

# The Fundamentals of Machine Learning

Willie Brink<sup>1</sup>, Nyalleng Moorosi<sup>2</sup>

<sup>1</sup>Stellenbosch University, South Africa

<sup>2</sup>Council for Scientific and Industrial Research, South Africa

Deep Learning Indaba 2017

# About this tutorial

- ▶ We'll provide a very gentle and intuition-friendly introduction to the basics of Machine Learning.
- ▶ We might brush over (and most likely skip a few) important issues, but luckily a lot of these will be revisited over the next few days.
- ▶ Overview:
  - ▶ the what, why and where of ML
  - ▶ a few typical ML tasks
  - ▶ optimization and probability (two often overlooked but **key aspects** of ML!)
  - ▶ practical considerations

# What is ML?

- ▶ The aim of Machine Learning is to give a computer the ability to **find patterns and underlying structure in data**, in order to understand what is happening and to predict what will happen.
- ▶ Many ML problems boil down to that of constructing a mathematical relationship between some input and output, given many examples, such that new examples will also be explained well.  
*(The principle of generalization.)*

# Why is ML a thing?

- ▶ Hard problems in high dimensions, like many modern CV or NLP problems, require complex models which would be difficult (if not impossible) for a human to engineer and hard-code.
- ▶ Machines can discover hidden, non-obvious patterns.
- ▶ The proliferation of data as well as the rise of cloud-based storage and computing have opened up unprecedented possibilities for discovery and advancement in just about every scientific discipline.

*Quick experiment: how many disciplines are in this room?*

# ML as data-driven modelling

- ▶ Get data.
- ▶ Decide what the machine should learn from the data, and which aspects (features) of the data are useful.
- ▶ Make assumptions, pick a suitable model/algorithm.
- ▶ Train the model. *This is where the machine learns.*
- ▶ Test how well the model performs (generalizes to unseen data).
- ▶ Like any mathematical modelling process, this one can be **iterative**.

# Common learning paradigms

- ▶ **Supervised learning**

- ▶ Learn a model from a given set of input-output pairs, in order to predict the output of new inputs.

- ▶ **Unsupervised learning**

- ▶ Discover patterns and learn the structure of unlabelled data.

- ▶ **Reinforcement learning**

- ▶ Learn what actions to take in a given situation, based on rewards and penalties.

- ▶ **Semi-supervised learning**

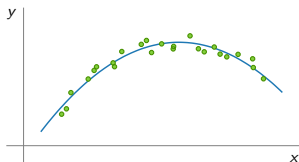
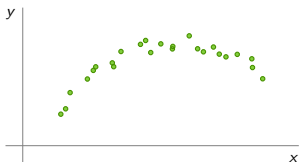
- ▶ Learn from a partially labelled dataset.

- ▶ **Deep learning**

- ▶ Any of these, but with very complex models and lots of data!

# A typical ML task: regression

- ▶ Fit a continuous functional relationship between input-output pairs.



- ▶ Example applications:
  - ▶ weather forecasting
  - ▶ house pricing prediction
  - ▶ epidemiology

# A typical ML task: classification

- ▶ Learn decision boundaries between the different classes in a dataset.

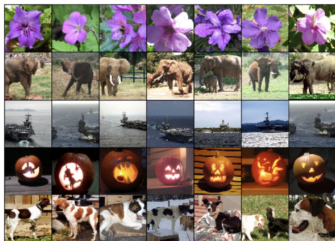


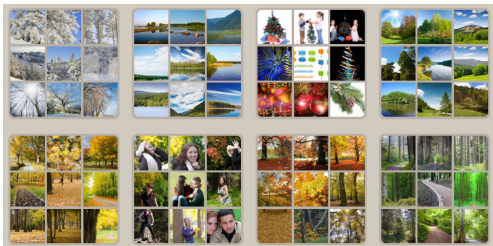
image-net.org

- ▶ Example applications:
  - ▶ handwritten character recognition
  - ▶ e-mail spam filter
  - ▶ cancer diagnosis



# A typical ML task: clustering

- ▶ Group similar datapoints into meaningful clusters.



[pixolution.org/lab/palm](http://pixolution.org/lab/palm)

- ▶ Example applications:
  - ▶ identify clients with similar insurance risk profiles
  - ▶ discover links between symptoms and diseases
  - ▶ build visual dictionaries

# A typical ML task: reinforcement learning

- ▶ Learn a winning strategy through constant feedback.



