacy is not a binary value

Data Collection, Preparation, and Exploration

ETL - **Extract** from **sources**, **Transform** data at **source**, **sink**, staging area, **Load** into **sink**

Sources: file, database, event log, website, etc.

Sinks: Python, R, SQLite, RDBMS, etc.

Data Sources: Structured Data (app DB), Semi-structured (event log, api server log), unstructured (wikipedia, images, video)

Role of Schema:

- **Schema** specifies **structure** and **types** of data
- Traditional DB's are **schema-on-write**
- Newer (noSQL) DB's are **schema-on-read** (**schemaless**)
- **Structured** (SQL) vs **Unstructured** (XML, XQuery, DOM - Document Object Model)
- JSON is **schemaless**
- **XML -** MarkLogic, **JSON -** mongoDB

Tabular Data:

- **Table** is collection of **rows** and **columns**
- **Row** has **index**, **column** has **name**, **cell** is specified by (**index, name**), **cell** doesn't need **value**
- Often stores as CSV or TSV

Processing Collected Data:

- **SAX:** parses XML data (filter out relevant data)
- **HTML** parsers: Beautiful Soup, TagSoup, Taggle
- Most large websites **discourage** screen-scraping and provide **Web Service APIs**
- **Web service** is defined as a software system designed to support interoperable machine-to-machine interaction over a network

    Two Kinds:

- **SOAP** (HTTP POST/GET), **REST** (contains all information, keep things simple, set of uniquely identified resources, set of well defined operations, uses Hypermedia)

Preparation Problems:

Parsing text into fields (separator issues)
Naming conventions (e.g., NYC vs. New York)
Missing required field (e.g., key field)
Different representations (e.g., 2 vs Two)
Fields too long (get truncated)
Primary key violation (from un- to structured or during integration)
Redundant/Duplicate Records
Formatting issues (especially dates)
Licensing issues and privacy concerns keep you from using the data as you would like

Statistics View:

- We want ideal samples, but in practice we have non-ideal samples (**distortion, selection bias, censorship, dependence**)
- Numeric outliers

Tools:

- **OpenRefine, DataWrangler**

Exploring:

- **Matplotlib, Ggplot, D3.js**
- **Histogram:** Single variable, discrete or continuous (normal, skewed, long-tailed → log-log plot)
    - **Multimodal Data:** Two or more distinct peaks
- **Scatterplot**: Two variables
- **Stacked Plot**: More than two variables when discrete
- **Parallel Coordinate Plot:** More than two variables, arbitrary number of other variables
- **Radar Chart:** Similar to **Parallel Coordinate Plot**
- **PCA (Principal Component Analysis):** Allows visualization of high-dimension continuous data in lower dimension (2D) by removing arbitrary dimensions

Form Expectations of what the data should look like to guard against pipeline errors without imposing biased expectations

Data Statistics

Measurement: Want to measure properties of data, basic (min, max, mean, etc.) and relationships (between columns via scatter plot, regression, causality, etc.); for **Models,** accuracy (how well does model match the data) and performance (memory, runtime)

- Many datasets are **samples** from an **infinite population**
- Want measures of **population**, but only have access to a **sample**
- **Samples** vary from one sample to the next and may be biased

Unbiased: sample mean, sample median
Biased: min, max, sample variance
X refers to population, **x refers to sample**

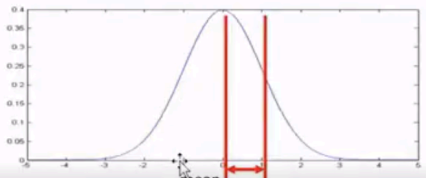
The mean of a set of values is just the average of the values.

Variance is a measure of the width of a distribution. Specifically, the variance is the mean squared deviation of points from the mean:

$$Var(X) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

The standard deviation is the square root of variance.

The normal distribution is completed characterized by mean and variance.



Central Limit Theorem: distribution of sum of a set of n identically distributed random variables approaches a normal distribution as n→inf

Normal distribution assumption: common parametric stat tests assume normally distributed data

Non normal distributions: poisson, exponential, zipf/pareto/yule, binomial/multinomial

Rhine Paradox

Hypothesis Testing: standard procedure for testing a claim about a property of a population

- Want to prove a hypothesis HA but it's hard so we try to disprove the **null hypothesis H0**
- **Test statistic** is some measurement we can make on data which is likely to be **big under HA** but **small under H0**
- **Procedure:**
    1. Start with question
    2. Establish hypothesis
    3. Determine test and sampling
    4. Choose Type 1 error rate
    5. State decision rule
    6. Gather sample data
    7. Calculate test statistics
    8. State statistical conclusion
    9. Make decision of inference from conclusion

Null hypothesis is a statement that the value of a population parameter is equal to some claimed value (**either reject H0 or fail to reject H0**)

Alternative hypothesis (H1) is the statement that the parameter has a value that differs from H0 (!=, <, >)

***** Frame the claim so that it becomes alternative hypothesis**

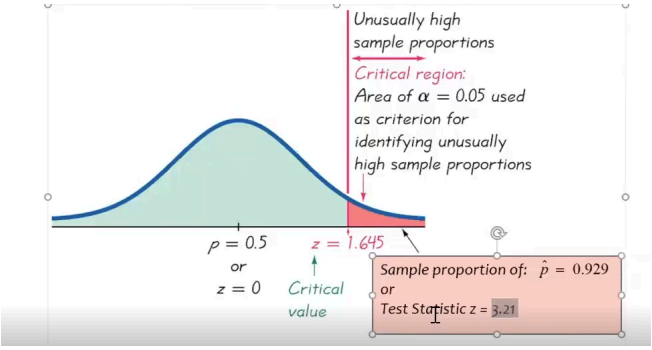
Test statistic is a value (score) with the assumption that the null hypothesis is true

Test Statistic - Formulas

Test statistic for proportion	$z = \dfrac{\hat{p} - p}{\sqrt{\dfrac{pq}{n}}}$
Test statistic for mean	$z = \dfrac{\bar{x} - \mu}{\dfrac{\sigma}{\sqrt{n}}}$ or $t = \dfrac{\bar{x} - \mu}{\dfrac{s}{\sqrt{n}}}$
Test statistic for standard deviation	$\chi^2 = \dfrac{(n-1)s^2}{\sigma^2}$

Z score is for mean 0 and standard dev 1

T score is for standard deviation of sample



Significance level: probability that test statistic will fall in critical region (alpha = 0.05, 0.01, 0.001)
Critical value: any value that separates critical region from the values of the test statistic that do not lead to the rejection of the null hypothesis
P-value: probability of getting a value of the test that is at least as extreme as one in sample data (critical region in left tail → p-value = area to left of test statistic, critical region in right tail → p-value = area to right of test statistic, critical region in two tails → p-value = twice the area beyond test statistic)

- If the p is low, null must go

Type 1 error is rejecting the null hypothesis when it is actually true (**false negative**) (alpha)
Type 2 error is mistake of failing to reject null hypothesis when it is actually false (**false positive**) (beta)

- To decrease alpha and beta, increase the sample size

T-test: Compare two groups

Single-sample: Compute the test statistic:

$$t = \frac{\bar{x}}{\bar{\sigma}}$$

where $\bar{x}$ is the sample mean and $\bar{\sigma}$ is the sample standard deviation, which is the square root of the sample variance Var(x). This is so-called t-statistic.

If X is normally distributed, t is **almost** normally distributed, but not quite because of the presence of $\bar{\sigma}$. It has a **t-distribution**.

You use the single-sample test for **one group** of individuals in **two conditions**. Just subtract the two measurements for each person, and use the difference for the single sample t-test.
This is called a **within-subjects** design.

In this test, there are **two samples** x_1 and x_2 of sizes n_1 and n_2. A t-statistic is constructed from their sample means and sample standard deviations:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sigma_{\bar{x}_1 - \bar{x}_2}}$$

where: $\sigma_{\bar{x}_1 - \bar{x}_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$ and σ_1 and σ_2 are sample sdevs.

You should try to understand the formula, but you shouldn't need to use it. Most stats. software exposes a function that takes the samples x_1 and x_2 as inputs directly.

This design is called a **between-subjects** test.

CHI-squared: Compare the counts in a contingency table

Chi-squared test

Often you will be faced with discrete (count) data. Given a table like this:

	Prob(X)	Count(X)
X=0	0.3	10
X=1	0.7	50

Where Prob(X) is part of a null hypothesis about the data (e.g., that a coin is fair).

The CHI-squared statistic lets you test whether an observation is consistent with the data:

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

O_i is an observed count, and E_i is the expected value of that count. It has a *chi-squared distribution*, whose p-values you compute to do the test.

Fischer's exact test:

Fisher's exact test

In case we only have counts under different conditions

	Count1(X)	Count2(X)
X=0	a	b
X=1	c	d

We can use Fisher's exact test (n = a+b+c+d):

$$p = \frac{\binom{a+b}{a}\binom{c+d}{c}}{\binom{n}{a+c}} = \frac{(a+b)!\ (c+d)!\ (a+c)!\ (b+d)!}{a!\ \ b!\ \ c!\ \ d!\ \ n!}$$

Which gives the probability directly (its not a statistic; it is exact).

ANOVA: Compare outcomes under several discrete interventions

ANOVA

ANOVA (ANalysis Of VAriance) allows testing of multiple differences in a single test. Suppose our experiment design has an independent variable Y with four levels:

Y			
Primary School	High School	College	Grad degree
4.1	4.5	4.2	3.8

The table shows the mean values of a response variable (e.g., avg number of Facebook posts per day) in each group.

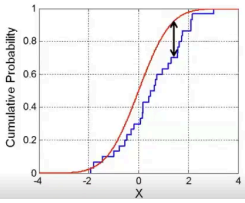
We would like to know in a single test whether the response variable depends on Y, at some particular significance such as 0.05.

ANOVA tests can be much more complicated, with multiple dependent variables, hierarchies of variables, correlated measurements, etc.

Nonparametric tests:

- **KS Test**

- The K-S (Kolmogorov-Smirnov) test is a very useful test for checking whether two (continuous or discrete) distributions are the same.
- In the **one-sided test**, an observed distribution (e.g. some observed values or a histogram) is compared against a reference distribution.
- In the **two-sided test**, two observed distributions are compared.
- The K-S statistic is just the **max distance between the CDFs** of the two distributions.
- While the statistic is simple, its distribution is not!
- But it is available in most stat packages.



- **Bootstrapping:** resampling the sample, **bootstrap test for regression**

Bootstrap Confidence Interval Tests

Then a test statistic outside the 95% Confidence Interval (CI) would be considered significant at 0.05, and probably not drawn from the same population.

E.g., suppose the data are differences in running times between two algorithms. If the 95% bootstrap CI does not contain zero, then original distribution probably has a mean other than zero, i.e. the running times are different.

We can also test for values other than zero. If the 95% CI contains only values greater than 2, we conclude that the difference in running times is significantly larger than 2.

Natural Language Processing:

- **Syntax, semantics, part-of-speech, bag-of-words, n-gram**

Assuming we have a dictionary mapping words to a unique integer id, a bag-of-words featurization of a sentence could look like this:

Sentence:	The	cat	sat	on	the	mat
word id's:	1	12	5	3	1	14

The BoW featurization would be the vector:

Vector	2,	0,	1,	0,	1,	0,	0,	0,	0,	0,	0,	1,	0,	1
position	1		3		5							12		14

In practice this would be stored as a sparse vector of (id, count)s:
(1,2),(3,1),(5,1),(12,1),(14,1)

N-grams

Sentence:	The	cat	sat	on	the	mat
2-grams:	the-cat,	cat-sat,	sat-on,	on-the,	the-mat	

Notice how even these short n-grams "make sense" as linguistic units. For the other sentence we would have different features:

Sentence:	The	mat	sat	on	the	cat
2-grams:	the-mat,	mat-sat,	sat-on,	on-the,	the-cat	

- **Size** becomes very large with an increasing n
- **K-skip-n-gram** gets sequence of words length-n at most k distance from each other
- **Part of speech** (noun, adj, verb, etc.)
- **POS Taggers(HMM MaxEnt)**
- **Named Entity Recognition**
- **Grammars** comprise rules that specify acceptable sentences in the language
- **Parse trees**
- **Probabilistic Context-Free Grammars -** learn probabilities for each rule and find the most likely sequence of productions

Tabular Data Processing:

Data model - collection of concepts for describing data
Schema - description of particular collection of data, using given data model **(specifies name of relation, plus name and type of each column)**
Relational database - set of relations
Instance - actual data at a given time

- *relation-list* : A list of relation names
  - possibly with a *range-variable* after each name
- *target-list* : A list of attributes of tables in *relation-list*
- *qualification* : Comparisons combined using AND, OR and NOT.
  - Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of =≠<>≤≥
- *DISTINCT*: optional keyword indicating that the answer should not contain duplicates.
  - In SQL SELECT, the default is that duplicates are *not* eliminated! (Result is called a "multiset")

Inner Join - occurs in both tables
FULL Outer Join - occurs in either table, if not in both there are NULL
Left Outer Join - occurs in right table but not left table
Left Semi Join - at least one representation in left table joins with right table
Cross Join - cross product, all occurrences of both tables combined
Normalization - minimizing data redundancy
GROUP BY - aggregate data
SQLite - fixed number of named columns of specified type
Pandas:

- **Series**: named, ordered dictionary
- **DataFrame**: a table with named columns
- **Operations**: Map, filter, sort/group by, aggregate, pivot, relational operations

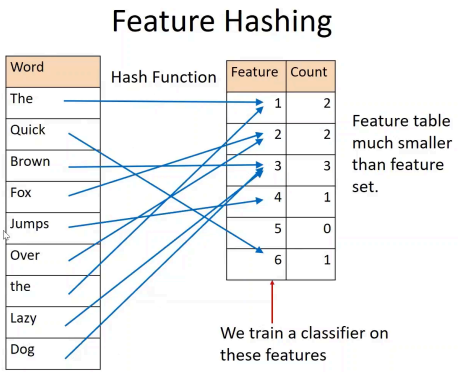
OLAP DBs - stores **aggregates** of data values along many dimensions
NoSQL
Key-value stores
Hive

Data Modeling - Featurization:

Featurization: mapping data from raw data to feature set
Sample should represent entire population, but also structure of sample itself (if neither, we see **over-fitting**)
Train/test split - can lead to over-fitting still
Train/validation/test split - validation is used to test initially, so that the test data isn't seen
ROC (Receiver-Operating-Characteristic) - Area(ROC) -> 1, better model
Lift plot - difference between actual classifier and random classifier
Featurization:

- **Method 1: Ablation -** remove a feature and determine if performance is significantly worse (t-test Q_0 and Q_1) (bootstrap sampling)
- **Method 2: Mutual information -** knowledge of one feature influences the distribution of another, rank features and keep highest
- **Method 3: CHI-Squared -** captures difference between expected and actual distributions (the higher the score, the more correlated the feature is)

Feature Hashing



They work well because the false positive cost (target dog ads to non-dog-lovers) is low compared to false negative cost (miss opportunity to target a dog-lover)
Trying to create bins which have some type of meaning
Important application of feature hashing is to interaction features (NGrams)

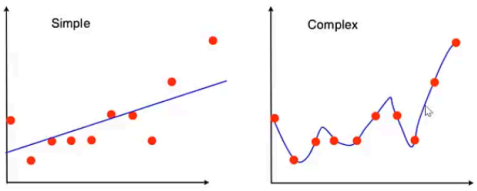
Supervised Learning - KNN and Naive Bayes:

Supervised: Given input/output labels, classification (class labels) or regression (continuous)
Unsupervised: Given only samples of data, clustering (discrete) or matrix factorization/kalman filtering/unsupervised neural networks (continuous)
Predicting from samples: We are most interested in the population, but we only have access to a sample, so we want a model which is generalizable

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.
i.e. $$Bias = \mathrm{E}[f_D(X) - y]$$
E[] is taken over points X and datasets D

Variance: if we train models $f_D(X)$ on many training sets D, variance is the variance of the estimates:
$$Variance = \mathrm{E}\left[\left(f_D(X) - \bar{f}(X)\right)^2\right]$$
Where $\bar{f}(X) = \mathrm{E}[f_D(X)]$ is the average prediction on X.

Complex models: More parameters, lower bias, higher variance
Simple models: Few parameters, higher bias, lower variance



Total expected error: Bias^2 + Variance
Variance strongly dominates, **overfitting** vs **bias** strongly dominates, **underfitting**
k-Nearest-neighbors: Find the k most similar data given a query
k-NN issues: the data is the model, so accuracy generally improves with more data, minimal configuration (k-number of neighbors, weighting of neighbors-inverse distance, similarity metric)
k-NN for classification and regression
k-NN distance measures: euclidean distance (simple/fast), cosine distance (good for documents/images), jaccard distance (sets), hamming distance (strings), manhattan distance (coordinate-wise distance)
Small k →low bias, high variance, **large k** →high bias, low variance
Choosing k: cross validation (60/20/20%), predict, tune, evaluate

The curse of dimensionality: phenomena that occur in high dimensions that do not occur in low dimensionality (in particular, data in high dimensions are sparser than in lower dimensions)
Bayes' Theorem - prior probability, conditional probability, posterior probability
Laplace Smoothing
Good of NB Classifiers - Simple and fast, compact model, very well-behaved numerically, can work well with sparse data
Bad of NB Classifiers - Subject to error and bias, can't model patterns, typically not as accurate

Supervised Learning - Regression, SVM, SGD, Decision Trees:

Linear Regression - least squares fit, gaussian noise, gradient descent, r^2
Logistic Regression - used with regularization, generalized naive bayes (usually better than naive bayes)

- Bayes rule for two classes c and $\neg c$:

$$\Pr(c|X) = \frac{\Pr(X|c)\Pr(c)}{\Pr(X)} = \frac{\Pr(X|c)\Pr(c)}{\Pr(X|c)\Pr(c) + \Pr(X|\neg c)\Pr(\neg c)}$$

- Dividing by the numerator:

$$= \frac{1}{1 + \frac{\Pr(X|\neg c)\Pr(\neg c)}{\Pr(X|c)\Pr(c)}}$$

Linearly separable data - separate classes by a line, plane, etc.
Perceptron algorithm - cycle through points, and if misclassified, adjust function

Algorithm:

- Set t=1, start with all-zeroes weight vector w_1.
- Given example x, predict positive iff $w_t \cdot x \geq 0$.
  - On a mistake, update as follows:
    - Mistake on positive, update $w_{t+1} \leftarrow w_t + x$
    - Mistake on negative, update $w_{t+1} \leftarrow w_t - x$

Best W - maximum margin solution (most stable under perturbations of inputs)
Support Vector Machine (SVM) - maximizing the distance between support vectors

- Since $\mathbf{w}^\top \mathbf{x} + b = 0$ and $c(\mathbf{w}^\top \mathbf{x} + b) = 0$ define the same plane, we have the freedom to choose the normalization of $\mathbf{w}$

- Choose normalization such that $\mathbf{w}^\top \mathbf{x}_+ + b = +1$ and $\mathbf{w}^\top \mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively

- Then the margin is given by

$$\frac{\mathbf{w}}{||\mathbf{w}||} \cdot (\mathbf{x}_+ - \mathbf{x}_-) = \frac{\mathbf{w}^\top (\mathbf{x}_+ - \mathbf{x}_-)}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$$

- Learning the SVM can be formulated as an optimization:

$$\max_{\mathbf{w}} \frac{2}{||\mathbf{w}||} \text{ subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{cases} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases} \text{ for } i = 1 \ldots$$

- Or equivalently

$$\min_{\mathbf{w}} ||\mathbf{w}||^2 \text{ subject to } y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 \text{ for } i = 1 \ldots N$$

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

Slack - (ignore data point) in general there is a trade off between margin and mistakes on training data
Application - object detection
Newton's Method - Iterative approach to optimization (can struggle finding global optimal)

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

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.

L-BFGS (limited memory) - Gradient descent (not as accurate as newton, but much more efficient)

Recall that we have $f : \mathbb{R}^n \to \mathbb{R}$, convex and differentiable, want to solve

$$\min_{x \in \mathbb{R}^n} f(x),$$

i.e., find $x^\star$ such that $f(x^\star) = \min f(x)$

Gradient descent: choose initial $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

t_k : learning rate

Stochastic Gradient Descent (SGD) - subset of X (not as accurate), but eventually converge (takes more steps but still quicker), or randomly choose subset of features, limitations when talking about feature frequency (where one learning rate doesn't work for all features)

AdaGrad (adaptive-rate SGD) - smooths modifications by magnitude

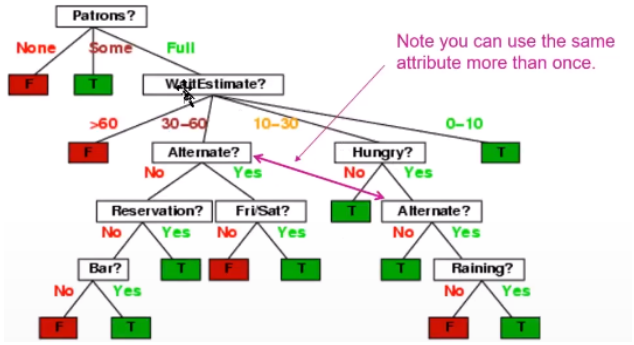
With ADAGRAD gradients are scaled by

$$\frac{1}{\sqrt{\sum_{k=1}^{t} {g_k}^2}}$$

Which is proportional to $t^{-0.5}$ assuming gradient magnitudes are roughly constant. This **inverse power rate** is quite efficient for "easy" learning problems like linear or logistic regression.

SGD learning rate - exponential schedule, constant schedule, linear schedule

Decision Trees:



Decision Tree Learning - might have 2^k nodes, can do greedy approach to recursively choose decision feature for each node

Choosing an attribute - ideally a good feature splits examples into subsets that are "all positive" or "all negative"

Entropy - captures the lack of structure in data (low means more structure/more information)

Information Gain

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)

Choosing Best Features

At each node, we choose the feature f which **maximizes the information gain**.

This tends to be produce mixtures of classes at each node that **are more and more "pure"** as you go down the tree.

If a node has examples all of one class c, we make it a leaf and output "c". Otherwise, when we hit the depth limit, we output **the most popular class** at that node.

As tree depth increases, **bias** decreases and **variance** increases

Ensemble Methods - Bagging, stacking, boosting (address issue of determining bias vs variance)

- **Bagging:** (binary typically) train learners on different samples then combine by voting
- **Stacking**: (linear regression) combine models using second stage learner
- **Boosting:** (same dataset with weights, repeats process) train learners on filtered output of other learners

Random Forest (bagging ensemble): grow k trees on datasets sampled (bootstrap sampling) from original dataset, draw K bootstrap samples of size N, grow each tree by selecting random m out of p features, aggregate predictions (usually m is sqrt(p) or less)

- We want to take a vote between different learners, so this ensures **diversity** by **bootstrapping** and **randomizing** features
- **Pros**: very popular, easy to implement, parallelizes easily
- **Cons**: not state-of-the-art accuracy, needs many passes over data, easy to overfit

Boosted Decision Trees: RF's are trained independently, but these are trained sequentially

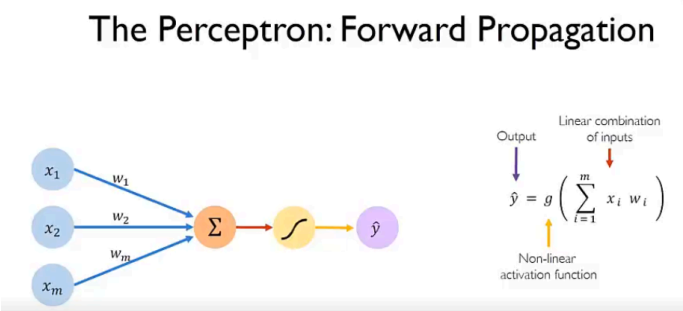
Random forests vs Boosted Trees: RF 10s of deep large trees (variance reduction through aggregate), BD uses 1000s of shallow small trees (bias reduction through boosting, variance already low)

Deep Learning:

Big data, hardware, software

Stack perceptrons to form neural networks

Forward propagation:



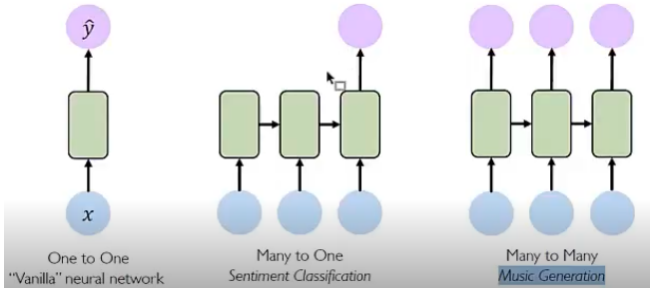
Activation functions: example, sigmoid function (given inputs try and return corresponding output), allow us to introduce non-linearities
Hidden layer: can't leverage control on inputs and outputs
Quantifying loss: takes predicted and actual labels
Binary cross entropy loss (measures difference between predicted and expected), mean squared error loss
Loss optimization: achieve the lowest loss, take small step in opposite direction of gradient, repeat until convergence; initialize weights randomly, loop until convergence (compute gradient, update weights), return weights
Computing gradients - backpropagation
Loss functions can be difficult to optimize: learning rate too small (local min), learning rate too big (won't converge); dealing with this (try different learning rates), (adaptive learning rate - Adam, momentum, adadelta, adagrad)
Stochastic gradient descent: instead of computing gradient for every data point, pick random sample, compute gradient, update weights, until convergence (much faster but less accurate) each step might be slightly off, and will take more steps, but is still faster overall (MINIBATCH)
Overfitting - regularization: technique used to discourage complex model (Regularization 1: Dropout, randomly set some activations to 0) (Regularization 2: Early stopping, stop training before overfitting)
Predicting next word:

- Idea 1: Use fixed window (problem, can't model long-term dependencies)
- Idea 2: Use entire sequence as set of counts (problem, counts don't preserve order)
- Idea 3: Use really big fixed window (problem, no parameter sharing - issues with shuffle)

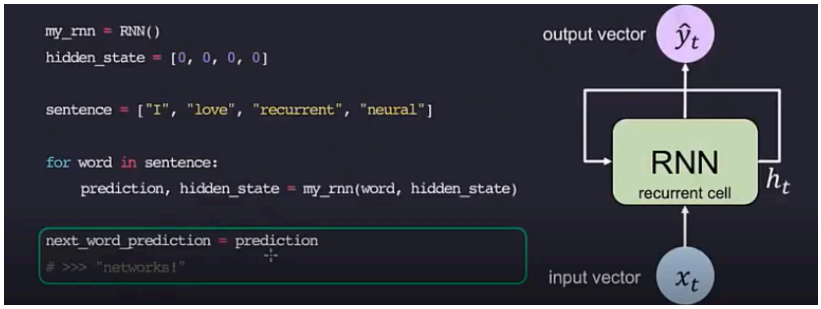
To model sequences, we need to:

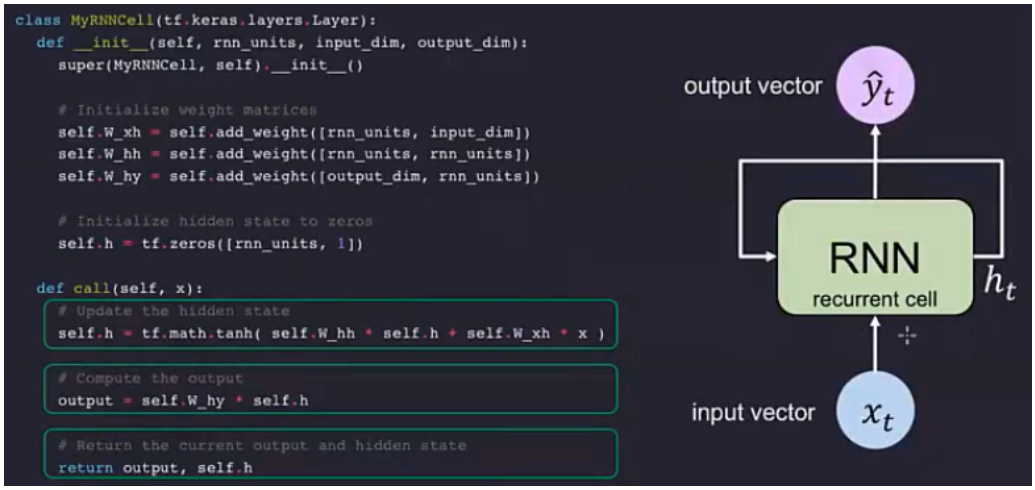
- Handle **variable length** sequences
- Track **long-term dependencies**
- Maintain information about **order**
- **Share parameters** across sequence

Recurrent Neural Networks (RNN):



- Maintains internal state which keeps information already received
- Provide input, parameters, AND old state





Backpropagation Through Time (BPTT)

Vanishing/exploding gradients

- Trick 1: activation functions
- Trick 2: parameter initialization
- Trick 3: gated cells (LSTM)

Long short term memory (LSTM): contain interacting layers that control information flow; information is added or removed through layers called gates

RNN Applications: music generation (input: sheet music, output: next character), sentiment classification (input: sequence of words, output: probability of positive sentiment), machine translation

RNNs:

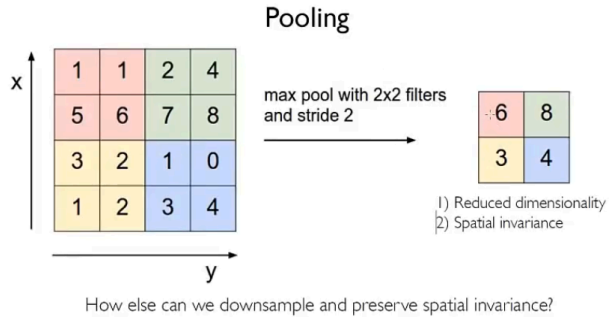
- Well suited for sequence modeling
- Model sequences via recurrence relations
- Train with backprop through time
- Gated cells like LSTMs let us model long-term dependencies
- Models for music generation, classification, machine translation

Computer Vision:

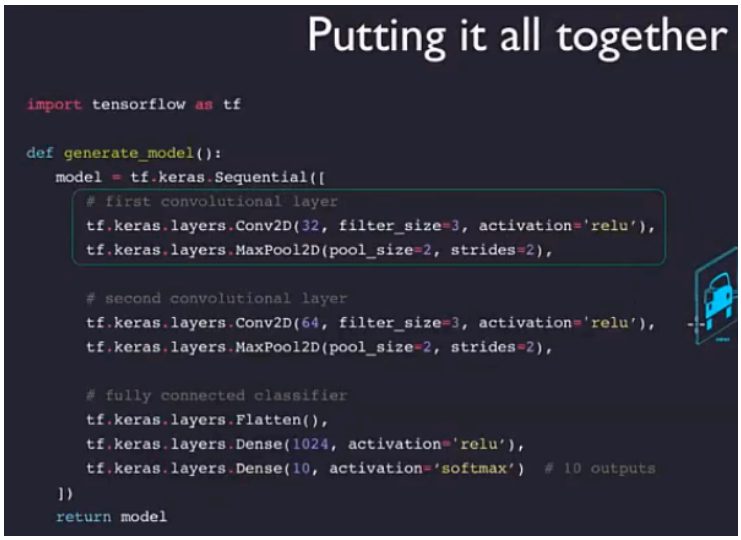
- Matrix of rgb values
- Hierarchy of features
- Using spatial structure
- Apply set of weights to extract local features, use multiple filters, spatially share parameters
- Feature map

Convolution Neural Network (CNN):

1. Convolution, apply filters with learned weights to generate feature maps
2. Non-linearity, often ReLU
3. Pooling, downsampling operation on each feature map



How else can we downsample and preserve spatial invariance?



R-CNN: object detection, image captioning

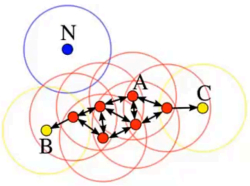
Unsupervised Learning:

Clustering - Why? Underlying process, segmentation, compression
Cluster bias - Humans conceptualize the world in groups, maybe it is more used than it should be
Continuum vs Cluster
Hierarchical clustering (clusters form hierarchy and can be computed bottom-up/top-down)**, flat clustering** (no inter-cluster structure)**, hard clustering** (items assigned to a unique cluster)**, soft clustering** (cluster is real-valued function)
K-means:

- Based on euclidean distance traditionally (intra-cluster measure)
- Greedy approach (Lloyd's algorithm) finds closest cluster center for each item, assigns it to the cluster, and then recomputes the cluster centroid as the means of items for the newly-assigned items in the cluster
- Iterate for fixed number of iterations or until no/little change is being made
- Random sample (pick random subset k) or K-Means++ (iteratively construct random sample with good spacing; more chance to points farther away)
- Finding an exact solution is difficult
- Solution isn't optimal and varies by initial points
- Performance is O(nk) per iteration, n(iterations), k(# of clusters)
- Favors a spherical shape

DBSCAN:

- Looks to more density based clustering
- Core points have at least 'minPts' neighbors in a sphere of diameter epsilon around them



- **Core points** can **directly reach** neighbors in their ε-sphere.
  - By definition, non-core points cannot directly reach another point.
- Point q is **density-reachable** from p if there is a series of points $p=p_1,p_2,\ldots p_n=q$ s.t. p_{i+1} is directly reachable from p_i
- All points not density-reachable from any other points are **outliers**.

- A cluster is a set of points which are mutually **density-connected**

DBSCAN Algorithm

```
DBSCAN(D, eps, MinPts) {
   C = 0
   for each point P in dataset D {
      if P is visited
         continue next point
      mark P as visited
      NeighborPts = regionQuery(P, eps)
      if sizeof(NeighborPts) < MinPts
         mark P as NOISE
      else {
         C = next cluster
         expandCluster(P, NeighborPts, C, eps, MinPts)
      }
   }
}
```

- O(n logn)
- If data doesn't fit in memory, O(n^2)

Performance:

- Dataset size (model quality improves with size), model size (bigger models perform better generally)
- Offline: Model training, Online: Model prediction
- Algorithmic improvements, computational improvements, cluster scale-up

Unsupervised Learning - Deep Generative Models:

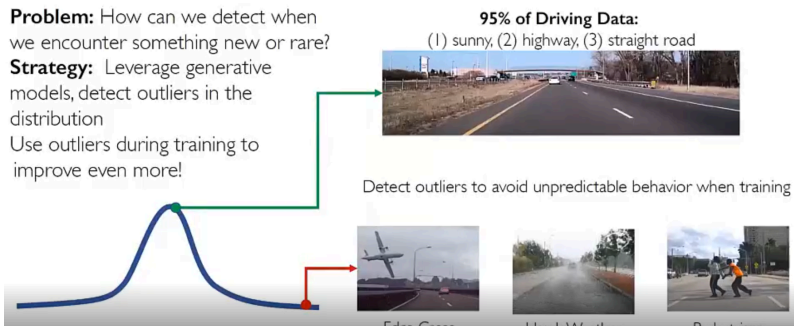
Unsupervised:

- **Data:** x, no labels
- **Goal:** Learn some hidden or underlying structure of the data

Supervised:

- Learn function to map x → y

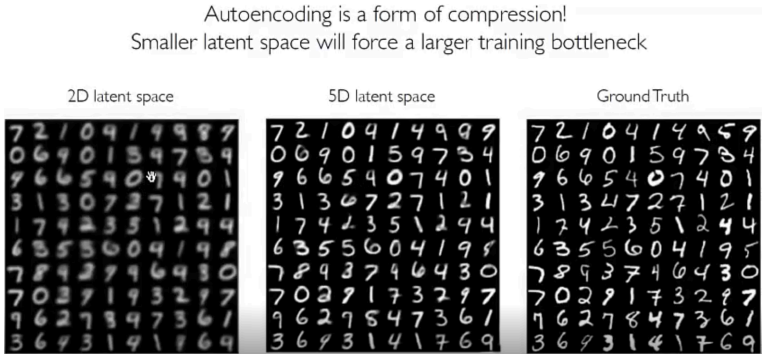
Generative modeling:

- Take as input training samples from some distribution and learn a model that represents distribution
- **Debiasing:** capable of uncovering underlying latent variables in a dataset
- **Outlier detection:** leverage generative models to detect outliers so we can know if we encounter something new or rare
- 

Latent variable: you have observed data, but you want to know the variables and structure that generates behavior

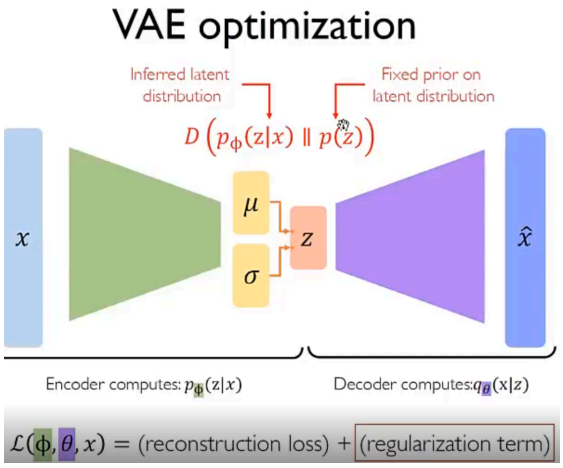
Autoencoders:

- **Encoder:** Learning a lower-dimensional feature representation from unlabeled training data; learns mapping from data x to low-dimensional latent space z
- **Decoder:** Learning a mapping back from latent space z to a reconstructed observation x_hat
- $\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$

- 

- **Bottleneck hidden layer** forces network to learn compressed latent representation
- **Reconstruction loss** forces latent representation to capture as much information about data as possible

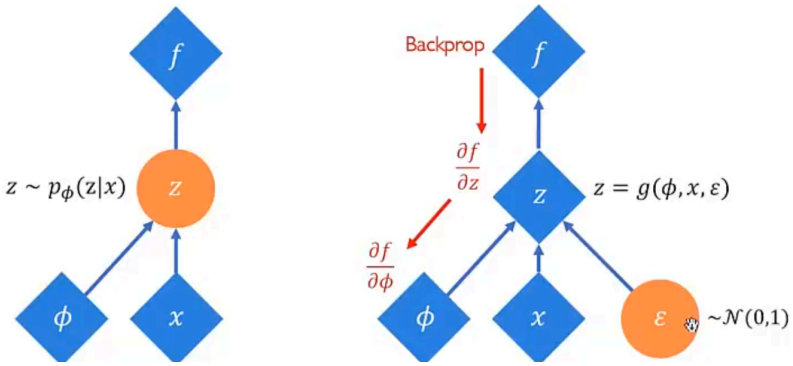
Variational Autoencoders (VAEs):

- **Deterministic approach** can perfectly reconstruct, but leads to overfitting, hence VAEs
- Using a stochastic approach learns distribution (normal) of features instead of features themselves
- Probabilistic approach to autoencoders, having mean and st dev vectors
- 

- **Common choice of prior:**

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

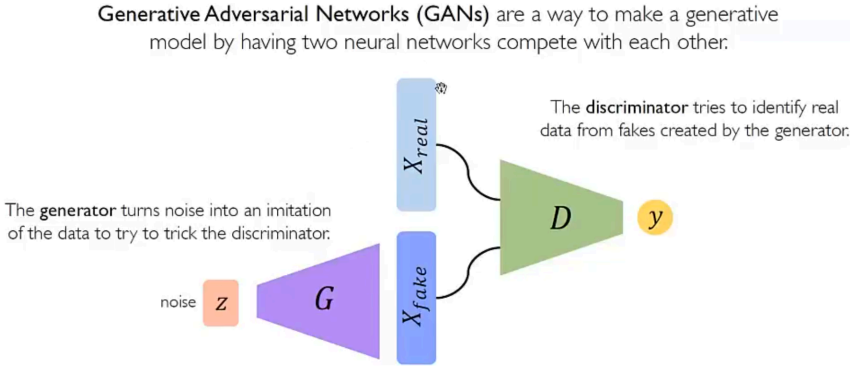
- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (ie. memorizing the data)
- $$= -\frac{1}{2}\sum_{j=0}^{k-1}\left(\sigma_j + \mu_j^2 - 1 - \log \sigma_j\right)$$ KL-divergence between the two distributions
- **Problem:** we can't backpropagate gradients through sampling layers

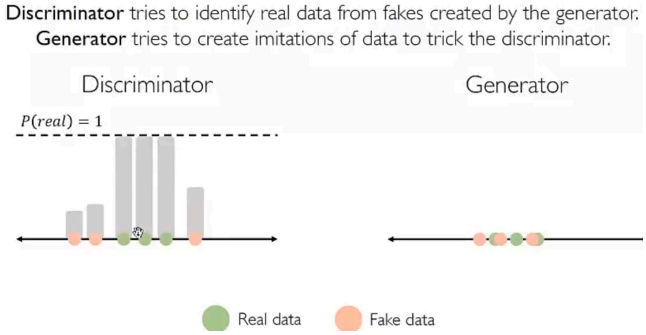


- **Latent perturbation:** slowly increase/decrease a single latent variable and keep other variables fixed (disentanglement, want to detangle latent variables that are correlated with each other)
- **Learn latent structure, estimate distribution, adaptively resample data, learn from fair distribution**
- **Summary:** compress representation, reconstruction allows for unsupervised, reparameterization trick, interpret hidden latent variables, generate new examples

Generative Adversarial Networks (GANs):

- **Idea:** Don't explicitly model density, instead just sample to generate new instances
- **Problem:** sample from complex distribution
- **Solution:** sample from something simple (noise), lean a transformation to training distribution





Train GAN jointly via **minimax** game:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

Discriminator wants to maximize objective s.t. $D(x)$ close to 1, $D(G(z))$ close to 0.
Generator wants to minimize objective s.t. $D(G(z))$ close to 1.

- **Recent advances:** training GANs at different resolutions, CycleGAN (learns transformations across domains with unpaired data), style-based transfer, transforming speech

Visualization:

Why? Gain insight, provide qualitative overview, search patterns, help find regions and suitable parameters, provide a visual of proof
A: support reasoning (analysis), **B:** inform and persuade (communication)
Principles:

- **Simplify** (remove junk, leverage interactive and dynamic charts)
- **Understand magnitudes** (using brightness, areas can be deceptive)
- **Use color** (choose colors based on information, use online resources)
- **Use structure**

Methods:

- Pixel-oriented visualization techniques
- Geometric visualization
- Parallel coordinates
- Icon-based
- Dimensional stacking

Toolkits:

- Chart selection - Juice analytics, Andrew Abela
- Interactive toolkits - D3, Vega, Bokeh