AN INTRODUCTION TO MACHINE LEARNING FOR PARTICLE PHYSICS

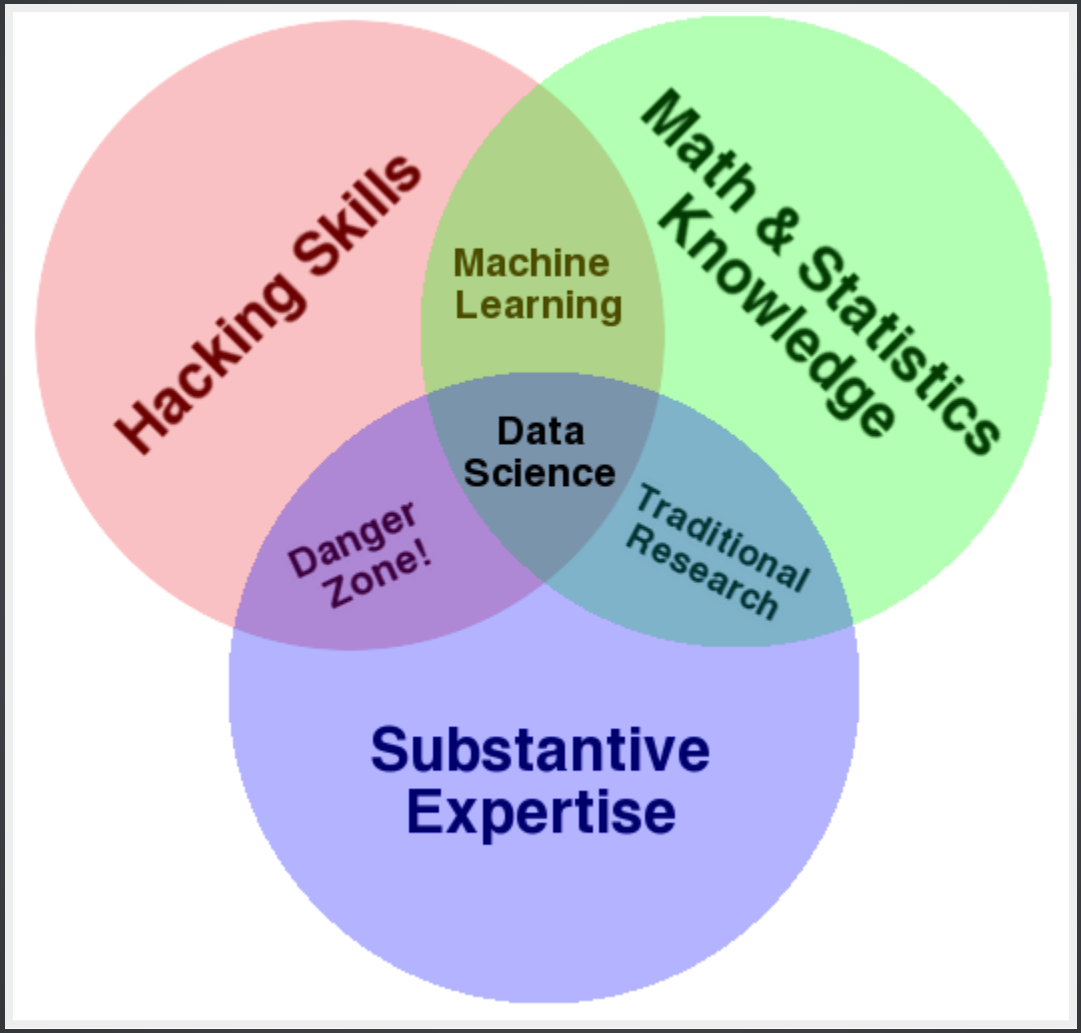
ANDREW JOHN LOWE

3 AUGUST 2016

ABOUT THE SPEAKER

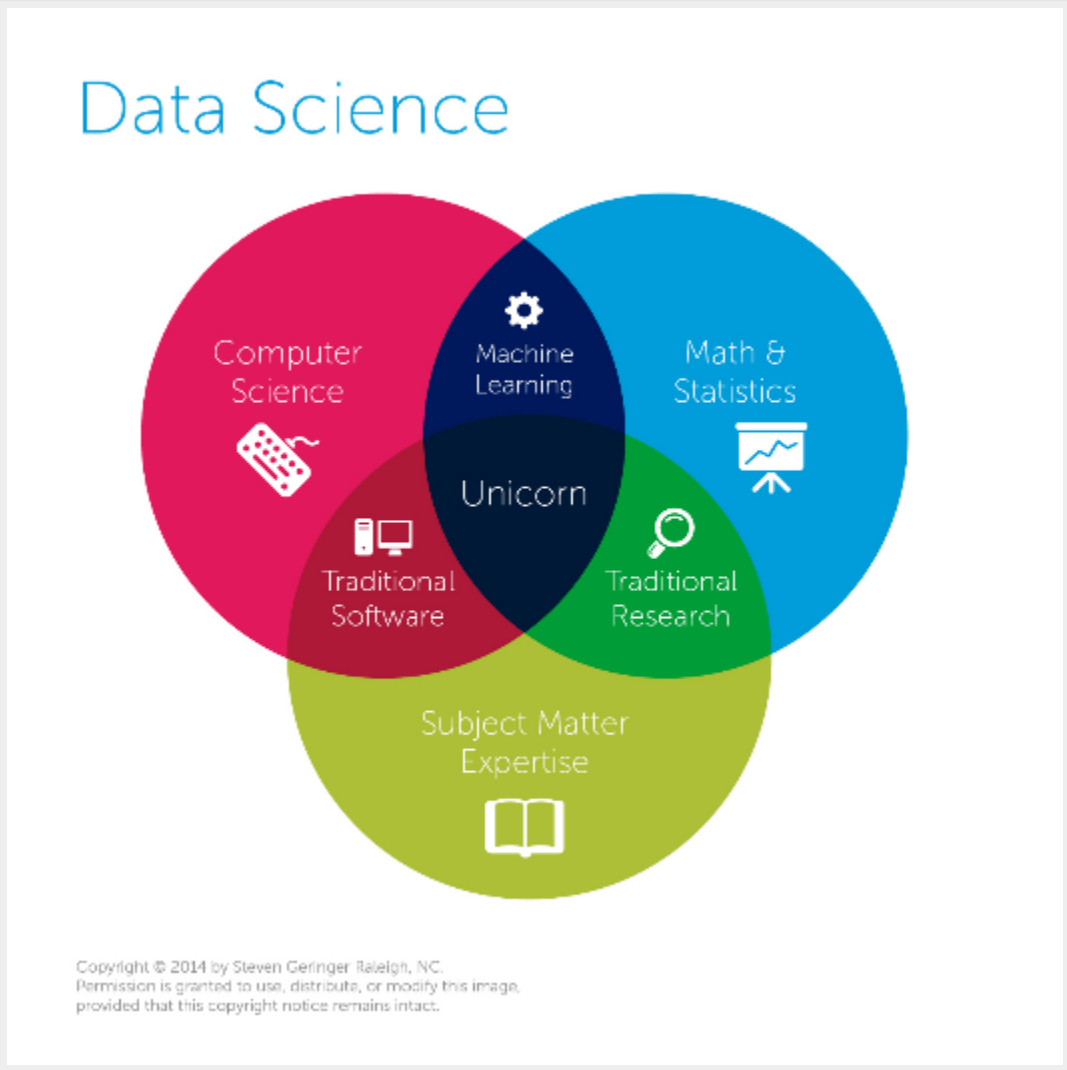
- **Research Fellow**, Wigner Research Centre for Physics, Hungary (2013–present)
- **Postdoctoral Fellow**, California State University, Fresno, USA (2010–2012)
- **Postdoctoral Fellow**, Indiana University, USA (2008–2009)
- **PhD student**, Royal Holloway, University of London, UK (2001–2008)
- **MSc student**, Royal Holloway, University of London, UK (2000–2001)
- **Assistant Research Scientist**, National Physical Laboratory, UK (1998–2000)
- **BSc student**, Royal Holloway, University of London, UK (1993–1996)

WHAT IS DATA SCIENCE?



The Data Science Venn Diagram, by Drew Conway

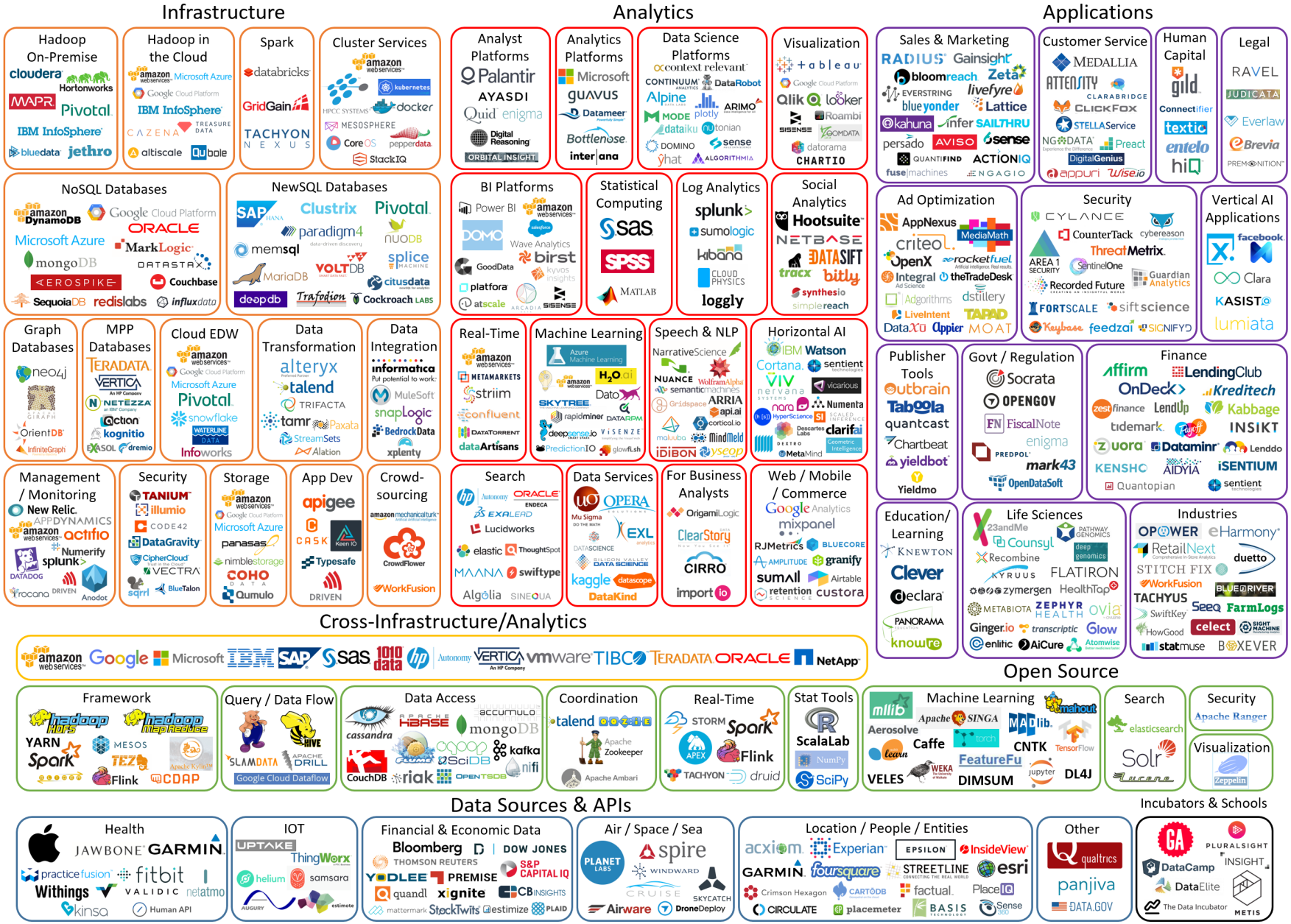
ANOTHER INTERPRETATION



People with all these skills are rare!

WHAT DOES THE “BIG DATA” LANDSCAPE LOOK LIKE OUTSIDE PARTICLE PHYSICS?

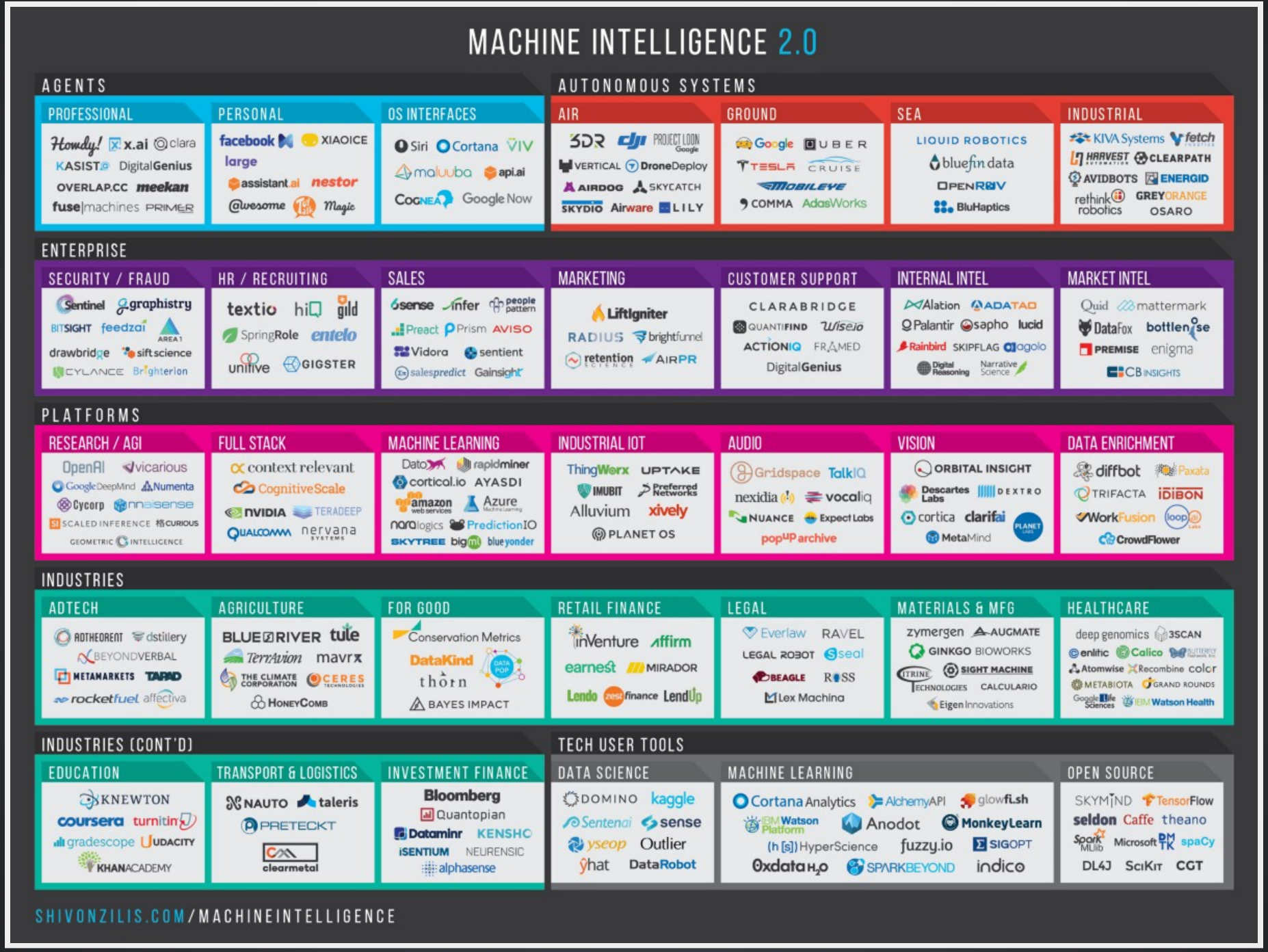
Big Data Landscape 2016 (Version 3.0)



Last Updated 3/23/2016

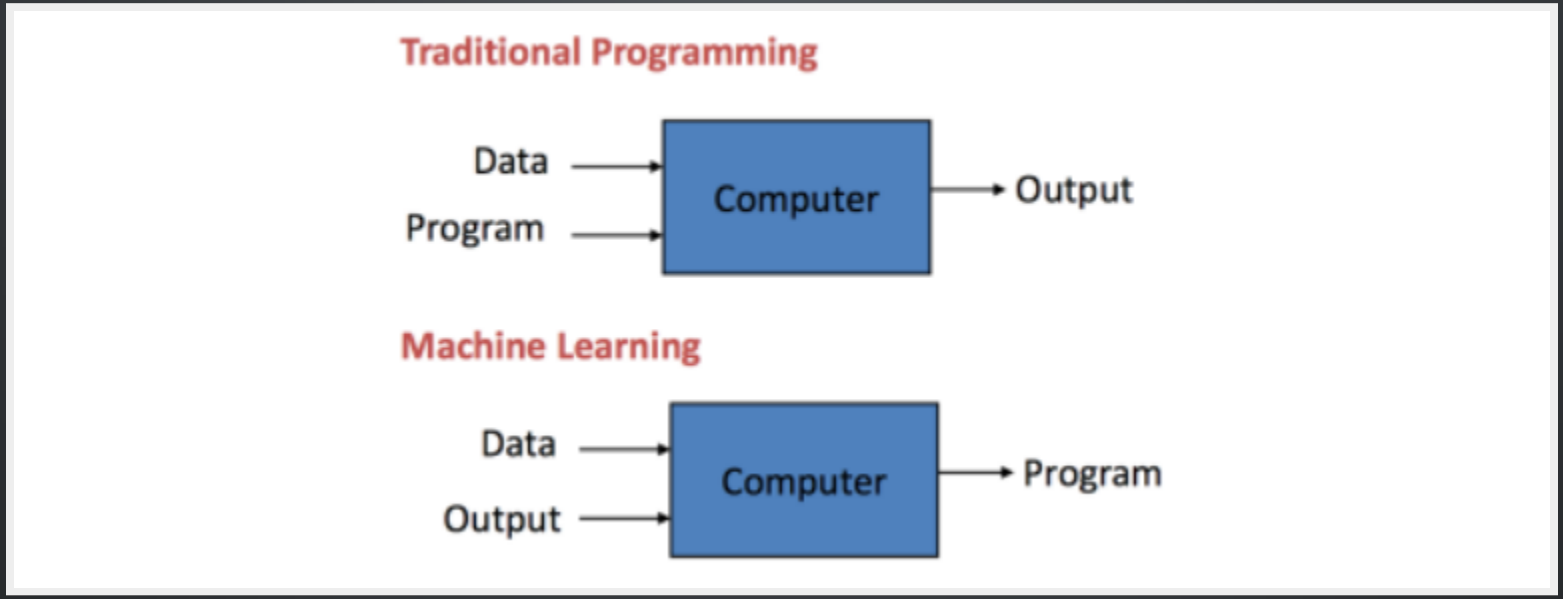
FIRSTMARK

WHERE IS MACHINE LEARNING USED?


MACHINE INTELLIGENCE 2.0
AGENTS
PROFESSIONAL
Howdy!
x.ai
clara
KASISTO
DigitalGenius
OVERLAP.CC
meekan
fuse|machines
PRIMER
PERSONAL
facebook
XIAOICE
large
assistant.ai
nestor
@wesome
Magic
OS INTERFACES
Siri
Cortana
VIV
maluuba
api.ai
CogNEA
Google Now
AUTONOMOUS SYSTEMS
AIR
3DR
dji
PROJECT LOON
Google
VERTICAL
DroneDeploy
AIRDOG
SKYCATCH
SKYDIO
Airware
LILY
GROUND
Google
UBER
TESLA
CRUISE
MOBILEYE
COMMA
AdasWorks
SEA
LIQUID ROBOTICS
bluefin data
OPENROV
BluHaptics
INDUSTRIAL
KIVA Systems
fetch
HARVEST
CLEARPATH
AVIDBOTS
ENERGID
rethink robotics
GREYORANGE
OSARO
ENTERPRISE
SECURITY / FRAUD
Sentinel
graphistry
BITSIGHT
feedzai
AREA 1
drawbridge
sift science
CYLANCE
Brighterion
HR / RECRUITING
textio
hiQ
gild
SpringRole
entelo
unifive
GIGSTER
SALES
6sense
infer
people pattern
Preact
Prism
AVISO
Vidora
sentient
salespredict
Gainsight
MARKETING
LiftIgniter
RADIUS
brightfunnel
retention SCIENCE
AIRPR
CUSTOMER SUPPORT
CLARABRIDGE
QUANTIFIND
Wisedo
ACTIONIQ
FRAMED
DigitalGenius
INTERNAL INTEL
Alation
ADATAO
Palantir
sapho
lucid
Rainbird
SKIPFLAG
algolo
Digital Reasoning
Narrative Science
MARKET INTEL
Quid
mattermark
DataFox
bottlenose
PREMISE
enigma
CB INSIGHTS
PLATFORMS
RESEARCH / AGI
OpenAI
vicarious
Google DeepMind
Numenta
Cycorp
nnaisense
SCALED INFERENCE
CURIOUS
GEOMETRIC INTELLIGENCE
FULL STACK
context relevant
CognitiveScale
NVIDIA
TERADEEP
QUALCOMM
nervana SYSTEMS
MACHINE LEARNING
Dato
rapidminer
cortical.io
AYASDI
amazon web services
Azure Machine Learning
narrologics
PredictionIO
SKYTREE
bigml
blueyonder
INDUSTRIAL IOT
ThingWorx
UPTAKE
IMUBIT
Preferred Networks
Alluvium
xively
PLANET OS
AUDIO
Gridspace
TalkIQ
nexidia
vocaliq
NUANCE
Expect Labs
popup archive
VISION
ORBITAL INSIGHT
Descartes Labs
DEXTRO
cortica
clarifai
PLANET LABS
MetaMind
DATA ENRICHMENT
diffbot
Paxata
TRIFACTA
iDIBON
WorkFusion
loop AI Labs
CrowdFlower
INDUSTRIES
ADTECH
ADTHEORENT
dstillery
BEYONDVERBAL
METAMARKETS
TAPAD
rocketfuel
affectiva
AGRICULTURE
BLUE RIVER
tule
TerrAvion
mavrx
THE CLIMATE CORPORATION
CERES TECHNOLOGIES
HONEYCOMB
FOR GOOD
Conservation Metrics
DataKind
DATA POP
thorn
BAYES IMPACT
RETAIL FINANCE
inVenture
affirm
earnest
MIRADOR
Lendo
zestfinance
LendUp
LEGAL
Everlaw
RAVEL
LEGAL ROBOT
seal
BEAGLE
ROSS
Lex Machina
MATERIALS & MFG
zymergen
AUGMATE
GINKGO BIOWORKS
ITRINE TECHNOLOGIES
SIGHT MACHINE
CALCULARIO
Eigen Innovations
HEALTHCARE
deep genomics
3SCAN
enlitic
Calico
Atomwise
Recombine
colcr
METABIOTA
GRAND ROUNDS
Google Life Sciences
IBM Watson Health
INDUSTRIES (CONT'D)
EDUCATION
KNEWTON
coursera
turnitin
gradescope
UDACITY
KHANACADEMY
TRANSPORT & LOGISTICS
NAUTO
taleris
PRETECKT
clearmetal
INVESTMENT FINANCE
Bloomberg
Quantopian
Dataminr
KENSHO
iSENTIUM
NEURENSIC
alphasense
TECH USER TOOLS
DATA SCIENCE
DOMINO
kaggle
Sentenai
sense
yseop
Outlier
yhat
DataRobot
MACHINE LEARNING
Cortana Analytics
AlchemyAPI
glowfish
IBM Watson Platform
Anodot
MonkeyLearn
(h[s]) HyperScience
fuzzy.io
SIGOPT
0xdata H2O
SPARKBEYOND
indico
OPEN SOURCE
SKYMIND
TensorFlow
seldon
Caffe
theano
Spark MLlib
Microsoft DMTK
spaCy
DL4J
SciKit
CGT
SHIVONZILIS.COM/MACHINEINTELLIGENCE

WHAT IS MACHINE LEARNING?

- Arthur Samuel (1959): Field of study that gives computers the ability to learn without being explicitly programmed
- Tom Mitchell (1998): A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E
- Traditional programming versus machine learning:



MACHINE LEARNING & PARTICLE PHYSICS

- Machine learning is more or less what is commonly known in particle physics as multivariate analysis (MVA)
- Used for many years but faced widespread scepticism
- Use of multivariate pattern recognition algorithms was basically taboo in new particle searches until recently
- Much prejudice against using what were considered “black box” selection algorithms
- Artificial neural networks and Fisher discriminants (linear discriminant analysis) were used somewhat in the 1990’s
- Boosted Decision Trees (AdaBoost, 1996) is the favourite algorithm used for many analyses (1st use: 2004)

Successes, Challenges and Future Outlook of Multivariate Analysis In HEP, Helge Voss, 2015 J. Phys.: Conf. Ser. 608 (2015) 012058; Higgs Machine Learning Challenge visits CERN, 19 May 2015, CERN; Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification, Hai-Jun Yang *et al.*, Nucl.Instrum.Meth. A543 (2005) 577-584

STATISTICAL LEARNING VERSUS MACHINE LEARNING

- Machine learning arose as a subfield of Artificial Intelligence.
- Statistical learning arose as a subfield of Statistics
- There is much overlap
- Machine learning has a greater emphasis on large scale applications and prediction accuracy
- Statistical learning emphasizes models and their interpretability, and precision and uncertainty
- But the distinction has become more and more blurred, and there is a great deal of “cross-fertilisation”
- In the following, we’ll use the term “machine learning”, regardless of the origin of specific methods

TYPES OF MACHINE LEARNING

- **Supervised learning**: learn from examples and make predictions for new data
- **Unsupervised learning**: looking for patterns in the data
- Other types exist (*e.g.*, reinforcement learning, semi-supervised learning, recommender systems)

SUPERVISED LEARNING

- We have an outcome measurement Y (also called dependent variable, response, target)
- We have a vector of p predictor measurements X (also called regressors, covariates, features, attributes, independent variables)
- In the *regression* problem, Y is quantitative (*e.g.*, price, blood pressure, voltage)
- In the *classification* problem, Y is categorical (*e.g.*, dead/alive, signal/background, digit 0–9, particle type, malignant/benign)
- We have training data $(x_1, y_1), \ldots, (x_N, y_N)$: N observations (examples, instances, cases, events) of these measurements
- Learns a mapping from the inputs to the outputs

SUPERVISED LEARNING OBJECTIVES

On the basis of the training data we would like to:

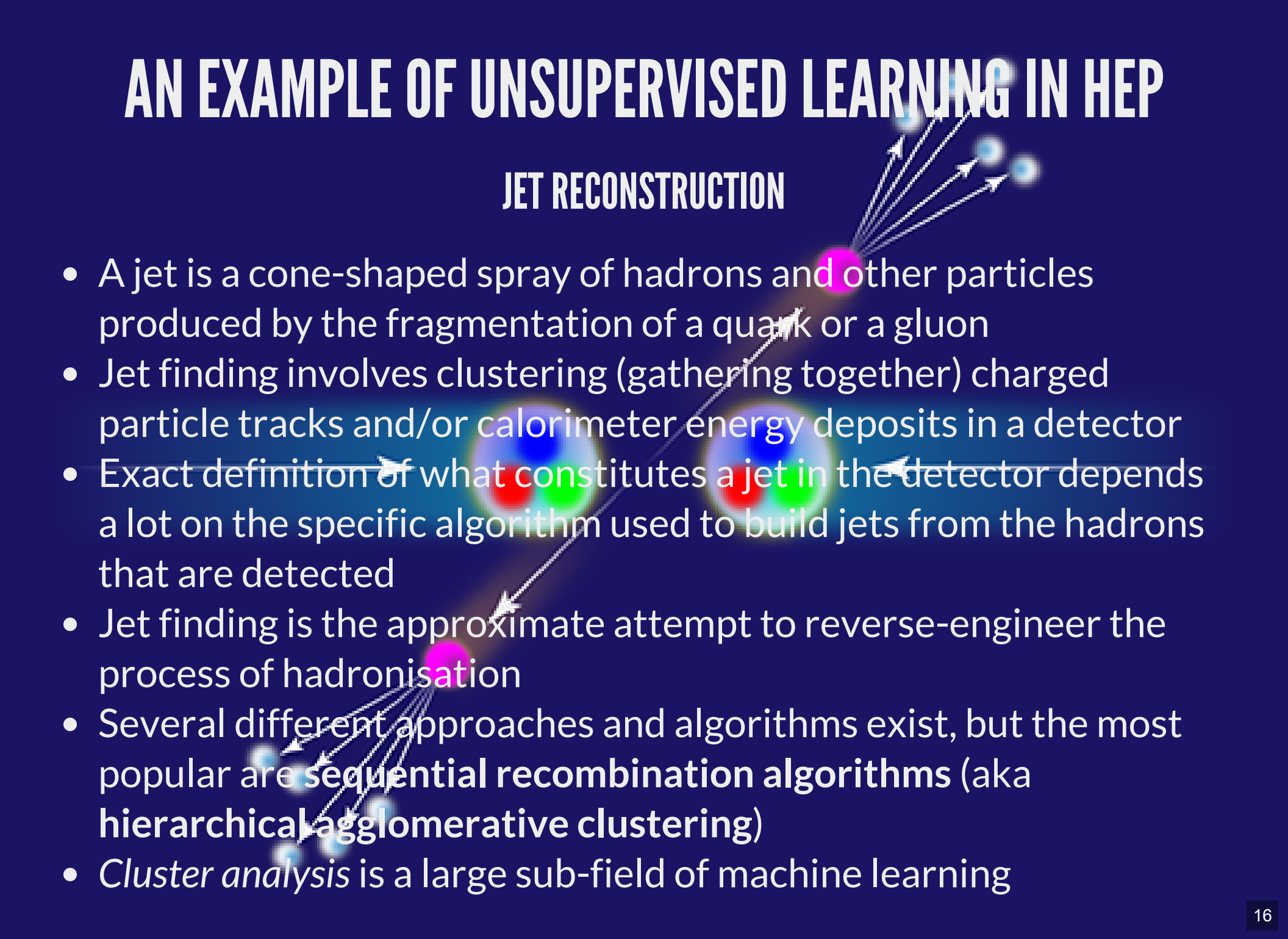
- Accurately predict unseen test cases
- Understand which inputs affect the outcome
- Assess the quality of our predictions and inferences

UNSUPERVISED LEARNING

- No outcome variable, just a set of predictors (features) measured on a set of samples
- Objective is more fuzzy: find groups of samples that behave similarly, find features that behave similarly, find linear combinations of features with the most variation
- Difficult to know how well your are doing
- Different from supervised learning, but can be useful as a pre-processing step for supervised learning

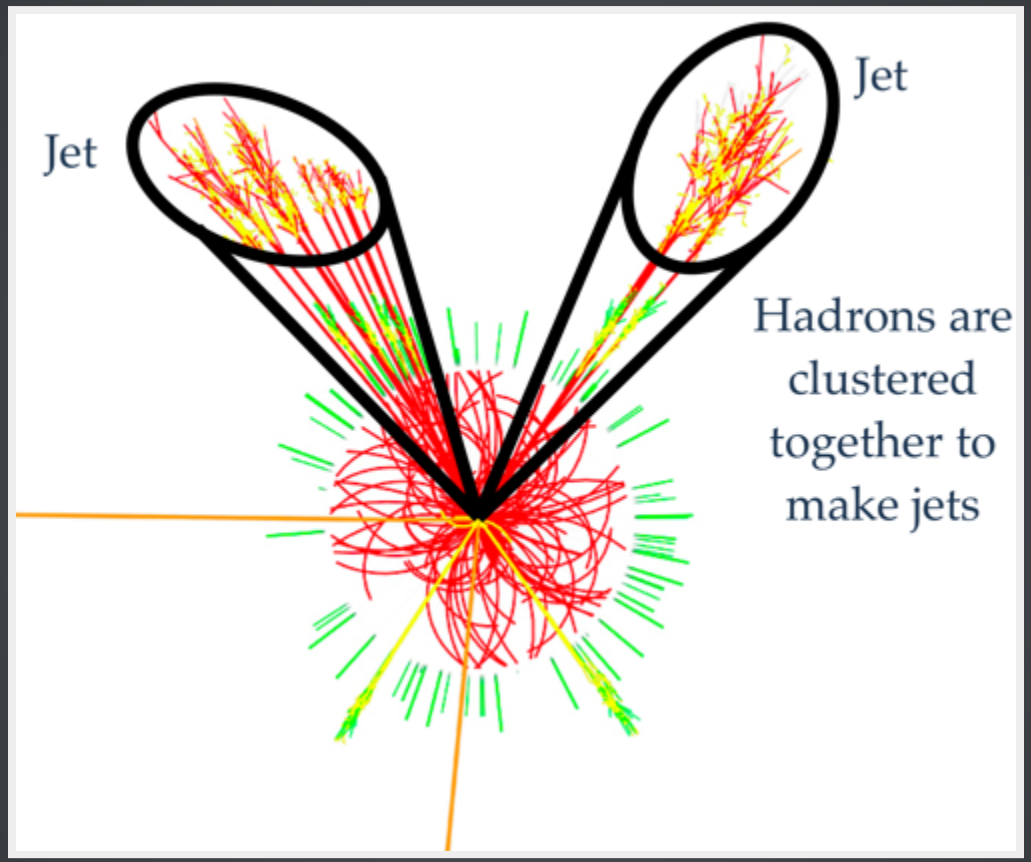
AN EXAMPLE OF UNSUPERVISED LEARNING IN HEP

JET RECONSTRUCTION



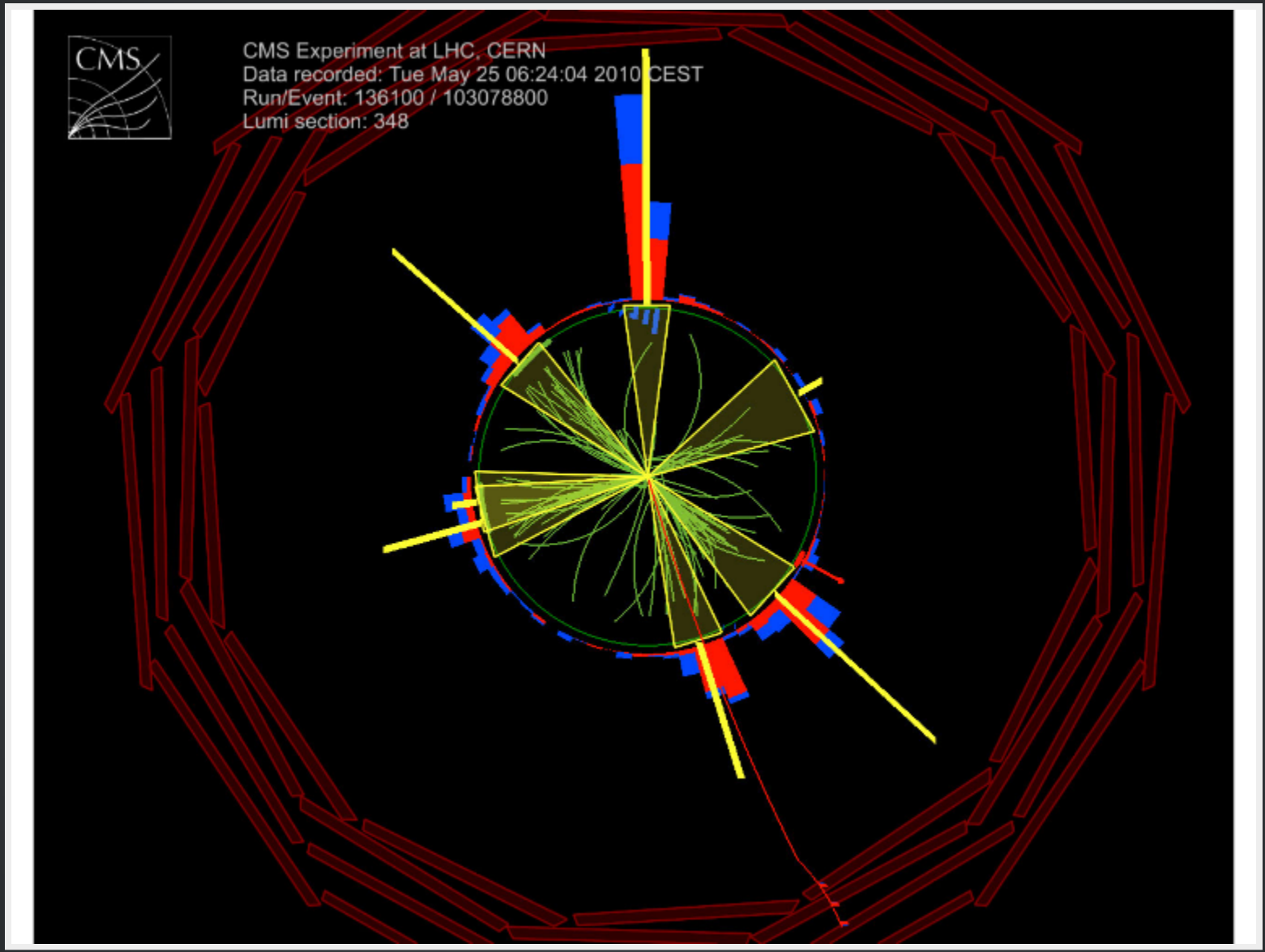
- A jet is a cone-shaped spray of hadrons and other particles produced by the fragmentation of a quark or a gluon
- Jet finding involves clustering (gathering together) charged particle tracks and/or calorimeter energy deposits in a detector
- Exact definition of what constitutes a jet in the detector depends a lot on the specific algorithm used to build jets from the hadrons that are detected
- Jet finding is the approximate attempt to reverse-engineer the process of hadronisation
- Several different approaches and algorithms exist, but the most popular are **sequential recombination algorithms** (aka **hierarchical agglomerative clustering**)
- *Cluster analysis* is a large sub-field of machine learning

JET RECONSTRUCTION



Jets are viewed as a proxy to the initial quarks and gluons that we can’t measure and are a common feature in high-energy particle collisions

A SIX-JET EVENT SEEN BY CMS



View along detector beam axis

LINEAR REGRESSION WITH ONE VARIABLE

- In regression problems, we are taking input variables and trying to fit the output onto a *continuous* expected result function
- We have a single output y (we are doing supervised learning) and a single feature x
- Our hypothesis function has the general form: $\hat{y} = \theta_0 + \theta_1 x$
- Choose θ_0, θ_1 so that $\hat{y}$ is close to y for training examples (x, y)
- For notational compactness, we can arrange the parameters in a vector:

$$\Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

COST (OR LOSS) FUNCTION

- We can measure the accuracy of our hypothesis function by using a **cost function** (also known as a **loss function**)
- Estimate parameters as the values that minimise the following cost function (sum of squared residuals):

$$J(\Theta) = \sum_{i=1}^{N} \left(y_i - \hat{y}_i \right)^2$$

- Squaring the residuals $y_i - \hat{y}_i$ ensures their contribution to J is always positive
- J is a measure of how bad our fit is to the training data

GRADIENT DESCENT

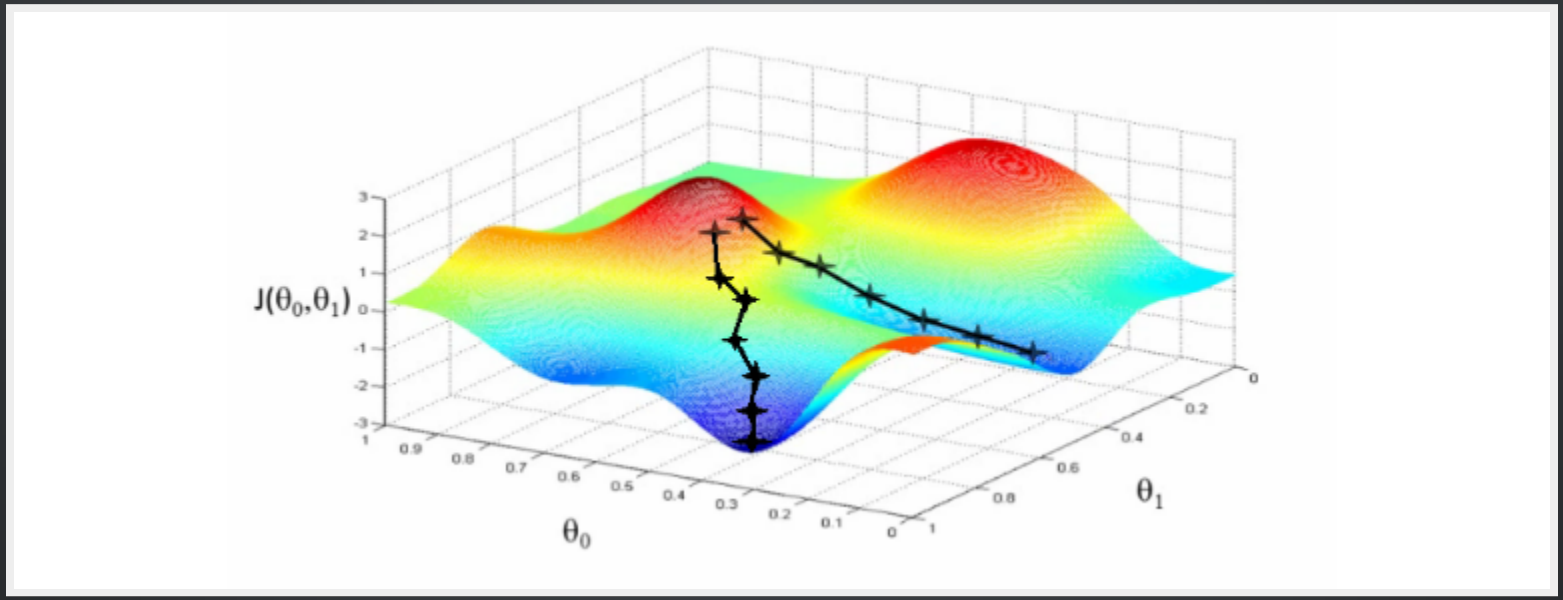
- We want to find the minimum of the cost function J
- The gradient descent algorithm is:
- Start at some $\Theta_0 = (\theta_0, \theta_1)$
- Repeat until convergence (measured by some stopping criterion):

$$\Theta_{j+1} = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_j)$$

- α is the *learning rate*
- Intuitively, what we are doing is walking down the cost function J in the direction of the tangent to J, with step size controlled by the learning rate α, to find the minimum of J

PROBLEMS WITH GRADIENT DESCENT

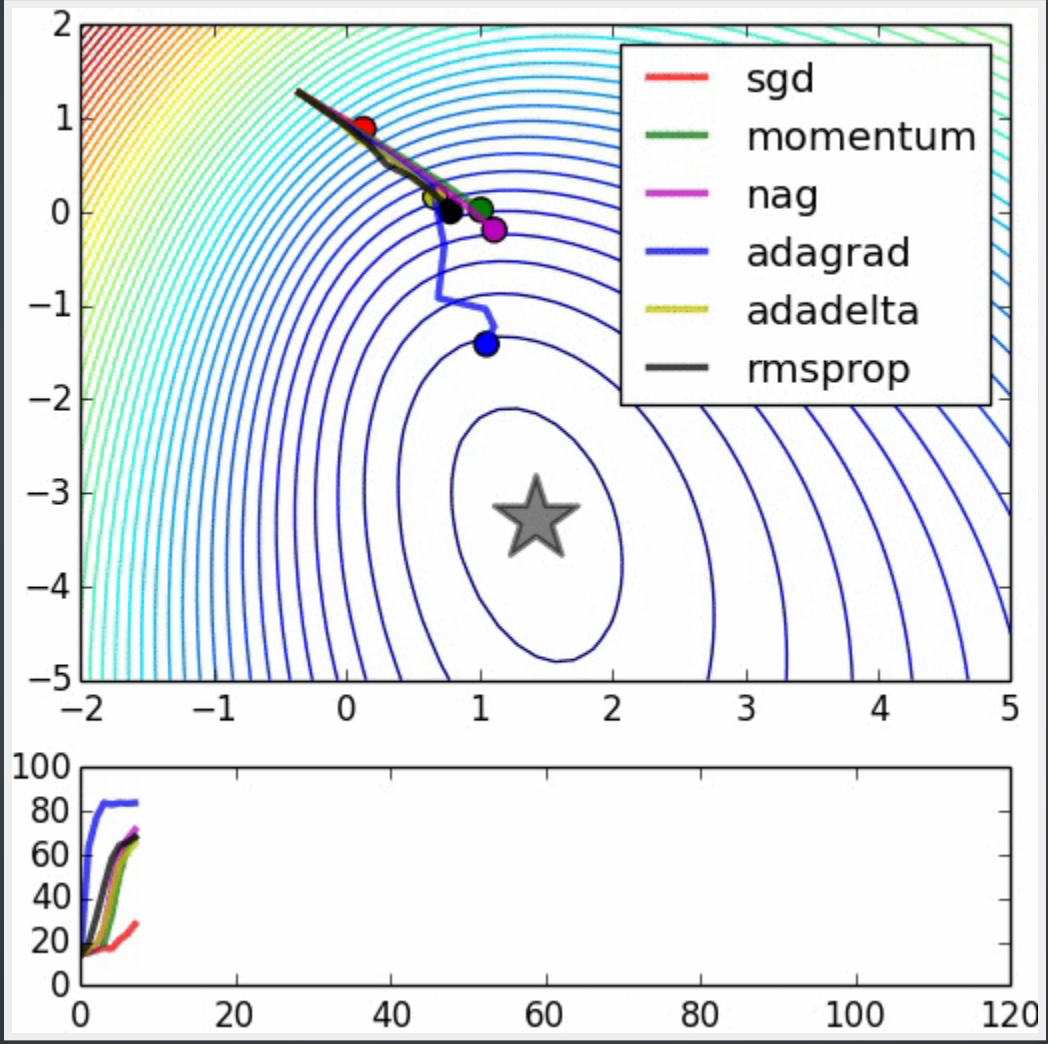
- If α is too small, gradient descent can be slow
- If α is too big, gradient descent might overshoot the minimum, fail to converge, or even diverge
- If J has more than one minimum, gradient descent might get stuck in a local minimum instead of finding the global minimum: the solution can be sensitive to starting point Θ_0



IMPROVEMENTS TO GRADIENT DESCENT

- More sophisticated algorithms than gradient descent are used in practice, but the general principle is the same
- If you are performing linear regression and your training sample has many observations, gradient descent would require a large summation before gradient descent can make a single step $\rightarrow$ can be very slow!
  - We can *parallelise* the computation by breaking the summation into chunks and spreading them across several CPUs
- For each step of gradient descent, we can perform the summation on a randomly chosen single training example; this is **stochastic gradient descent** (SGD)
  - SGD's path to a minimum jitters and is less direct,
  - SGD unlikely to converge at a minimum and will instead wander around it randomly, but usually result is close enough
- Other, more advanced, algorithms are available that are often faster than gradient descent and have the advantage that there is no need to manually pick α

COMPARISON OF GRADIENT-BASED MINIMISATION METHODS



MULTIVARIATE LINEAR REGRESSION

- Linear regression with multiple variables is a trivial extension
- We have a *feature vector* — our set of features:

$$X = [x_0, x_1, \cdots, x_p]$$

- For convenience of notation, we define $x_0 = 1$
- We define our hypothesis function as follows:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p = \Theta^T X$$

- Our cost function is (where y is the vector of all output values):

$$J(\Theta) = (y - X\Theta)^T (y - X\Theta)$$

- Our gradient descent rule becomes:

$$\Theta_{j+1} = \Theta_j - \alpha \nabla J(\Theta_j)$$

FEATURE SCALING

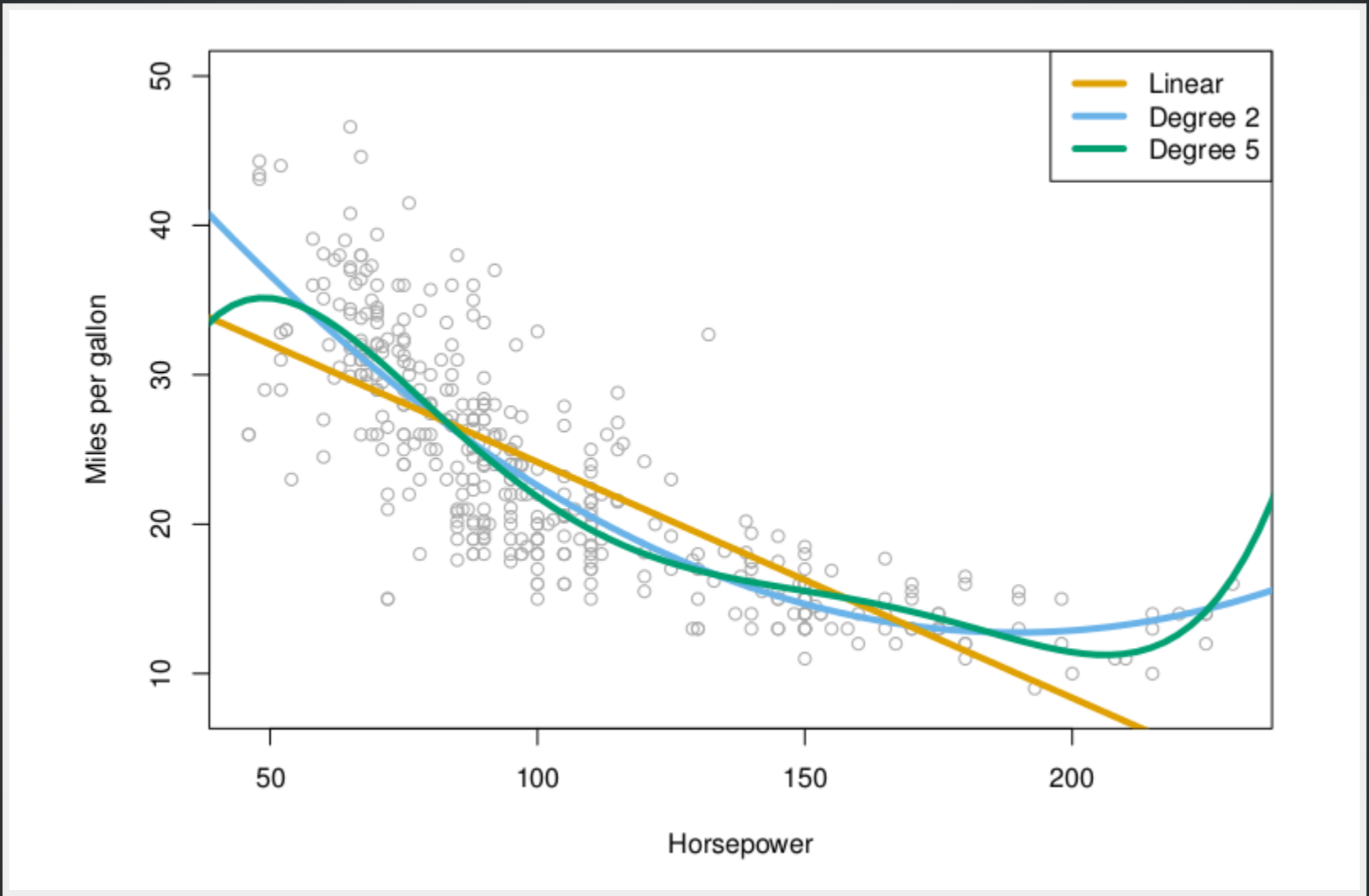
- Suppose we are performing multivariate linear regression with two features x_1, x_2 which differ greatly in scale
- $J(\Theta)$ will have the shape of a bowl that has been "stretched" — boat shaped, or like a long thin valley
- Gradient descent can be slow as it oscillates inefficiently down to the minimum, bouncing between opposite sides of the "valley"
- Speed up gradient descent by making the features similar in scale
- We can make the ranges of the features the same
- Usually apply z-score normalisation: $x_i' = (x_i - \bar{x}_i)/s_i$, where $\bar{x}_i$ and s_i are the sample mean and standard deviation of feature i
- Some machine learning methods (*e.g.*, support vector machines) require feature scaling, and it's good practice to do it anyway, although it's not always necessary
- Often log transform features with skewed distributions

POLYNOMIAL REGRESSION

- We can improve our features and the form of our hypothesis function in a couple different ways.
- Our hypothesis function need not be linear if that does not fit the data well
- We can add higher-order terms that can also account for interactions between features
- Feature scaling is required to avoid features blowing up (*e.g.*, if x_1 has range 1–10^3 then range of x_1^2 becomes 1–10^6)
- For a single variable, our hypothesis function looks like this:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \cdots + \theta_p x_1^p$$

POLYNOMIAL REGRESSION ON CAR DATA



What degree polynomial is best? We'll discuss how this choice is made later.

LOGISTIC REGRESSION

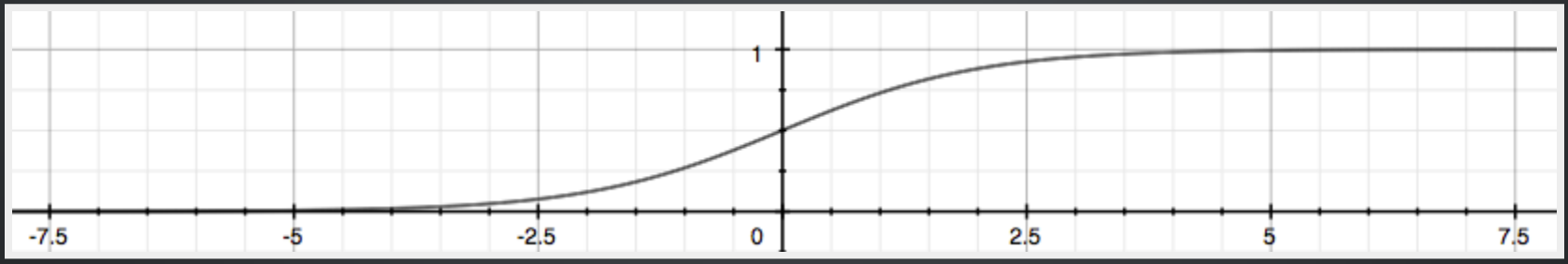
- Now switching from regression problems to classification problems
- Don't be confused by the name "logistic regression"; it is named that way for historical reasons
- To start with, consider a binary classification problem where $y \in \{0, 1\}$, where 0 is usually taken as the "negative class" and 1 as the "positive class" and
  - 0 = background, 1 = signal
  - 0 = good email, 1 = spam
  - *etc.*
- We could use linear regression to predict the class probabilities, and map all predictions $<$ 0.5 as 1 and $>$ 0.5 as 1, but we run into problems with probabilities less than 0 or greater than 1
- Our hypothesis function should satisfy: $0 \leq \hat{y} \leq 1$

LOGISTIC FUNCTION

- The function

$$f(x) = \frac{1}{1 + e^{-x}}$$

shown here, maps any real number to the (0,1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification



LOGISTIC REGRESSION HYPOTHESIS REPRESENTATION

- Logistic regression uses the function:

$$P(X) = \frac{e^{\theta_0+\theta_1 x_1+\cdots+\theta_p x_p}}{1+e^{\theta_0+\theta_1 x_1+\cdots+\theta_p x_p}}$$

- or, more compactly:

$$P(X) = \frac{e^{\Theta^T X}}{1+e^{\Theta^T X}}$$

- A bit of rearrangement gives:

$$\log\left(\frac{P(X)}{1-P(X)}\right) = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p = \Theta^T X$$

- This transformation is called the *log odds* or *logit* of $P(X)$

FROM PROBABILITIES TO CLASS PREDICTIONS

- To estimate the parameters Θ, we can use gradient descent with the cost function:

$$J(\Theta) = (-y^T \log(\hat{y}) - (1 - y)^T \log(1 - \hat{y}))$$

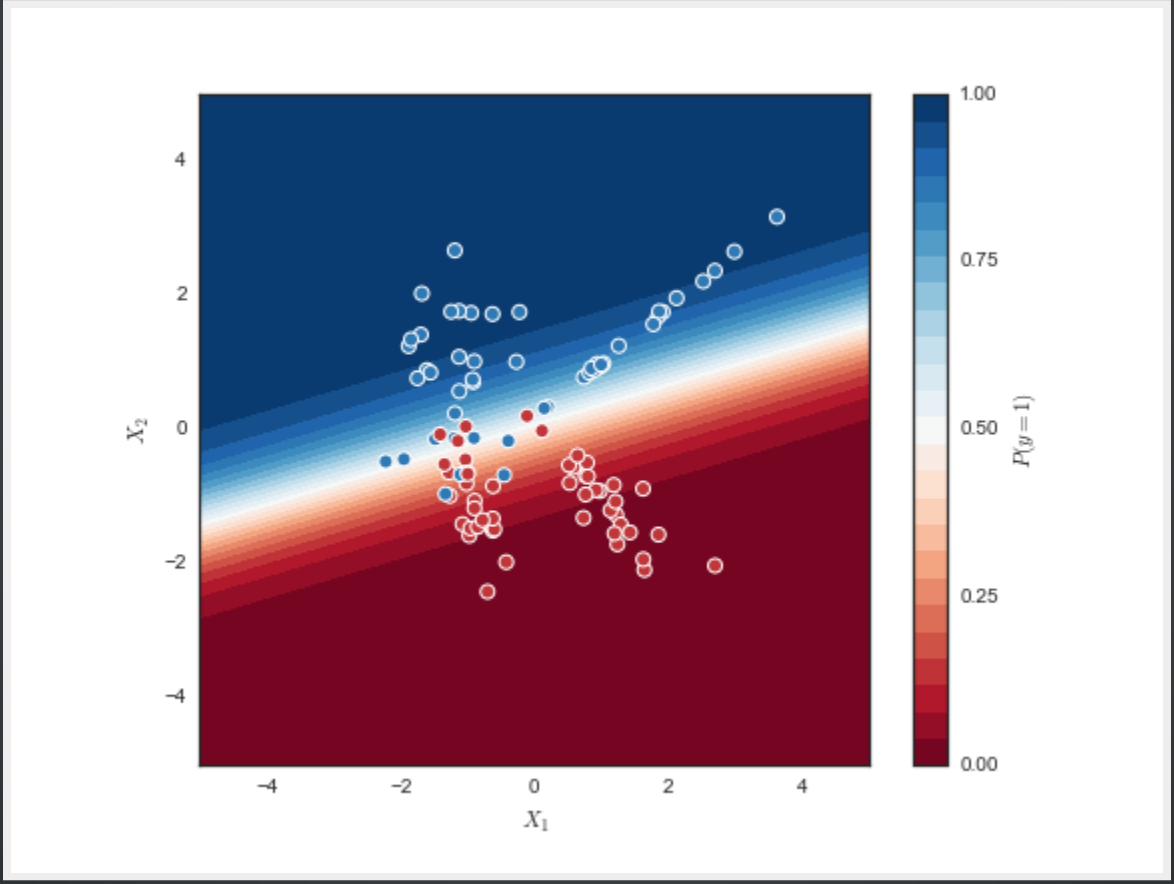
- In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

$$\hat{y} \geq 0.5 \implies y = 1$$
$$\hat{y} < 0.5 \implies y = 0$$

- The *decision boundary* is the line that separates the area where $y = 0$ and where $y = 1$

LOGISTIC REGRESSION DECISION BOUNDARY

- Logistic regression classifies new samples based on any threshold you want, so it doesn't inherently have one *decision boundary*; here we show a contour plot for a range of threshold values:



CONFUSION MATRIX

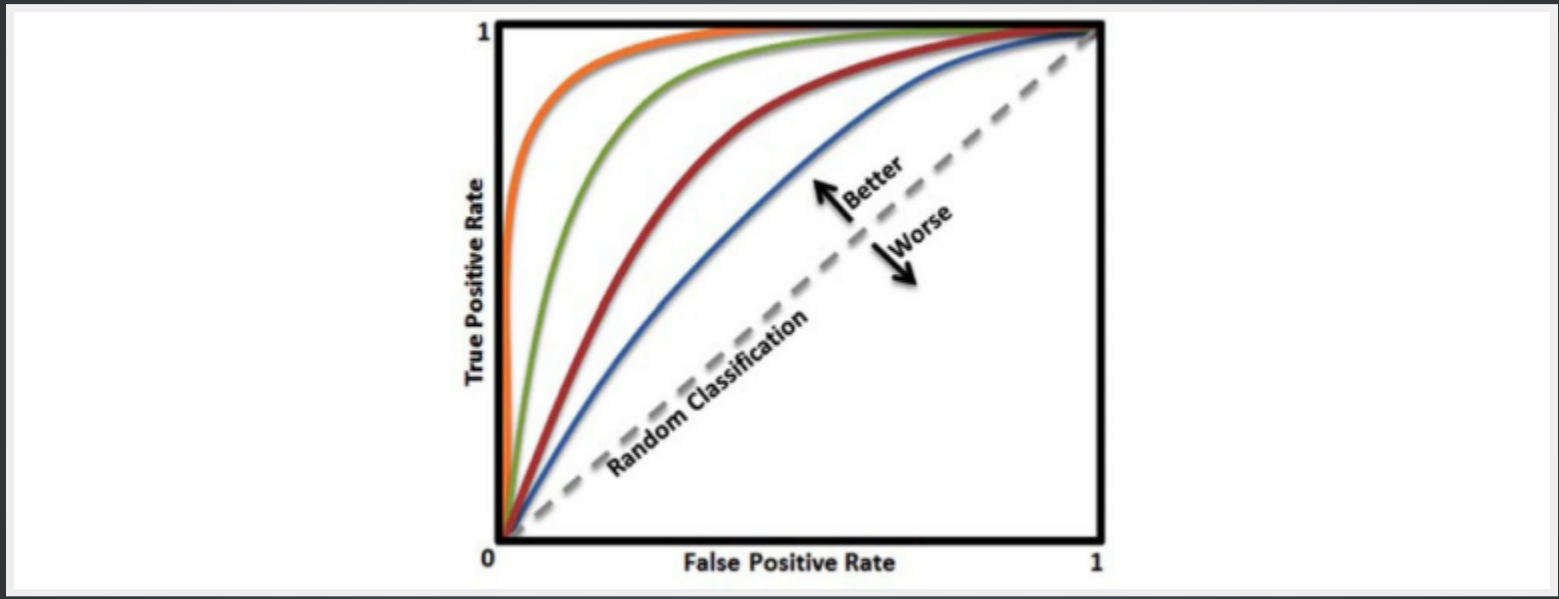
- A confusion matrix is a (contingency) table that is often used to describe the performance of a classification model, for example:

		Predicted	
		Positive	Negative
True	Positive	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

- Types of errors:
  - *False positive rate*: The fraction of negative examples that are classified as positive
  - *False negative rate*: The fraction of positive examples that are classified as negative
- We can change the two error rates by changing the threshold from 0.5 to some other value in [0, 1]

ROC CURVE

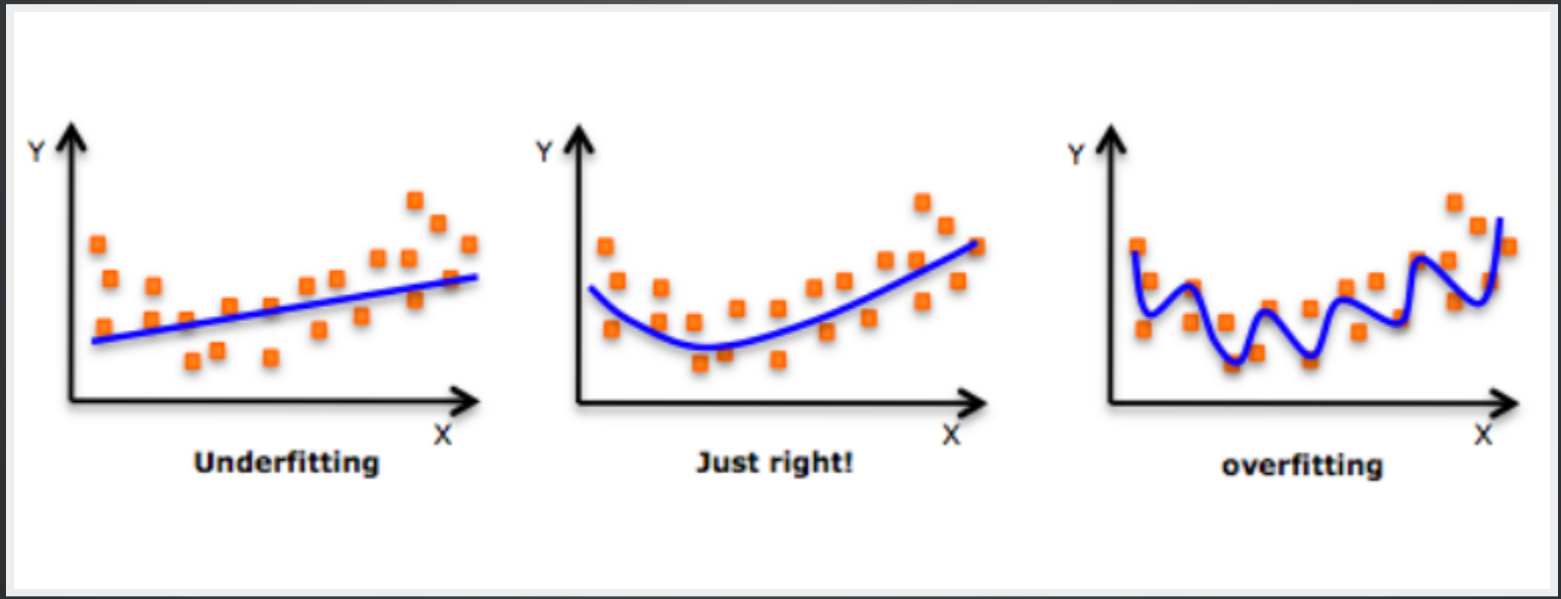
- A *receiver operating characteristic (ROC)*, or *ROC curve*, is a graphical plot that illustrates the performance of a binary classifier:



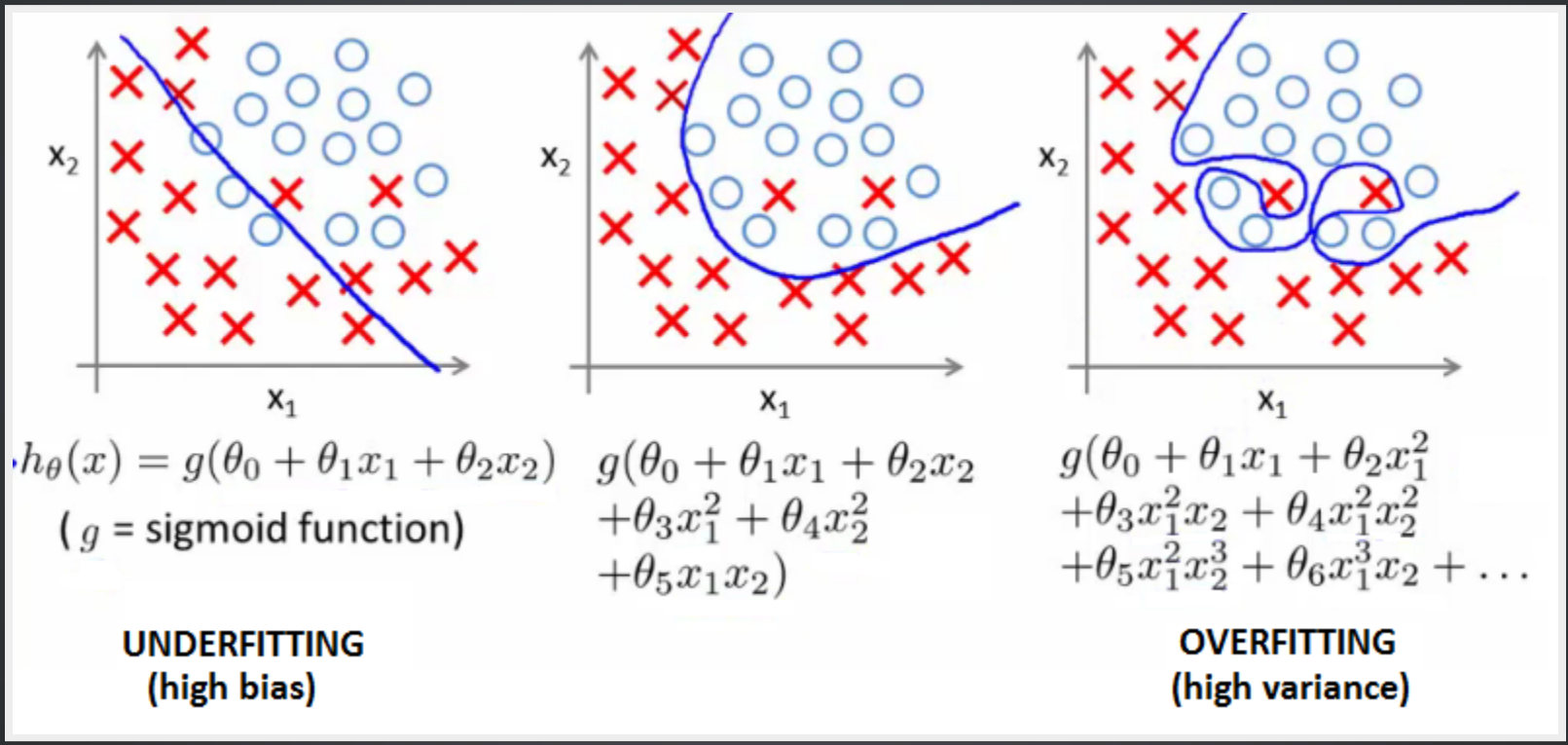
- Displays both the false positive and false negative rates
- The ROC curve is traced out as we change the threshold
- Sometimes we use the *AUC* or *area under the curve* to summarise the overall performance (higher AUC is good)

THE PROBLEM OF OVERFITTING

- If we have too many features, the learned hypothesis may fit the training data very well but fail to generalize to new examples
- Example: linear regression



AN EXAMPLE OF OVERFITTING IN CLASSIFICATION



REGULARISATION

- Regularization is designed to address the problem of overfitting
- This technique that *constrains* or *regularises* the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero
- It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance

RIDGE REGRESSION

- The least squares fitting procedure in linear regression estimates the parameters Θ for p features using the values that minimise:

$$J(\Theta) = \sum_{i=1}^{N} \left(y_i - \hat{y}_i\right)^2 = \sum_{i=1}^{N} \left(y_i - \theta_0 + \sum_{j=1}^{p} \theta_j x_{ij}\right)^2$$

- In contrast, the ridge regression coefficient estimates are the values that minimize (where $\lambda \geq 0$ is a tuning parameter to be determined separately):

$$J(\Theta) = \sum_{i=1}^{N} \left(y_i - \theta_0 + \sum_{j=1}^{p} \theta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p} \theta_j^2$$

RIDGE REGRESSION: CONTINUED

- As with least squares, ridge regression seeks coefficient estimates that fit the data well, by making the RSS small
- However, the second term in the cost function, $\lambda \sum_j \theta_j^2$, called the *shrinkage penalty*, is small when the θ are close to zero, and so it has the effect of shrinking the estimates of θ towards zero
- The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates
- The shrinkage penalty sets the budget for how large the coefficient estimates can be; we minimise the RSS subject to $\sum_j \theta_j^2 \leq s$
- Selecting a good value for λ is critical; use *cross-validation*
- Regularisation used for both regression and classification
- Feature scaling must be used due to the sum of squared coefficients in the shrinkage term

REGULARISATION: SOME INTUITION

- By shrinking some of the coefficient estimates towards zero, we reduce the influence of some of the features on the model, thereby making it less complex; λ the model flexibility
- We can use regularisation to smooth the output of our hypothesis function to reduce overfitting
- If λ is chosen to be too large, it may smooth out the function too much and cause underfitting
- To help you think about this:
  - Imagine you're performing a least squares fit to some data with sticks of spaghetti; how long you cook the spaghetti for controls how well you can bend it to fit the data!
  - If you overcook, you'll be able to fit every data point, in effect *memorising* the training data — but the resultant model will not generalise to new data points, and is useless

SELECTING λ FOR RIDGE REGRESSION

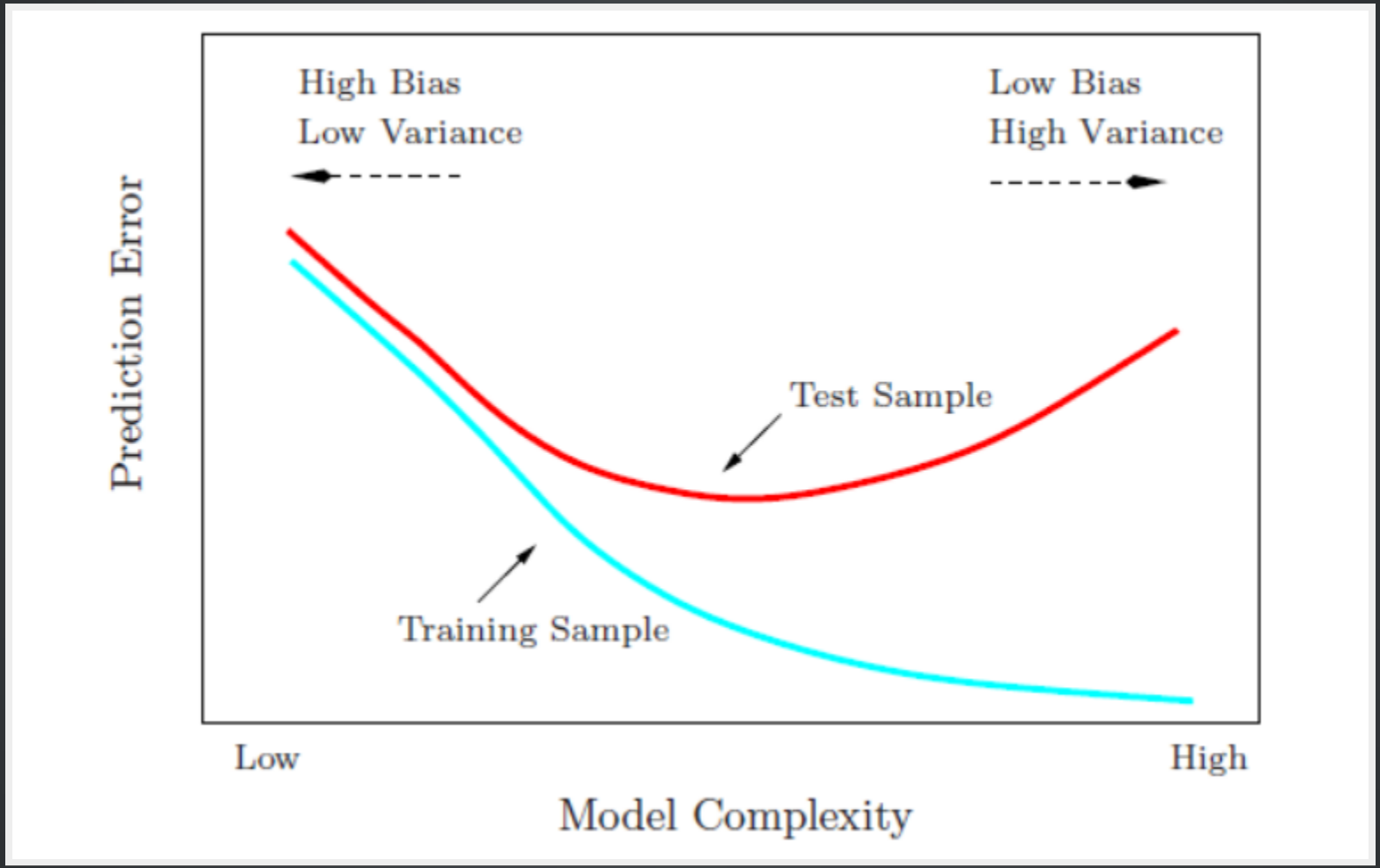
(A BRIEF LOOK-AHEAD AT CROSS-VALIDATION, BEFORE WE EXPLORE IN MORE DEPTH)

- Ridge regression requires a method to determine which of the models under consideration is best
- That is, we require a method selecting a value for the tuning parameter λ
- *Cross-validation* provides a simple way to tackle this problem. We choose a grid of λ values, and compute the cross-validation error rate for each value of λ
- We then select the tuning parameter value for which the cross-validation error is smallest
- Finally, the model is re-fit using all of the available observations and the selected value of the tuning parameter

TRAINING ERROR VERSUS TEST ERROR

- The *test error* is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method
- In contrast, the *training error* can be easily calculated by applying the statistical learning method to the observations used in its training
- But the training error rate often is quite different from the test error rate, and in particular the former can **dramatically underestimate** the latter

TRAINING- VERSUS TEST-SET PERFORMANCE



- Left: underfitting, right: overfitting
- Optimal model is where the test error is at a minimum

VALIDATION SET APPROACH TO ESTIMATE TEST ERROR

- Here we randomly divide the available set of samples into two parts: a *training set* and a *validation* or *hold-out set*
- The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set
- Validation set error provides an estimate of the test error
- Most common scheme used in HEP, if any is used at all!

MODEL SELECTION AND TRAIN/VALIDATION/TEST SETS

- If we perform model selection or tune paramaters such as polynomial degree (for polynomial regression) or the regularisation parameter λ using the test set, the error estimate is very likely to be overly optimistic
- To solve this, we can introduce a third set, the *cross-validation set*, to serve as an intermediate set that we can use to perform tuning and model selection, *e.g.*:
  - Optimise the parameters in Θ using the training set for each tuning parameter value
  - Pick the tuning parameter with the lowest error evaluated on the CV set
  - Estimate the generalisation error using the test set
- A typical train/CV/test split is 60%/20%/20%

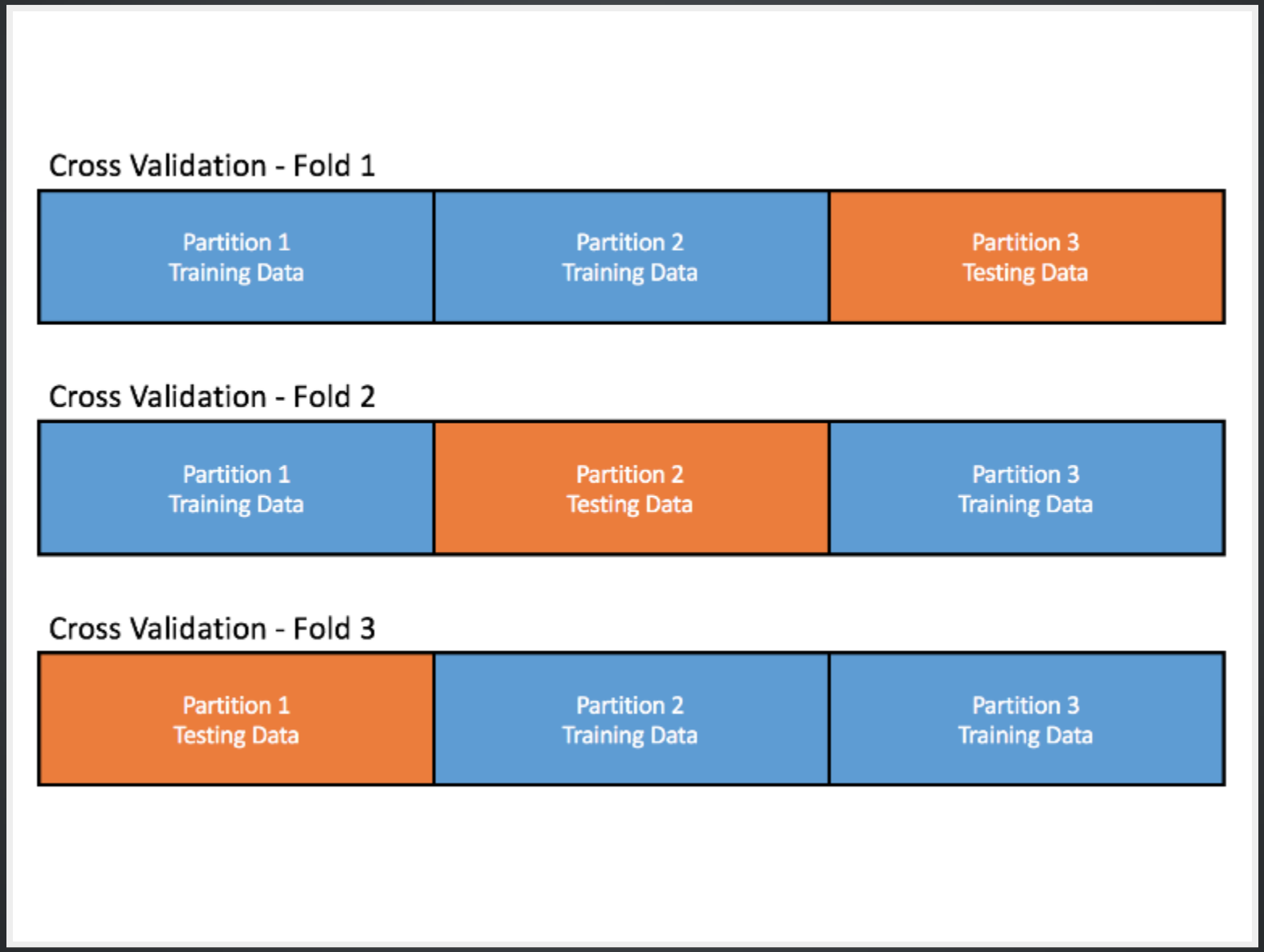
DRAWBACKS OF VALIDATION SET APPROACH

- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set
- In the validation approach, only a subset of the observations — those that are included in the training set rather than in the validation set — are used to fit the model
- This suggests that the validation set error may tend to *overestimate* the test error for the model fit on the entire data set

K-FOLD CROSS-VALIDATION

- Widely used approach for estimating test error
- Estimates can be used to select best model, and to give an idea of the test error of the final chosen model
- Idea is to randomly divide the data into K equal-sized parts. We leave out part k, fit the model to the other $K - 1$ parts (combined), and then obtain predictions for the left-out kth part
- This is done in turn for each part $k = 1, 2, \ldots, K$
- The k results from the folds can then be averaged to produce a single estimation
- The advantage of this method over the simple hold-out method is that all observations are used for both training and validation, and each observation is used for validation exactly once
- Typical values for K are 5 or 10

K-FOLD CROSS-VALIDATION ILLUSTRATED



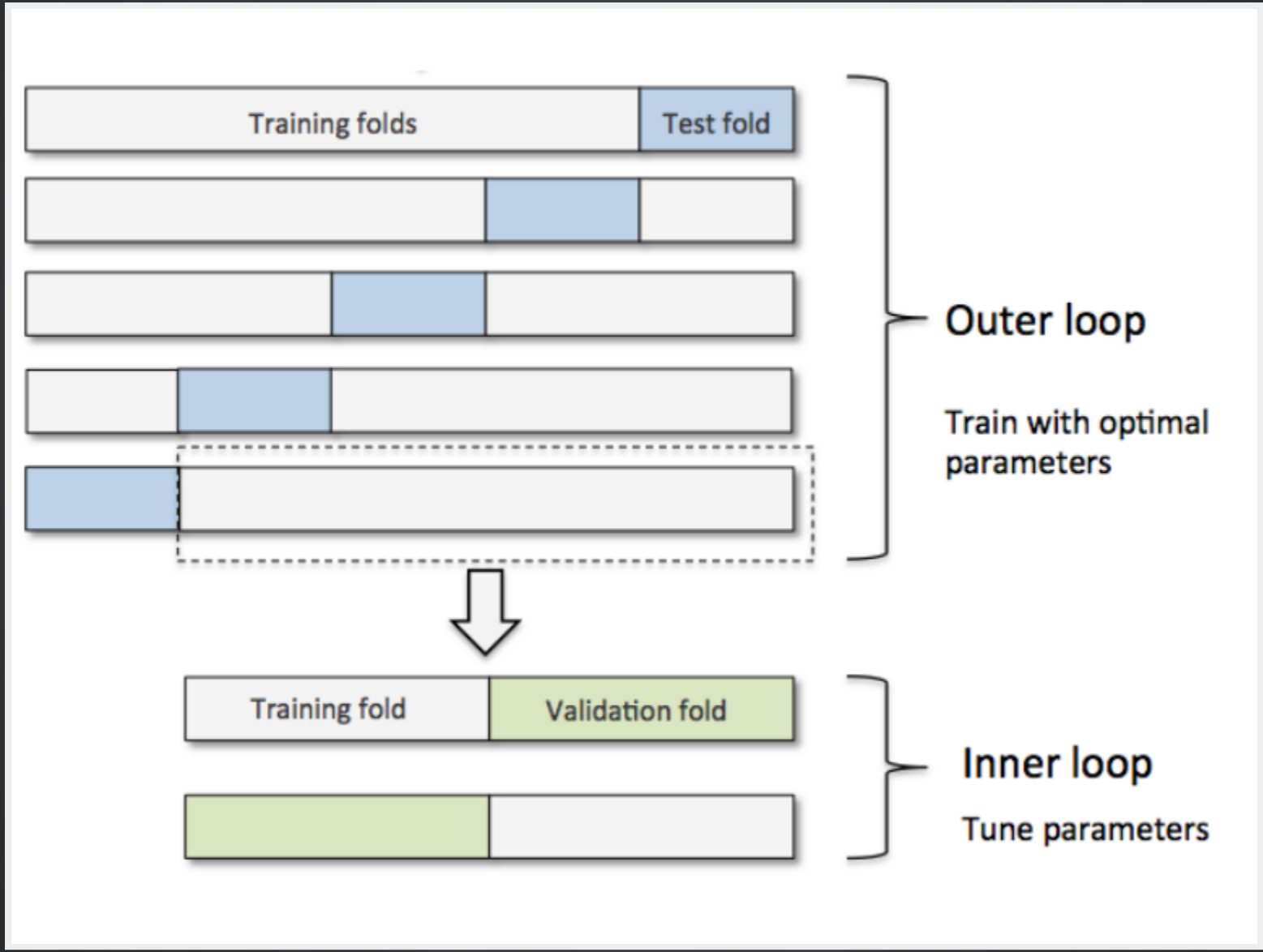
WHAT TO DO NEXT?

- What do we do next, after having used cross-validation to choose a value of a tuning parameter (*e.g.*, λ)?
- It may be an obvious point, but worth being clear: we now fit our estimator to the *entire* training set using the tuning parameter value we obtained
- Deploy model!

MODEL SELECTION VERSUS MODEL EVALUATION

- Cross-validation is used for both model selection/tuning and to evaluate the test error of the final chosen model
- Should we use the error obtained on the CV set during model selection/tuning as an estimate of the test error of the final chosen model? No!
- The CV error estimate is likely to be overly optimistic, because we obtained it using the same observations as those used to select/tune the model to optimise performance!
- The solution is to use an *outer loop* of cross-validation to estimate the performance of the final model

NESTED K-FOLD CROSS VALIDATION ILLUSTRATED



NESTED CROSS VALIDATION: SOME FINAL DETAILS

- If you use cross validation to estimate the hyperparameters of a model and then use those hyperparameters to fit a model to the whole dataset, then that is fine, provided that you recognise that the cross validation estimate of performance is likely to be (possibly substantially) optimistically biased
- This is because part of the model (the hyperparameters) have been selected to optimise the cross validation performance, so if the cross validation statistic has a non-zero variance (and it will) there is the possibility of overfitting the model selection criterion
- If you want to choose the hyperparameters and estimate the performance of the resulting model then you need to perform a nested cross-validation, where the outer cross validation is used to assess the performance of the model, and the inner cross validation is used to determine the hyperparameters

INFORMATION LEAKAGE

- Information leakage is the creation of unexpected additional information in the training data, allowing a model or machine learning algorithm to make unrealistically good predictions
- Leakage is a pervasive challenge in applied machine learning, causing models to over-represent their generalization error and often rendering them useless in the real world
- Given that methods such as cross validation and other steps to protect against information leakage are largely unknown in the HEP community, it's reasonable to assume that it occurs in HEP analyses frequently

INFORMATION LEAKAGE EXAMPLES

- If you evaluate the performance of your final model using test data, and tune your model to get a better result, you are using the test data for tuning!
- If you apply feature scaling on the whole data set before creating partitions or folds, the means and standard deviations (and any data transformation parameters) have seen the test data!
- If you choose the features for your model before creating partitions or folds, you are using information from the test data, and you get choose uninformative features that are spuriously correlated with the output →| useless final model!
- If you tune *any* cuts using the test data, your final model is likely to have overly optimistic performance!
  - This is true even if you are using simple one-dimensional rectangular cuts on variables — *i.e.*, the traditional cut-based method commonly used in HEP!

FEATURE SELECTION

PROBLEMS OF TOO MANY FEATURES

- Correlated features can skew prediction
- Irrelevant features (not correlated to class variable) cause unnecessary blowup of the model space
- Irrelevant features can drown the information provided by informative features in noise
- Irrelevant features in a model reduce its explanatory value (also when predictive accuracy is not reduced)
- Training may be slower and more computationally expensive
- Increased risk of overfitting

REDUNDANT & IRRELEVANT FEATURES

- What should we do when it is likely that the data contains many redundant or irrelevant features?
- **Redundant features** are those which provide no more information than the currently selected features
- **Irrelevant features** provide no useful information in any context
- Various methods exist to remove them:
  - *Wrapper methods* consider the selection of a set of features as a search problem, where different combinations are prepared, evaluated and compared to other combinations — typically slow but give good performance
  - *Filter methods* apply a statistical measure to assign a scoring to each feature — fast, but consider each feature independently
  - *Embedded methods* learn which features best contribute to the accuracy of the model while the model is being created

CHOOSING A CLASSIFIER

- We have only considered simple linear models so far
- Linear models are seldom correct, but they can be informative!
- In some cases, the true relationship between the input and output might actually be close to linear — so a more complex model will not increase performance much
- Simple models may do as well as complex models if the features are uniformative
- In any case, your “go-to” algorithm before trying anything else should be a simple linear model; this will give you a baseline with which to compare against
- Boosted decision trees and neural nets are fashionable in HEP nowadays, but in some cases a simple model might get reasonable performance and be more interpretable, and is probably easy to tune and faster to train

One advantage of crude models is that we know they are crude and will not try to read too much from them. With more sophisticated models,

> *… there is an awful temptation to squeeze the lemon until it is dry and to present a picture of the future which through its very precision and verisimilitude carries conviction. Yet a man who uses an imaginary map, thinking it is a true one, is like to be worse off than someone with no map at all; for he will fail to inquire whenever he can, to observe every detail on his way, and to search continuously with all his senses and all his intelligence for indications of where he should go.*

From *Small is Beautiful* by E. F. Schumacher

THAT'S PROBABLY ENOUGH THEORY FOR NOW!