- ▶ Example applications:
  - ▶ autonomous navigation
  - ▶ evaluate trading strategies
  - ▶ learning to fly a helicopter

## A typical ML task: make a recommendation

- ▶ Generate personalized recommendation, through collaborative filtering of sparse user ratings.

user	Book1	Book2	Book3	Book4	Book5	Book6
A	3	2		5		
B	4		3	0		5
C	1	4		3	2	
D		3	2		4	

- ▶ Example applications:
  - ▶ which movies or books might a particular user enjoy
  - ▶ which news articles are relevant for a particular user
  - ▶ which song should be played next

# Characteristics of an ML model

- ▶ Consider the ML task of polynomial regression: given training data  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $\dots$ ,  $(x_N, y_N)$ , fit an  $n$ th degree polynomial of the form

$$y = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0.$$

- ▶ The degree of the polynomial is a **hyperparameter** of the model. Its value should be chosen prior to training (akin to model selection).
- ▶ The coefficients  $a_n, a_{n-1}, \dots, a_1, a_0$  are parameters for which optimal values are learned during training (model fitting).

# Learning as an optimization problem

- ▶ Consider a model with parameter set  $\theta$ . Training involves finding values for  $\theta$  that lets the model fit the data.
- ▶ The idea is to **minimize** an error, or difference between model predictions and true output in the training set.
  - ▶ Define an error/loss/cost function  $E$  of the model parameters.
  - ▶ A popular minimization technique is **gradient descent**:

$$\theta^{t+1} = \theta^t - \eta \frac{\partial E}{\partial \theta}$$

- ▶ The learning rate  $\eta$  can be updated (by some policy) to aid convergence.
- ▶ Common issues to look out for!
  - ▶ Convergence to a local, non-global minimum.
  - ▶ Convergence to a non-optimum (if the learning rate becomes too small too quickly).

# Probability

- ▶ Thinking about ML in terms of probability distributions can be extremely useful and powerful, esp. for reasoning under uncertainty.
- ▶ Your training data are samples from some underlying distribution, and a crucial underlying assumption is that that data in the test set are sampled from **the same** distribution.
- ▶ **Discriminative modelling:**  
Predict a most likely label  $y$  given a sample  $x$ , i.e. solve  $\arg \max_y p(y|x)$ .  
*Find the boundary between classes, and use it to classify new data.*
- ▶ **Generative modelling:**  
Find  $y$  by modelling the joint  $p(x, y)$ , i.e. solve  $\arg \max_y p(x|y)p(y)$ .  
*Model the distribution of each class, and match new data to these distributions.*

# Practical 1

- ▶ Implement a simple linear classifier.  
*Minimize cross-entropy through gradient descent.*
- ▶ Run the linear classifier on nonlinear data in a “swiss roll” structure.
- ▶ Introduce a nonlinear activation function in the scores, thereby implementing a nonlinear classifier.

# Practical considerations

- ▶ Let's take a closer look at a few practical aspects of ...
  - ▶ data collection and features
  - ▶ models and training
  - ▶ improving your model




# Practical considerations

- ▶ Let's take a closer look at a few practical aspects of ...
  - ▶ **data collection and features**
  - ▶ models and training
  - ▶ improving your model

# Data exploration and cleanup

- ▶ **Explore your data**: print out a few frames, plot stuff, eyeball potential correlations between inputs and outputs or between input dimensions, ...
- ▶ Deal with dirty data, missing data, outliers, etc. (domain knowledge crucial)
- ▶ Preprocess or filter the data (e.g. de-noise)?  
*Maybe not! We don't want to throw out useful information!*
- ▶ Most learning algorithms accept only numerical data, so some **encoding** may be necessary. *E.g. one-hot encoding on categorical data:*

ID	Gender
0	female
1	male
2	not specified
3	not specified
4	female



ID	male	female	not specified
0	0	1	0
1	1	0	0
2	0	0	1
3	0	0	1
4	0	1	0

- ▶ Next, design the inputs: turn raw data into relevant data (features).

# Feature engineering

- ▶ Until recently, this has been viewed as a **very important** step. Now, since the surge of DL, it's becoming almost taboo in certain circles!  
*With enough data, the machine should learn which features are relevant.*
- ▶ But in the absence of a lot of data, careful feature engineering can make all the difference (good features > good algorithms).
- ▶ Features should be **informative** and **discriminative**.
- ▶ Domain knowledge can be crucially useful here.  
*Which features might cause the required output?*
- ▶ A few feature selection strategies:
  - ▶ remove features one-by-one and measure performance
  - ▶ remove features with low variance (not discriminative)
  - ▶ remove features that are highly correlated to others (not informative)

# Feature transformations

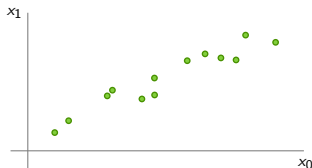
- ▶ Feature **normalization** can help remove bias in situations where different features are scaled differently.

House	Size (m <sup>2</sup> )	Bedrooms
0	208	3
1	460	5
2	240	2
3	373	4



House	Size	Bedrooms
0	-0.956	-0.387
1	1.190	1.162
2	-0.684	-1.162
3	0.449	0.387

- ▶ Dimensionality reduction (e.g. PCA).



- ▶ Potential downside: dimensions lose their intuitive meaning.

# Practical considerations

- ▶ Let's take a closer look at a few practical aspects of ...
  - ▶ data collection and features
  - ▶ **models and training**
  - ▶ improving your model

# Choosing a model

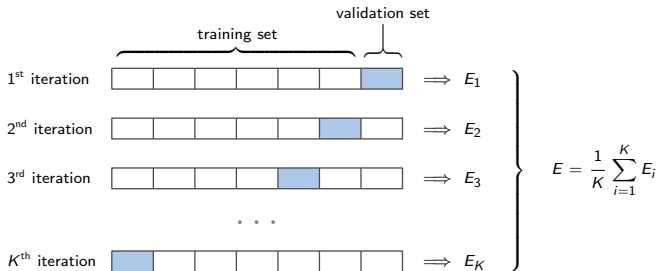
- ▶ The choice depends on the data and what we want to do with it...
  - ▶ Is the training data labelled or unlabelled?
  - ▶ Do we want categorical/ordinal/continuous output?
  - ▶ How complex do we think is the relationship between inputs and outputs?
- ▶ Many models to choose from!
  - ▶ **Regression:** linear, nonlinear, ridge, lasso, ...
  - ▶ **Classification:** naive Bayes, logistic regression, SVM, neural nets, ...
  - ▶ **Clustering:** k-means, mean-shift, EM, ...
- ▶ No one model is inherently better than any other. (the “no free lunch” theorem)  
*It really depends on the situation.*
- ▶ Choosing the appropriate model and tuning its hyperparameters can be tricky, and may have to become part of the training phase.

# Measuring performance

- ▶ The value of the loss function after training (or how well training labels are reproduced) may give insight into how well the model assumptions fit the data, but it is not an indication of the model's ability to generalize!
- ▶ Withhold part of our data from the training phase: the **test set**.
- ▶ Can we use our test set to select the best model, or to find good values for the hyperparameters? No! That'd be like training on the test set!  
*The performance measure on the test set should be unbiased.*
- ▶ Instead, extract a subset of the training set: the **validation set**.
- ▶ Train on the training set, and use the validation set for an indication of generalizability and to optimize over model structure and hyperparameters.

# K-fold cross validation

- ▶ The training/validation split reduces the data available for training.
- ▶ **Idea:** partition the training set into  $K$  folds, and give every fold a chance to act as validation set. Average the results, for a more unbiased indication of how our model might generalize.





# Performance metrics

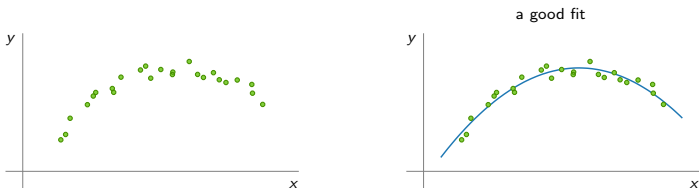
- ▶ How exactly do we measure the performance of a trained model (when validating or testing)?
- ▶ Many options! Remember: it is very important to be clear about which metric you use. *A number by itself doesn't mean much.*
  - ▶ **Regression:** RMSE, correlation coefficients, ...
  - ▶ **Classification:** confusion matrix, precision and recall, ...
  - ▶ **Clustering:** validity indices, inter-cluster density, ...
  - ▶ **Recommender systems:** rank accuracy, click-through rates, ...
- ▶ Note: most of these compare model output to a “ground truth”.
- ▶ Based on the application, which kinds of errors are more tolerable?
- ▶ It might also be insightful to ponder the **statistical significance** of your performance measurements.

# