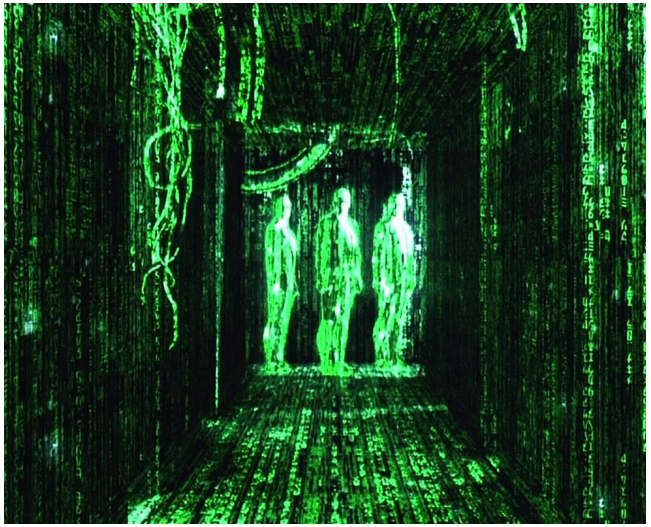
Introduction to Machine Learning 2018

How many are here to become this?

Source: Andrew Ng's book

Machine learning for us

Self-driving cars

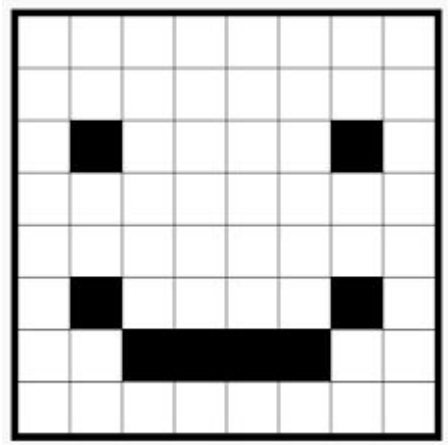


Or the MATRIX!

But real machines are still struggling with this problem

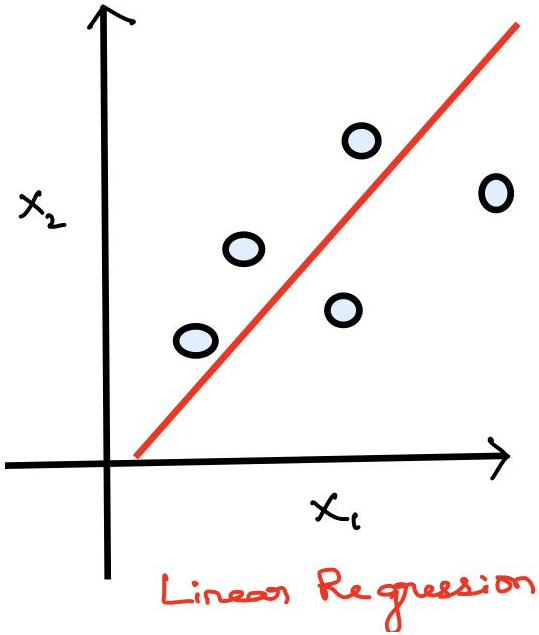
New York Times reported in **1958** that the invention was the beginning of a computer that would "**be able to walk, talk, see, write, reproduce itself and be conscious of its existence.**"

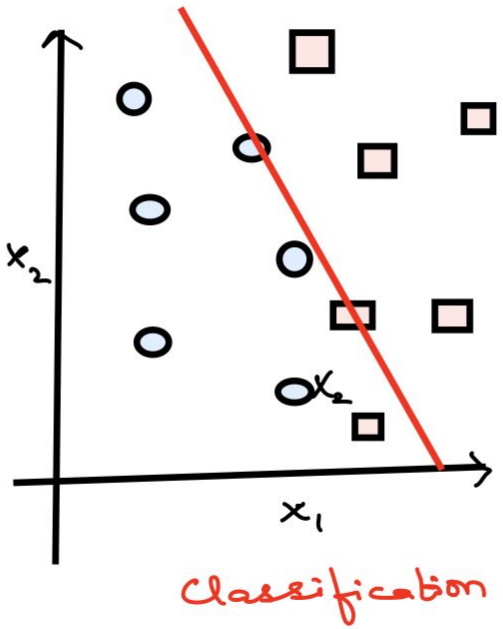
Behind the hood

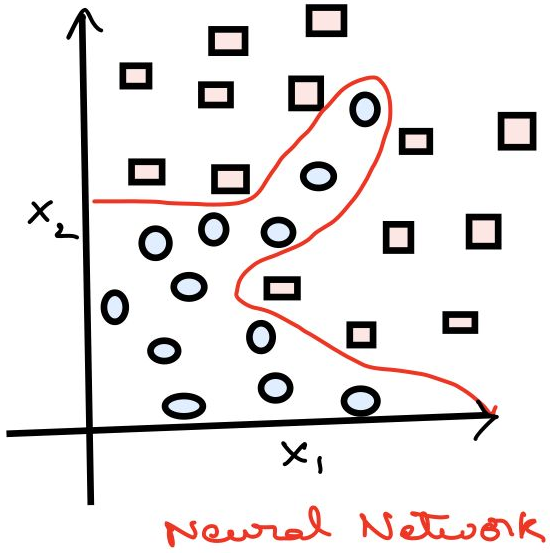


chihuahua

muffin







What is machine learning?

Mathematical hypothesis based on data and use the hypothesis (models) to predict results on new data

Types:

1. Supervised
2. Unsupervised
3. Semi-supervised
4. Reinforcement learning

Common models used in machine learning

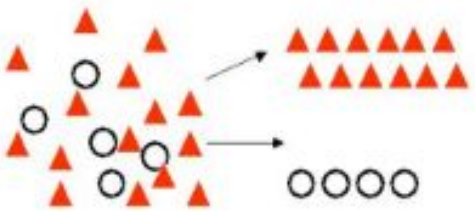
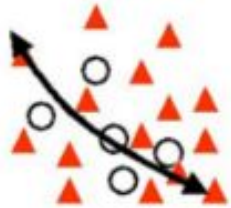
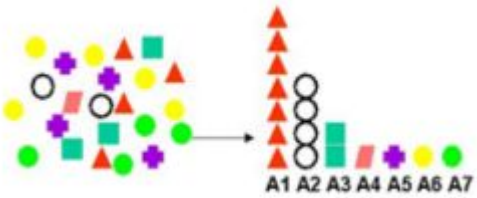
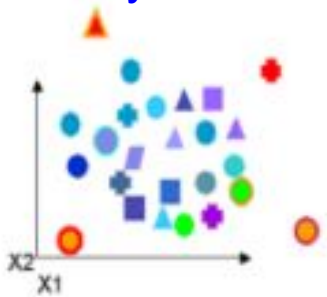
Techniques	Applications
Classification 	Response/ no response yes/ no
Regression 	Continuous numerical outcome
Attribute Ranking 	Finding importance based on strength of relationship with target attribute

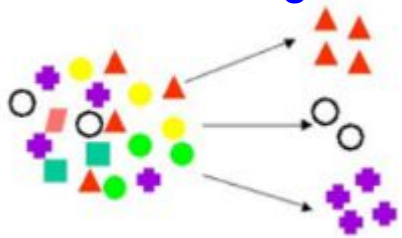
Image source: Futuretext

Anomaly detection



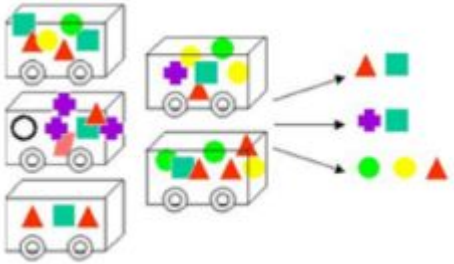
Identifies unusual cases

Clustering



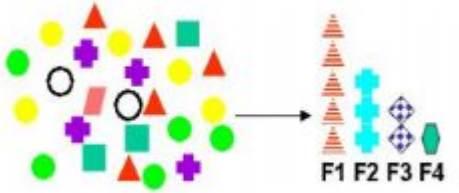
Finding natural groupings

Association



Finds rules associated with naturally co occurring terms

Feature selection



Produces new attributes based on a linear combination of existing attributes

Image source: Futuretext

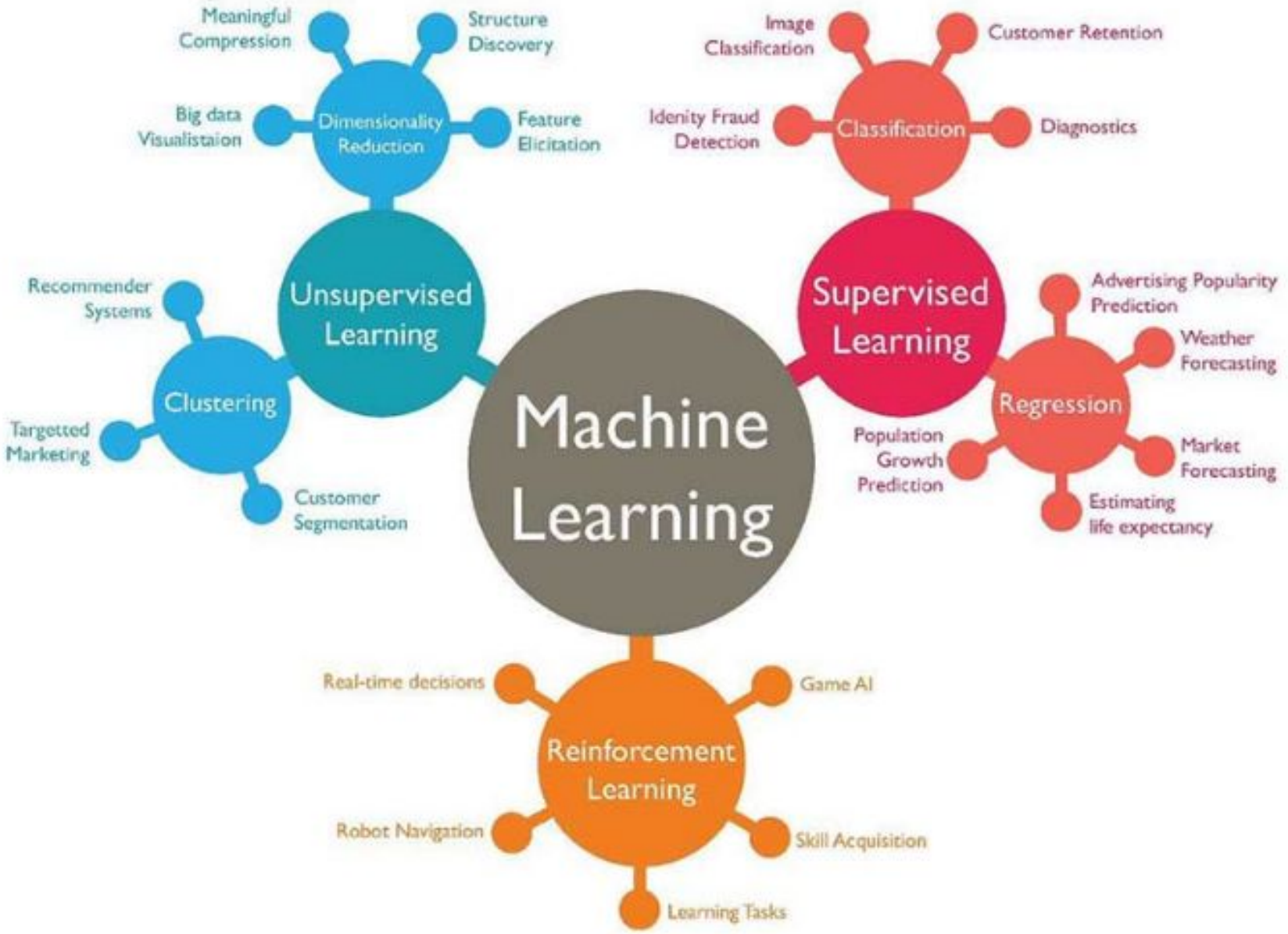
Different ML models

Dimensionality Reduction
Clustering

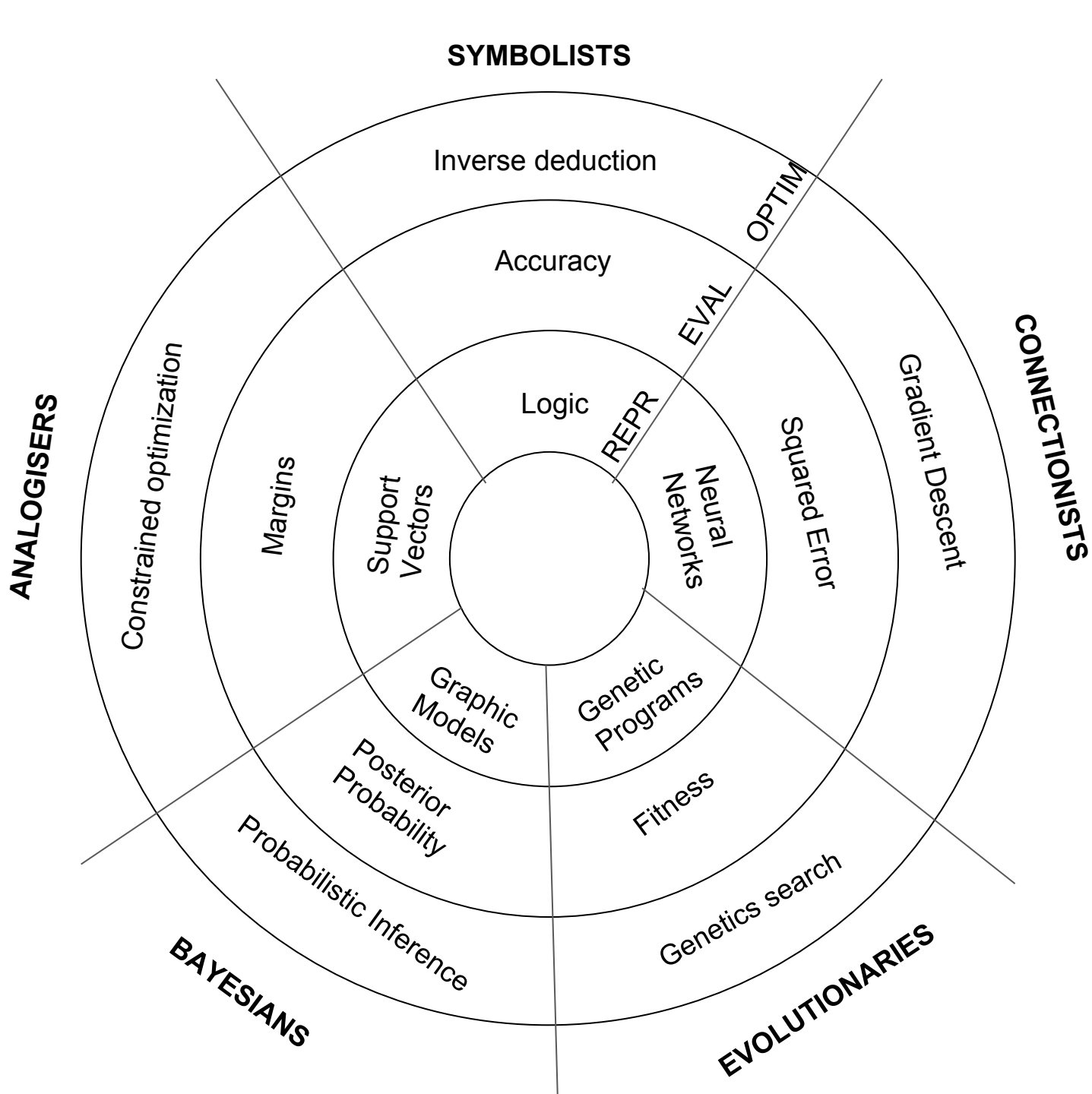
Nearest Neighbours
SVM
Random Forest & Decision Trees

Linear Model & Matrix Algebra
Linear & Nonlinear Regression
Artificial Neural Networks
Deep learning

Type of Machine Learning algorithms



Source: Data Science Central

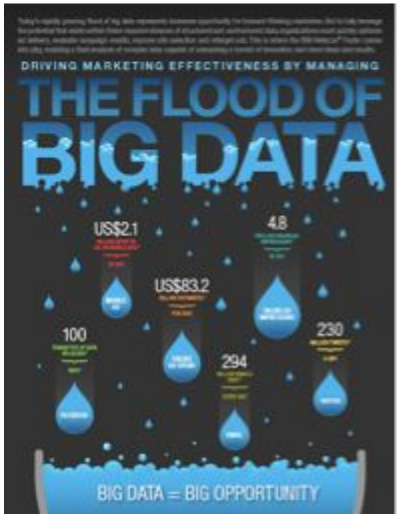

SYMBOLISTS
CONNECTIONISTS
EVOLUTIONARIES
BAYESIANS
ANALOGISERS
Inverse deduction
Gradient Descent
Genetics search
Probabilistic Inference
Constrained optimization
OPTIM
EVAL
REPR
Accuracy
Squared Error
Fitness
Posterior Probability
Margins
Graphic Models
Genetic Programs
Support Vectors
Neural Networks
Logic

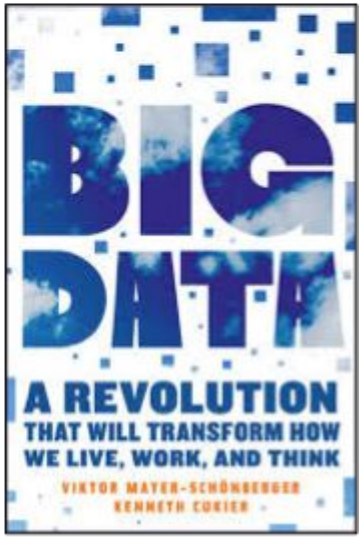
Machine Learning: why now

Big data world

IBM estimates that 90% of world's data has been created in the last two years

Commercial World Data: Financial & Retail Data





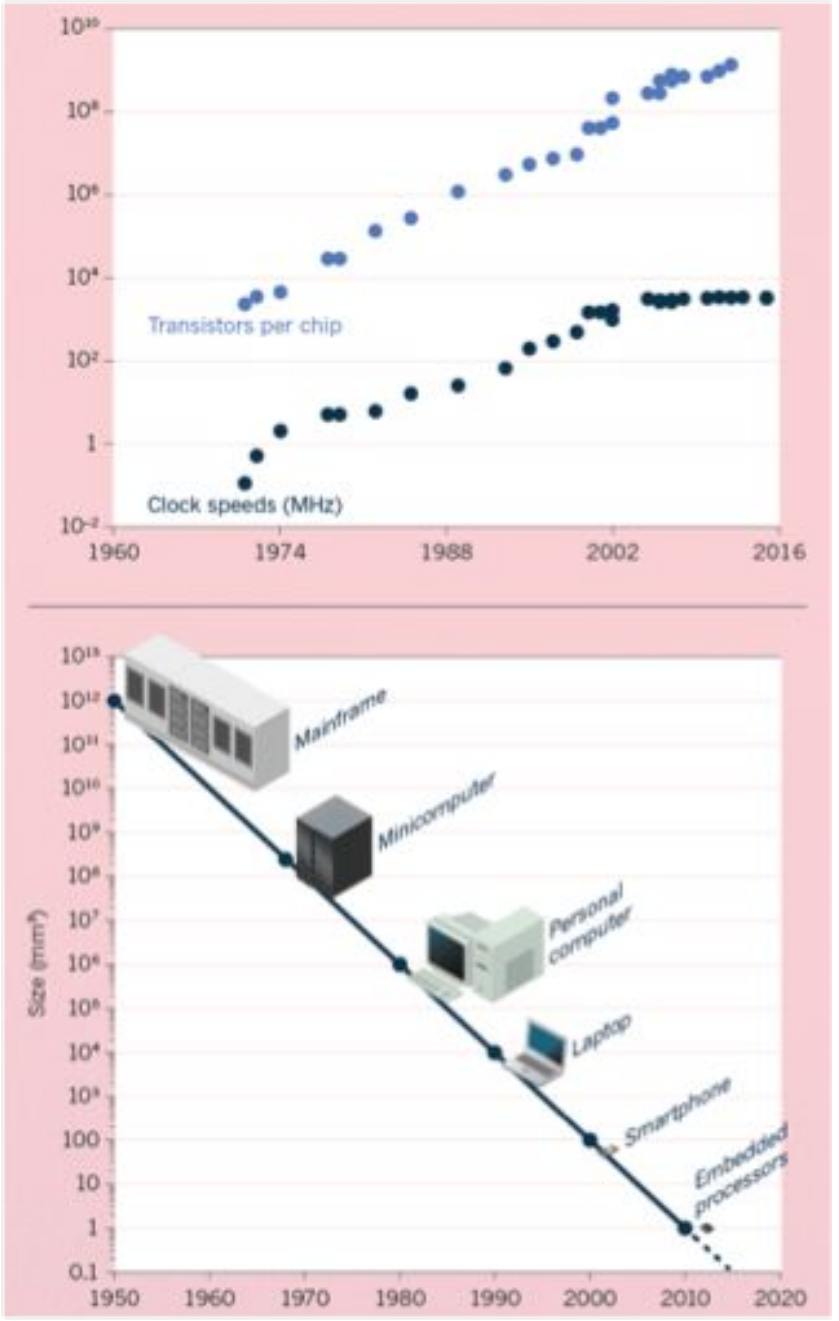
*

Human brain	2.5 PB
Spotify	10 PB
Ebay	90 PB
Facebook	300 PB
Google	15000 PB

***Royal Society website**

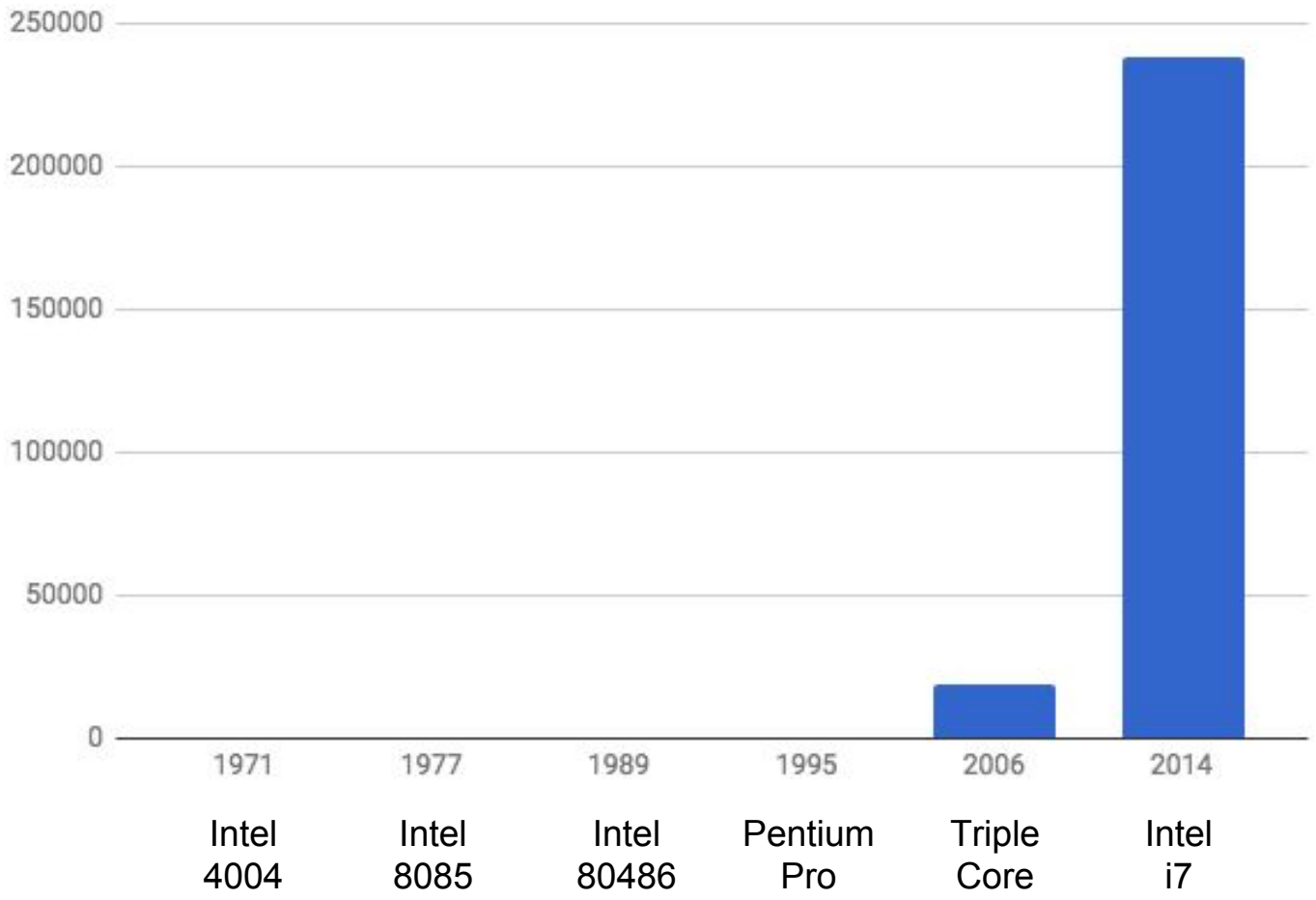
Moore's Law: Exponential Scaling of Computer Technology

- Exponential increase in the number of transistors per chip.
- Led to improvements in speed and miniaturization.
- Drove widespread adoption and novel applications of computer technology.

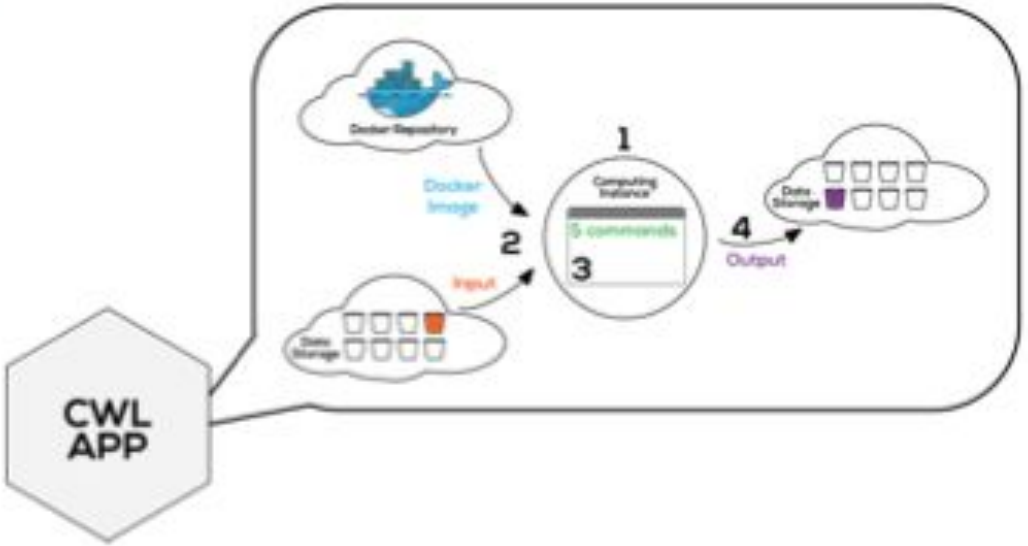


[Waldrop ('15) Nature]

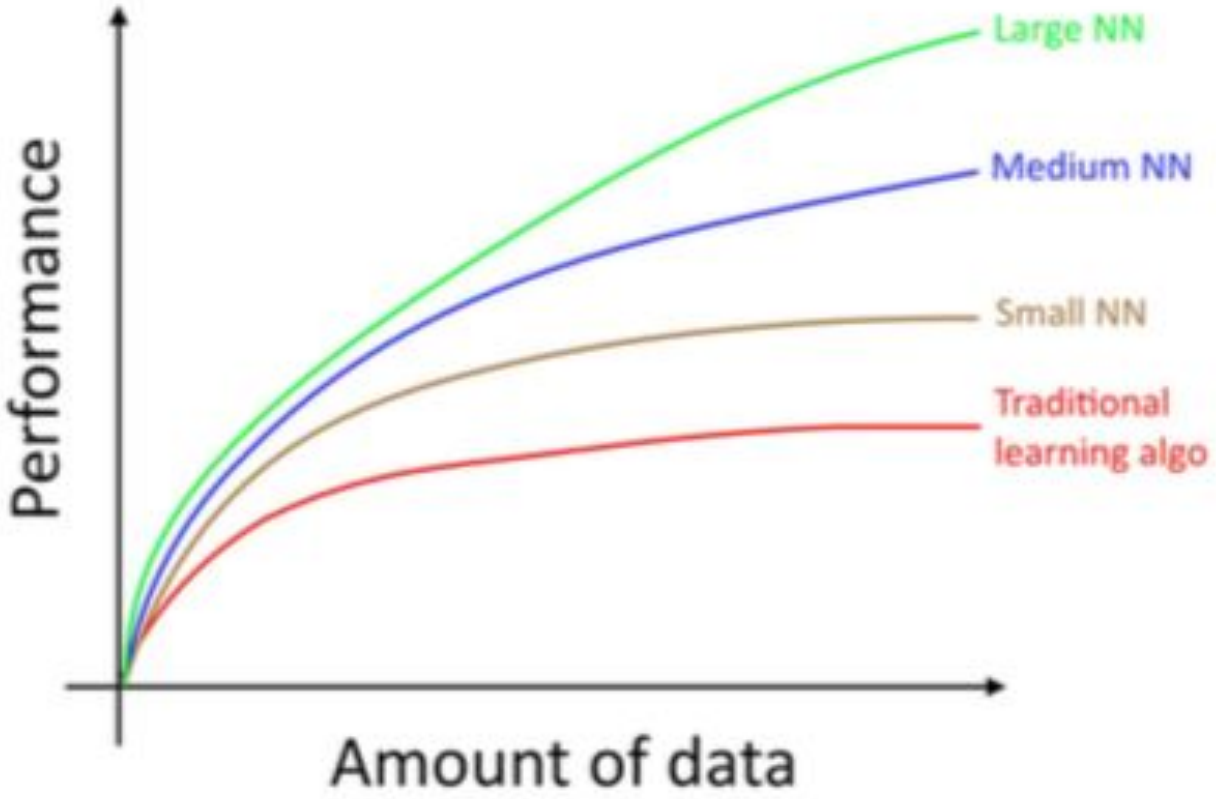
Processing speeds



Cloud computing

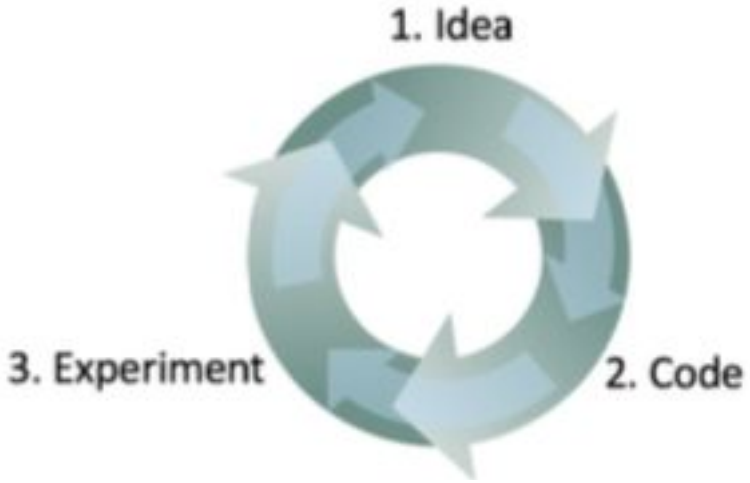


As a result!



Source: Andrew Ng's book

Typical ML algorithm development



Dev/Test set & Metrics

Keep in mind:

1. The distributions should be identical
2. Overfitting dev set
3. The metric is measuring something else
4. Establish metric and dev/test set soon

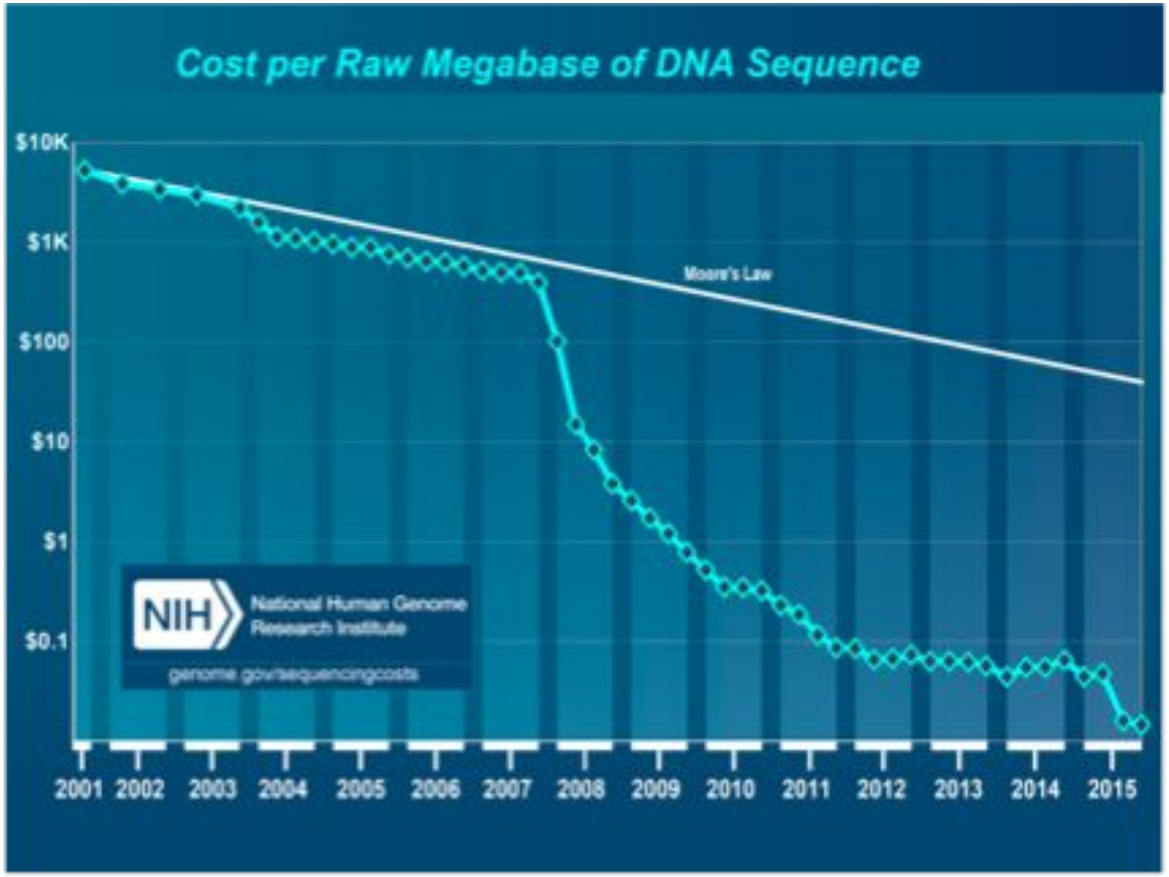
Machine Learning: why biology?

Part III

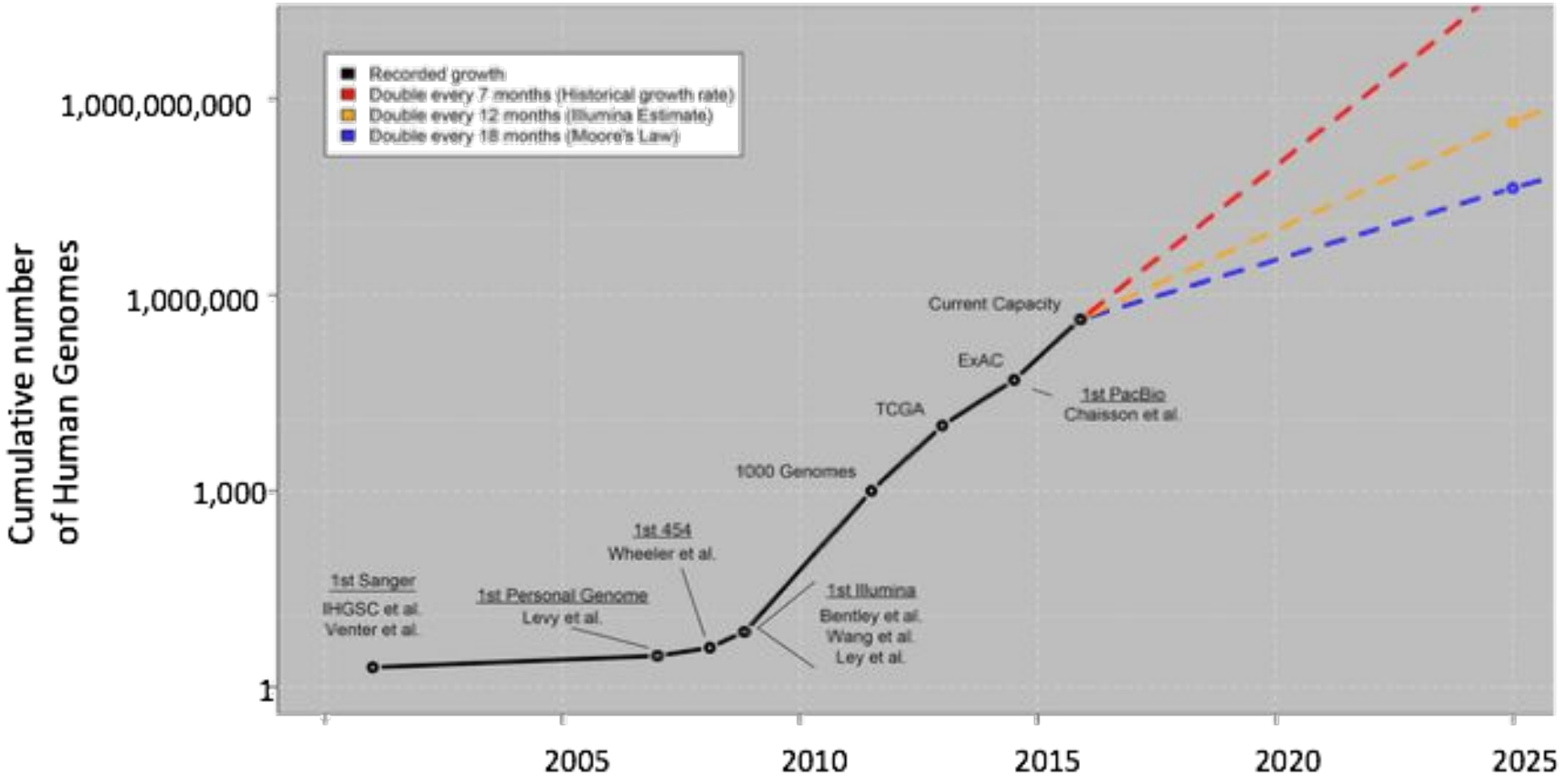
Sequencing data explosion: faster than moore's law over time

DNA sequencing has gone through technological S-curves

- In the early 2000's, improvements in Sanger sequencing produced a scaling pattern similar to Moore's law.
- The advent of NGS was a shift to a new technology with dramatic decrease in cost).

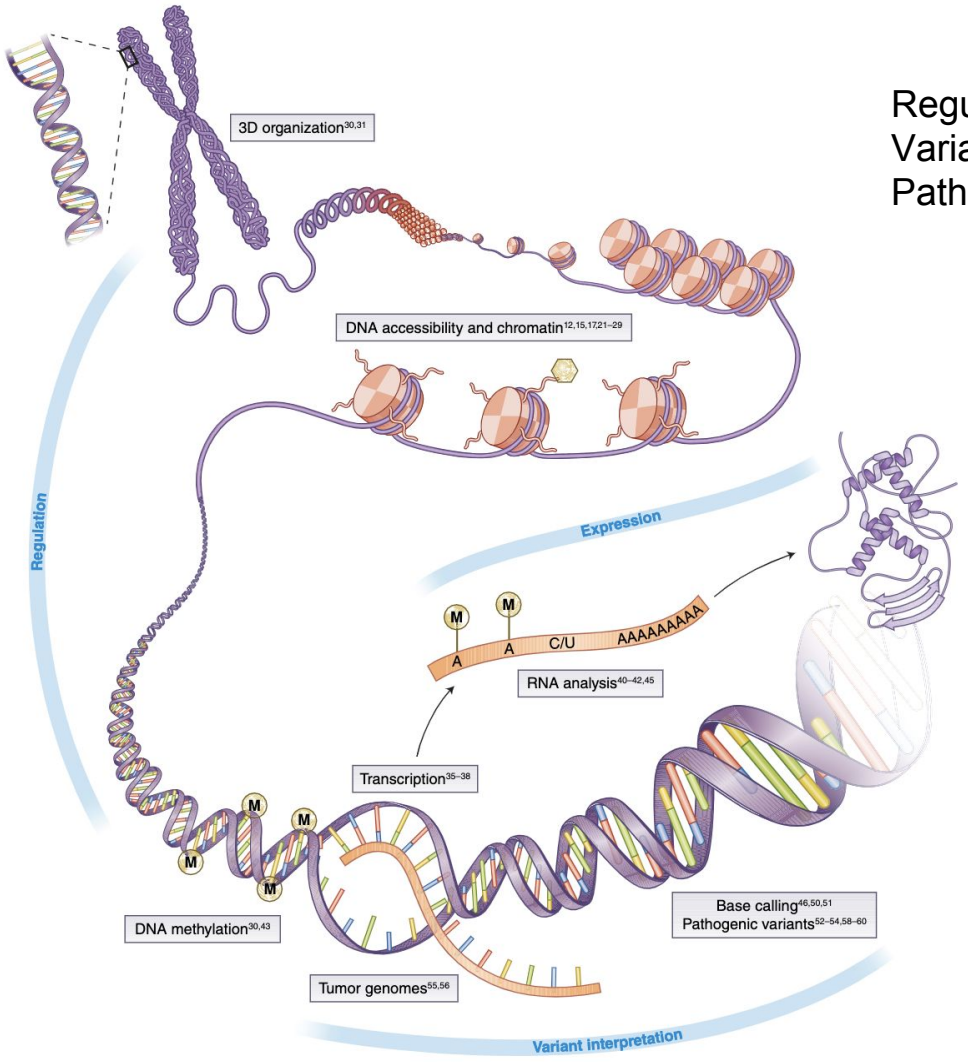


Where are we heading?



By 2019 total NGS data to exceed 2 Exabytes

Applications of Deep learning in genomics

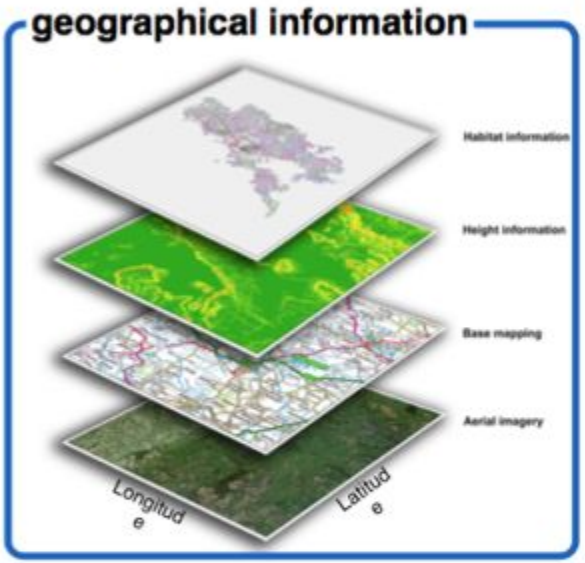


Regulatory genomics,
Variant calling and
Pathogenicity scores.

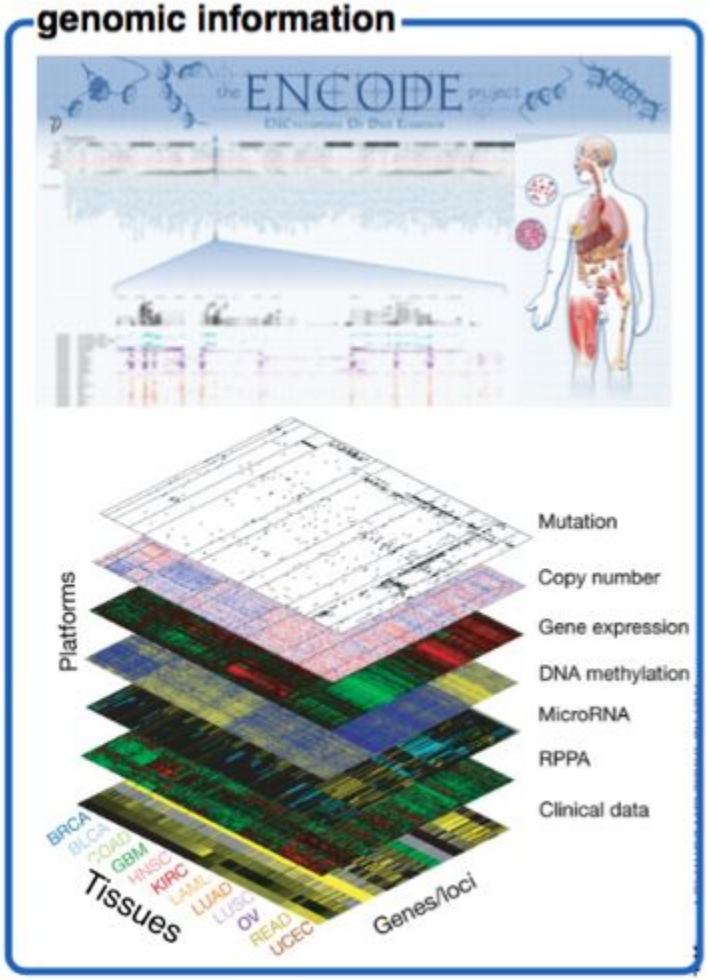
Zou, J. et al., 2018. A primer on deep learning in genomics. *Nature genetics*. Available at: http://dx.doi.org/10.1038/s41588-018-0295-5.

More and more data!

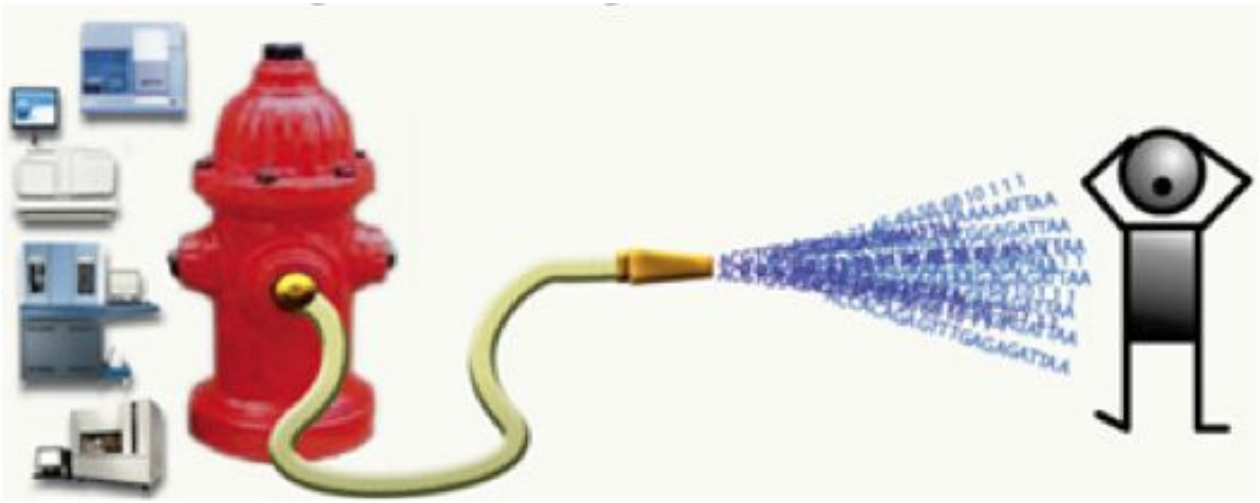
Human genome annotation — a non-intuitive map



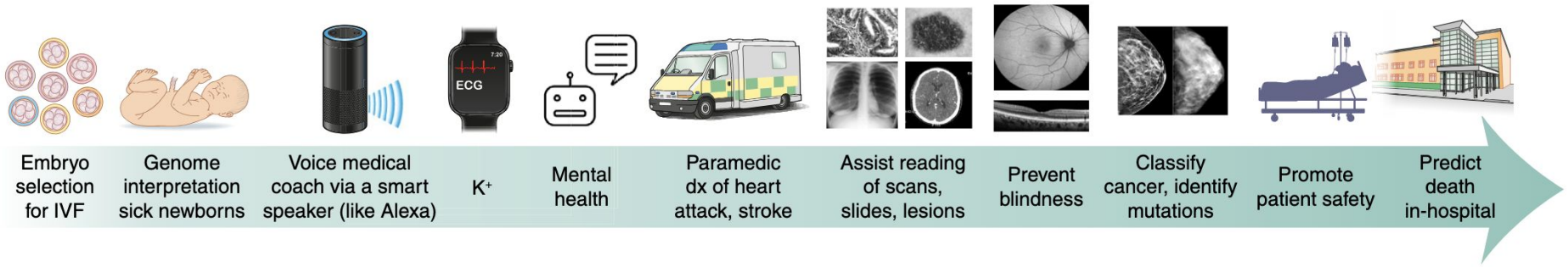
- Large-scale organisation providing an overview of the genome
- Integration of heterogeneous data



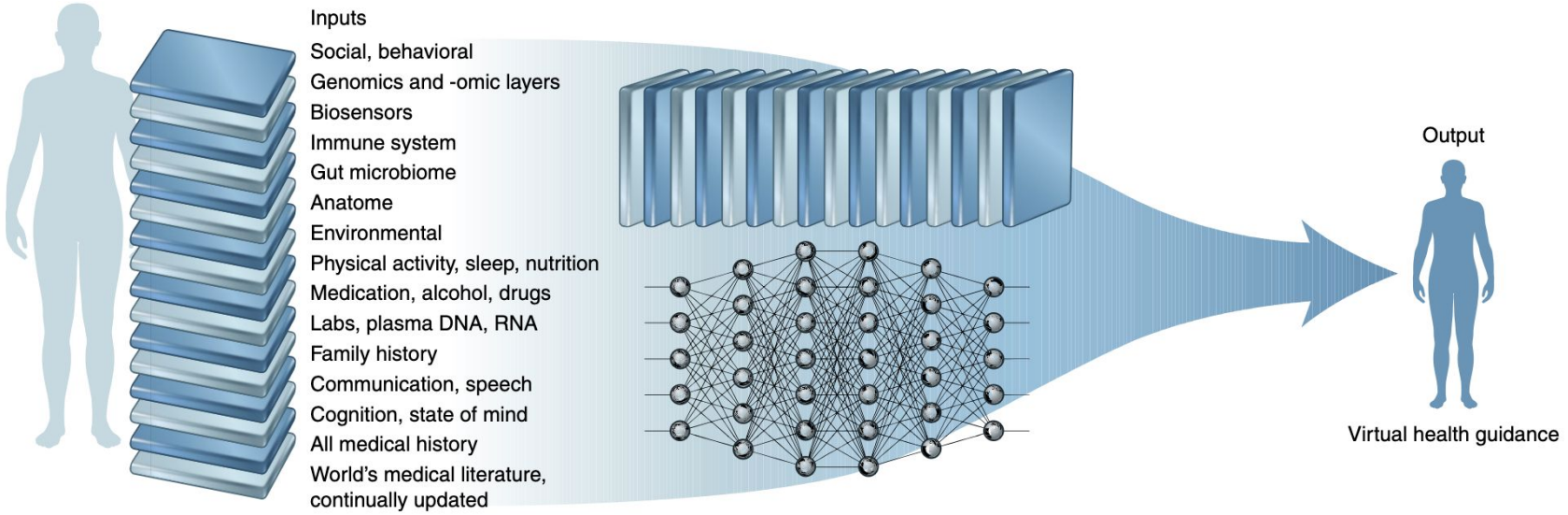
Data overload?



Examples of AI applications across the human life



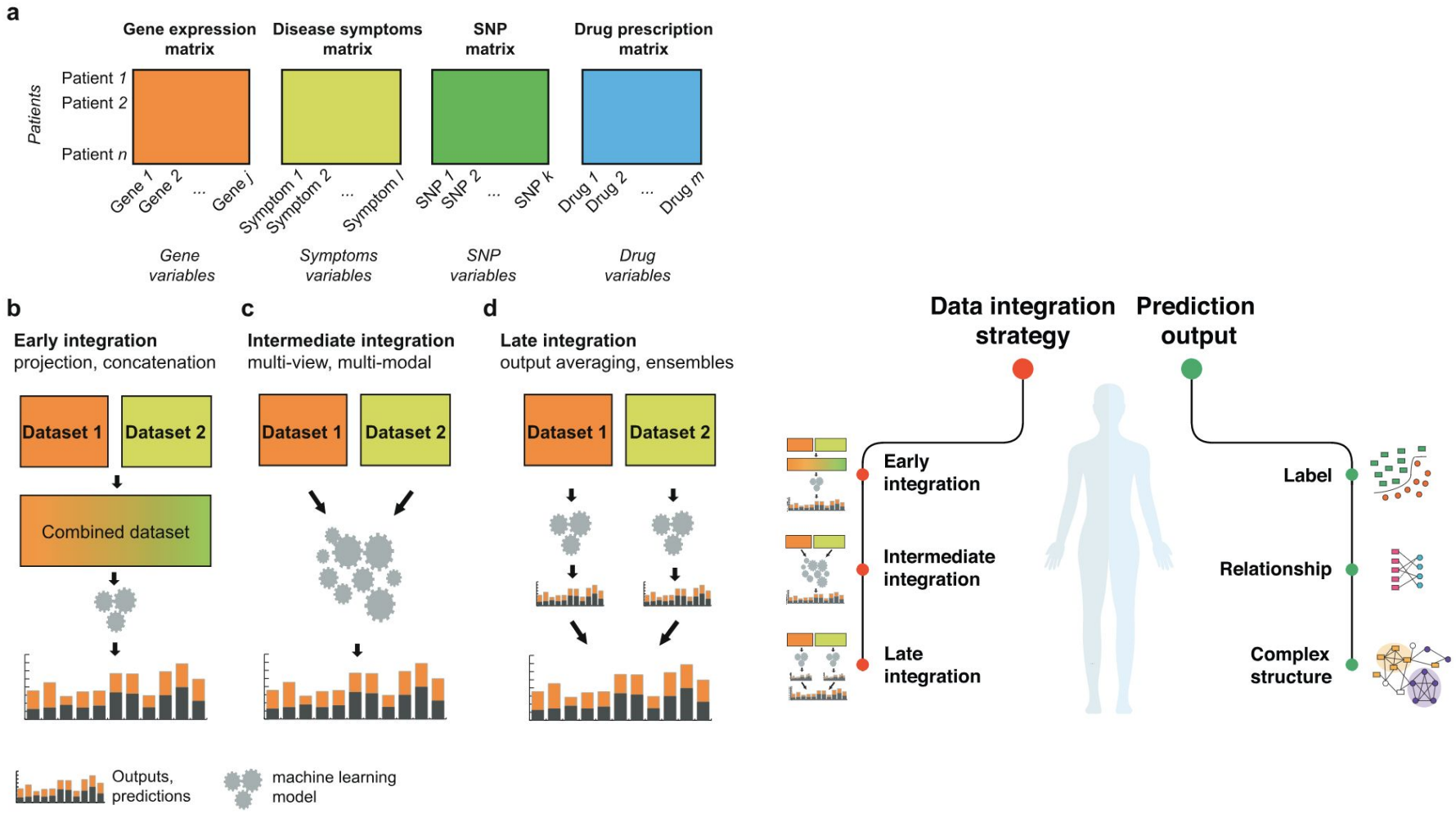
Multi-modal data inputs & algorithms to provide individualized guidance



Topol, E.J., 2019. High-performance medicine: the convergence of human and artificial intelligence. Nature medicine, 25(1), pp.44–56.

Multi-modal data inputs & algorithms to provide individualized guidance

high-dimensional, incomplete, biased, heterogeneous, dynamic, and noisy.



Zitnik, M. et al., 2019. Machine Learning for Integrating Data in Biology and Medicine: Principles, Practice, and Opportunities. An international journal on information fusion, 50, pp.71–91.

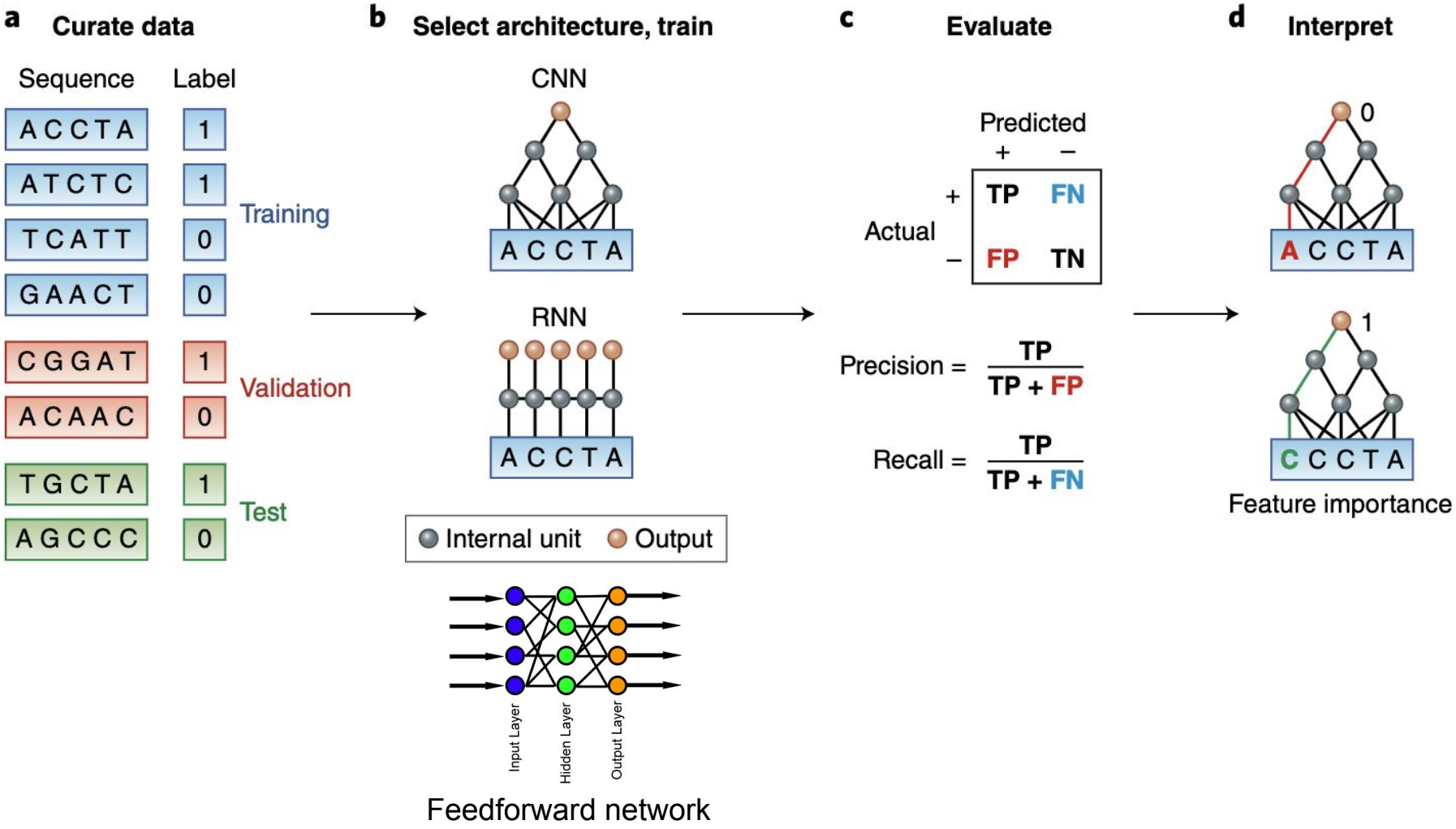
Current and potential AI applications in medicine

Basic biomedical research	Translational research	Clinical practice
Automated experiments	Biomarker discovery	Disease diagnosis
Automated data collection	Drug–target prioritization	Interpretation of patient genomes
Gene function annotation	Drug discovery	Treatment selection
Prediction of transcription factor binding sites	Drug repurposing	Automated surgery
Simulation of molecular dynamics	Prediction of chemical toxicity	Patient monitoring
Literature mining	Genetic variant annotation	Patient risk stratification for primary prevention

Yu, K.-H., Beam, A.L. & Kohane, I.S., 2018. Artificial intelligence in healthcare. *Nature Biomedical Engineering*, 2(10), pp.719–731.

Deep learning workflow in genomics

Deep learning is an umbrella term that refers to the recent advances in neural networks and the corresponding training platforms (e.g.,TensorFlow and PyTorch)



Zou, J. et al., 2018. A primer on deep learning in genomics. *Nature genetics*. Available at: http://dx.doi.org/10.1038/s41588-018-0295-5.

Deep learning resources

All the resources, except for those listed in the 'specific for genomics' block, are relevant for deep learning in general. Because the field is developing rapidly, this information is likely to be considerably different in the future.

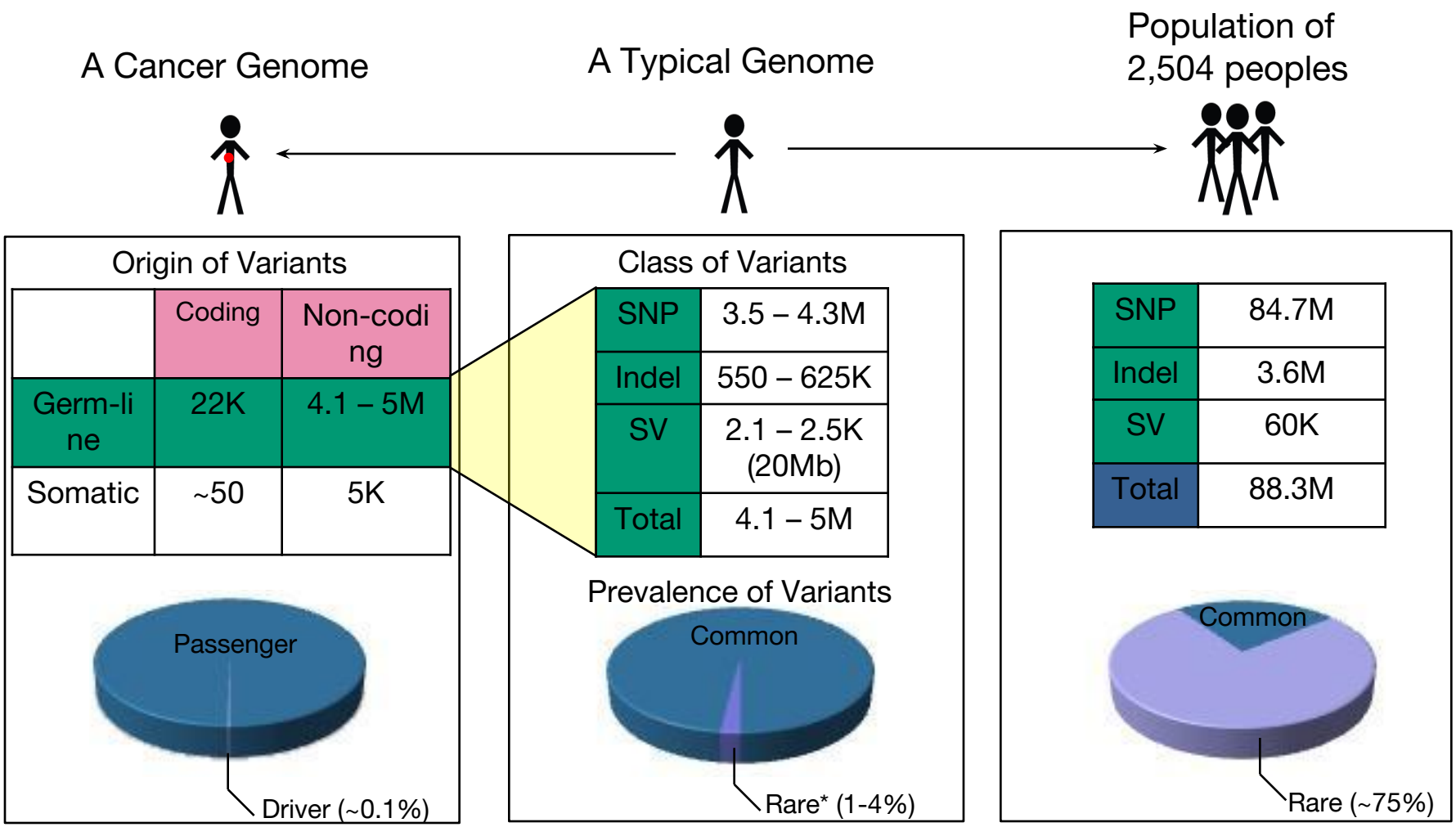
Resource type	Name	URL	Comment
Cloud platform	Amazon EC2	https://aws.amazon.com/ec2/	Most popular cloud platform
	Microsoft Azure	https://azure.microsoft.com/	Second-largest cloud platform
Plug-and-play cloud GPU services	FloydHub	https://www.floydhub.com/	All startups in the GPU service space; pay-by-the-hour model on top of basic monthly subscriptions
	PaperSpace	https://www.paperspace.com/	
	Valohai	https://valohai.com/	
	Google CloudML	https://cloud.google.com/ml-engine/	Can run your own models on Google's hardware, including tensor processor units
	Google Colaboratory	https://colab.research.google.com/	Notebook environment with free GPUs (during 12 h)
Design services for deep learning models	Fabrik	https://github.com/Cloud-CV/Fabrik/	Model export to Keras code; no training
	IBM Data Cloud	https://datascience.ibm.com/	Model export to Keras, PyTorch, TensorFlow or Caffe
	DeepCognition	http://deepcognition.ai/	Training and evaluation included
Prebuilt images with CUDA support	Docker Hub	https://hub.docker.com/r/nvidia/cuda/	Docker images from NVIDIA with CUDA/cuDNN GPU support
	Amazon Deep Learning AMIs	https://aws.amazon.com/machine-learning/amis/	Amazon Machine Images (AMIs) with GPU support
Software libraries (general)	Keras	https://keras.io/	More high-level than TensorFlow (below) but can be integrated with it in many ways
	TensorFlow	https://www.tensorflow.org/	Developed by Google; most popular deep learning framework
	PyTorch	http://pytorch.org/	Developed by Facebook
Software libraries (specific for genomics)	DragoNN	https://kundajelab.github.io/dragonn/	Tutorials included
	Kipoi	http://kipoi.org/	Model zoo for deep learning in genomics
Educational resources	fast.ai	http://www.fast.ai/	E.g., Deep Learning for Coders 1 and 2
	Coursera	https://www.coursera.org/specializations/deep-learning/	Deep-learning-specialization course package
	Textbook	http://neuralnetworksanddeeplearning.com/	Free online textbook with example code
	Fast.ai tips on configuring a deep learning environment	https://github.com/reshamas/fastai_deeplearn_part1/blob/master/README.md#platforms-for-using-fastai-gpu-required/	Instructions for configuring deep learning frameworks for a variety of platforms; from the fast.ai course but general; the details of these procedures change quickly
	Setting up TensorFlow with GPU on Google Cloud Engine	https://medium.com/google-cloud/jupyter-tensorflow-nvidia-gpu-docker-google-compute-engine-4a146f085f17/	Recipe for Docker-based setup of Google Cloud instance with TensorFlow, GPU support and Jupiter Notebooks

Zou, J. et al., 2018. A primer on deep learning in genomics. *Nature genetics*. Available at: http://dx.doi.org/10.1038/s41588-018-0295-5.

Biological problems that we follow and work on

Part III

Human genetic variation



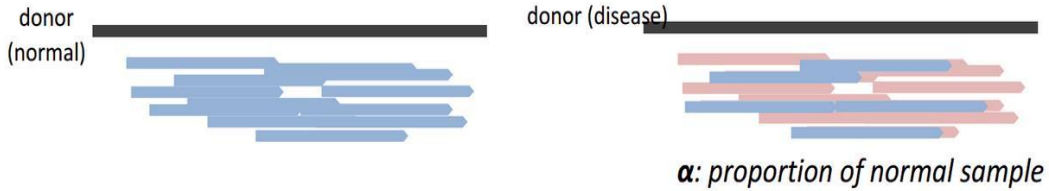
* Variants with allele frequency < 0.5% are considered as rare variants in 1000 genomes project.

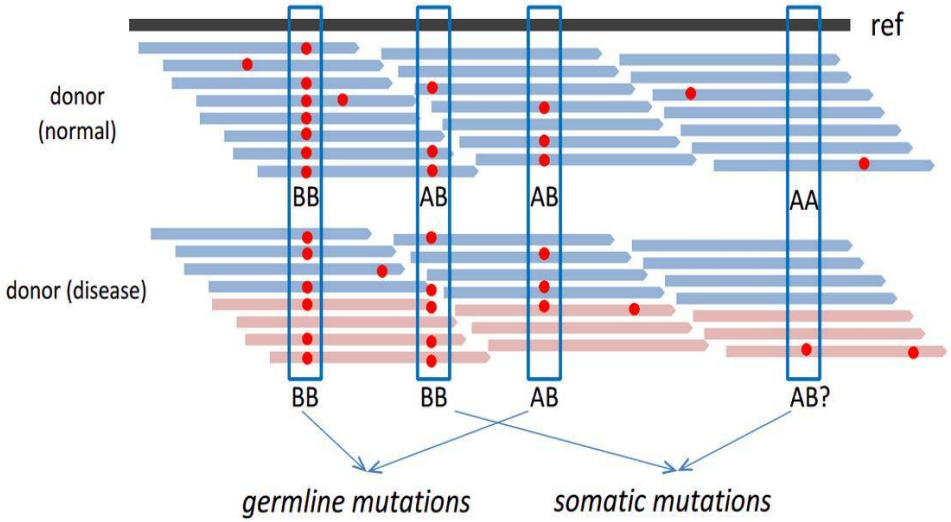
The 1000 Genomes Project Consortium, Nature. 2015. 526:68-74
Khurana E. et al. Nat. Rev. Genet. 2016. 17:93-108

Variant calling

Searching the haystack

3.5 million SNPs



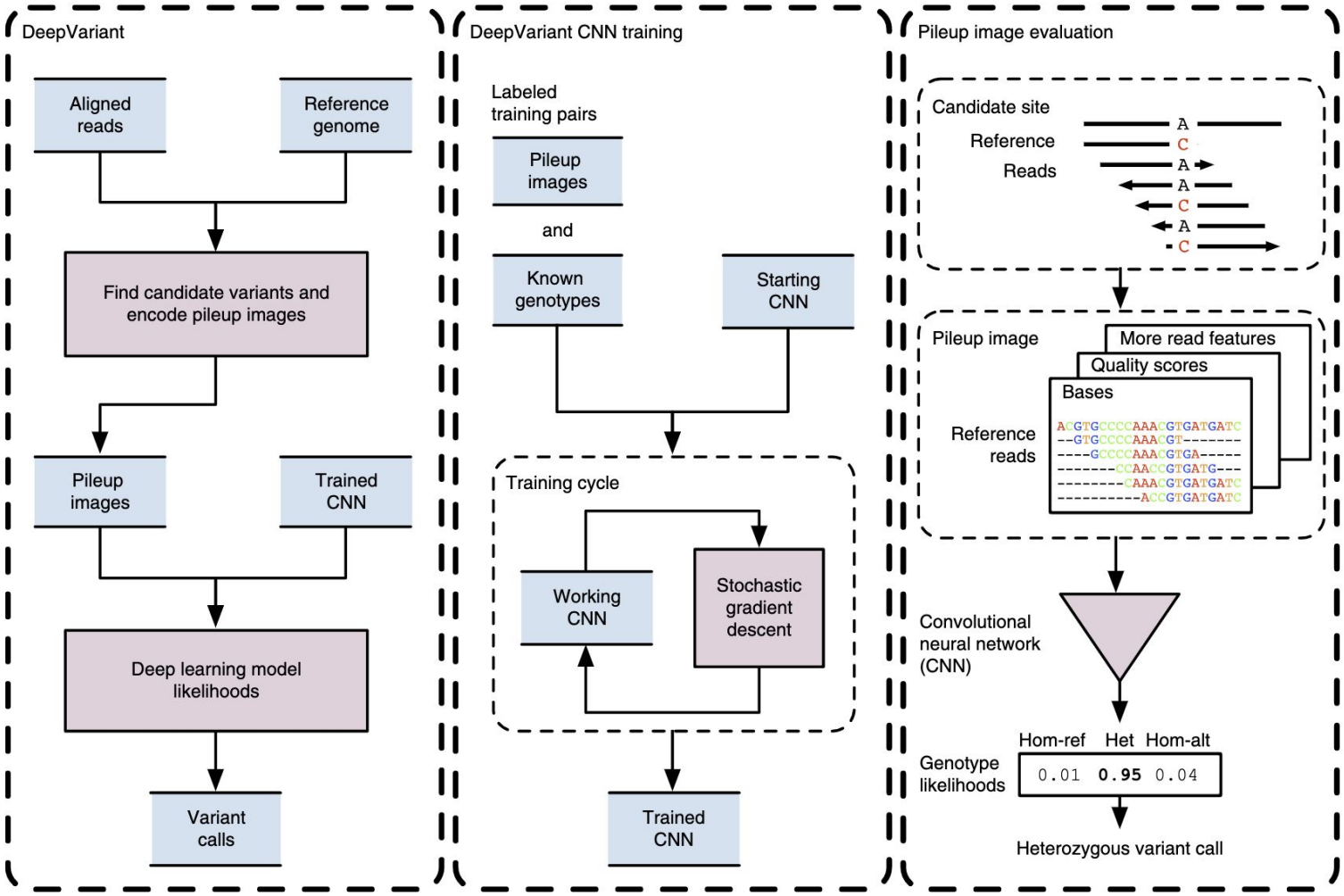


Alignment -> Improving -> Variant calling -> Filtering

Resulting file type: vcf

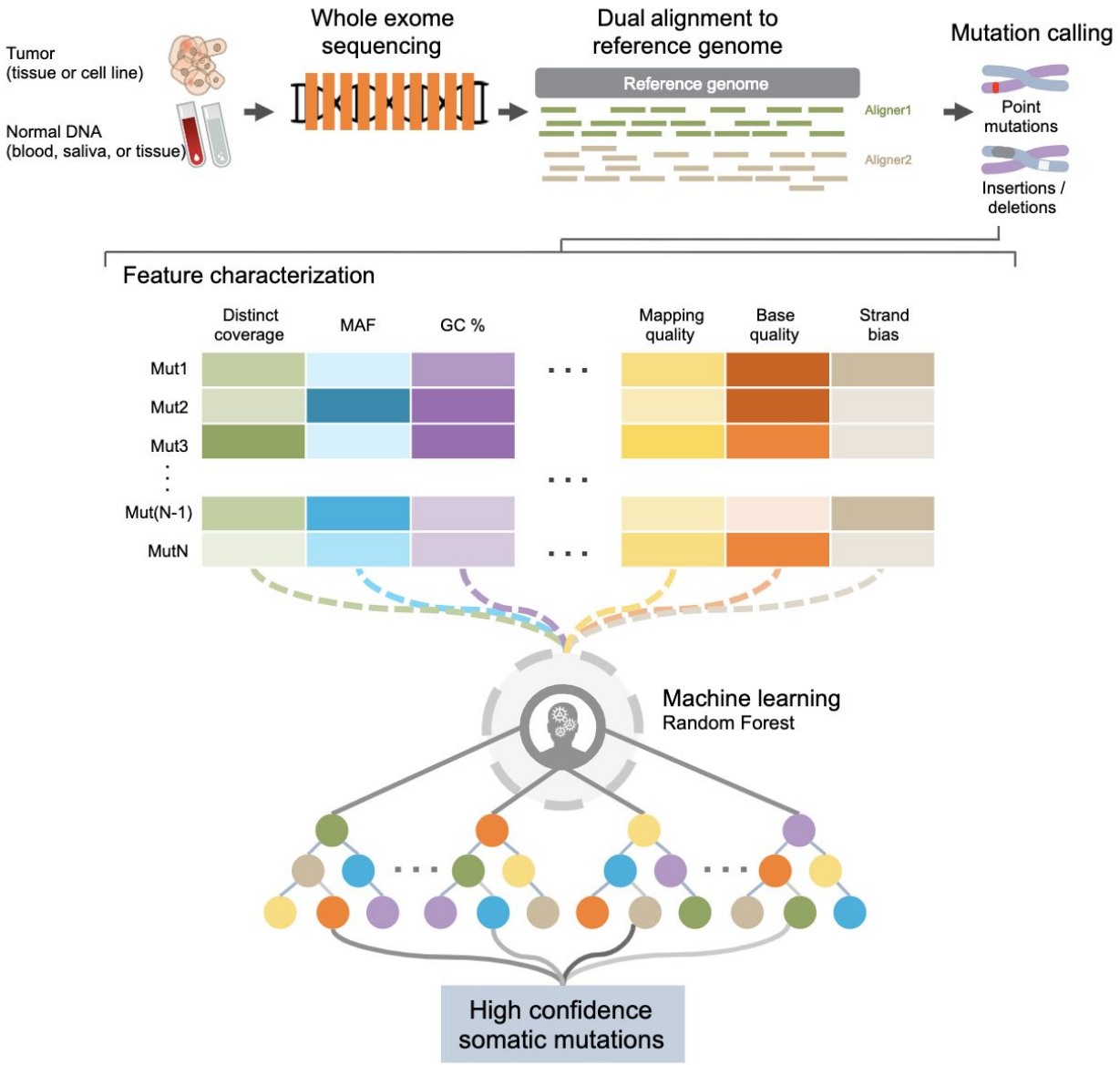
"What are the differences to the reference genome?"

Variant caller using Deep Neural Networks



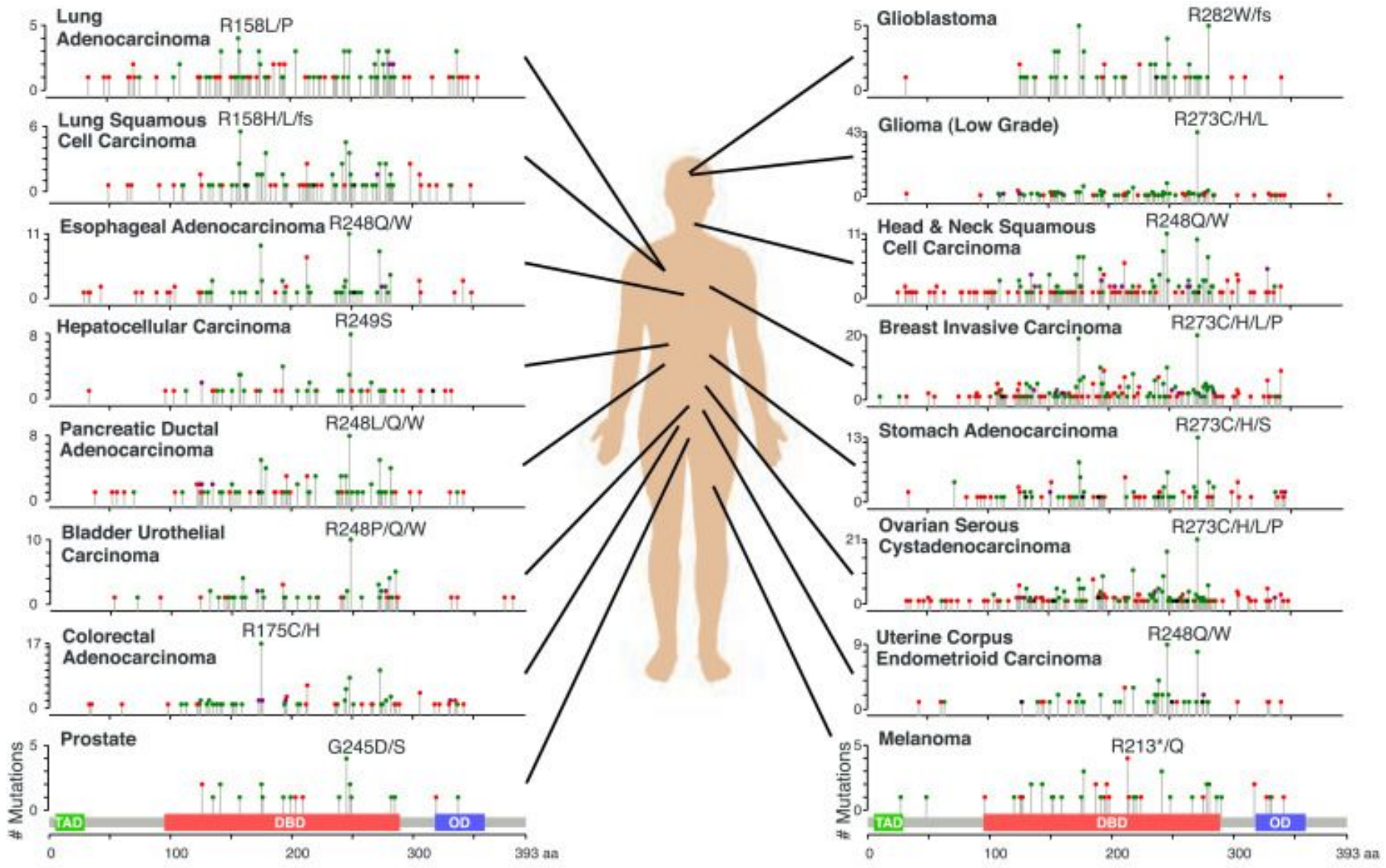
Poplin, R. et al., 2018. A universal SNP and small-indel variant caller using deep neural networks. *Nature biotechnology*, 36(10), pp.983–987.

Somatic mutation discovery using Random Forest



Wood, D.E. et al., 2018. A machine learning approach for somatic mutation discovery. *Science translational medicine*, 10(457), http://dx.doi.org/10.1126/scitranslmed.aar7939.

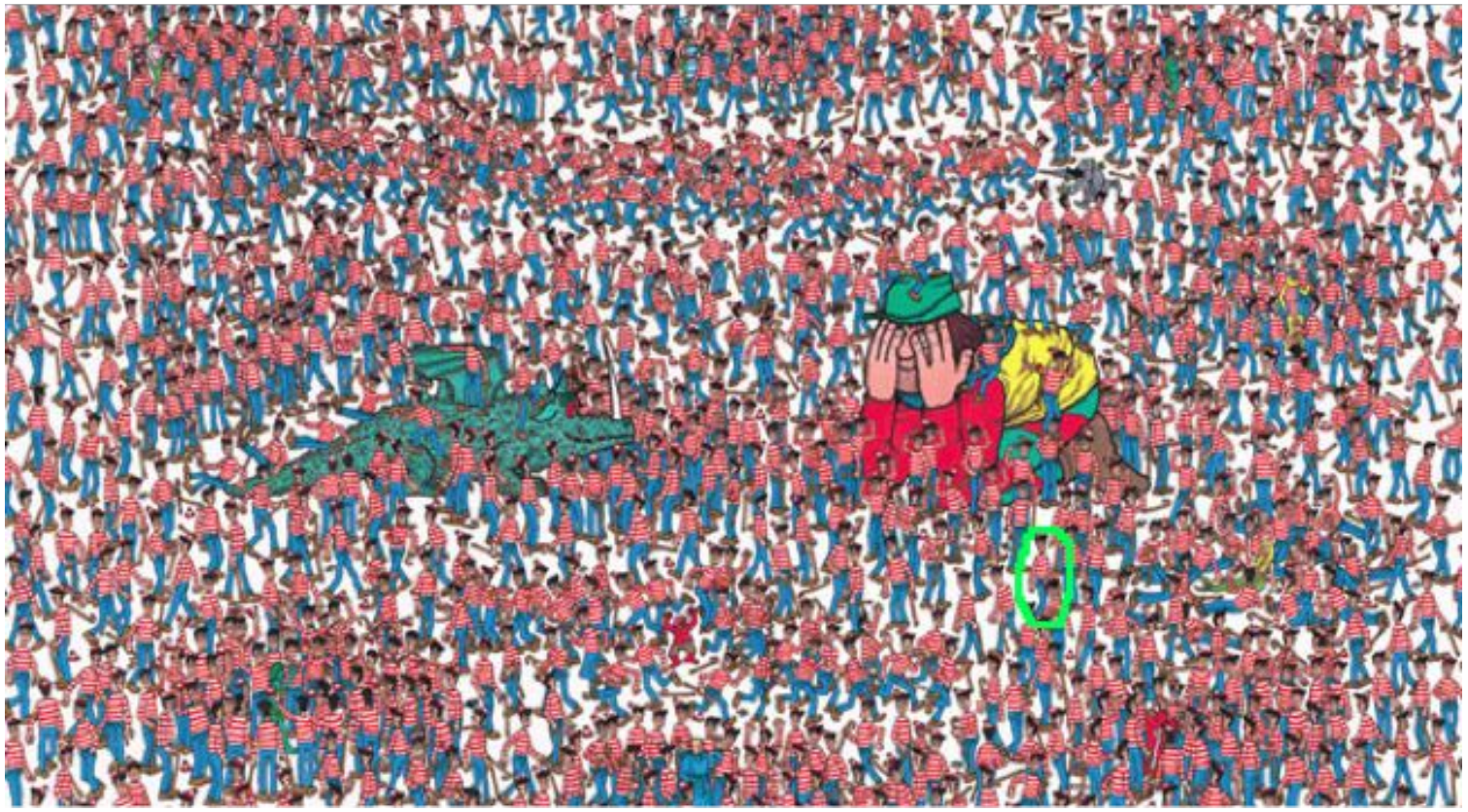
Variant prioritization



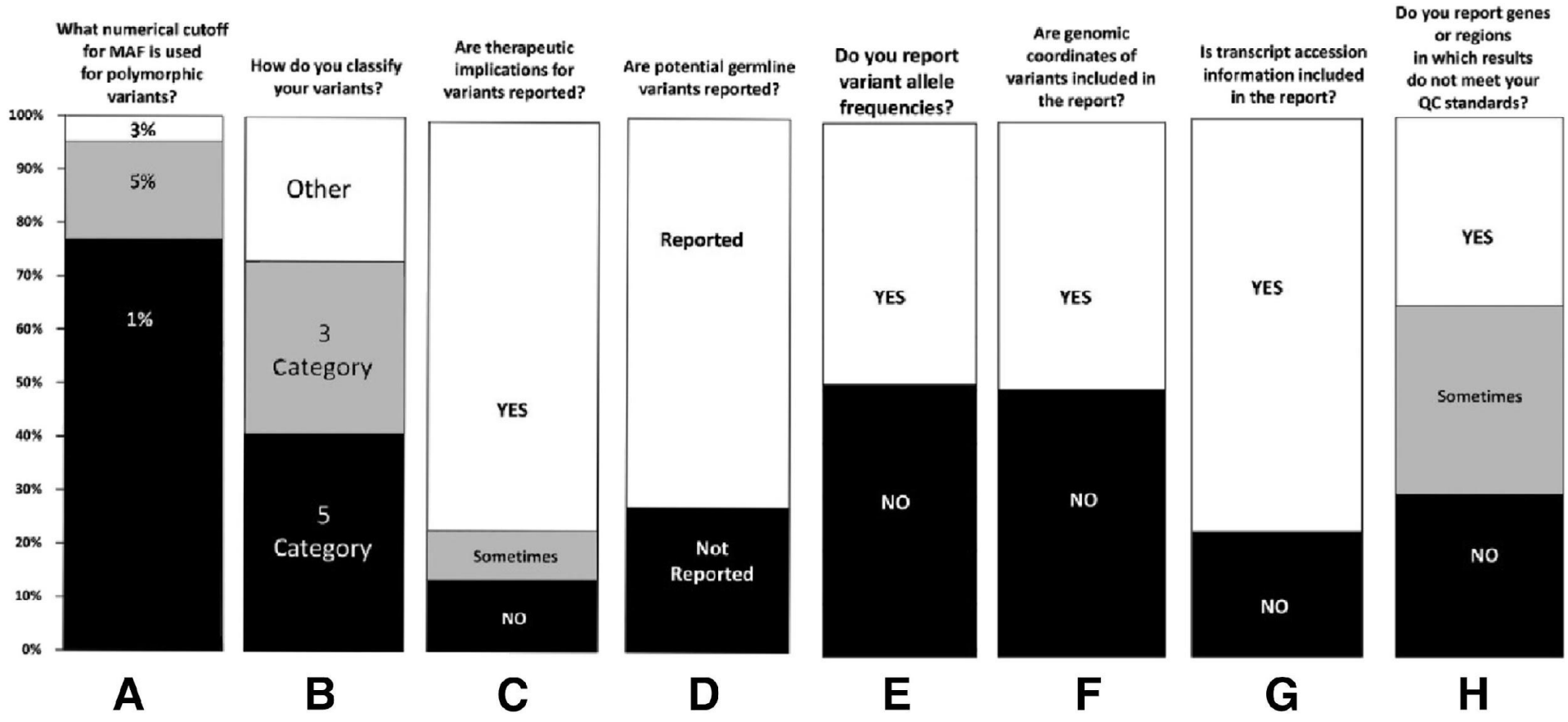
Cell 2017 170, 1062-1078DOI: (10.1016/j.cell.2017.08.028)

How does it impact diseases?

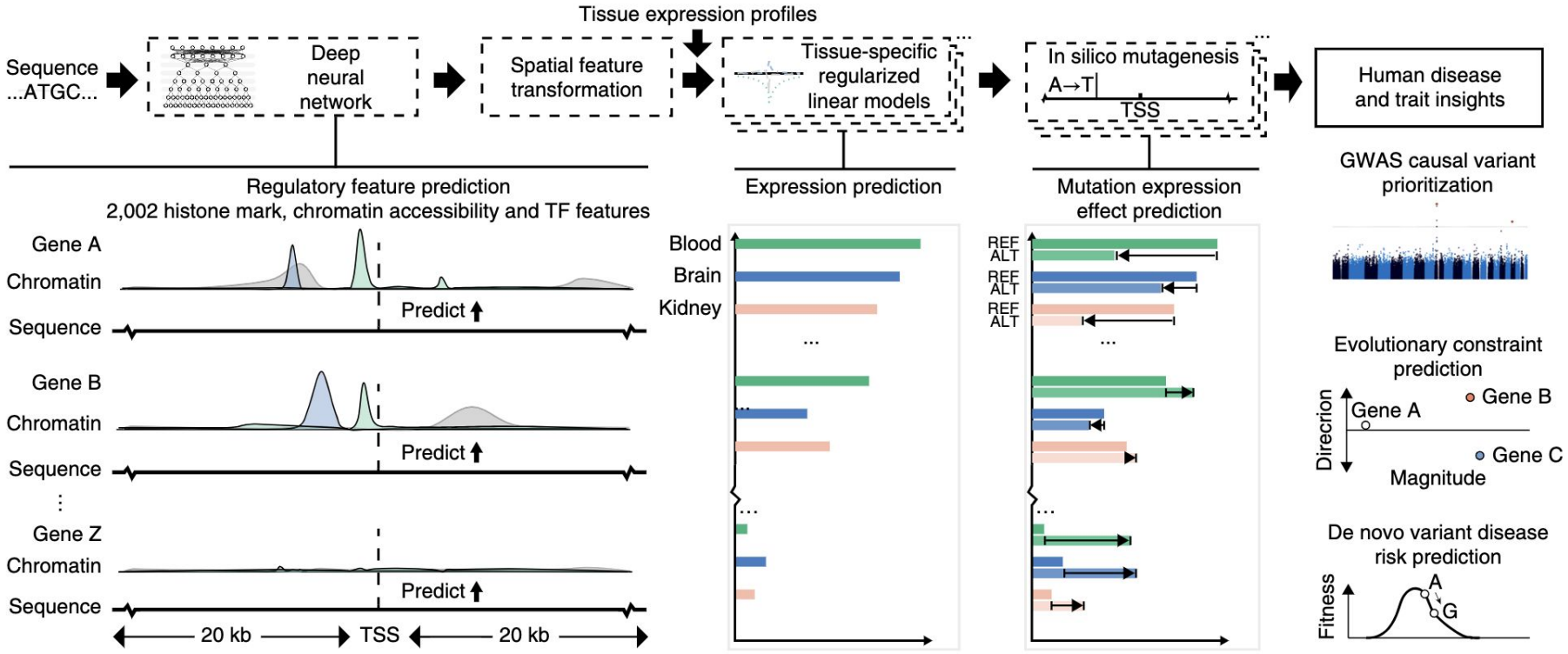
(Finding the key mutations in ~3M Germline variants &
~5K Somatic Variants in a Tumor Sample)



Arbitrary classification of pathogenicity of variants

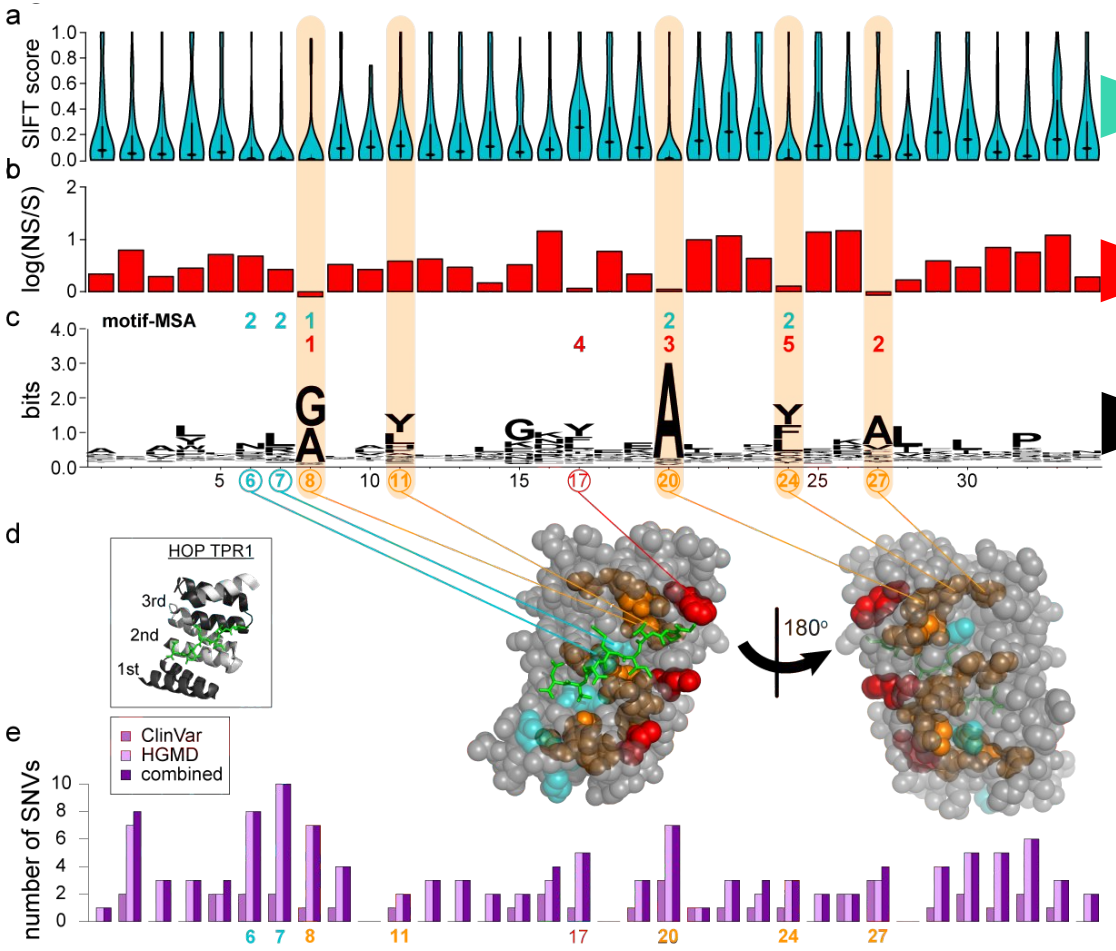


Deep Learning for ab initio prediction of variant effects



Zhou, J. et al., 2018. Deep learning sequence-based ab initio prediction of variant effects on expression and disease risk. *Nature genetics*, 50(8), pp.1171–1179.

Variants in protein coding regions



Selection in PPI motifs

How to check possible significance:

-> Burial within structure

-> more SNVs implicated in diseases in ClinVar and HGMD

Deep Generative Models to capture genetic variations on proteins

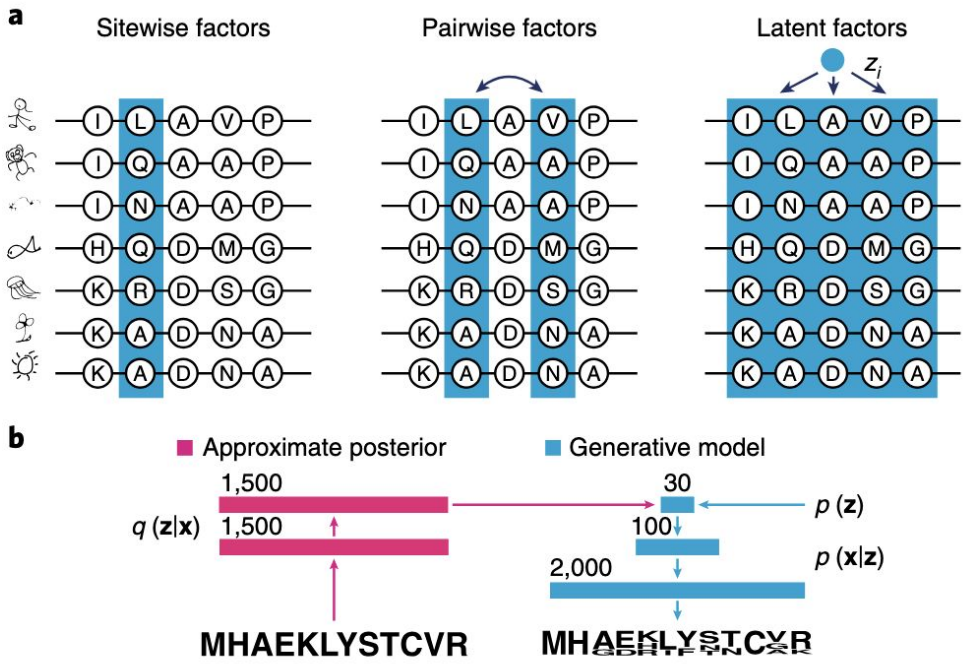


Fig. 1 | A nonlinear latent-variable model captures higher-order dependencies in proteins and RNAs. a, Comparison of a nonlinear latent-variable model with site-independent and pairwise models. **b**, The dependency $p(\mathbf{x}|\mathbf{z})$ (blue) of the sequence **x** on the latent variable **z** is modeled by a neural network, and inference and learning are made tractable by joint training with an approximate inference network $q(\mathbf{z}|\mathbf{x})$ (pink). This combination of model and inference is also known as a variational autoencoder. The size of the latent variables **z** and hidden dimensions of the neural network are shown.

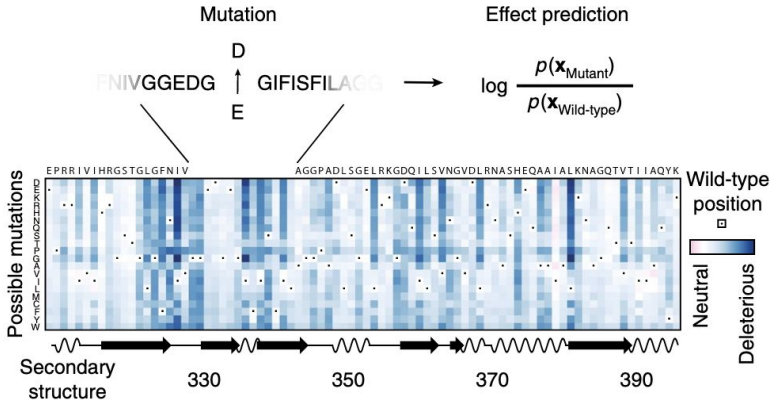
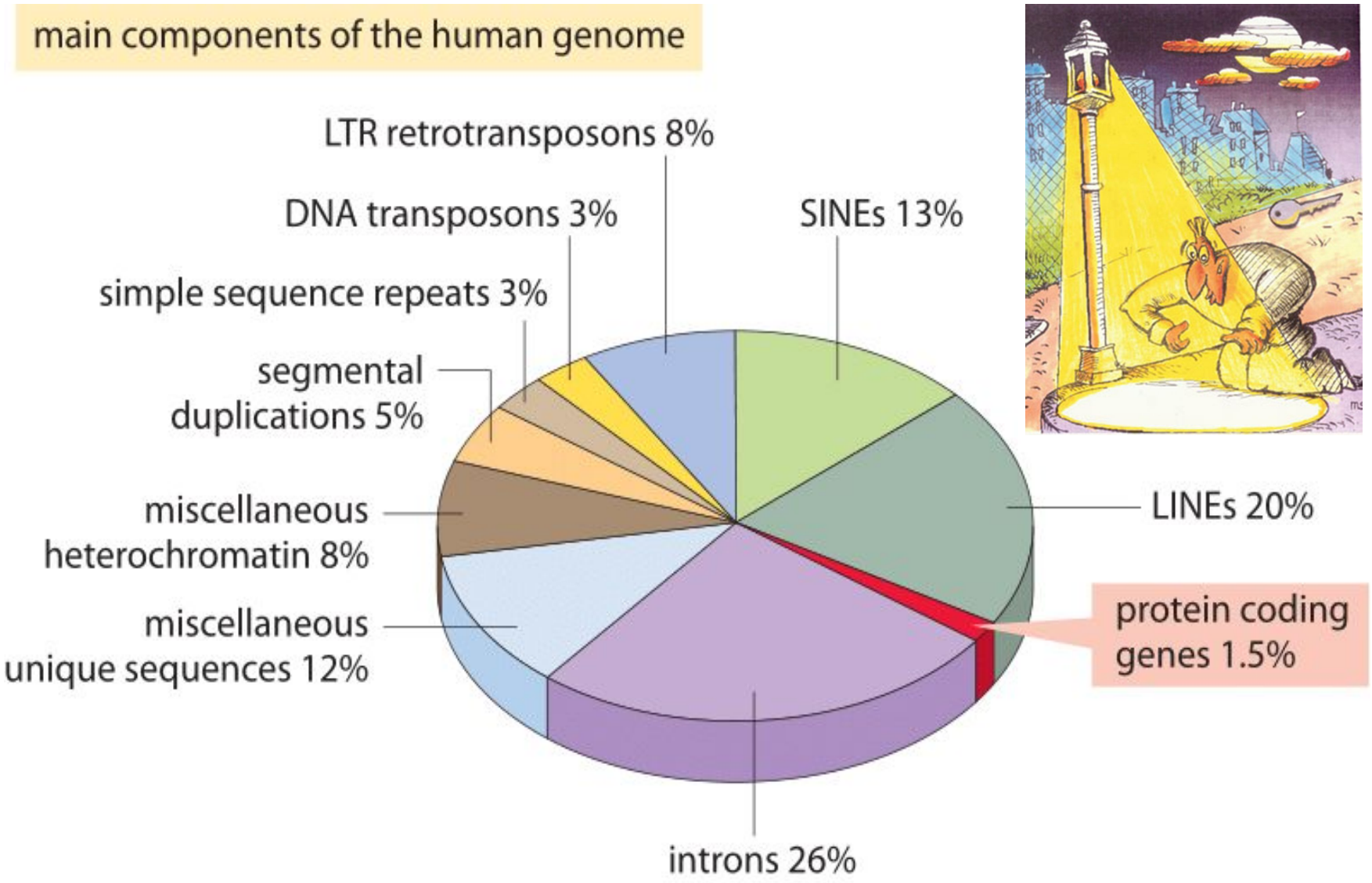


Fig. 2 | Mutation effects can be quantified by likelihood ratios. After fitting a probabilistic model to a family of homologous sequences, we heuristically quantified the effect of mutation as the log-ratio of mutant likelihood to wild-type likelihood (as approximated by the ELBO; Methods). Bottom: mutation-effect scores for positions 310–393 in the PDZ domain.

DeepSequence

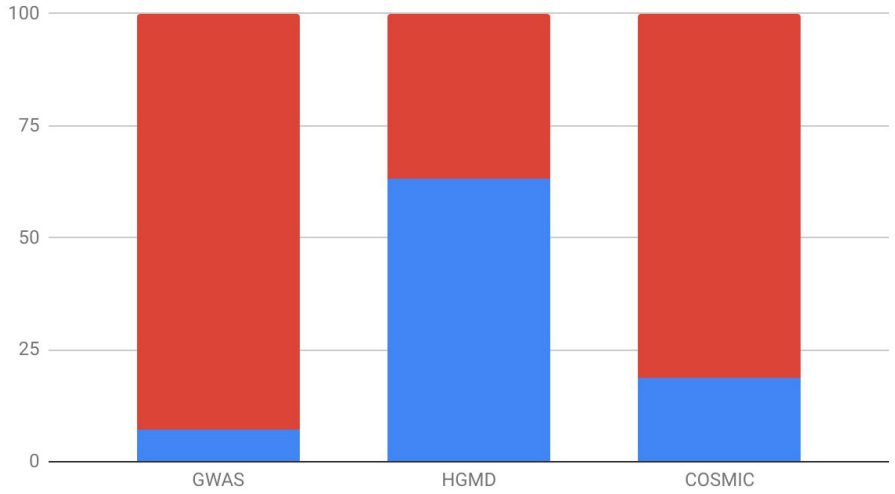
Riesselman, A.J., Ingraham, J.B. & Marks, D.S., 2018. Deep generative models of genetic variation capture the effects of mutations. *Nature methods*, 15(10), pp.816–822.

Problem 2

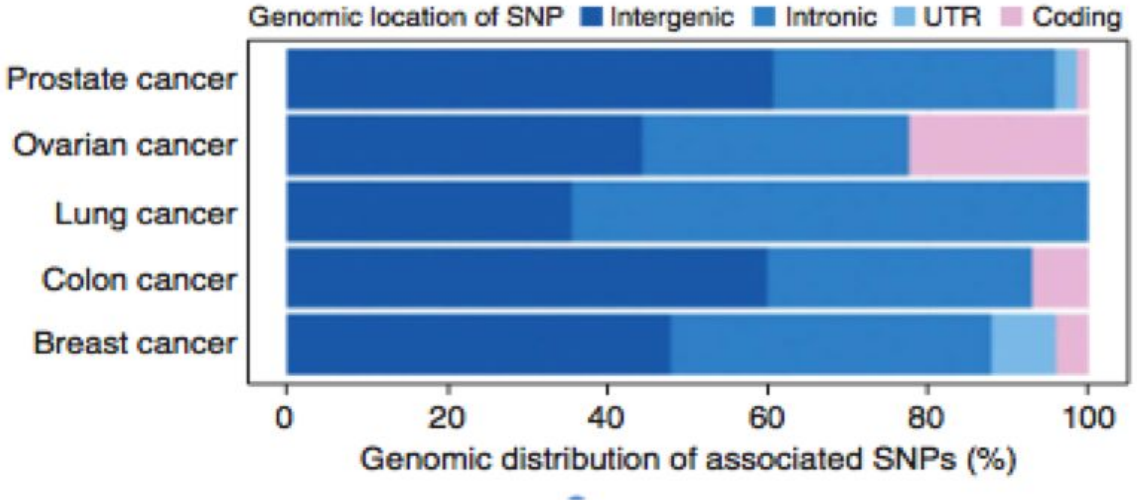


Ref 1

Cancer associated variants

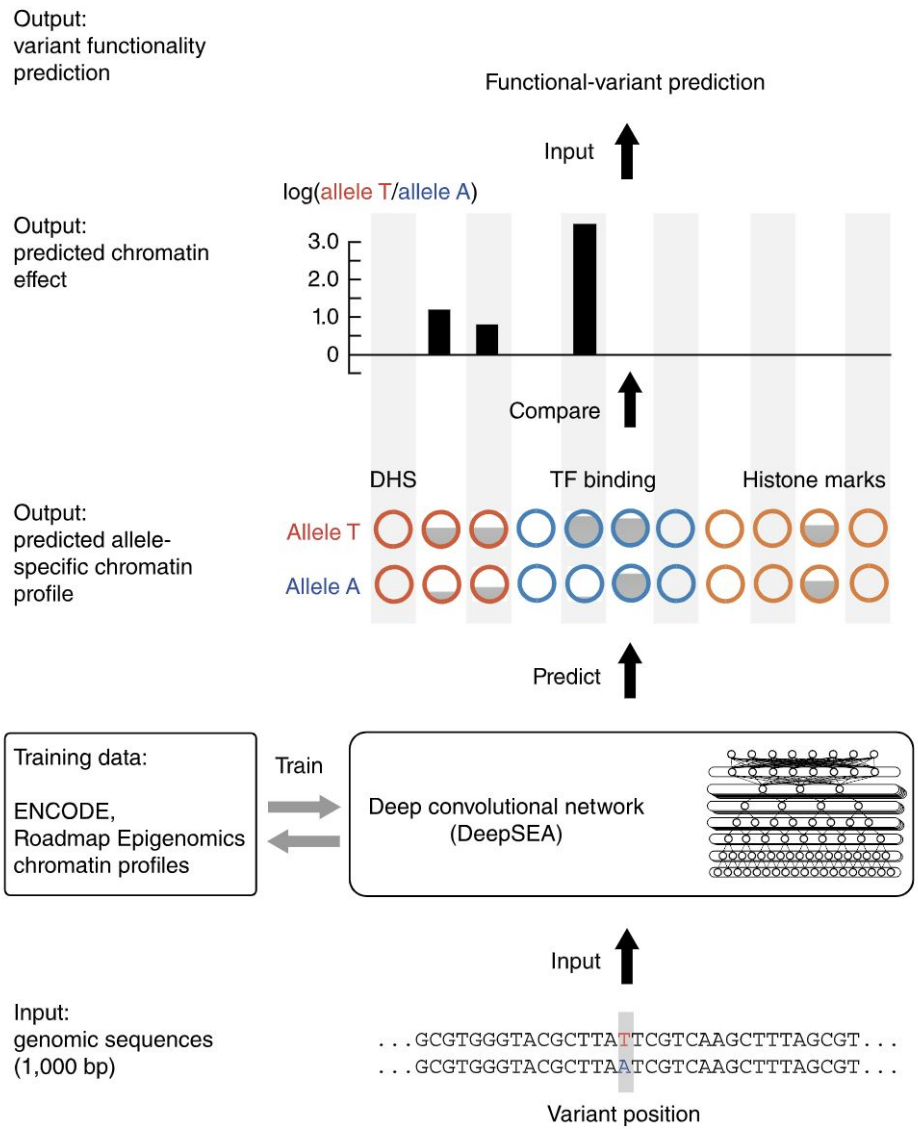


Prabakaran Lab, Manuscript in preparation



S. W. Cheetham et al., Long non coding RNAs and the genetics of Cancer. British Journal of Cancer, 108 (12), 2419-2425 (2013)

DeepSEA pipeline



strategy for predicting chromatin effects of noncoding variants.

Zhou, J. & Troyanskaya, O.G., 2015. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10), pp.931–934.

Properties of predictive models of six tools

Table 1 Properties of predictive models for six tools

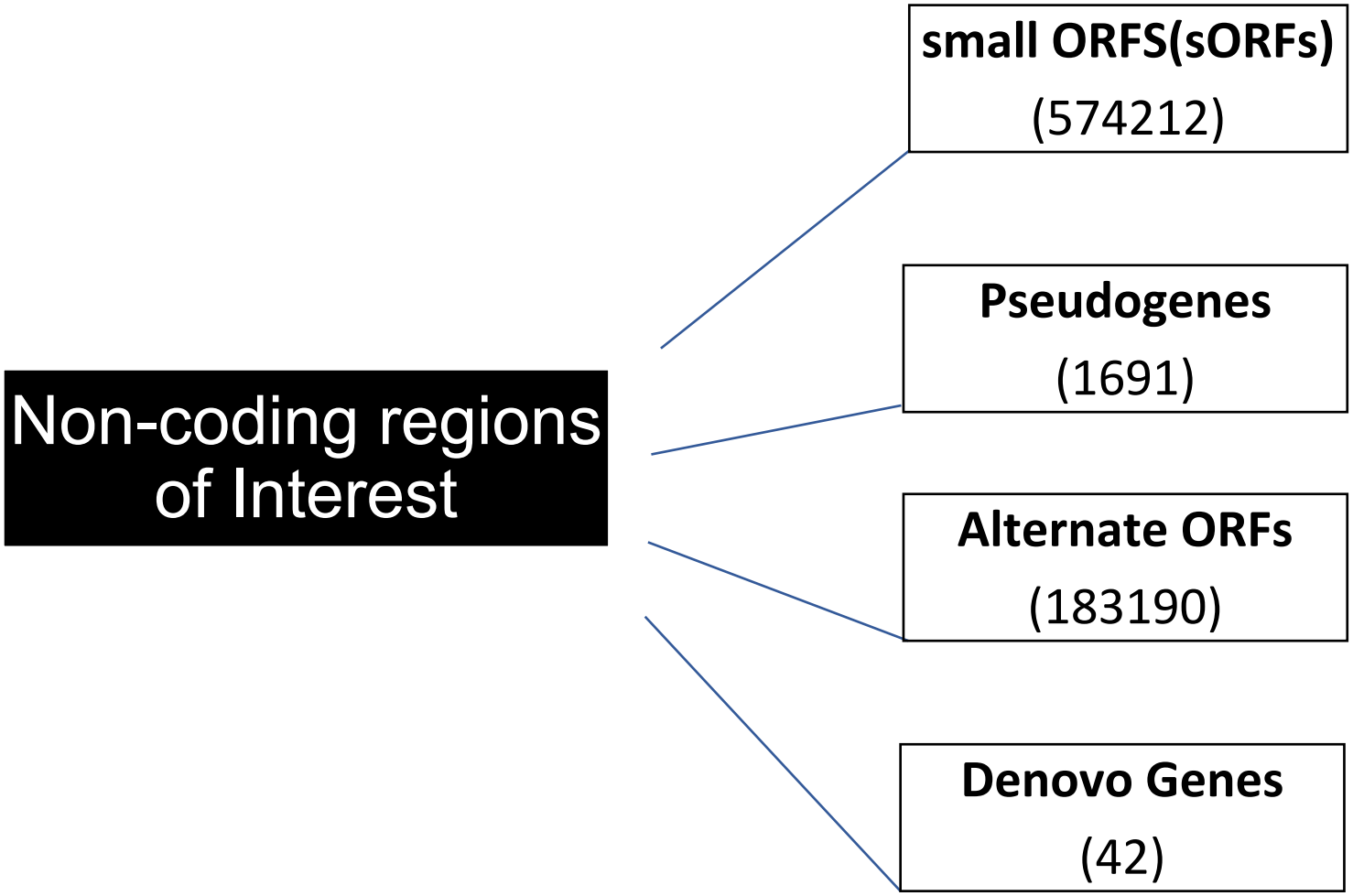
Method	Assumption of pathogenicity	Predictors	Modeling approaches	Performance (AUROC)[a]
CADD	Evolutionary fitness	Evolutionary parameters, ENCODE summaries, functional annotations, population frequencies	Support vector machines	0.92[b]
CATO	Molecular functions	Cell type- and tissue-specific assays, evolutionary parameters, functional annotations	Logistic regression	NA[c]
DeepSEA	Molecular functions	Local sequences, evolutionary parameters	Deep learning, Logistic regression	0.85
EIGEN	None[d]	Evolutionary parameters, ENCODE summaries, population frequencies	Unsupervised learning	0.79
GWAVA	DAVs vs. CPPs	Evolutionary parameters, ENCODE summaries, population frequencies	Random forests	0.97
LINSIGHT	Evolutionary fitness	Evolutionary parameters, ENCODE summaries, functional annotations	Generalized linear model	0.96

AUROC = area under the receiver operator characteristic curve, DAV = disease-associated variant, CPP = common population polymorphism
[a]Highest AUROC values in classifying DAVs and CPPs reported in the original publications
[b]CADD reported AUROC values that mixed coding and noncoding variants
[c]CATO predicts transcription factor occupancy instead of pathogenicity
[d]EIGEN uses an unsupervised learning approach and thus makes no assumption of pathogenicity during training

Liu, L. et al., 2019. Biological relevance of computationally predicted pathogenicity of noncoding variants. *Nature communications*, 10(1), p.330.

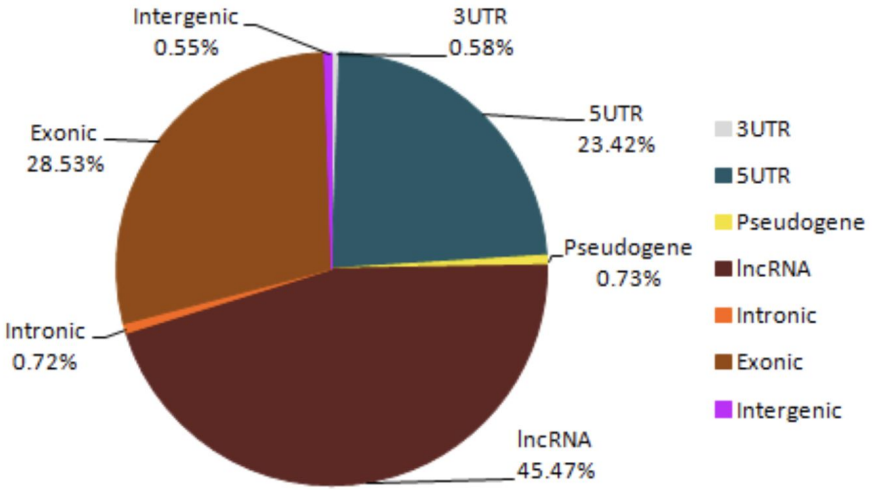
Problem 2 questions

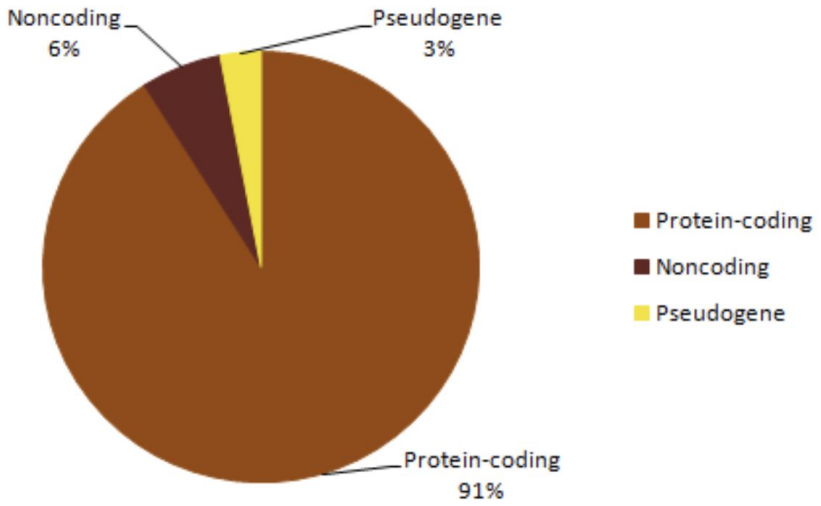
1. Regions of noncoding genome

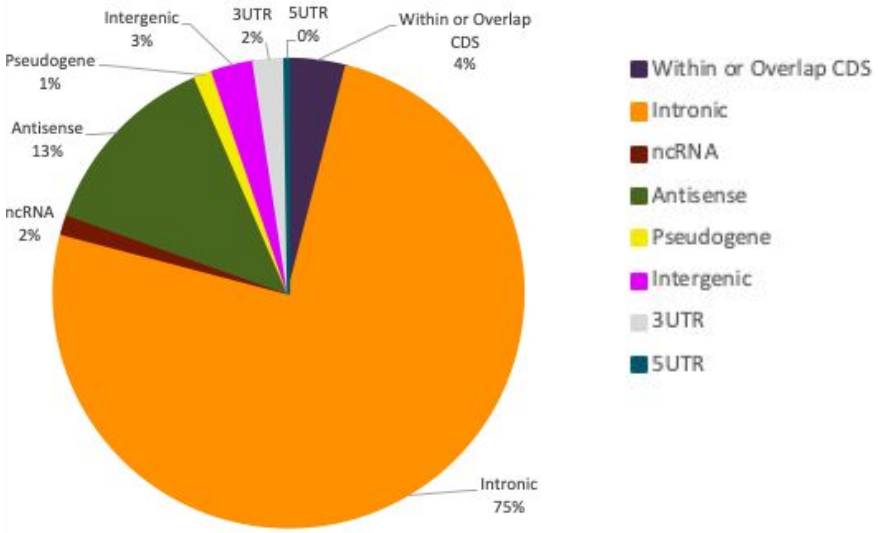


2. Prioritizing variants in these regions using machine learning approaches

Evidence for translations in mouse B & T cells







Prabakaran Lab, Manuscript submitted

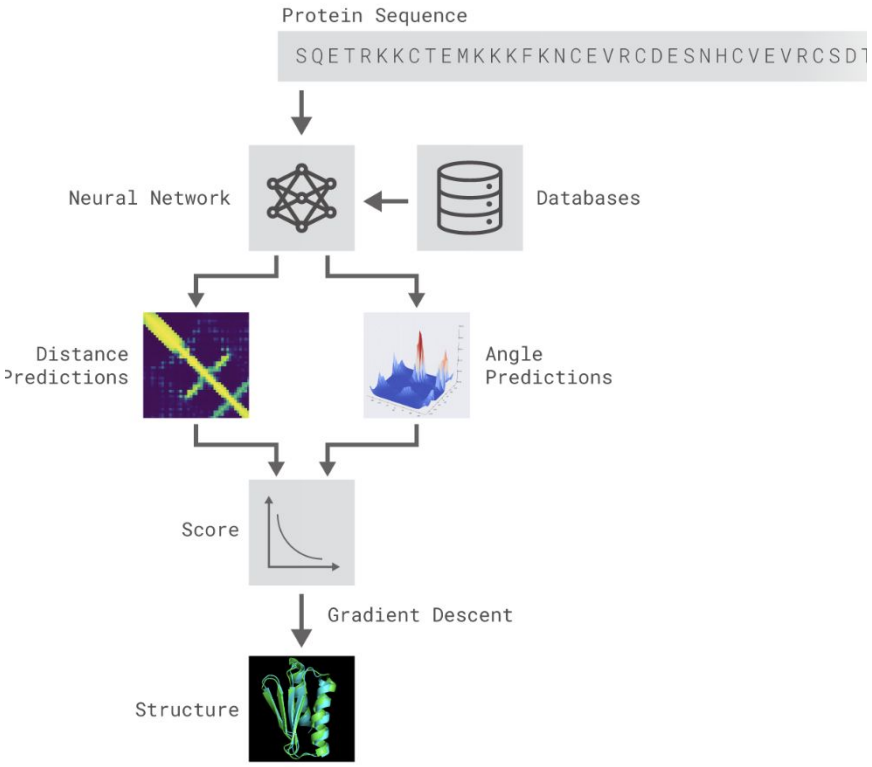
DeepMind Alpha Fold

AlphaFold: Using AI for scientific discovery

Today we're excited to share DeepMind's first significant milestone in demonstrating how artificial intelligence research can drive and accelerate new scientific discoveries. With a strongly interdisciplinary approach to our work, DeepMind has brought together experts from the fields of structural biology, physics, and machine learning to apply cutting-edge techniques to predict the 3D structure of a protein based solely on its genetic sequence.

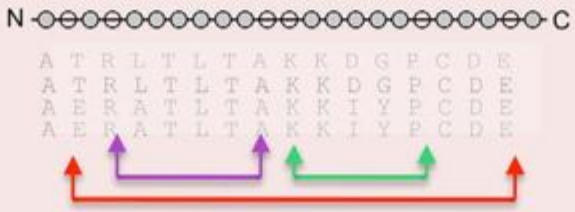
Our system, **AlphaFold**, which we have been working on for the past two years, builds on years of prior research in using vast genomic data to predict protein structure. The 3D models of proteins that AlphaFold generates are far more accurate than any that have come before—making significant progress on one of the core challenges in biology.

What is the protein folding problem?



Structural genomics of sORFs

Align evolutionary diverged sequences



Calculate covariance matrix for each pair of sequence positions for all pairs of amino acids (A,B)

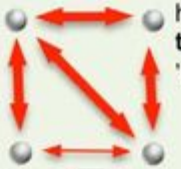
$$C_{ij}(A,B) = f_{ij}(A,B) - f_i(A)P_j(B)$$

$$C_{ij}^{-1}(A,B) = -e_{ij}(A,B)_{i \neq j}$$

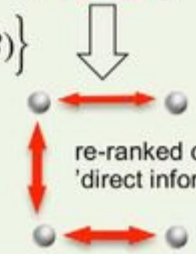
Identify maximally informative pair couplings using **statistical model** of entire protein to infer residue-residue co-evolution

$$P_{ij}^{Dir}(A,B) = \frac{1}{Z}\exp\left\{e_{ij}(A,B) + \tilde{h}_i(A) + \tilde{h}_j(B)\right\}$$

$$DI_{ij} = \sum_{A,B=1}^{q} P_{ij}^{Dir}(A,B)\ln\frac{P_{ij}^{Dir}(A,B)}{f_i(A)f_j(B)}$$



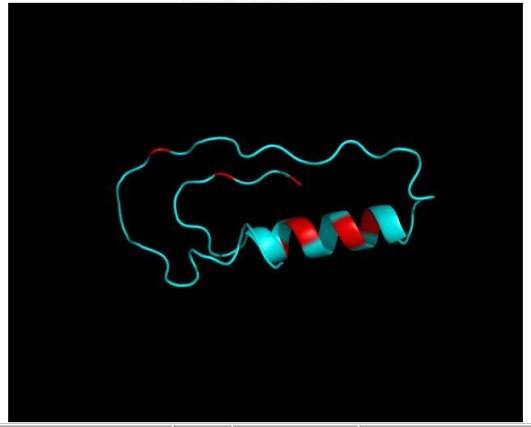
high ranking **transitive** 'indirect correlations'



re-ranked correlations 'direct information' = DI

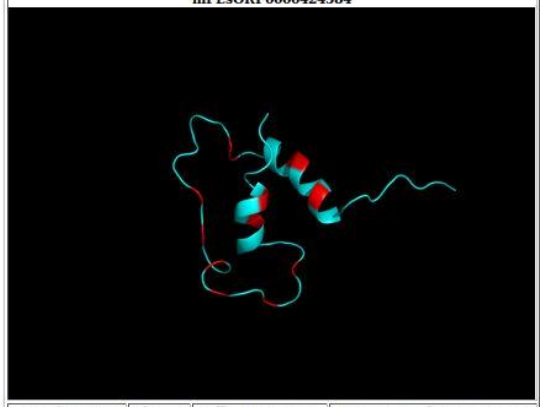
Examples of sORFs with mutations

mPLsORF0000239729



Mutation ID(s)	Change	Effect on sORF	sORF secondary structure
COSM3816594-AA=p.W105* COSM3816595-AA=p.W105*	G>A	W16>* (PDB W17)	H
COSM4056367-AA=p.Y91H COSM4056368-AA=p.Y91H	T>C	Y2>H (PDB Y3)	E
COSM5632701-AA=p.L109V COSM5632702-AA=p.L109V	C>G	L20>V (PDB L21)	H
COSM6324258-AA=p.A94V COSM6324259-AA=p.A94V	C>T	A5>V (PDB A6)	C
COSM6761144-AA=p.E103D COSM6761145-AA=p.E103D	G>T	E14>D (PDB E15)	H
COSM964154-AA=p.D111Y	G>T	D22>Y (PDB D23)	H
COSM964155-AA=p.R129H	G>A	R40>H (PDB R41)	C

mPLsORF0000424584

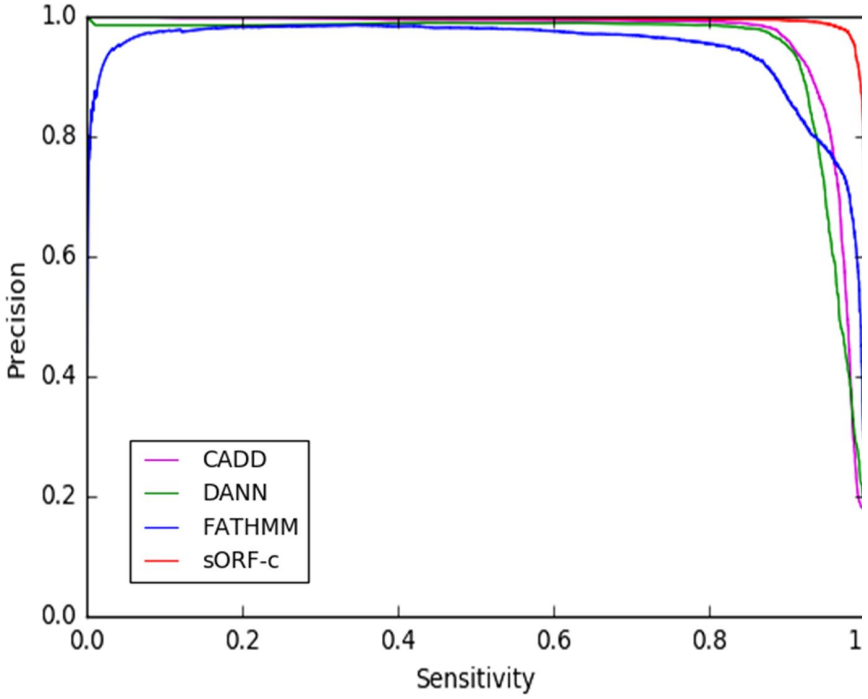


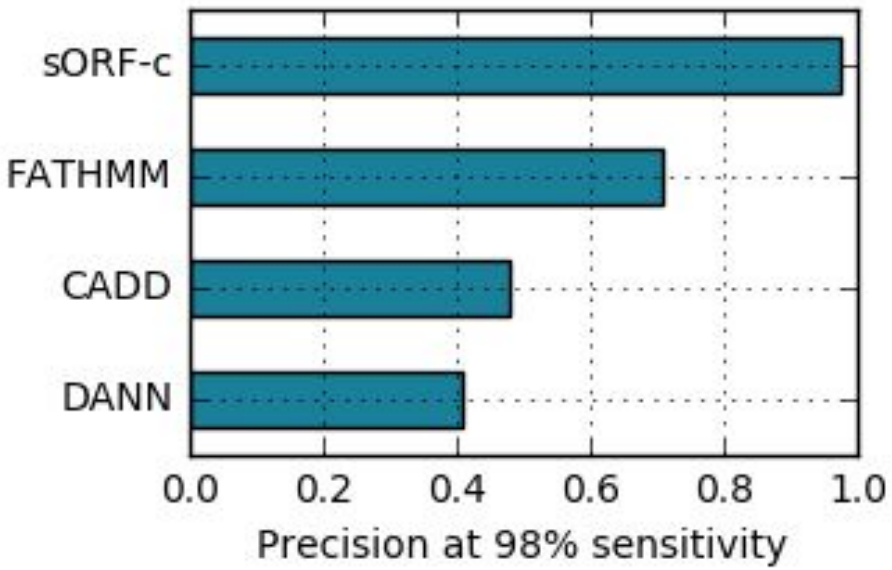
Mutation ID(s)	Change	Effect on sORF	sORF secondary structure
COSN1260962 COSN5770834	A>G	Y10>C (PDB Y10)	H
COSN18723188	C>T	P32>L (PDB P32)	C
COSN20080080	C>T	R13>C (PDB R13)	C
COSN20121983	C>T	I28>I (PDB I28)	C
COSN20648546 COSN21264986 COSN23079780	G>A	W50>* (PDB R50)	H
COSN20650206 COSN21677358 COSN22583798 COSN23084029	C>T	G42>G (PDB G42)	E
COSN20650207 COSN22687956 COSN23081188	T>C	W50>R (PDB R50)	H
COSN21264985 COSN22557586 COSN23085391	A>G	K49>K (PDB K49)	H
COSN21675097	T>C	Y43>Y (PDB Y43)	H
COSN23082255	G>A	K34>K (PDB K34)	H
COSN23087874	A>G	K26>E (PDB E26)	H
COSN24397730	G>A	R23>H (PDB R23)	H
COSN26666338	A>T	E67>V	H
COSN27001349 COSN8016762	G>A	T20>T (PDB T20)	C
COSN27001446	C>T	T20>M (PDB T20)	C
COSN27001936	G>T	R70>M	C
COSN27003013	G>A	G54>D (PDB G54)	C

Functional annotation features

Conservation-based annotations	PhastCons 46-way score; PhastCons 100-way score; PhastCons primates; PhastCons mammals; PhyloP vertebrates; PhyloP primates; PhyloP mammals; GERP++ neutral evolution score; GERP++ rejected substitution score; GERP++ rejected substitution p-value; GERP element p-value; bStatistic; Conserved TF binding sites; SiPhy 29-way log odds
Functional annotations	GWAVA region score; GWAVA TSS score; Is transversion; Amino acid change; Fitness Consequence score; SIFT score and category; Polyphen score and category; Grantham score; Segway; MutationTaster score; LRT score; PROVEAN score; MutationAssessor score; MetaLR score; MetaSVM score
Region-based annotations	Max H3K27 acetylation level; Max H3K27 methylation level; Max H3K27 trimethylation level; Max expression value; Peak signal and p-value of Dnase evidence for open chromatin; Peak signal and p-value of Faire evidence for open chromatin; Peak signal and p-value of Pol II evidence for open chromatin; Peak signal and p-value of CTCF evidence for open chromatin; Peak signal and p-value of Myc evidence for open chromatin; Max nucleosome track score; Number of TF binding sites; Distance to TSS; Distance to nearest exon; Relative position in transcript; Open chromatin code; Percent GC in 150bp region; Distance to splice site; Nature of splice site

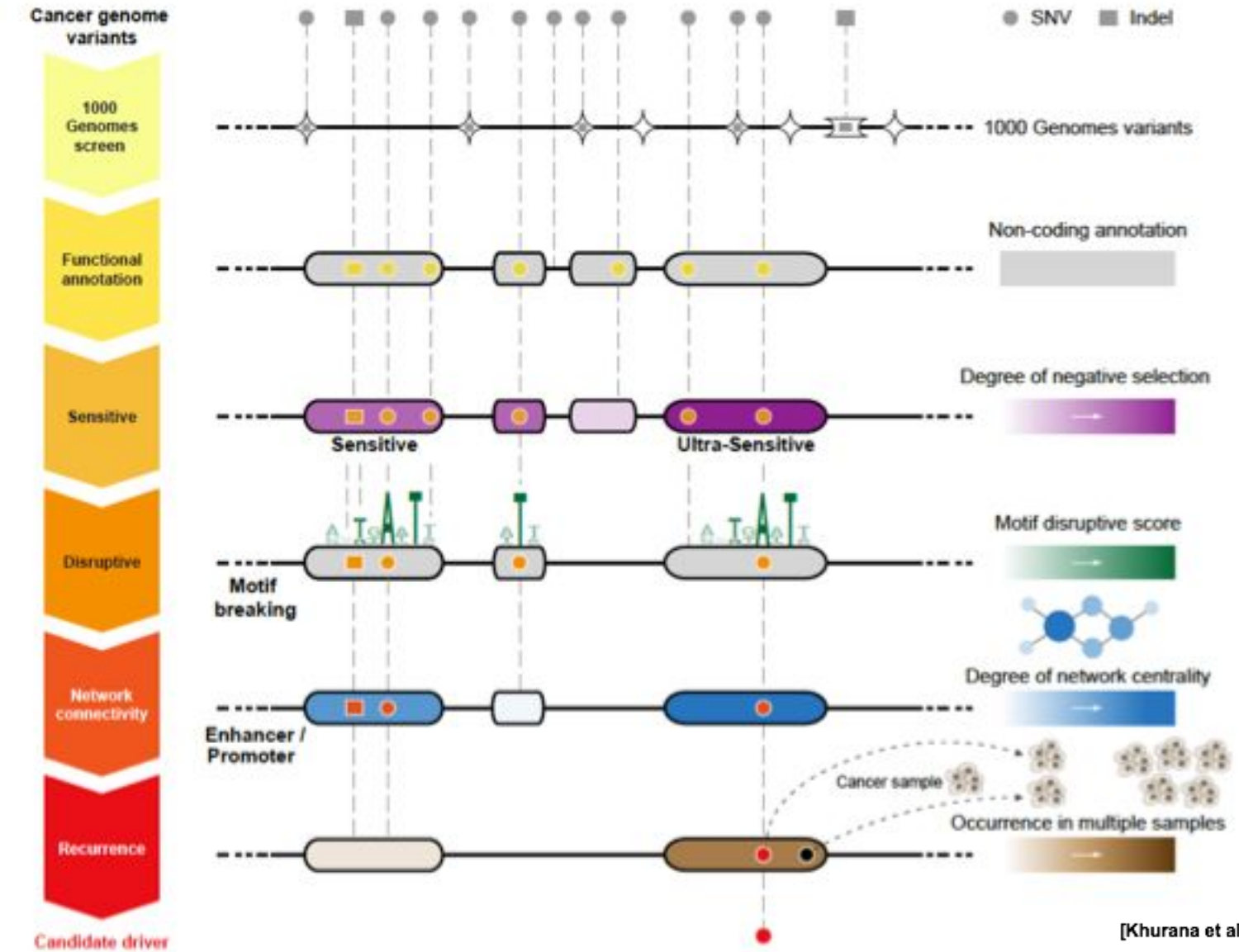
Benchmarking sORF-c perfomance





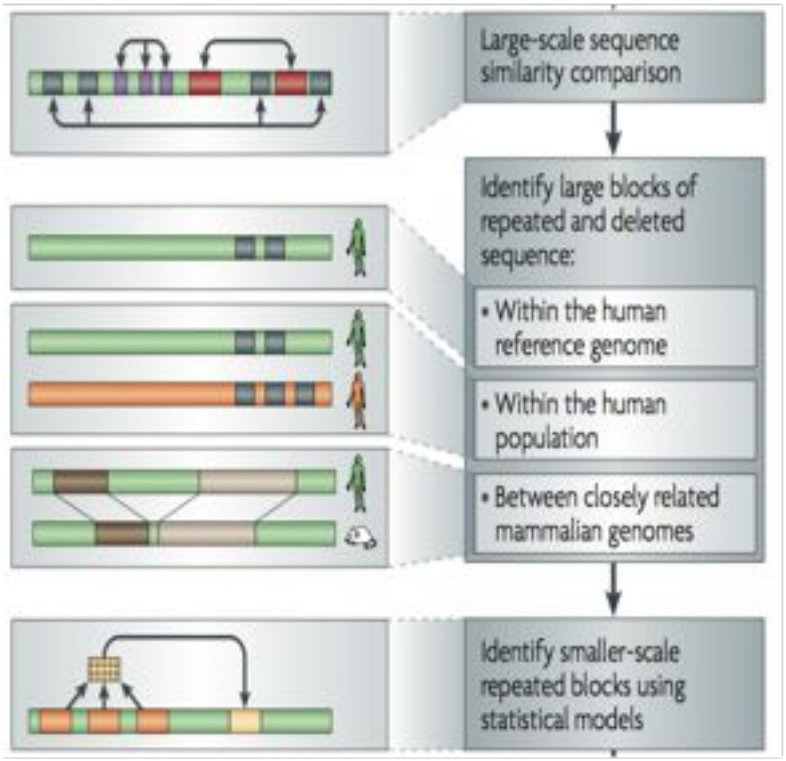
Jackson, F., Wayland, M.T. & Prabakaran, S., 2017. Identification And Prioritisation Of Variants In The Short Open-Reading Frame Regions Of The Human Genome.
bioRxiv, p.133645. Available at: http://biorxiv.org/content/early/2017/05/03/133645

Other attempts are variant prioritization in noncoding regions

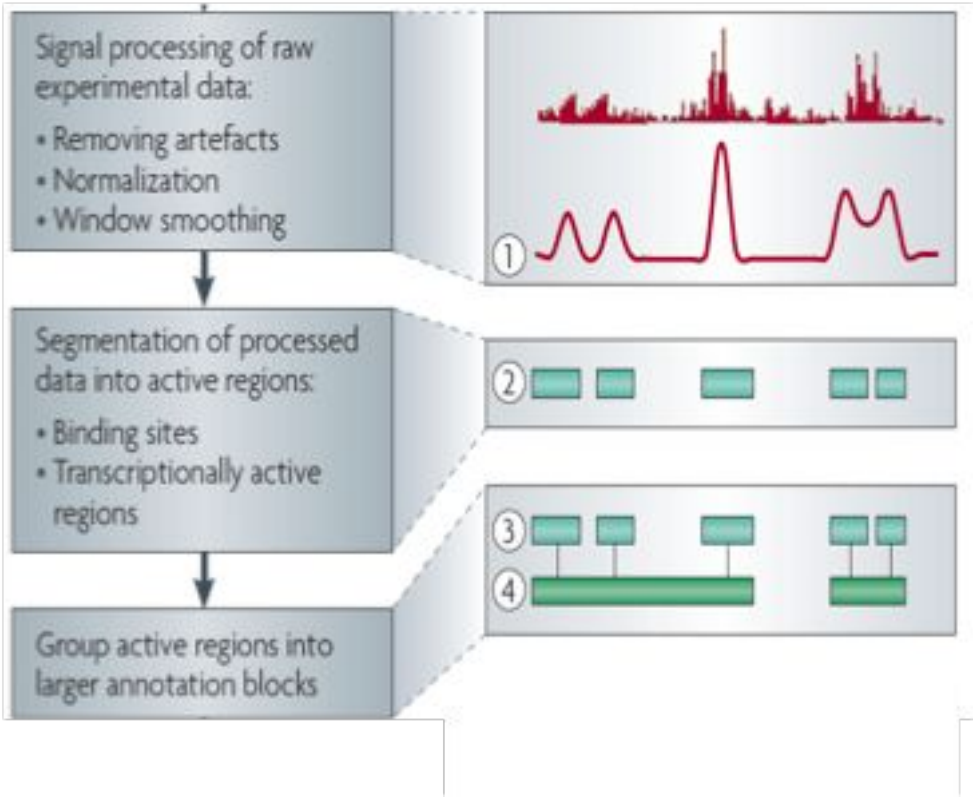


[Khurana et al., *Science* ('13)]

Examples of genome annotation

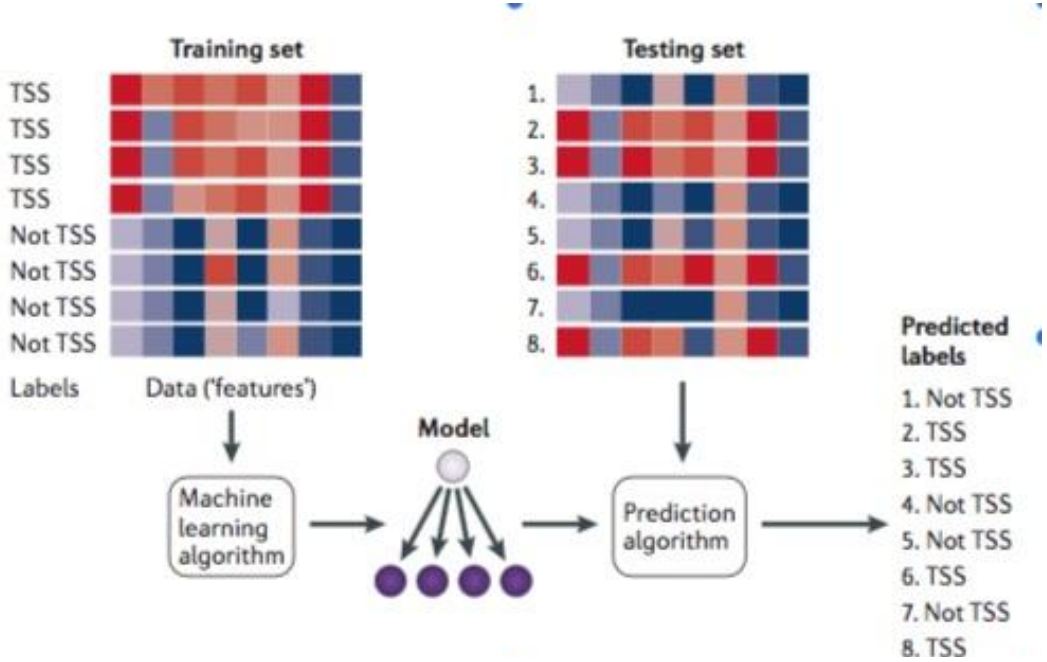


Functional Genomics
Chip-seq (Epigenome & seq. specific TF)
and ncRNA & un-annotated transcription



Annotation of transcription factor binding sites

Typical supervised ML workflow



Step 1: develop an algorithm that will lead to successful learning.

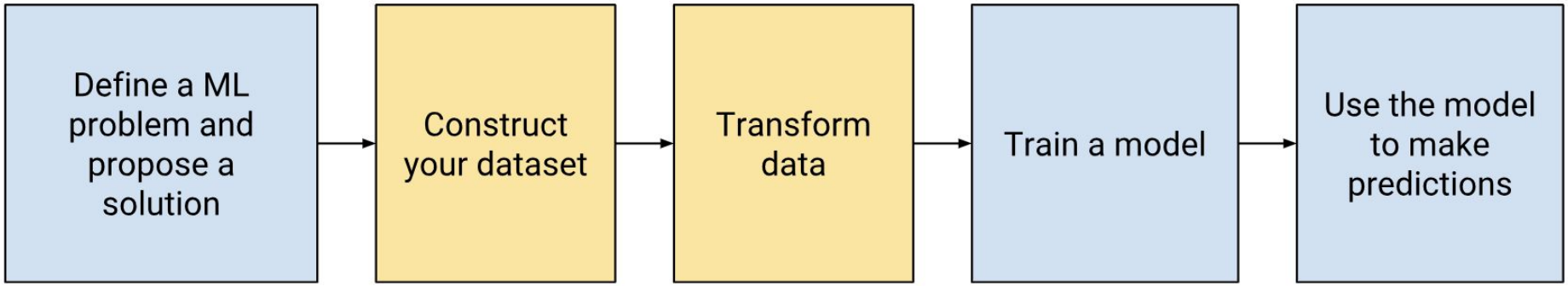
Step 2: the algorithm is provided with a large collection of TSS sequences as well as, optionally, a list of sequences that are known not to be TSSs. The annotation indicating whether a sequence is a TSS is known as the label. The algorithm processes these labelled sequences and stores a model.

Step 3: new unlabelled sequences are given to the algorithm, and it uses the model to predict labels (in this case, 'TSS' or 'not TSS') for each sequence. If the learning was successful, then all or most of the predicted labels will be correct.

Some more examples of ML in biology

1. To identify splice sites
2. To identify promoters, enhancers, or positioned nucleosomes
3. To annotating genes — including their untranslated regions (UTRs), introns and exons — along entire eukaryotic chromosomes
4. Gene expression (microarrays and RNAseq)
5. Gene Onotology term assignments
6. Infer gene networks
7. DNAase-seq, MNase-seq, FAIRE-seq, ChIP-seq analysis

Data types and partitioning



Behind the hood - 1

1. The algorithms take a subset of observations called as the **training data** and tests them on a different subset of data called as the **test data**.

2. **Dev (development set)** - this is used to tune parameters, select features, and make other decisions regarding the learning algorithm. It is also sometimes called the **hold-out cross validation set**

3. **Test set -** this is used to evaluate the performance of the algorithm, but not to make any decisions regarding what learning algorithm or parameters to use.

- Dev set and test set data should come from the same distribution

Behind the hood-II

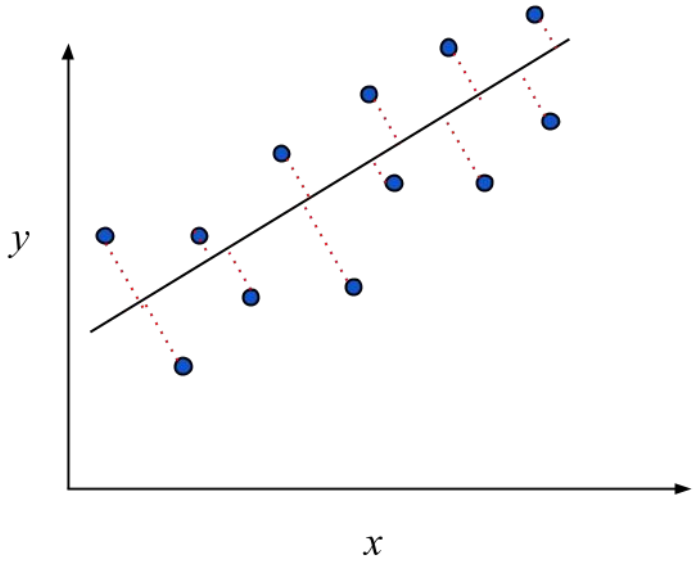
4. The error between the prediction of the outcome variable the actual data is evaluated as **test error**.

5. The **objective function** of the algorithm is to minimise these test errors by tuning the parameters of the hypothesis.

6. Models that successfully capture these desired outcomes are further evaluated for **Bias** and **Variance**(overfitting and underfitting).

For linear models

architecture



hypothesis

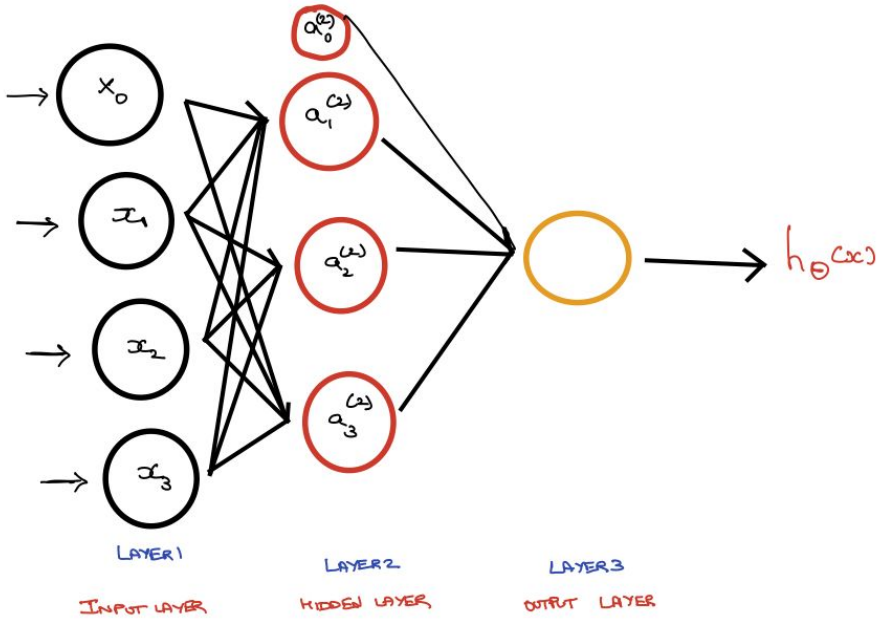
$$h_\theta(x) = \theta_1 x + \theta_0$$

Cost function = $$1/2n \sum_{i=1}^{n} (h_\theta(x^i) - y^i)^2$$

Machine learning algorithm tries to **minimise** this Cost Function

For neural network models

architecture



hypothesis

$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$
$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(1)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

Cost function =

$$CF(\Theta) = -1/m \left[\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) log(1 - (h_\Theta(x^{(i)}))_k) \right]$$

$$+\lambda/2m \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{S_{l+1}} (\Theta_{ji}^{(l)})^2$$

Data transformation

If you had to prioritize improving one of the areas below in your machine learning project, which would have the most impact?

Using the latest optimization algorithm

A more clever loss function

A deeper network

The quality and size of your data

How large Dev set & test set should be?

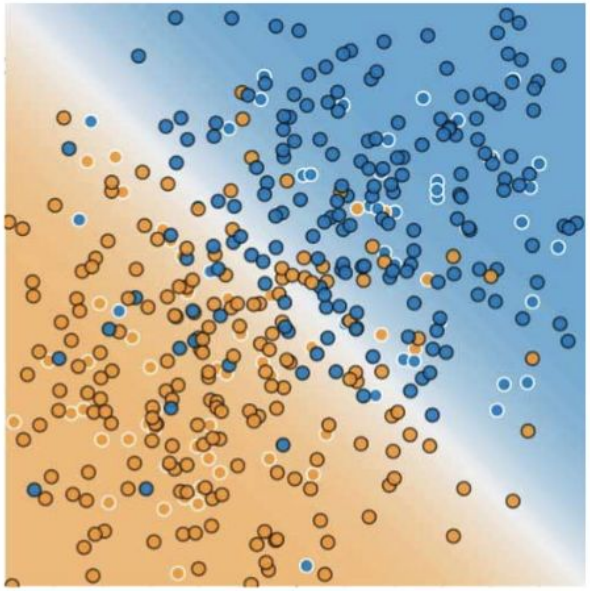
The dev set should be large enough to detect differences between algorithms that you are trying out.

if classifier A has an accuracy of 95.0% and classifier B has an accuracy of 95.1%, then a dev set of 100 examples would not be able to detect this 0.1% difference.

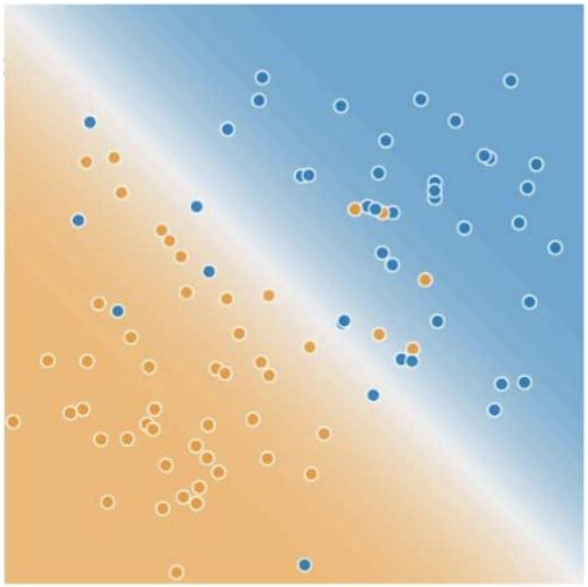
Dev sets with sizes from 1,000 to 10,000 examples are common. With 10,000 examples, there is a good chance of detecting an improvement of 0.1%

There is no need to have excessively large test sets beyond what is needed to evaluate the performance of your algorithms. For example 30% is large in the big data world

How large Dev set & test set should be?



Training Data



Test Data

More training better model

Better confidence intervals

For smaller dataset we do cross validation

Constructing data

Collect the raw data.
Identify feature and label sources.
Select a sampling strategy.
Split the data.

You're on a brand new machine learning project, about to select your first features. How many features should you pick?

Pick 1-3 features that seem to have strong predictive power.
Pick 4-6 features that seem to have strong predictive power.
Pick as many features as you can, so you can start observing which features have the strongest predictive power.

Know your data

Omitted values. For instance, a person forgot to enter a value for a house's age.

Duplicate examples. For example, a server mistakenly uploaded the same logs twice.

Bad labels. For instance, a person mislabeled a picture of an oak tree as a maple.

Bad feature values. For example, someone typed in an extra digit, or a thermometer was left out in the sun.

Histograms are a great mechanism for visualizing your data in the aggregate. In addition, getting statistics like the following can help:

Maximum and minimum
Mean and median
Standard deviation

Data transformation - sanity check

Explore, Clean, and Visualize Your Data

Explore and clean up your data before performing any transformations on it.

Examine several rows of data.

Check basic statistics.

Fix missing numerical entries.

What type of data do you have?

Transactional log - record a specific event

Attribute data contains snapshots of information.

Aggregate statistics create an attribute from multiple transactional logs

Identifying features

Labelled

housingMedianAge (feature)	totalRooms (feature)	totalBedrooms (feature)	medianHouseValue (label)
15	5612	1283	66900
19	7650	1901	80100
17	720	174	85700
14	1501	337	73400
20	1454	326	65500

UnLabelled

housingMedianAge (feature)	totalRooms (feature)	totalBedrooms (feature)
42	1686	361
34	1226	180
33	1077	271

Identifying label and sources

The best label is a **direct label** of what you want to predict

A **derived label** because it does not directly measure what you want to predict

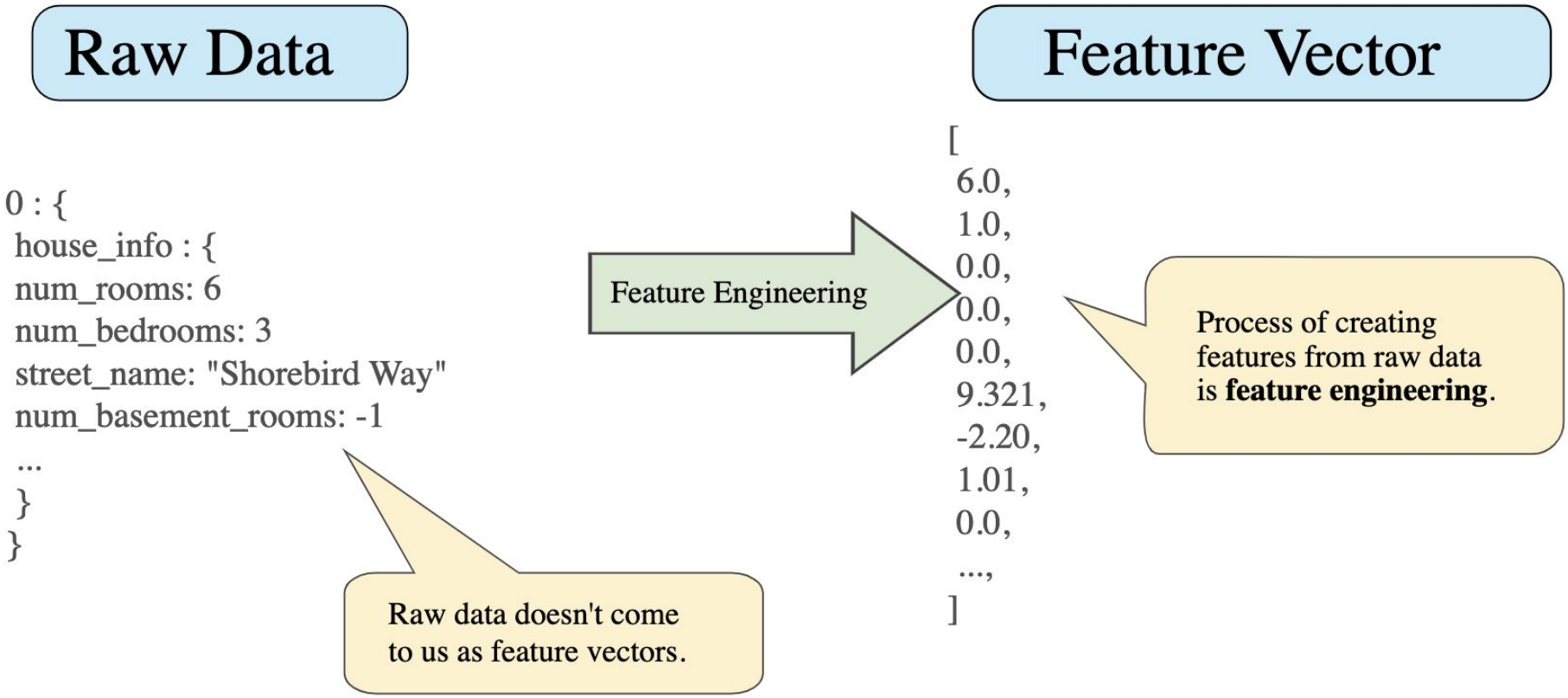
The output of your model could be either an Event or an Attribute. This results in the following two types of labels:

Direct label for Events, such as “Did the user click the top search result?”
Direct label for Attributes, such as “Will the advertiser spend more than $X in the next week?”

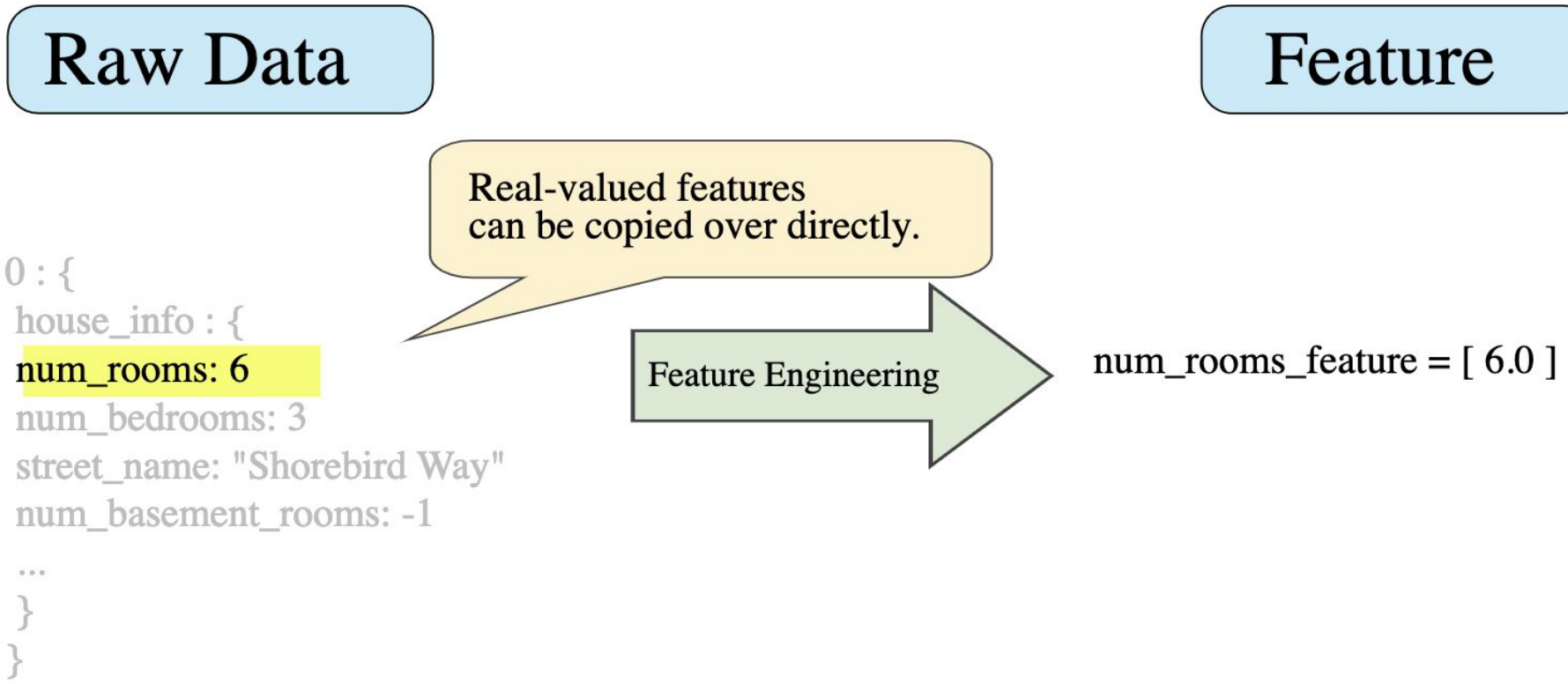
Qualities of good features

1. Avoid rarely used discrete feature values
2. Prefer clear and obvious meanings
3. Don't mix "magic" values with actual data
4. Feature shouldn't change over time

Feature engineering

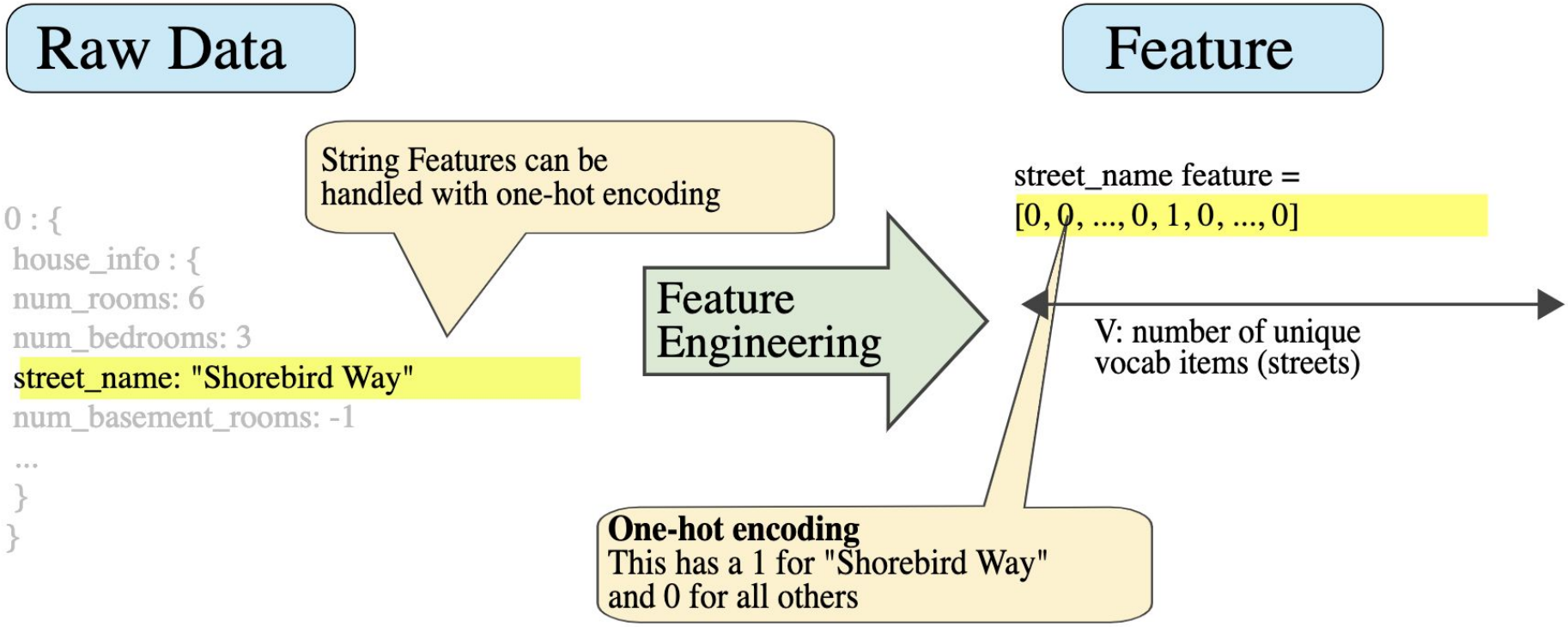


Mapping numeric values



Integer and floating-point data don't need a special encoding because they can be multiplied by a numeric weight.

Mapping categorical values



Features having a discrete set of possible values. For example, consider a categorical feature named house style, which has a discrete set of three possible values: Tudor, ranch, colonial

Numerical Data transformation - scaling

Scaling feature values

Scaling means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range (for example, 0 to 1 or -1 to +1)

Helps gradient descent converge more quickly.
Helps avoid the "NaN trap," in which one number in the model becomes a NaN (e.g., when a value exceeds the floating-point precision limit during training), and—due to math operations—every other number in the model also eventually becomes a NaN.
Helps the model learn appropriate weights for each feature.
Without feature scaling, the model will pay too much attention to the features having a wider range

One obvious way to scale numerical data is to linearly map [min value, max value] to a small scale, **such as [-1, +1].**

Another popular scaling tactic is to calculate the **Z score** of each value. The Z score relates the number of standard deviations away from the mean. In other words:

$$scaledvalue = (value - mean)/stddev.$$

Numerical Data transformation - normalization

You may need to apply two kinds of transformations to numeric data:

- **Normalizing** - transforming numeric data to the same scale as other numeric data.
- **Bucketing** - transforming numeric (usually continuous) data to categorical data.

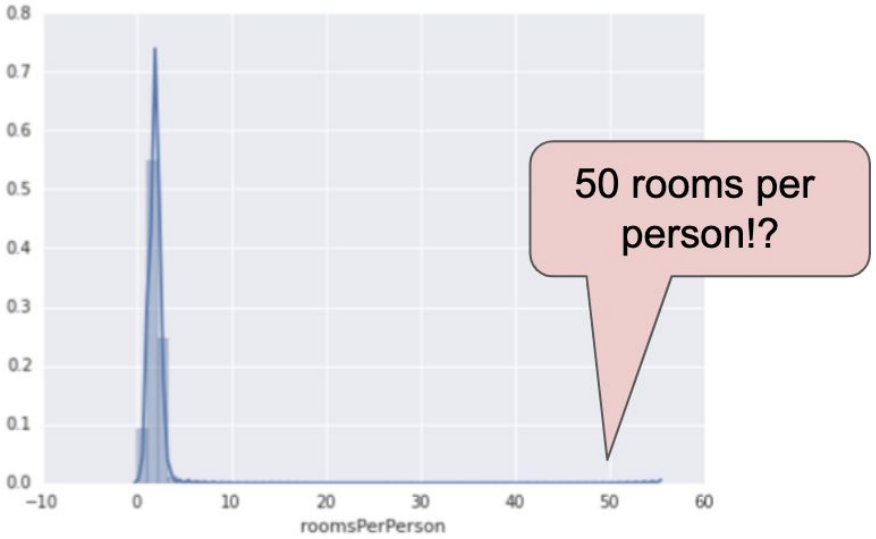
Why Normalize Numeric Features?

Normalization is necessary if you have very different values within the same feature (for example, city population). Without normalization, your training could blow up with NaNs if the gradient update is too large.

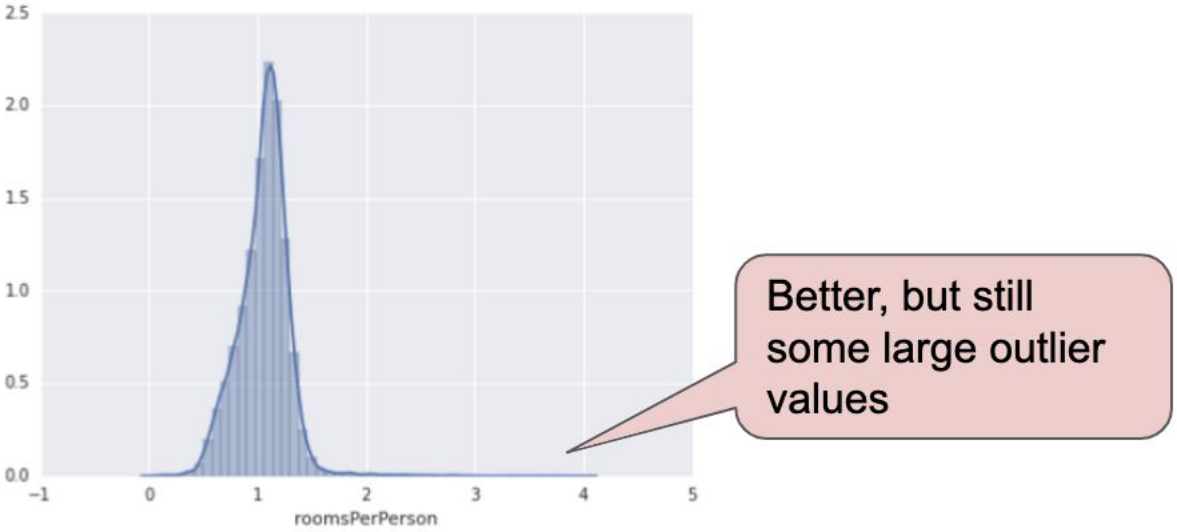
You might have two different features with widely different ranges (e.g., age and income), causing the gradient descent to "bounce" and slow down convergence

Data transformation - Normalisation

Handling extreme outliers



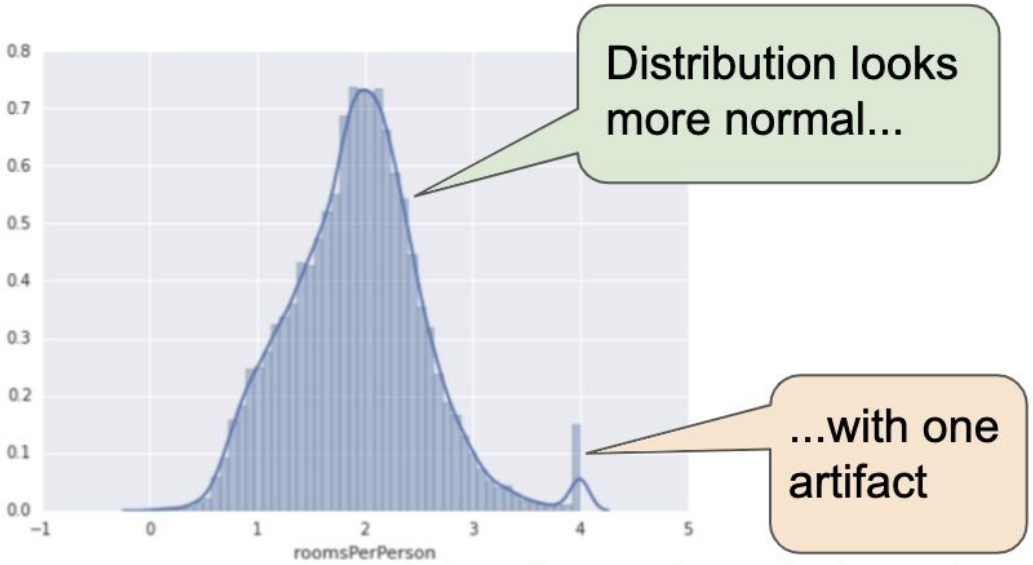

```
roomsPerPerson = totalRooms / population
```




```
roomsPerPerson = log((totalRooms / population) + 1)
```

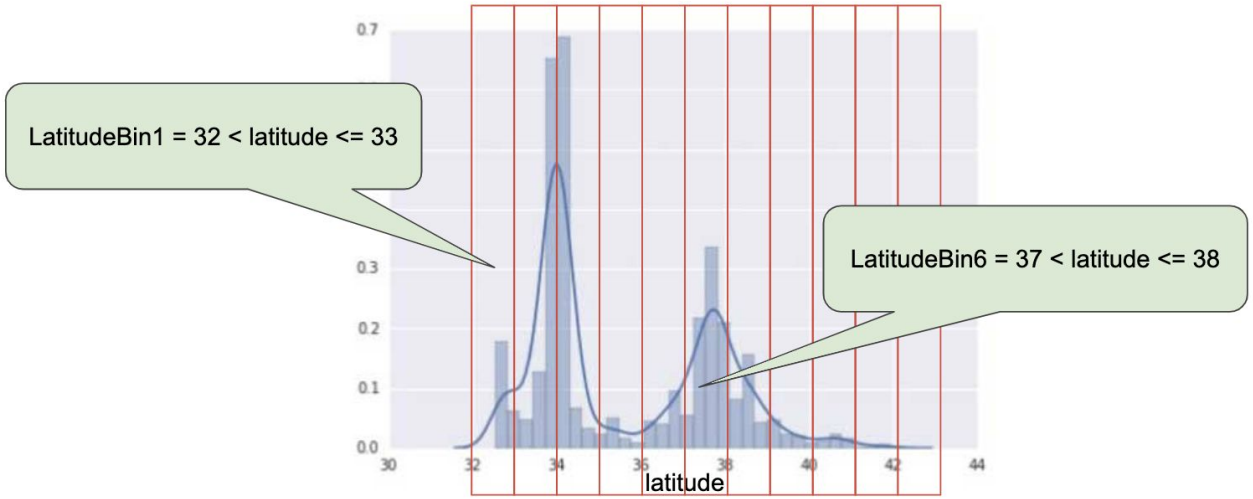
Data transformation - Normalisation

Clipping feature values at 4.0



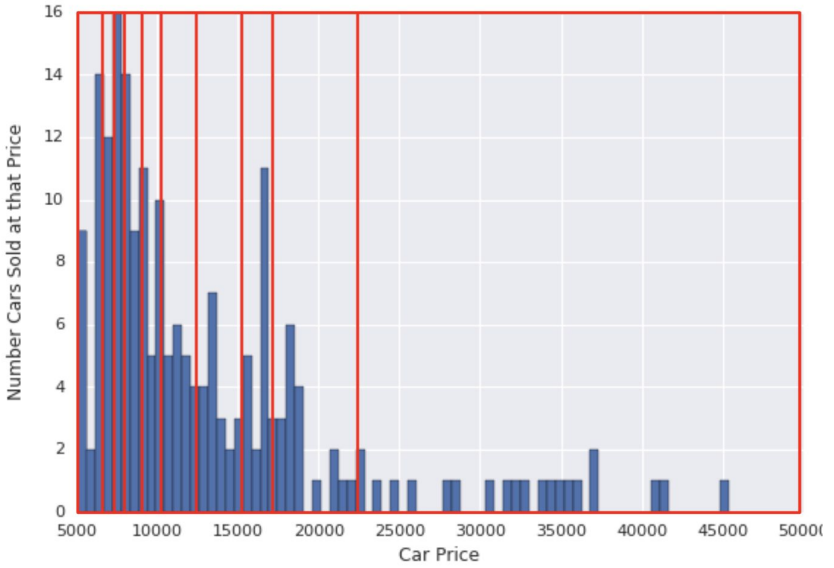

```
roomsPerPerson = min(totalRooms / population, 4)
```

Data transformation - Binning/ Bucketing



Buckets with equal bins

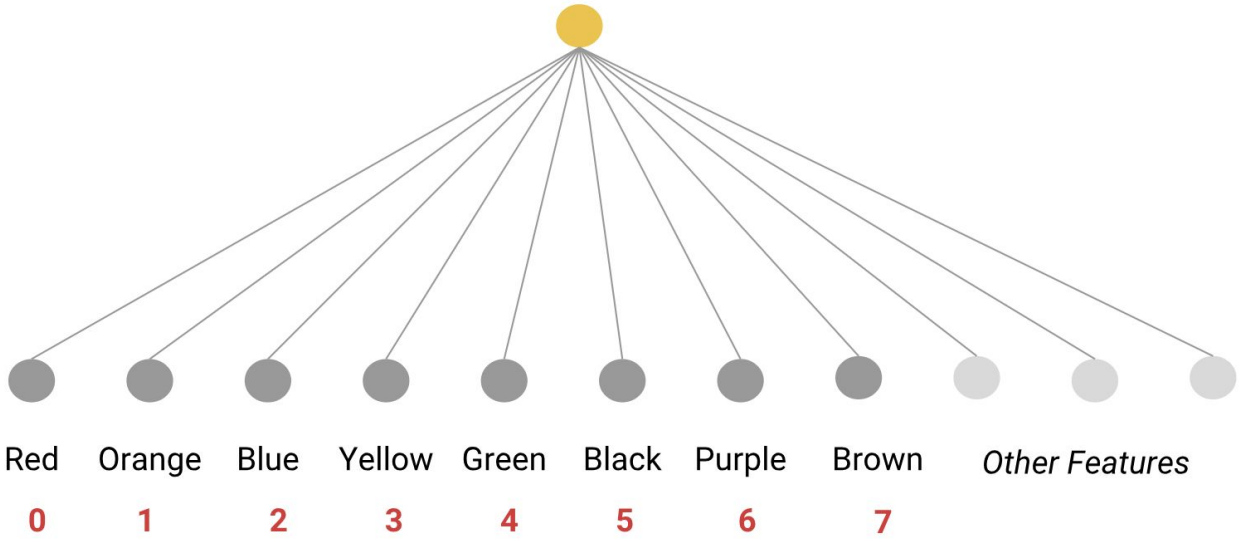
Buckets with quantile bins



Data transformation summary

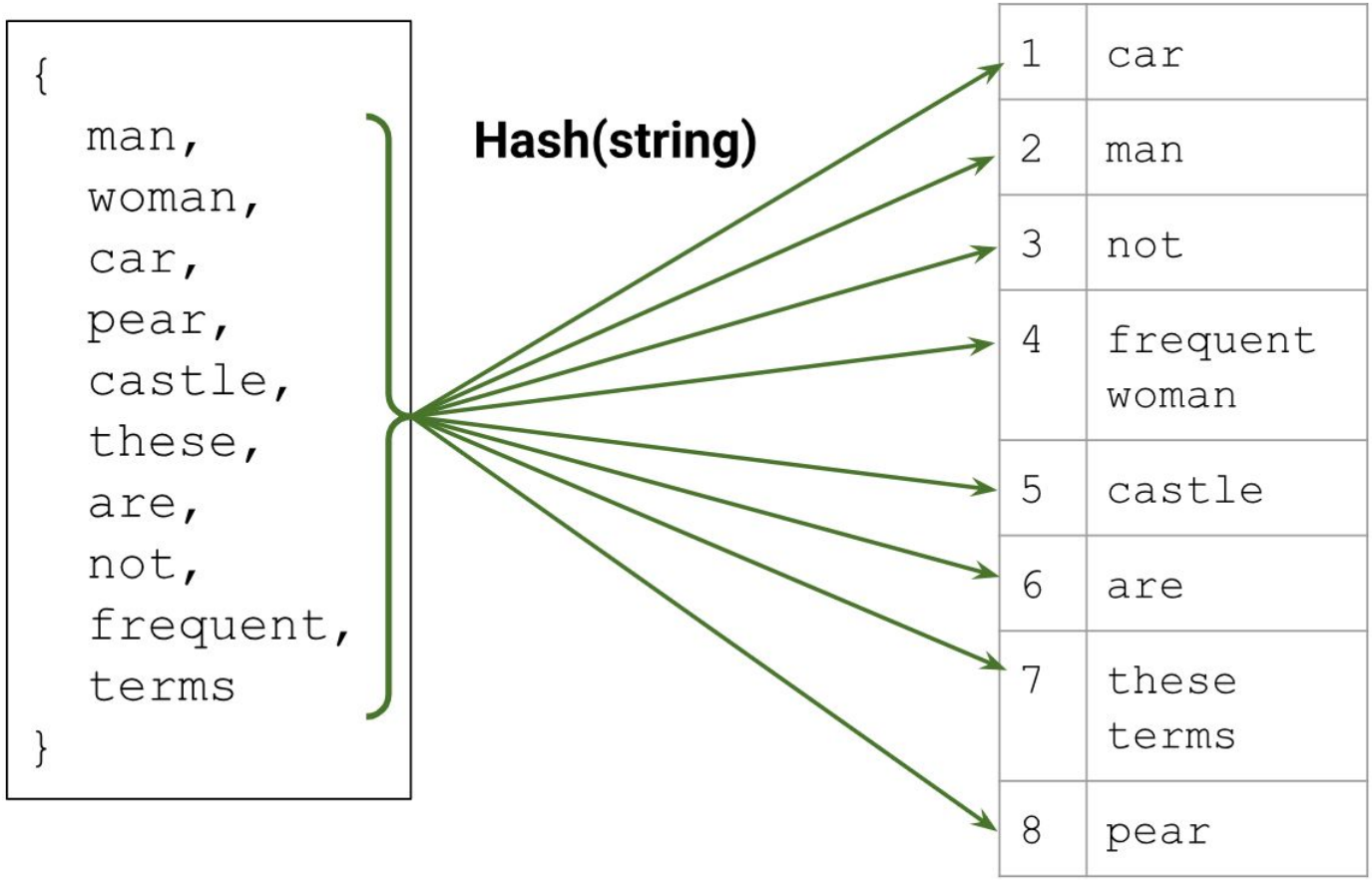
Normalization Technique	Formula	When to Use
Linear Scaling	$x' = (x - x_{min})/(x_{max} - x_{min})$	When the feature is more-or-less uniformly distributed across a fixed range.
Clipping	if x > max, then x' = max. if x < min, then x' = min	When the feature contains some extreme outliers.
Log Scaling	x' = log(x)	When the feature conforms to the power law.
Z-score	x' = (x - μ) / σ	When the feature distribution does not contain extreme outliers.

Categorical Data transformation



This sort of mapping is called a **vocabulary**.

Categorical Data transformation



Hashing

Reasons for Data transformation

Converting non-numeric features into numeric. You can't do matrix multiplication on a string

Resizing inputs to a fixed size. Linear models and feed-forward neural networks have a fixed number of input nodes, so your input data must always have the same size. For example, image models need to reshape the images in their dataset to a fixed size.
Optional quality transformations that may help the model perform better.

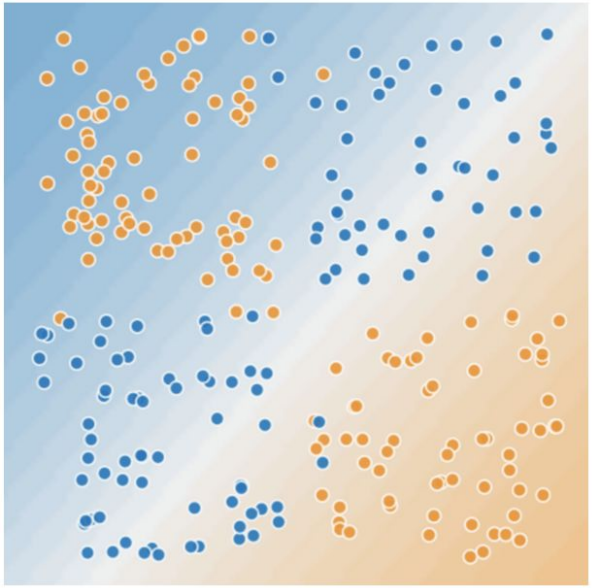
Tokenization or lower-casing of text features.

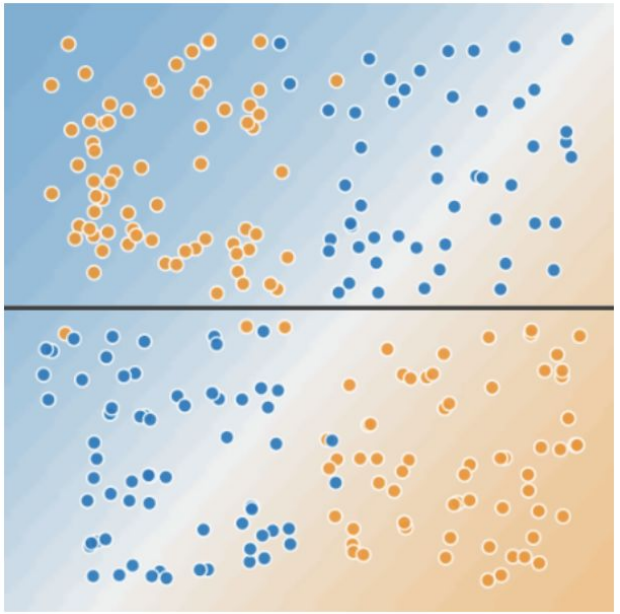
Normalized numeric features (most models perform better afterwards).

Allowing linear models to introduce non-linearities into the feature space.

Strictly speaking, quality transformations are not necessary--your model could still run without them. But using these techniques may enable the model to give better results.

Features crosses encodes nonlinearity





A feature cross is a synthetic feature that encodes nonlinearity in the feature space by multiplying two or more input features together. (The term cross comes from cross product.)

Let's create a feature cross named x_3 by crossing x_1 and : x_2

We treat this newly minted feature cross just like any other feature. The linear formula becomes:

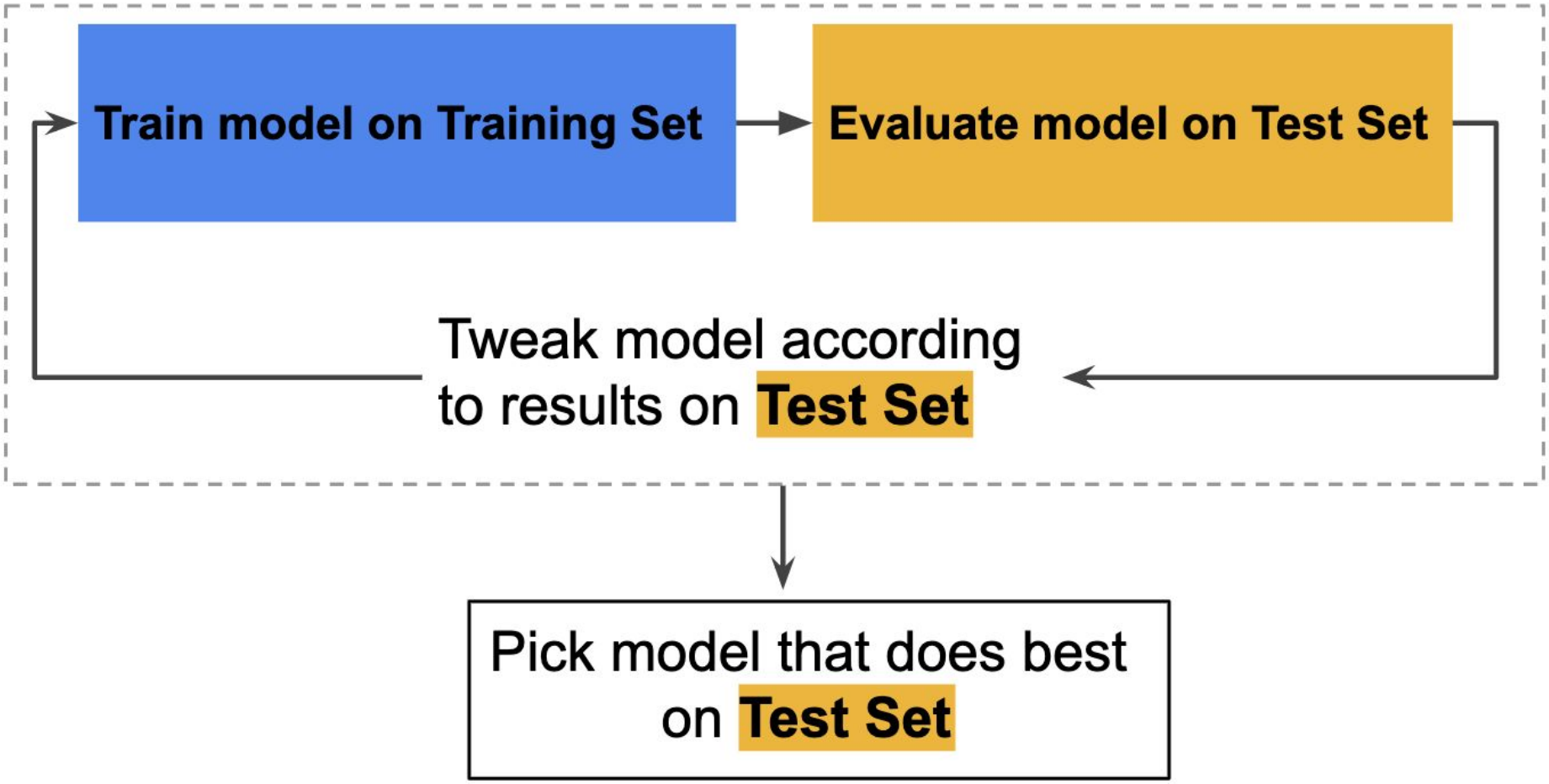
$$y = b + w_1x_1 + w_2x_2 + w_3x_3$$

A linear algorithm can learn a weight for just as it would for and . In other words, although encodes nonlinear information, you don't need to change how the linear model trains to determine the value of .

Partitioning dataset - part I

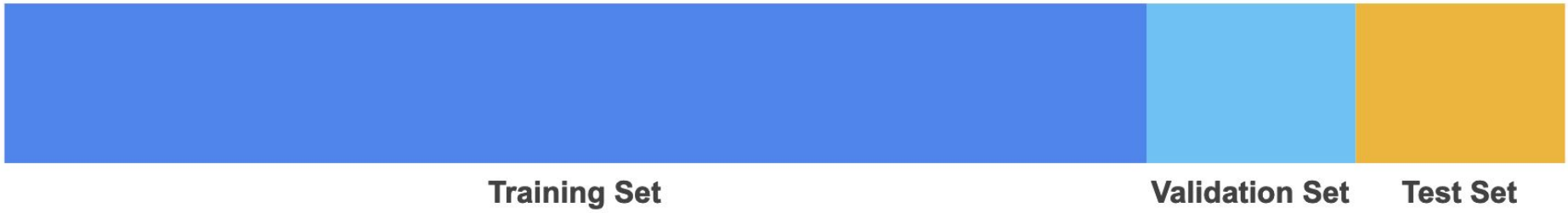
One large data set

Train-test workflow

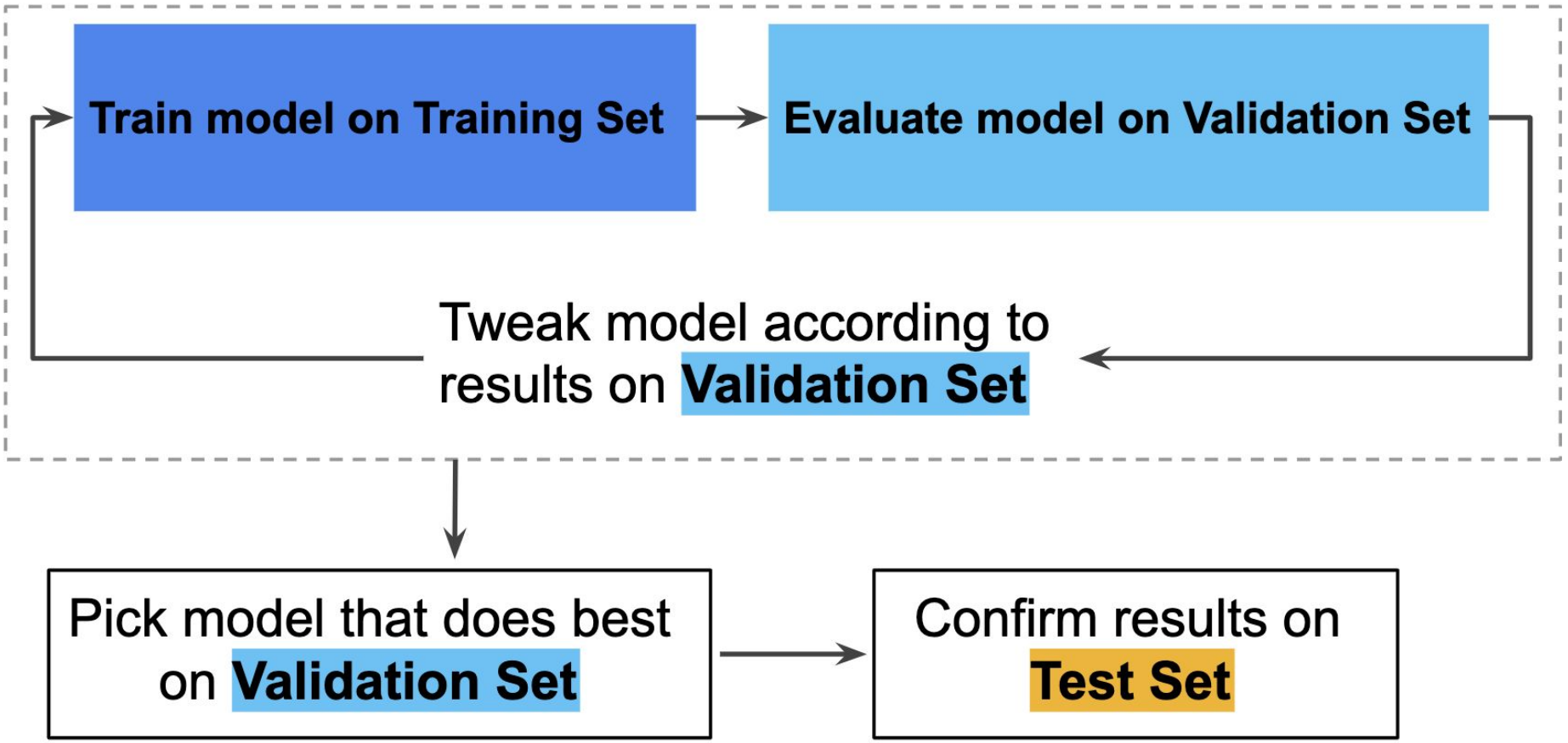


Problem: Overfitting on test set

Partitioning dataset - part II



Train-test workflow



Pick the model that does best on the validation set.
Double-check that model against the test set.

Three basic assumptions of the test/validation dataset

1. We draw examples **independently and identically (i.i.d.)** at random from the distribution

2. The distribution is **stationary**: It doesn't change over time

3. We always pull from the **same distribution**: Including training, validation, and test sets

Introduction to sampling

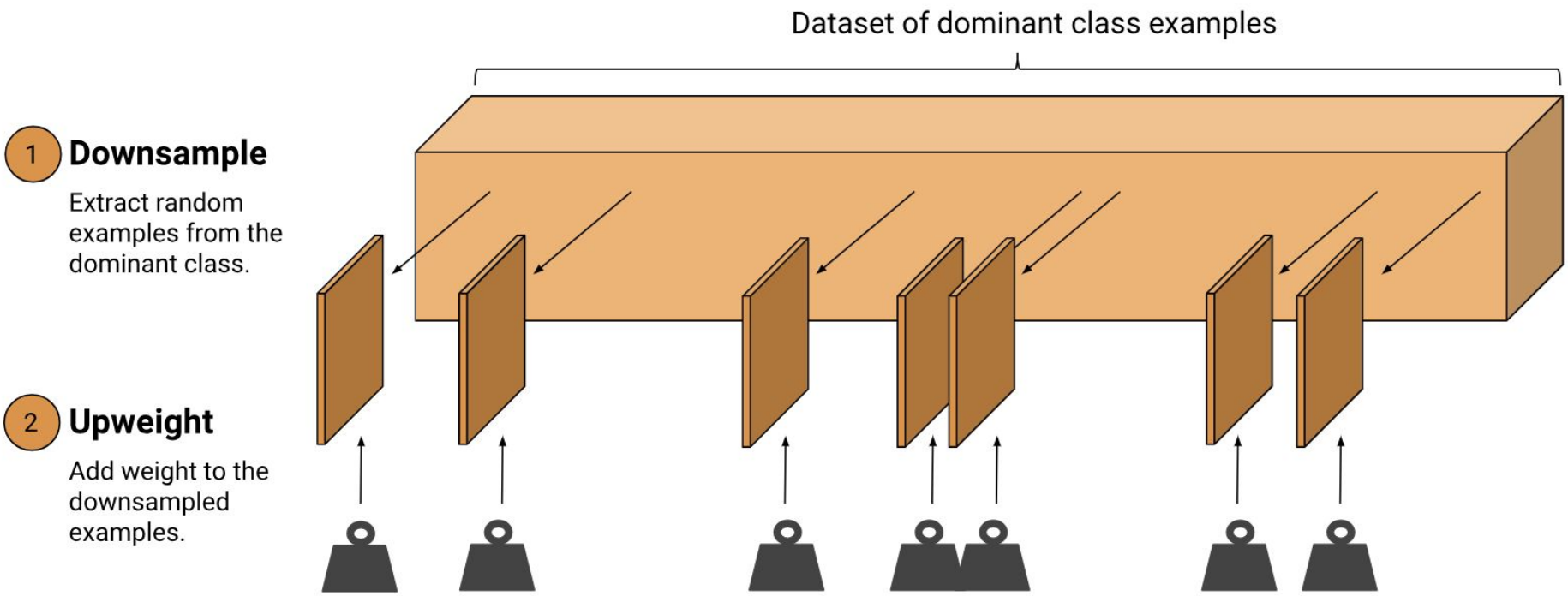
It's often a struggle to gather enough data for a machine learning project. Sometimes, however, there is too much data, and you must select a subset of examples for training.

Imbalanced data
A classification data set with skewed class proportions is called imbalanced. Classes that make up a large proportion of the data set are called majority classes. Those that make up a smaller proportion are minority classes.

Downsampling and Upweighting
An effective way to handle imbalanced data is to downsample and upweight the majority class. Let's start by defining those two new terms:

Downsampling (in this context) means training on a disproportionately low subset of the majority class examples.

Upweighting means adding an example weight to the downsampled class equal to the factor by which you downsampled.



Why Downsample and Upweight?
It may seem odd to add example weights after downsampling. We were trying to make our model improve on the minority class -- why would we upweight the majority? These are the resulting changes:

Faster convergence: During training, we see the minority class more often, which will help the model converge faster.
Disk space: By consolidating the majority class into fewer examples with larger weights, we spend less disk space storing them. This savings allows more disk space for the minority class, so we can collect a greater number and a wider range of examples from that class.
Calibration: Upweighting ensures our model is still calibrated; the outputs can still be interpreted as probabilities.

Issues with random splitting

When Random Splitting isn't the Best Approach
While random splitting is the best approach for many ML problems, it isn't always the right solution. For example, consider data sets in which the examples are naturally clustered into similar examples.

Suppose you want your model to classify the topic from the text of a news article. Why would a random split be problematic?

Practical considerations for random splitting

Make your data generation pipeline reproducible. Say you want to add a feature to see how it affects model quality. For a fair experiment, your datasets should be identical except for this new feature. If your data generation runs are not reproducible, you can't make these datasets.

In that spirit, make sure any randomization in data generation can be made deterministic:

Seed your random number generators (RNGs). Seeding ensures that the RNG outputs the same values in the same order each time you run it, recreating your dataset.

Use **invariant hash keys**. Hashing is a common way to split or sample data. You can hash each example, and use the resulting integer to decide in which split to place the example. The inputs to your hash function shouldn't change each time you run the data generation program. Don't use the current time or a random number in your hash, for example, if you want to recreate your hashes on demand.

The preceding approaches apply both to sampling and splitting your data.

Resampling methods

Resampling methods are an indispensable tool in modern statistics. They involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model. For example, in order to estimate the variability of a linear regression fit, we can repeatedly draw different samples from the training data, fit a linear regression to each new sample, and then examine the extent to which the resulting fits differ. Such an approach may allow us to obtain information that would not be available from fitting the model only once using the original training sample.

Cross validation & Bootstrapping

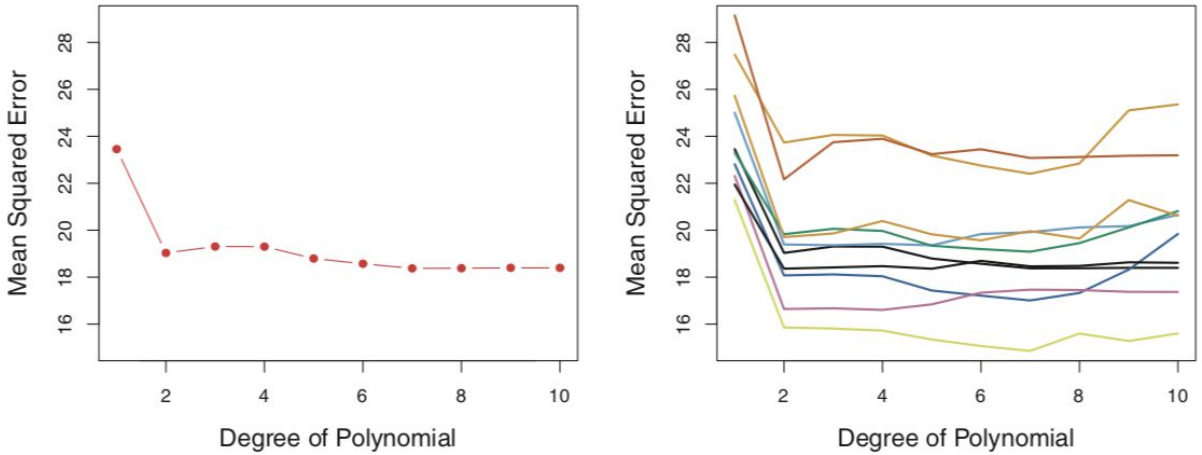
Both methods are important tools in the practical application of many statistical learning procedures.

For example, cross-validation can be used to estimate the test error associated with a given statistical learning method in order to evaluate its performance, or to select the appropriate level of flexibility.

The process of evaluating a model's performance is known as **model assessment**, whereas the process of selecting the proper level of flexibility for a model is known as **model selection**.

The bootstrap is used in several contexts, most commonly to provide a measure of accuracy of a parameter estimate or of a given statistical learning method.

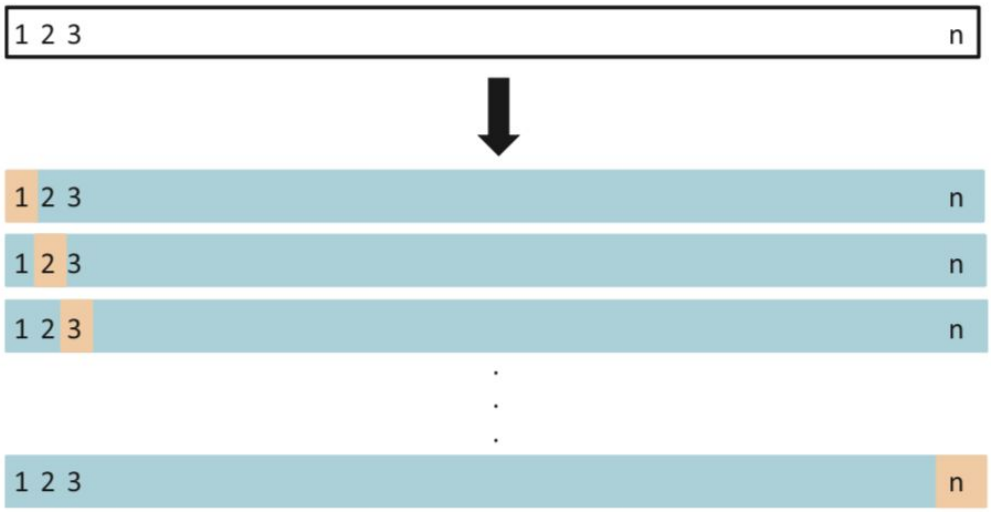
Validation set approach



the training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.

Leave-one-cross-validation

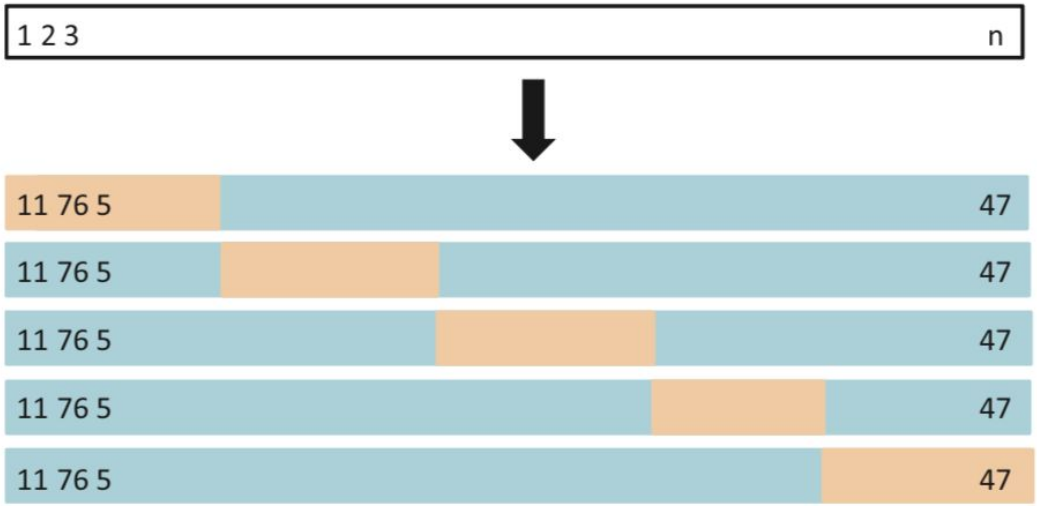
Leave-one-out cross-validation (LOOCV) is closely related to the validation leave-one- set approach, but it attempts to address that method's out drawbacks.



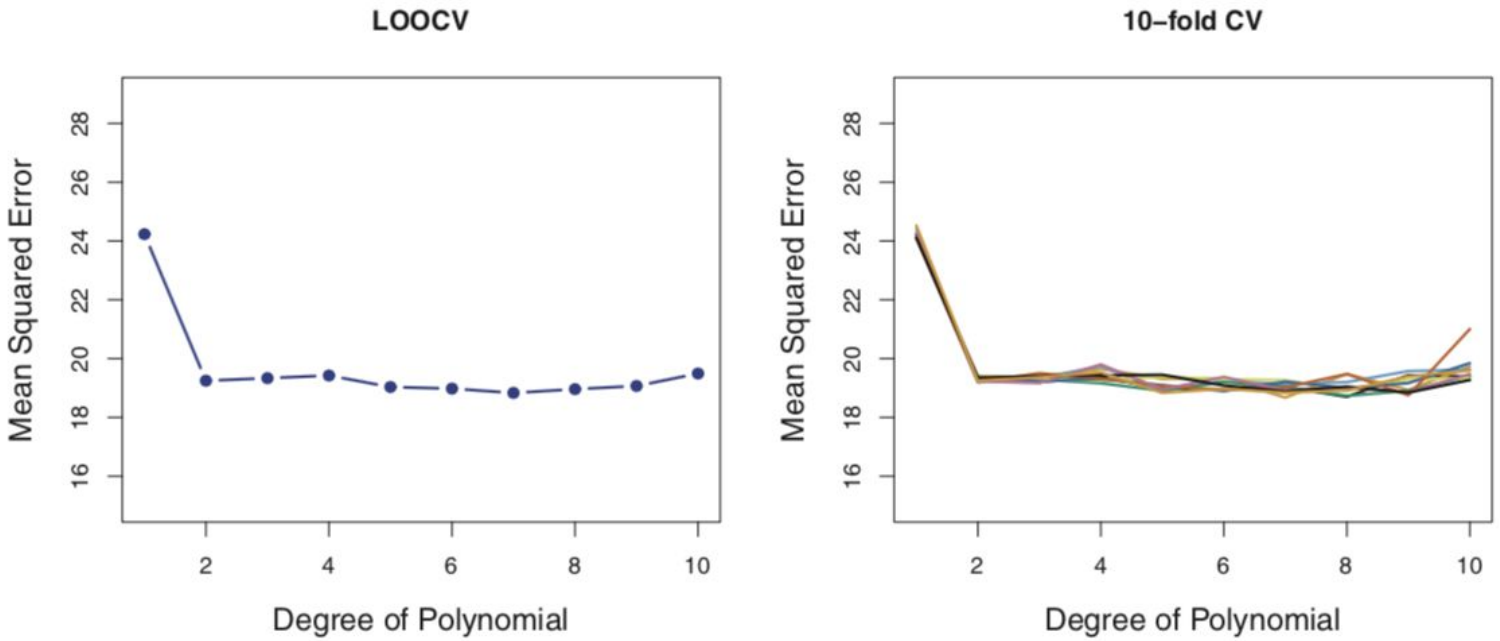
Like the validation set approach, LOOCV involves splitting the set of

observations into two parts. However, instead of creating two subsets of comparable size, a single observation (x_1,y_1) is used for the validation set, and the remaining observations $\{(x_2, y_2), \ldots, (x_n, y_n)\}$ make up the training set. The statistical learning method is fit on the $n - 1$ training observations, and a prediction $\hat{y}_1$ is made for the excluded observation, using its value x_1. Since (x_1, y_1) was not used in the fitting process, $MSE_1 = (y_1 - \hat{y}_1)^2$ provides an approximately unbiased estimate for the test error. But even though MSE_1 is unbiased for the test error, it is a poor estimate because it is highly variable, since it is based upon a single observation (x_1,y_1).

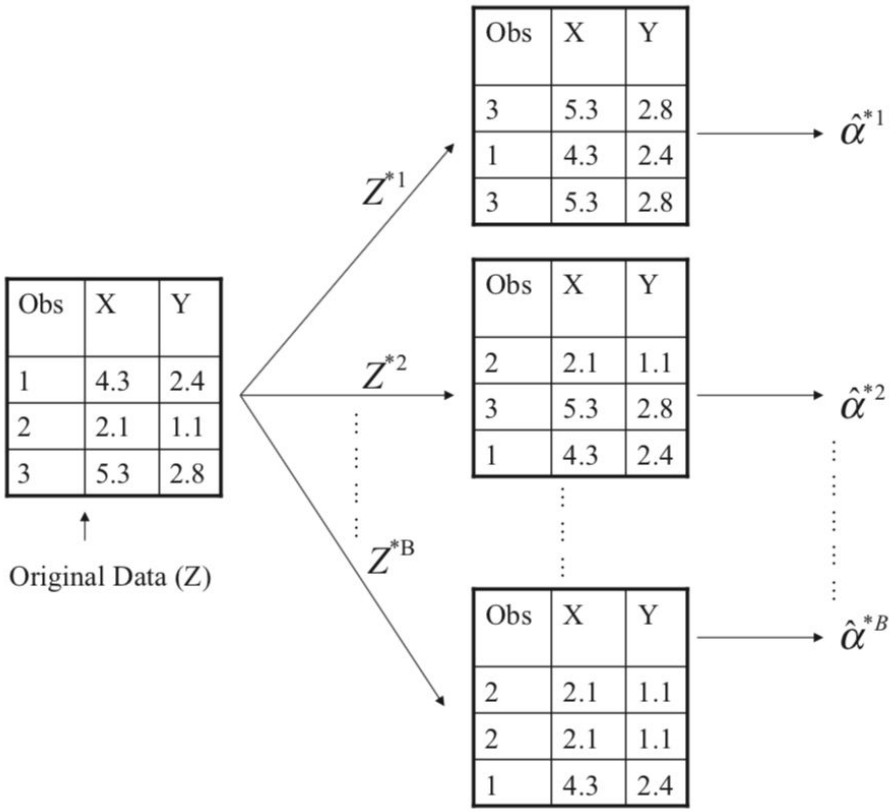
K-fold Cross validation



Bias Variance trade off



Bootstrapping



The bootstrap is a widely applicable and extremely powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method.

How to evaluate your ML algorithm

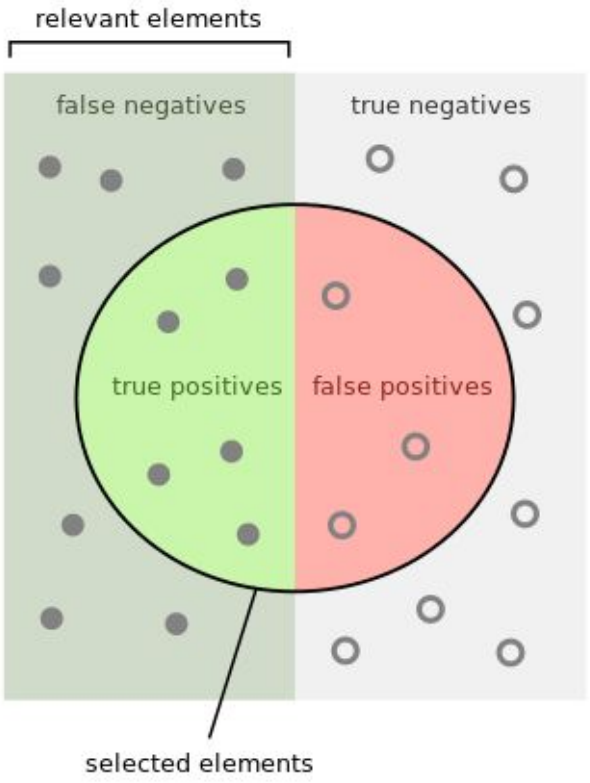
A single number metric is the easiest one!

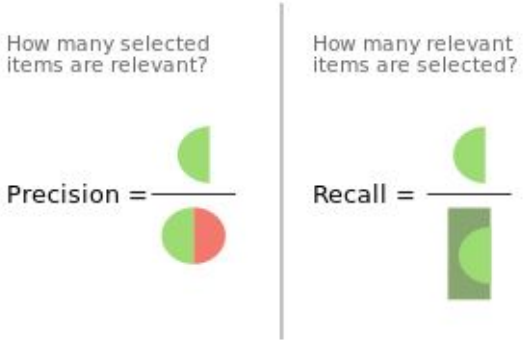
For example Accuracy

If a classifier A gives 97% accuracy and classifier B gives 95% accuracy, then we can conclude that A is better

Often Precision Recalls are used

Precision - Recall





precision is "how useful the search results are", and recall is "how complete the results are".

Classifier A 95% Precision and 90% Recall
Classifier B 98% Precision and 85% Recall

Average or an harmonic average - F1 score - of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

A single number is the best way to evaluate a ML algorithm

Source: Wikipedia

Other metrics

Run time

Size of the application

False positives - False negatives

Error Analysis

Error Analysis examining dev set examples that your algorithm misclassified, so that you can understand the underlying causes of the errors. This can help you prioritize projects.

Write down all the misclassified items and analyse them to efficiently evaluate and fix problems

If the Dev set is large, split it into two and evaluate one manually

Bias and Variance - I

There are two major sources of error in machine learning: bias and variance.

The algorithm's error rate on the training set - **bias**

The algorithm does on the dev (or test) set than the training set - **variance**

- Training error = 1% (bias)
- Dev error = 11% (variance=10%)

Overfitting

- Training error = 15% (bias)
- Dev error = 30% (variance = 15%)

High Bias and High Variance

- Training error = 15% (bias)
- Dev error = 16% (viance =1%)

Underfitting

- Training error = 0.5% (bias)
- Deverror = 1% (variance)

Perfect

Optimal error rate ("unavoidable bias")

Bias and Variance - II

To reduce bias

1. Increase the model size
2. Modifying features based on error analysis
3. Reduce or eliminate regularization
4. Modify model architecture

To reduce variance

1. Add more training data
2. Add regularization
3. Feature selection to decrease the number of inputs or features
4. Decrease the model size
5. Modify input features based on error analysis

Some pointers

Part IV

Unsupervised and semi-supervised examples

When a labelled training set is not available, unsupervised learning is required. For example, consider the interpretation of a heterogeneous collection of epigenomic data sets,ENCODE consortium.

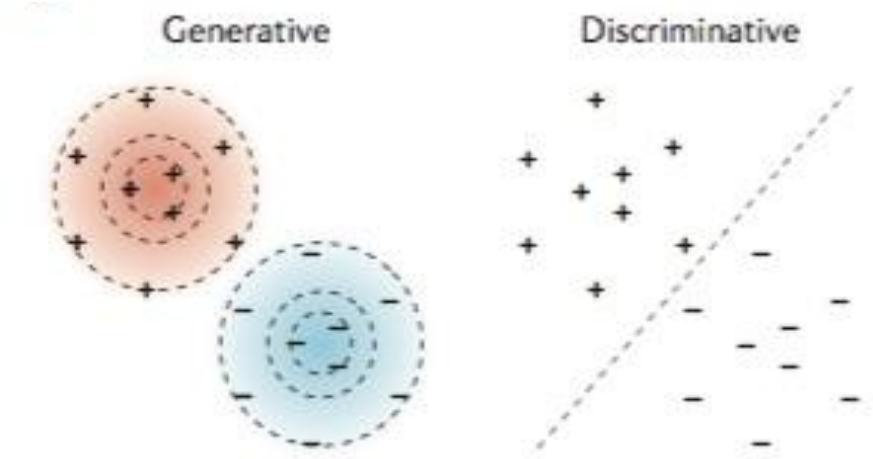
Gene-finding systems are often trained using a semi-supervised approach, in which the input is a collection of annotated genes and an unlabelled whole-genome sequence. The learning procedure begins by constructing an initial gene-finding model on the basis of the labelled subset of the training data alone. Next, the model is used to scan the genome, and tentative labels are assigned throughout the genome.

General wisdom

In general, supervised learning should be used **only when the training set and test set are expected to exhibit similar statistical properties.**

For example, a TSS data set generated by cap analysis of gene expression (CAGE) will not contain non-polyadenylated genes. If such genes also exhibit differences around the TSSs, then the resulting TSS predictor will be biased.

Generative vs discriminative models



There are trade-offs between accomplishing these two goals — methods that optimize **prediction accuracy often do so at the cost of interpretability.**

A researcher applying a machine learning method to this problem may either want to understand what properties of a sequence are the most important for determining whether a transcription factor will bind (that is, interpretation), or simply want to predict the locations of transcription factor binding as accurately as possible (that is, prediction).

Some more wisdom!

A widely used generative model of transcription factor binding uses a **position-specific frequency matrix (PSFM)**

Advantage:

Missing data
Interpretability
Probability framework

A simple example of a discriminative algorithm is the **support vector machine (SVM)**, the goal of which is to learn to output a value of 1 whenever it is given a positive training example and a value of –1 whenever it is given a negative training example.

Advantage:

when the amount of labelled training data is reasonably large, the discriminative approach will tend to find a better solution, in the sense that it will predict the desired outcome more accurately when tested on previously unseen data

Incorporating prior knowledge

Implicit prior knowledge:

Using chromatin data — DNase I accessibility and ChIP–seq profiles of histone modifications and transcription factor binding — one classifier distinguished regulatory regions that are close to a gene from regulatory regions that are far away from any gene.

Probabilistic priors: Uniform priors or Dirchlet priors

Non probabilistic priors: Discriminative methods can use similarity matrix - Kernels

Things to keep in mind

Heterogenous data:
Kernalisation
Bayesian framework
Redefine it as classification problem

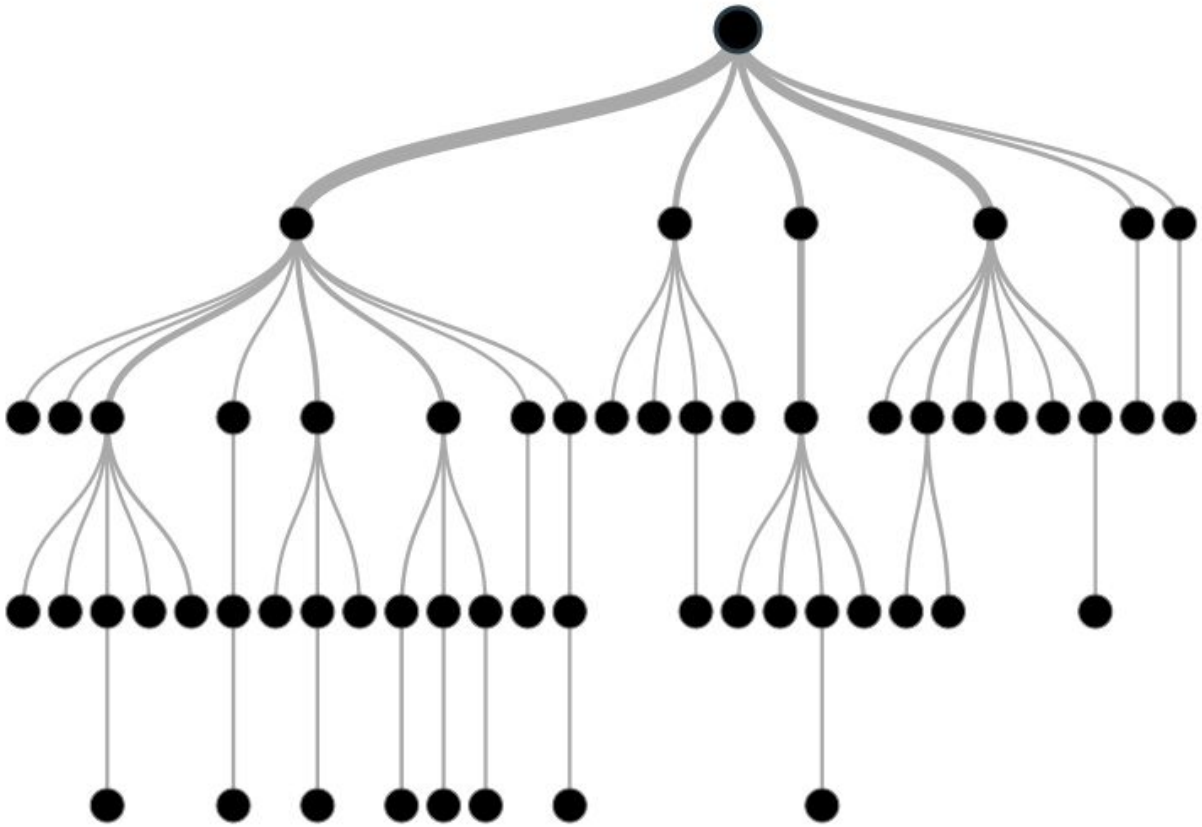
Feature selection:
No. of features. More features leads to overfitting
Reduce dimensions

Imbalanced class size:
Precision-Recall
A smaller subset of data

Missing data:
Replace with zero
Impute the values
Probability method called marginalisation

Libbrecht, M.W. & Noble, W.S., 2015. Machine learning applications in genetics and genomics. *Nature reviews. Genetics*, 16(6), pp.321–332.

Decision tree



Terminologies related to decision trees

Root nodule: the entire population that can get further divided into homogenous sets

Splitting: process of dividing a node into two or more sub-nodes

Decision node: When a sub-node splits into further sub-nodes

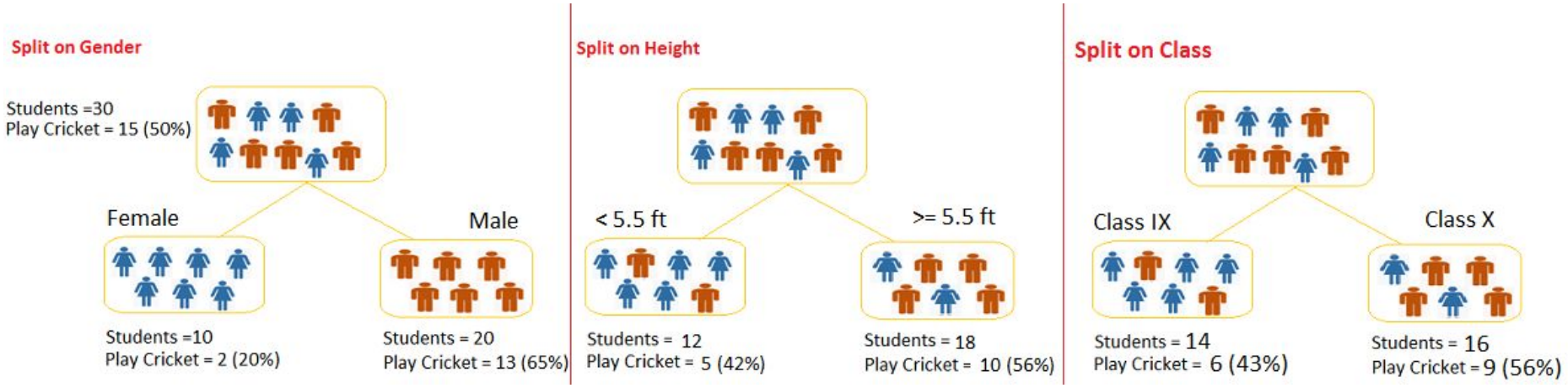
Leaf or terminal node: when a node does not split further it is called a terminal node.

Prunning: A loose stopping criteria is used to construct the tree and then the tree is cut back by removing branches that do not contribute to the generalisation accuracy.

Branch: a subsection of an entire tree

How does a tree know where to split ?

The classification tree searches through each dependent variable to find a single variable that splits the data into two or more groups and this process is repeated until the stopping criteria is invoked.



Gini index

Gini Index

If we select two items from a population at random then they must be of same class and the probability for this is 1 if population is pure.

a. It works with categorical target variable “Success” or “Failure”.
b. It performs only Binary splits
c. Higher the value of Gini higher the homogeneity.
d. CART (Classification and Regression Tree) uses Gini method to create binary splits.

Step 1: Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure p^2+q^2

Step 2: Calculate Gini for split using weighted Gini score of each node of that split.

Entropy

The more homogenous something is the less information is needed to describe it and hence it has gained information. Information theory has a measure to define this degree of disorganization in a system and it is known as Entropy. If a sample is completely homogeneous, then the entropy is zero and if it is equally divided (50% – 50%), it has entropy of one.

Entropy can be calculated using formula

$$\text{Entropy} = -p\log_2 p - q\log_2 q$$

Other statistical measures

Chi squared

And

Reduction in variance

Why and why not decision tree

Advantages of decision tree

1. Simple to understand and use
2. Algorithms are robust to noisy data
3. Useful in data exploration
4. decision tree is ‘non parametric’ in nature i.e. does not have any assumptions about the distribution of the variables
5. Nonlinear relationships between parameters do not affect tree performance

Disadvantages of decision tree

1.Overfitting is the common disadvantage of decision trees. It is taken care of partially by constraining the model parameter and by pruning.

2. It is not ideal for continuous variables as in it loses information

3. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the data set prior to fitting with the decision tree.

Parameters

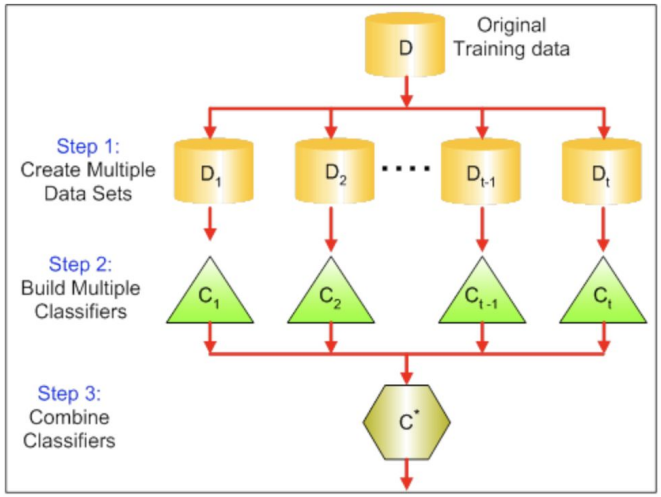
Some parameters used to defining a tree and constrain overfitting

1. Minimum sample for a node split
2. Minimum sample for a terminal node
3. Maximum depth of a tree
4. Maximum number of terminal nodes
5. Maximum features considered for split

Methods used in decision trees for trade off balance

***Ensemble** methods* involve group of predictive models to achieve a better accuracy and model stability. Ensemble methods are known to impart supreme boost to tree based models.

Bagging is a technique used to reduce the variance of predictions by combining the result of multiple classifiers modeled on different sub-samples of the same data set.



Methods used in decision trees for trade off balance

Boosting refers to a family of algorithms which converts weak learner to strong learner by combing the prediction of each weak learner using methods like average/ weighted average or by considering a prediction that has a higher vote. Gradient boosting and XGboost are examples of boosting algorithms.

Random Forest

It is a kind of ensemble learning method that combines a set of weak models to form a powerful model. In the process it **reduces dimensionality, removes outliers**, treats missing values, and more importantly it is both a regression and classification machine learning approach.

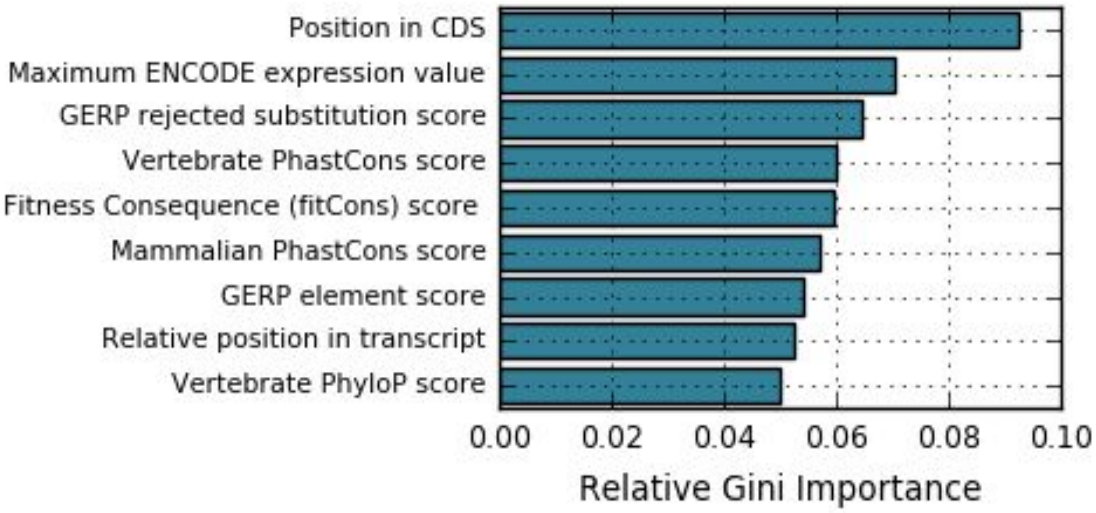
Advantages of Random Forest

This algorithm can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts.

One of benefits of Random forest is its power of handle large data set with higher dimensionality.

It can handle thousands of input variables and identify most significant variables so it is considered as one of the dimensionality reduction methods.

Further, the model outputs Importance of variable, which can be a very handy feature (on some random data set).



Advantages of Random Forest

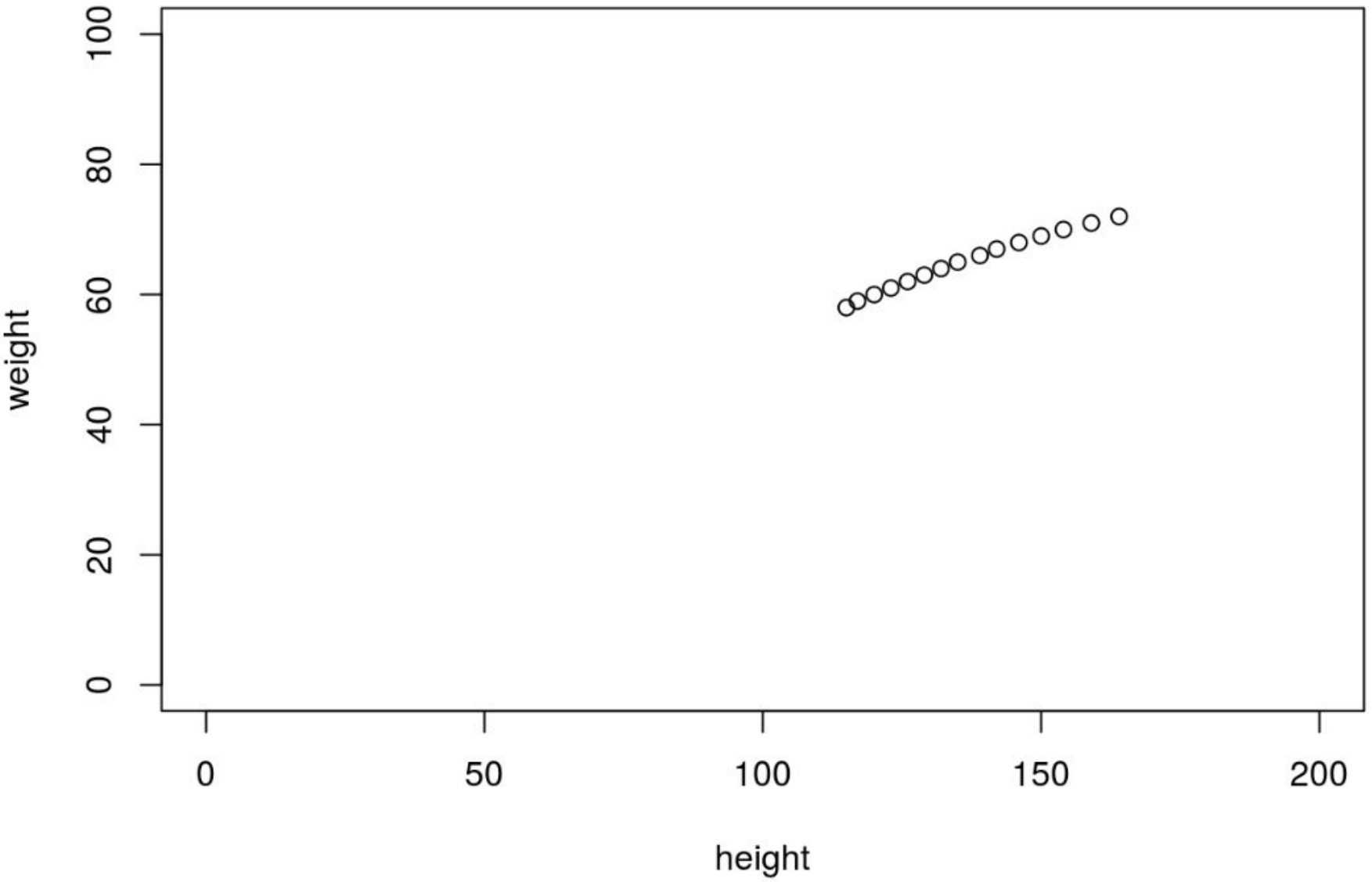
It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.

It has methods for balancing errors in data sets where classes are imbalanced.

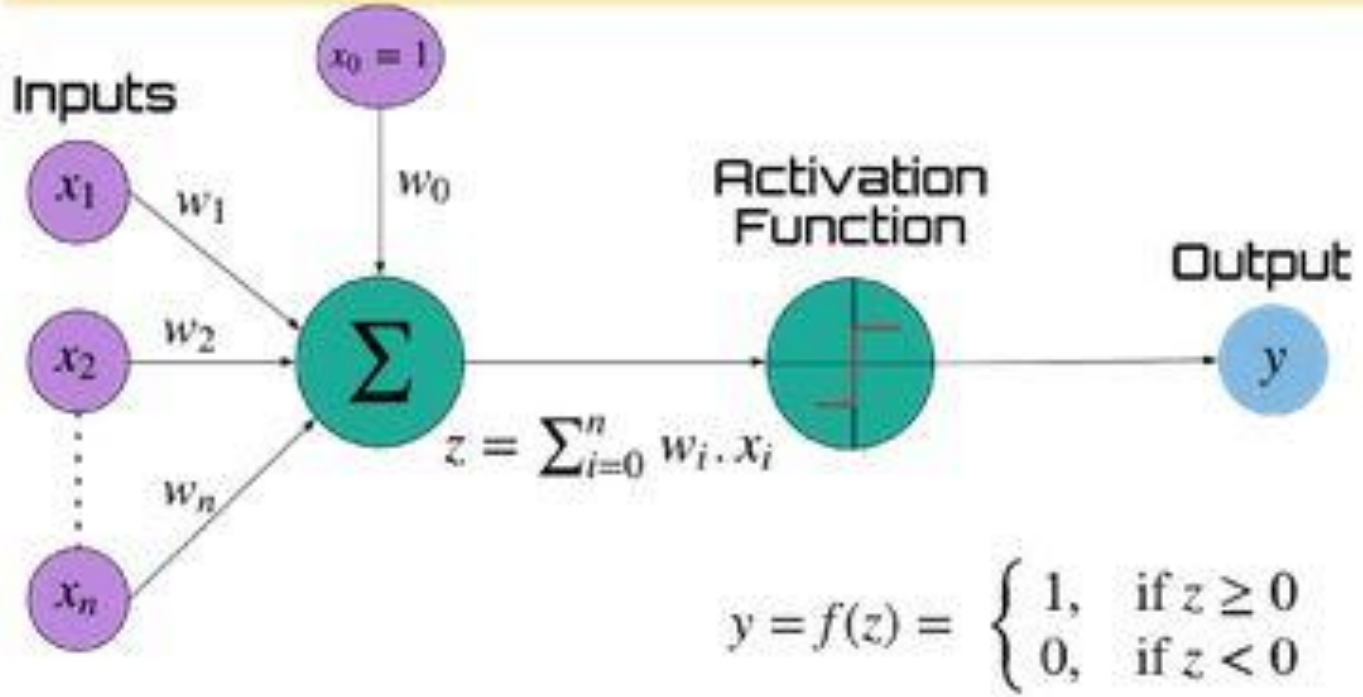
The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.

Random Forest involves sampling of the input data with replacement, bootstrap sampling.

Linear model



Perceptron: Threshold Activation



A Perceptron Model

Hypothesis

$$h_\theta(x) = \theta_{(0)} + \theta_{(1)}x$$

How do we evaluate?

$$\theta_{(i's)}$$

Let us assume $\theta_{(0)}$ = 25 and $\theta_{(1)}$ = 0

or

Let us assume $\theta_{(0)}$ = 0 and $\theta_{(1)}$ = -100

Our hope is that the hypothesis $h(x)$ accounts for all of the data with minimal error.

Mathematically:

We are tying to minimise $\theta_{(0)}$ and $\theta_{(1)}$

or **minimisation** of

$$(h_\theta(x) - y)^2$$

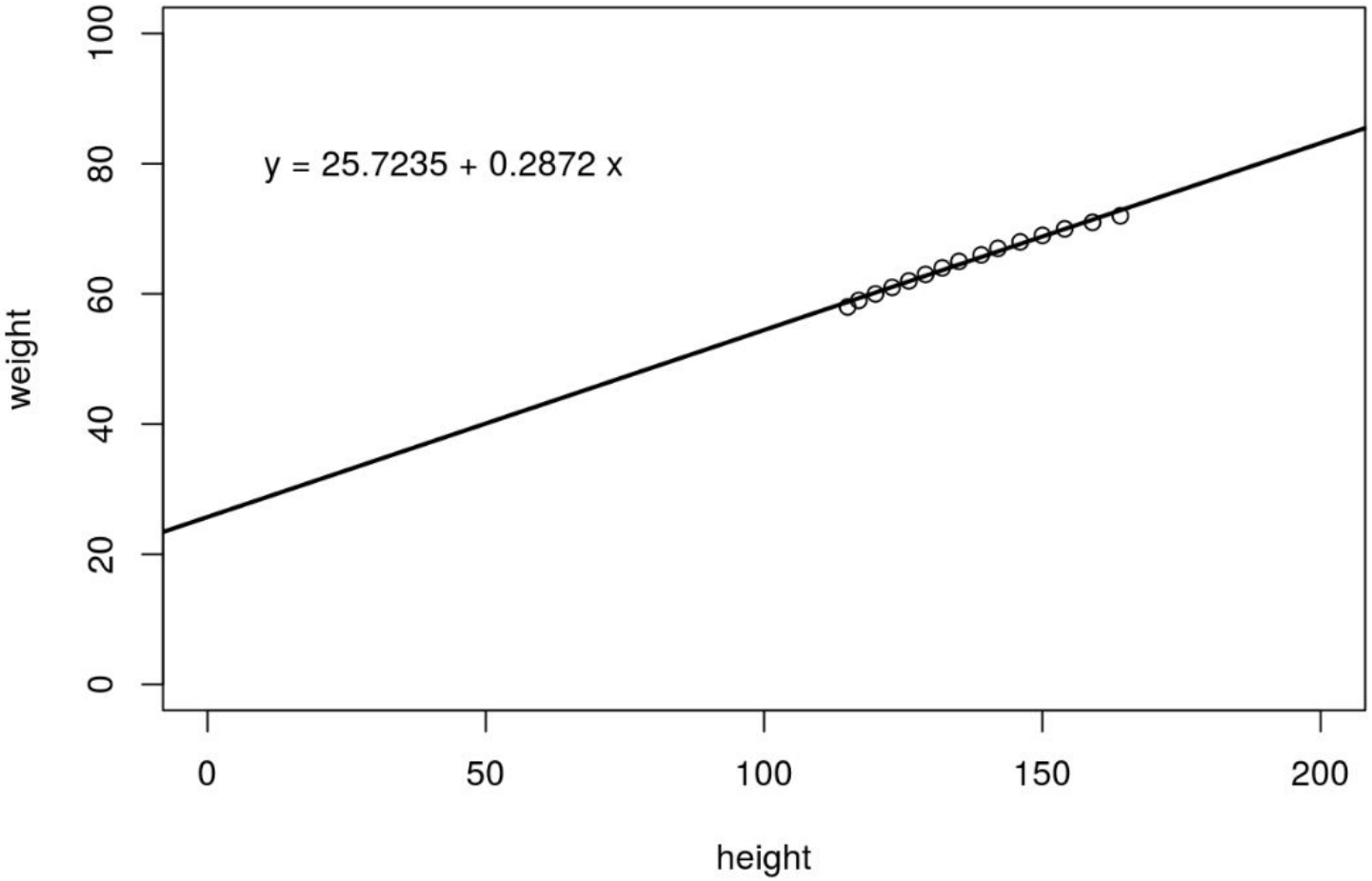
When done for all elements in the matrix it is called the **Cost Function**.

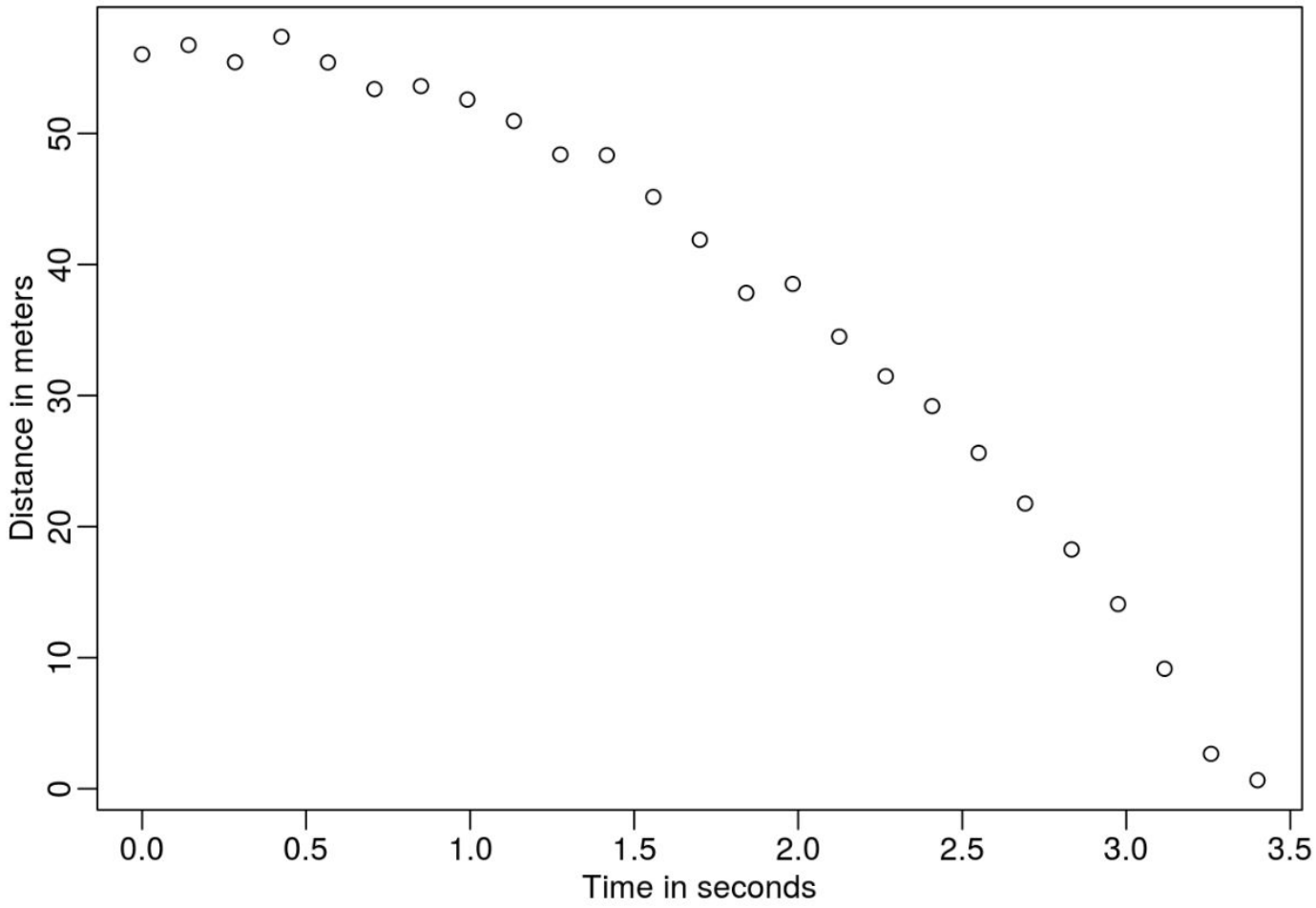
$$1/2n \sum_{i=1}^{n} (h_\theta(x^i) - y^i)^2$$

where $h_\theta(x^i) = \theta_0 + \theta_{1x^i}$

The **Cost Function**

$$CF(\theta_0, \theta_1) = 1/2n \sum_{i=1}^{n} (h_\theta(x^i) - y^i)^2$$

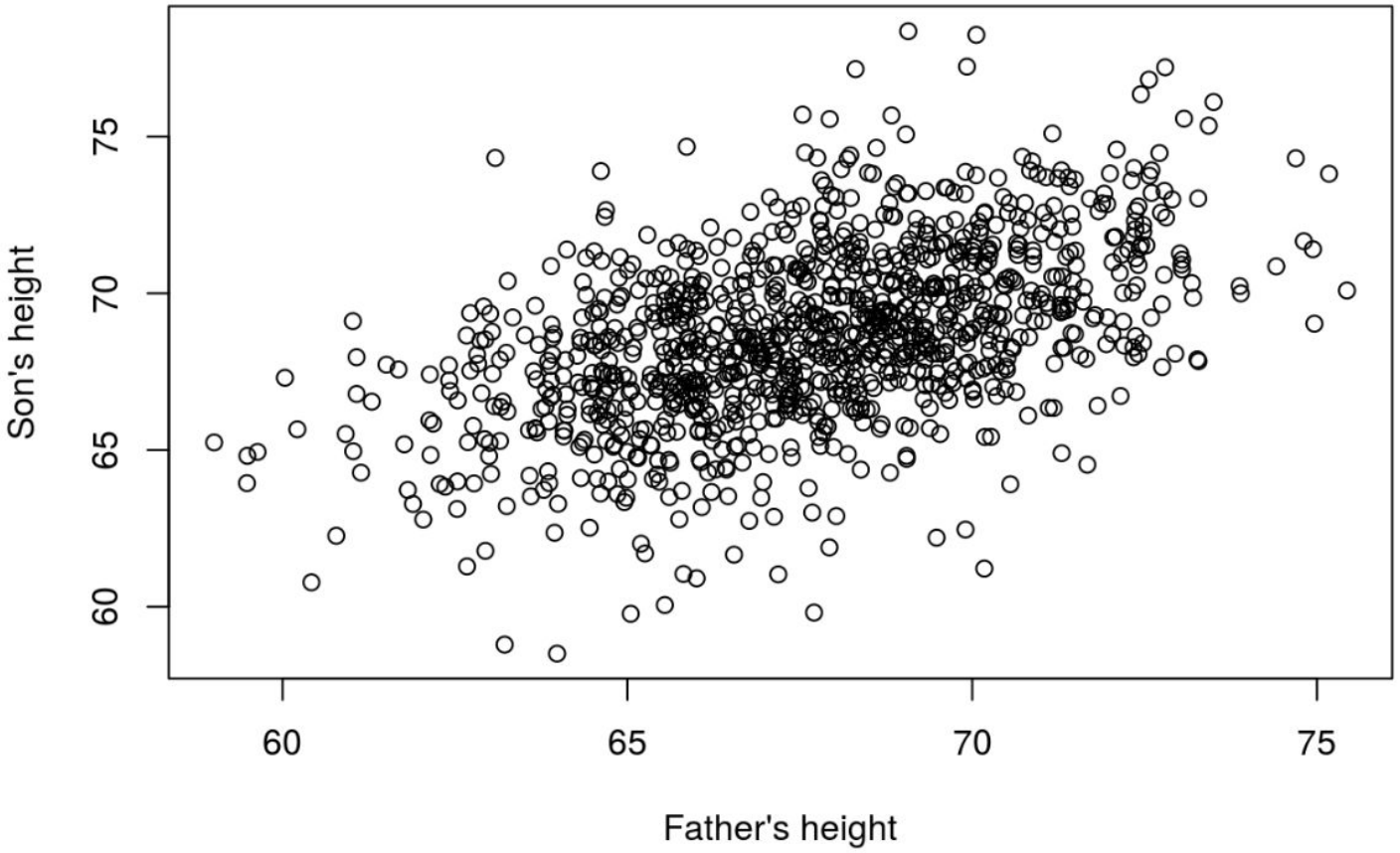

y = 25.7235 + 0.2872 x
weight
height
0
20
40
60
80
100
0
50
100
150
200



This data looks like it might follow the equation

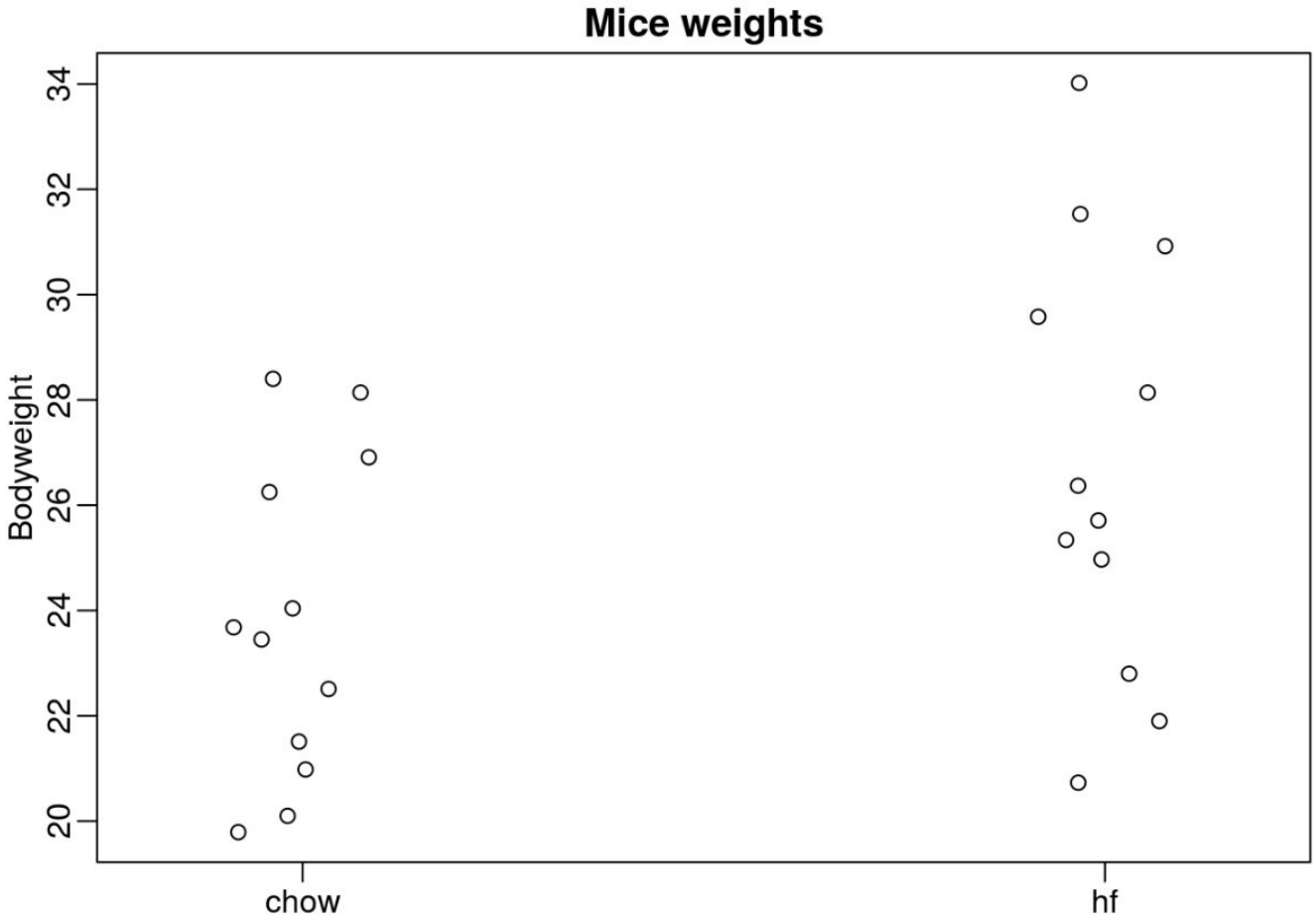
$$Y_i = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \varepsilon_i, i = 1, \ldots, n$$

With Y_i representing location, x_i representing the time, and ε_i accounting for measurement error. This is still a linear model because it is a linear combination of known quantities (the x's) referred to as predictors or covariates and unknown parameters (the θ's).



$$Y_i = \theta_0 + \theta_1 x_i + \varepsilon_i, i = 1, \ldots, N$$

This is also a linear model with x_i and Y_i, the father and son heights respectively, for the i-th pair and ε_i a term to account for the extra variability. Here we think of the fathers' heights as the predictor and being fixed (not random) so we use lower case. Measurement error alone can't explain all the variability seen in ε_i. This makes sense as there are other variables not in the model, for example, mothers' heights, genetic randomness, and environmental factors



We can estimate the difference in average weight between populations using a linear model of the form.

$$Y_i = \theta_0 + \theta_1 x_i + \varepsilon_i$$

with θ_0 the chow diet average weight, θ_i the difference between averages, $x_i = 1$ when mouse i gets the high fat (hf) diet, $x_i = 0$ when it gets the chow diet, and ε_i explains the differences between mice of the same population.

Estimating parameters θ

For the models above to be useful we have to estimate the unknown θs. In the second example, we want to describe a physical process for which we can't have unknown parameters. In the third example, we better understand inheritance by estimating how much, on average, the father's height affects the son's height. In the fourth example, we want to determine if there is in fact a difference: if $\theta_1 \neq 0$.

As explained above, we have to find the values that minimize the distance of the fitted model to the data. We come back to **Cost Function**.

$$\sum_{i=1}^{n}\left(Y_i - \left(\theta_0 + \sum_{j=1}^{p}\theta_j x_{i,j}\right)\right)^2$$

Once we find the minimum, we will call the values the least squares estimates (LSE) and denote them with $\hat{\theta}$. The quantity obtained when evaluating the least squares equation at the estimates is called the residual sum of squares (RSS). Since all these quantities depend on Y, they are random variables. The $\hat{\theta}$ s are random variables and we will eventually perform inference on them.

What actually happens when we invoke lm? Inside of lm, we will form a design matrix $\mathbf{X}$ and calculate the Cost function: $\boldsymbol{\beta}$, which minimizes the sum of squares. The formula for this solution is:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y}$$

We can calculate this in R using matrix multiplication operator %*%, the inverse function solve, and the transpose function t.

Getting back to the mice example

```r
levels(dat$Diet)
```

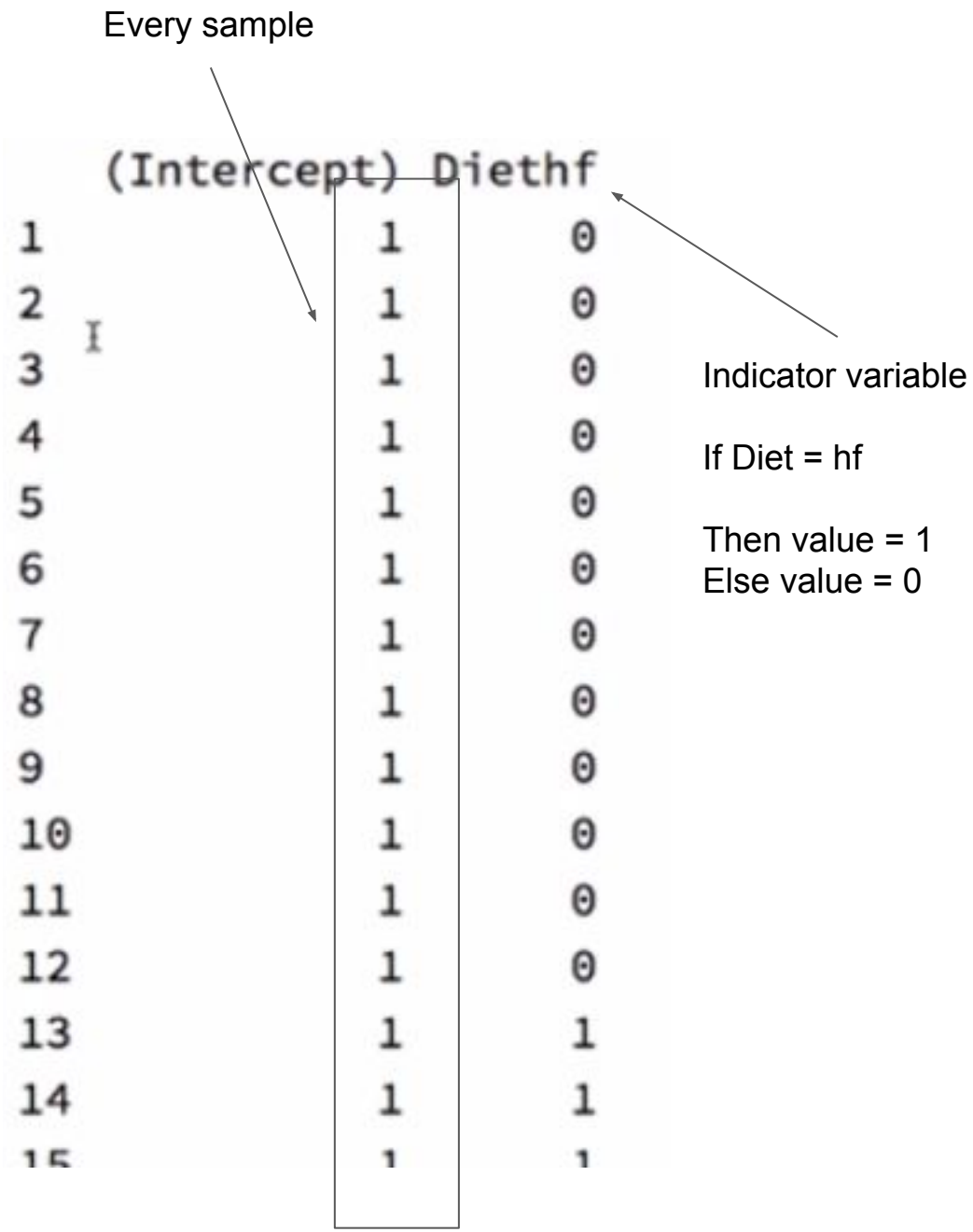
```
## [1] "chow" "hf"
```

Reference level

Non reference level

```r
X <- model.matrix(~ Diet, data=dat)
head(X)
```

Formula in R


Every sample
(Intercept) Diethf
1 1 0
2 1 0
3 1 0
4 1 0
5 1 0
6 1 0
7 1 0
8 1 0
9 1 0
10 1 0
11 1 0
12 1 0
13 1 1
14 1 1
Indicator variable
If Diet = hf
Then value = 1
Else value = 0

```r
Y <- dat$Bodyweight
X <- model.matrix(~ Diet, data=dat)
solve(t(X) %*% X) %*% t(X) %*% Y
```

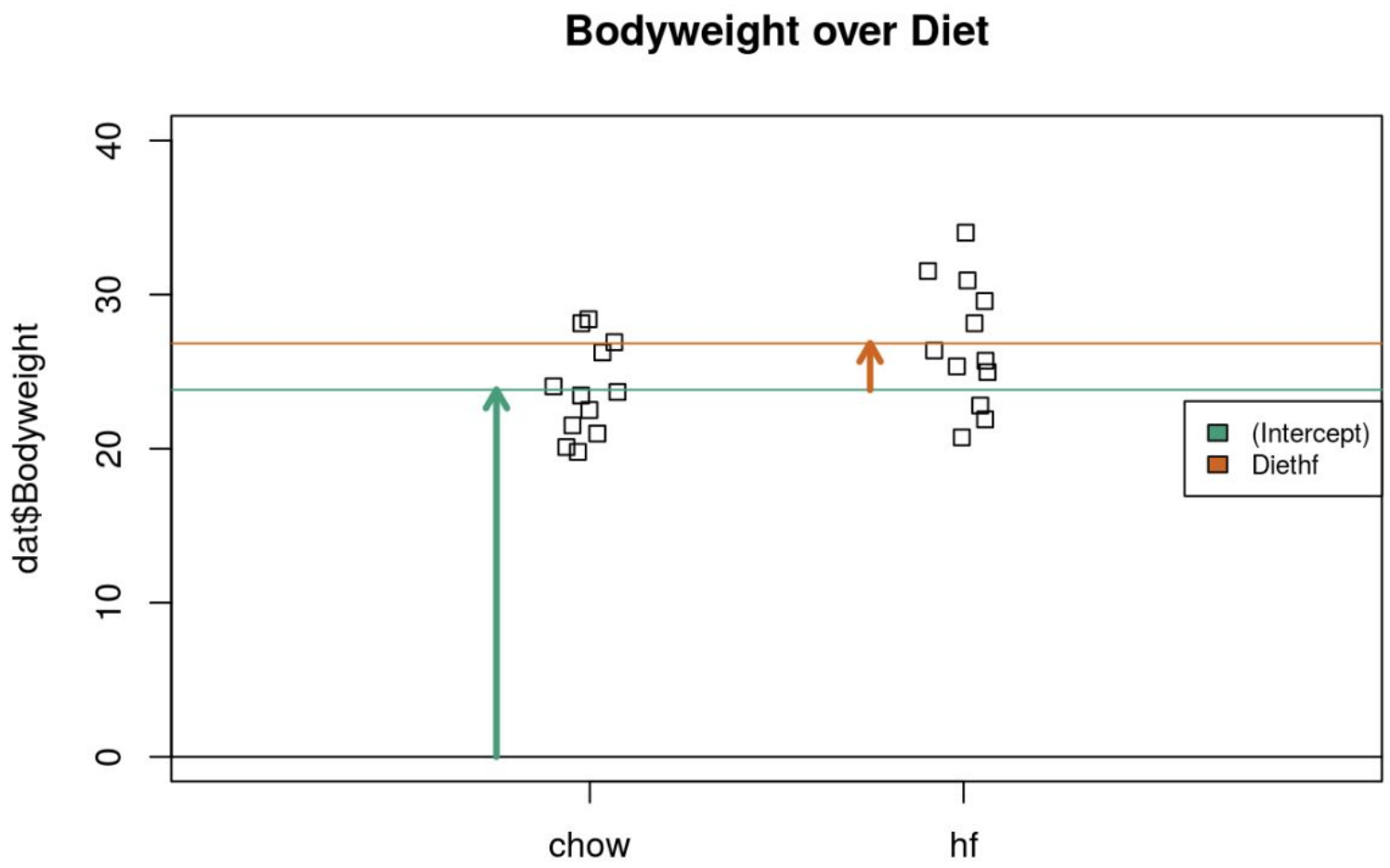
```
##                 [,1]
## (Intercept) 23.813333
## Diethf       3.020833
```

```r
fit <- lm(Bodyweight ~ Diet, data=dat)
summary(fit)
```

```
##
## Call:
## lm(formula = Bodyweight ~ Diet, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1042 -2.4358 -0.4138  2.8335  7.1858
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   23.813      1.039  22.912   <2e-16 ***
## Diethf         3.021      1.470   2.055   0.0519 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.6 on 22 degrees of freedom
## Multiple R-squared:  0.1611, Adjusted R-squared:  0.1229
## F-statistic: 4.224 on 1 and 22 DF,  p-value: 0.05192
```

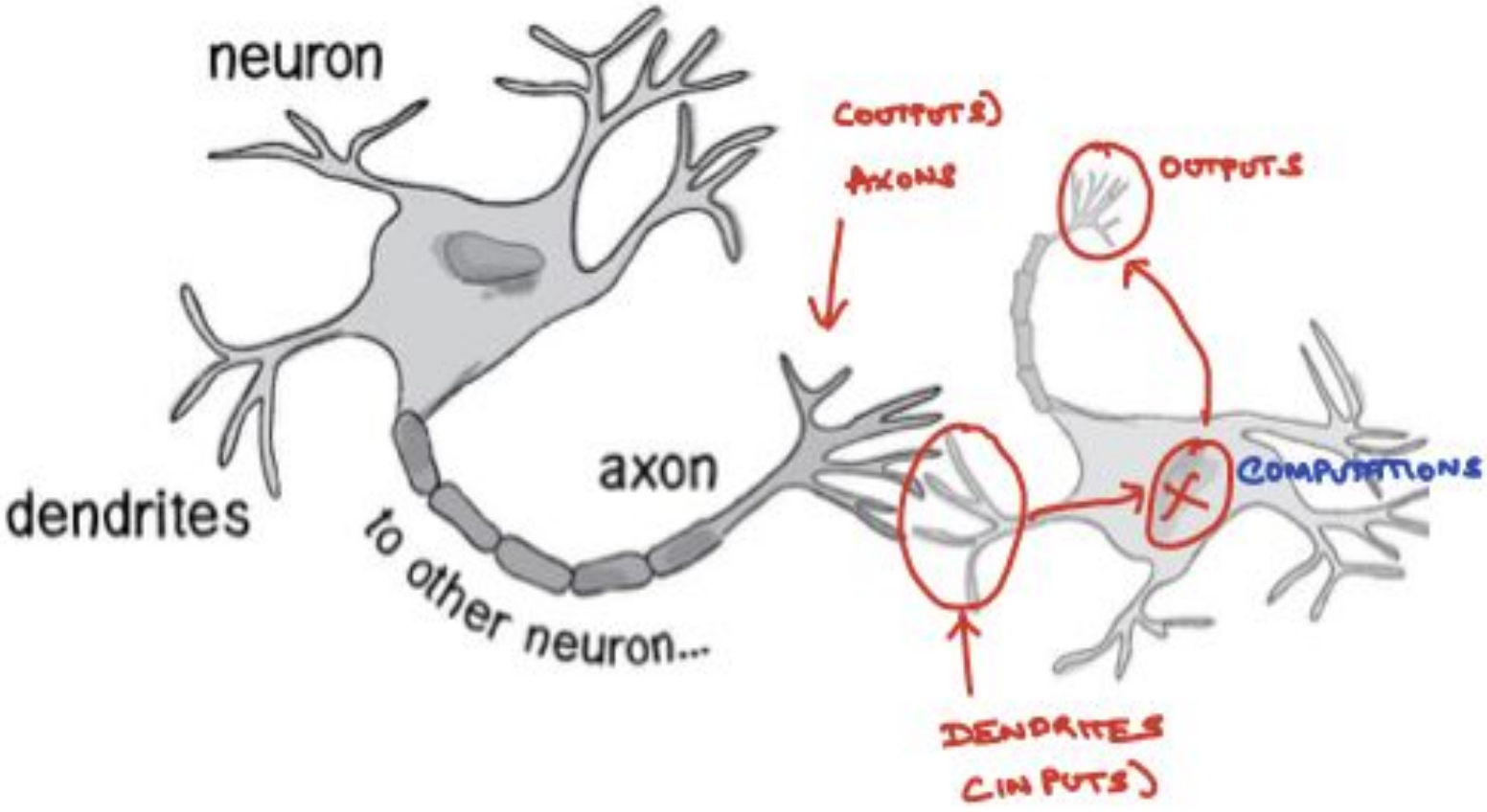
```r
(coefs <- coef(fit))
```

```
## (Intercept)      Diethf
##   23.813333    3.020833
```

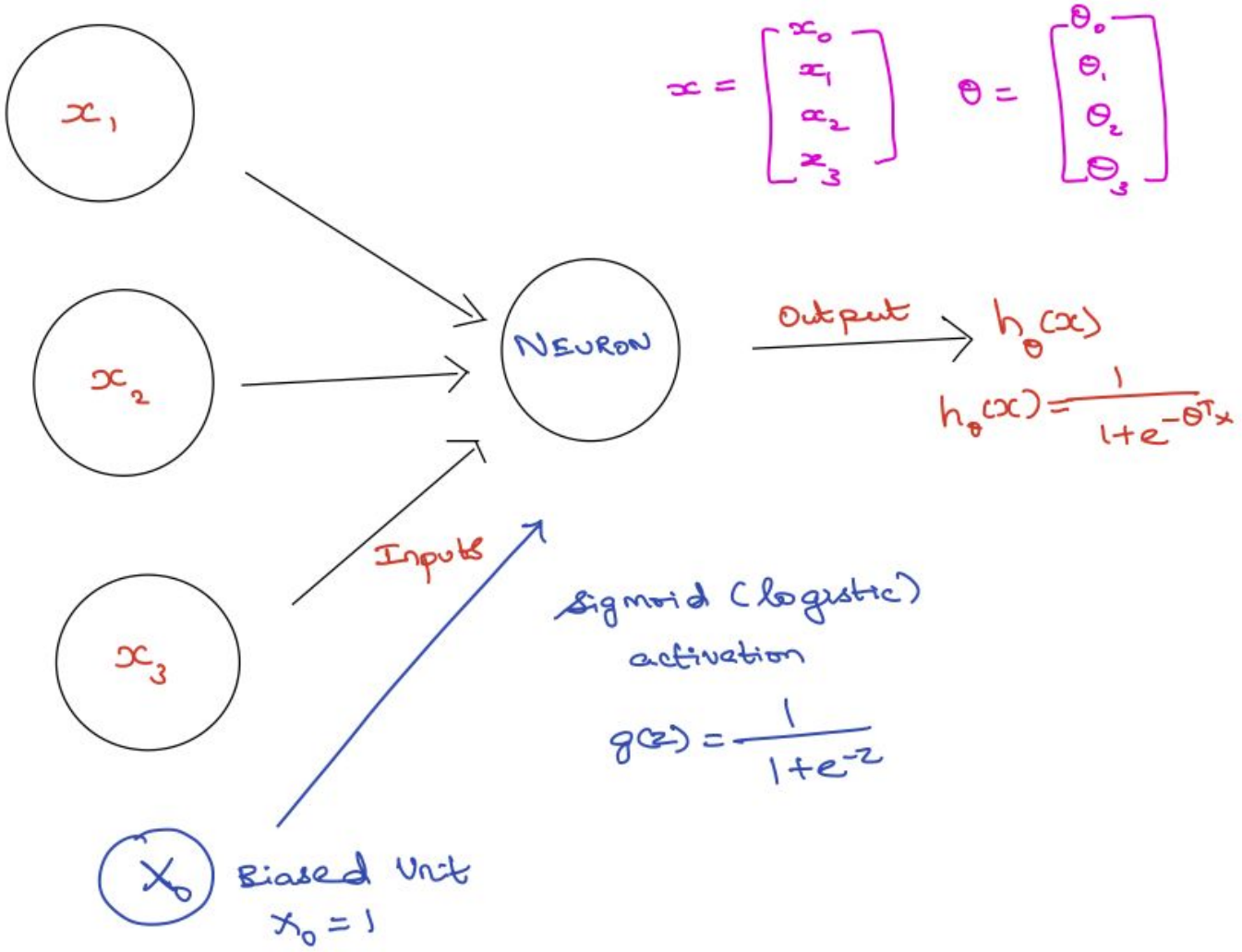


This is equivalent to doing a t-test

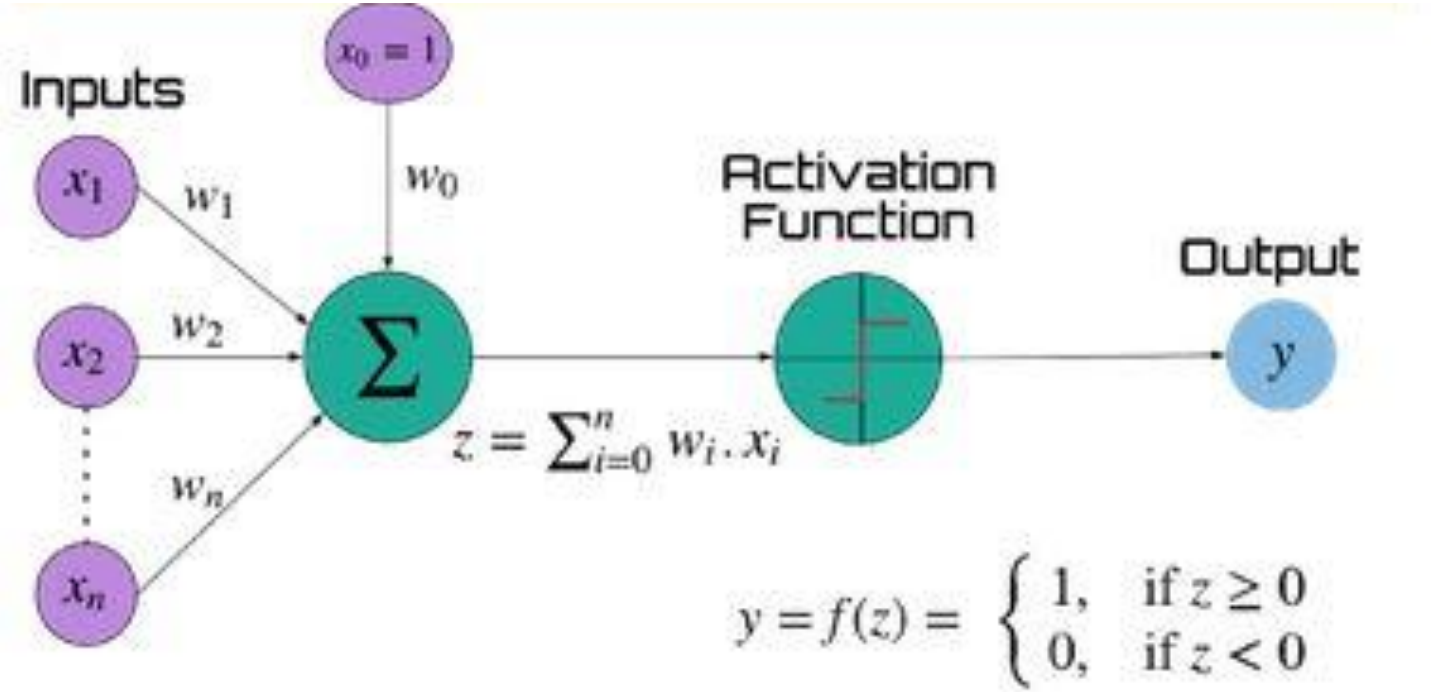
Artificial Neural Network

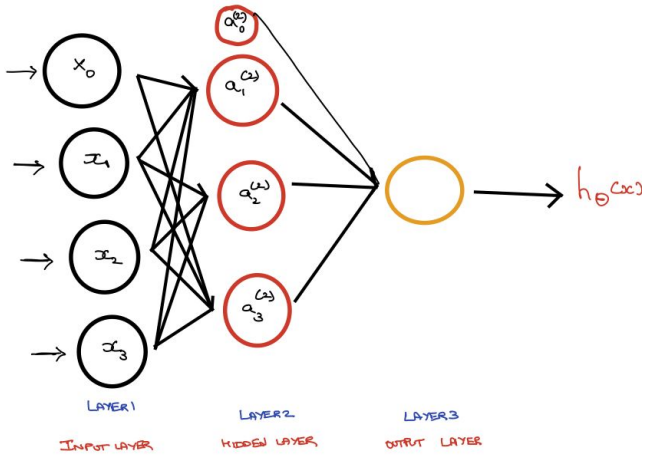


Perceptron Model



Artificial Neural Network





where

$a_i^{(j)}$ = activation of i in layer j

θ^i = matrix of weights controlling function mapping from layer j to layer j+1

$$
\begin{aligned}
a_1^{(2)} &= g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) \\
a_2^{(2)} &= g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) \\
a_3^{(2)} &= g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) \\
h_\theta(x) = a_1^{(3)} &= g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(1)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})
\end{aligned}
$$

Vectorized notations of inputs and activations.

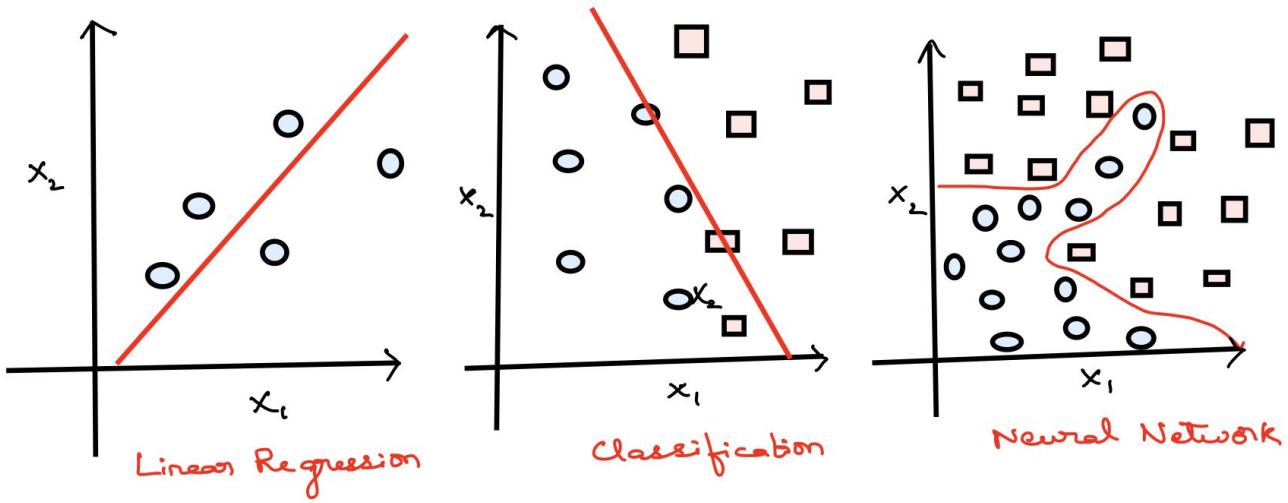
$$
x = \begin{bmatrix} x_o \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}
$$

$$
z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}
$$

Vectorized representation of activation of hidden layer and activation layer.

$$
\begin{aligned}
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)})
\end{aligned}
$$

$$
\begin{aligned}
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
h_\Theta(x) &= a^{(3)} = g(z^{(3)})
\end{aligned}
$$



Whereas for the third we could probably apply a logistic regression with a lot of nonlinear features like this

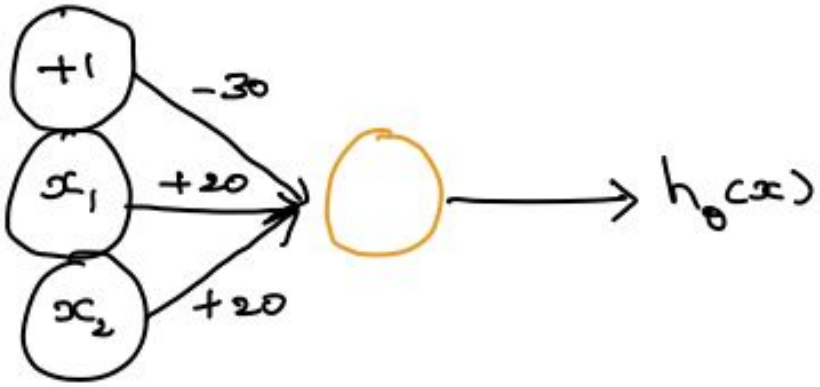
$$Y_i = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_i^2 x_2 + \theta_5 x_i^3 x_2 + \theta_6 x_i^3 x_2^2 \ldots)$$

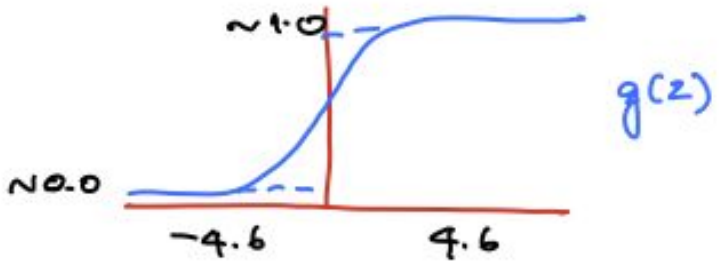
i.e. if we include enough polynomials we could arrive at an hypothesis that will separate the two classes.

This could perfectly work well if we just have two features, such as x_1 and x_2 but for almost all machine learning problems we usually have more than two features. Importantly if the number of features increase the number of quadratic terms increase as a function of $n^2/2$; where n is the number of features.

This would result in overfitting if the number of features increase.

Because ANN has felixibility to derive complex features from each layer of ptrons, it can be applied to any complex functional relationship and more importantly unlike generalized linear models (GLMs) it is not necessary to prespecify the type of relationship between covariates and response variables as for instance as linear combination. This makes ANN a valuable statistical tool.





$$h_\theta(x) = g(-30 + 20x_1 + 20x_2)$$

Truth table

x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Cost function and back propagation

Cost function of linear models

$$CF(\theta_0, \theta_1) = 1/2n \sum_{i=1}^{n} (h_\theta(x^i) - y^i)^2$$

Matrix solution to minimise the cost function in linear models

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

Cost funciton in ANN (it is a pretty scarry equation)

$$CF(\Theta) = -1/m \left[\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) log(1 - (h_\Theta(x^{(i)}))_k) \right]$$

$$+\lambda/2m \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{S_{l+1}} (\Theta_{ji}^{(l)})^2$$

We have to minimise this ANN cost function and it is done by back propogation. It is termed back propogation because of the fact that we compute the error from the outer most layer and go backwards.

A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

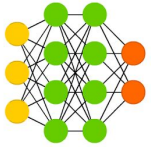
- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

Perceptron (P)

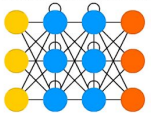
Feed Forward (FF)

Radial Basis Network (RBF)

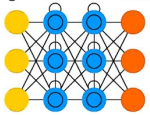
Deep Feed Forward (DFF)



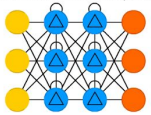
Recurrent Neural Network (RNN)



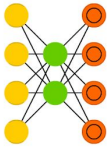
Long / Short Term Memory (LSTM)



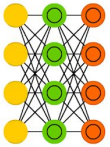
Gated Recurrent Unit (GRU)



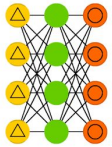
Auto Encoder (AE)



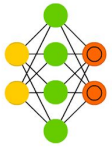
Variational AE (VAE)



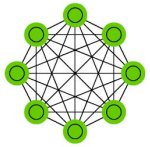
Denoising AE (DAE)



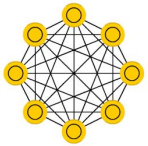
Sparse AE (SAE)



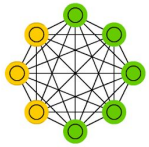
Markov Chain (MC)



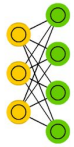
Hopfield Network (HN)



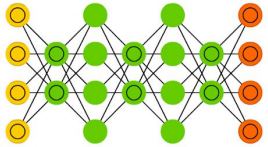
Boltzmann Machine (BM)



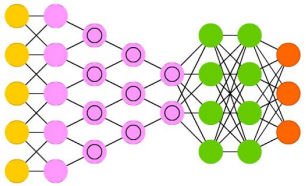
Restricted BM (RBM)



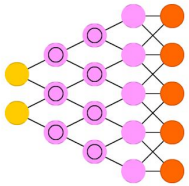
Deep Belief Network (DBN)



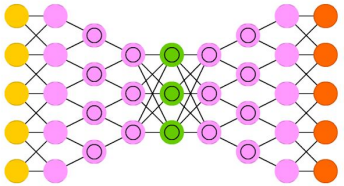
Deep Convolutional Network (DCN)



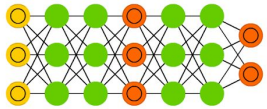
Deconvolutional Network (DN)



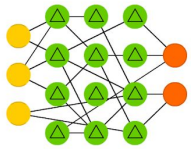
Deep Convolutional Inverse Graphics Network (DCIGN)



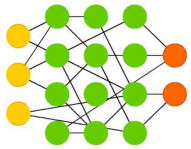
Generative Adversarial Network (GAN)



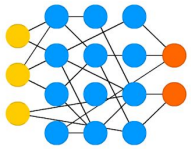
Liquid State Machine (LSM)



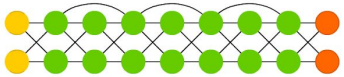
Extreme Learning Machine (ELM)



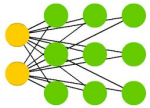
Echo State Network (ESN)



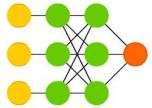
Deep Residual Network (DRN)



Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)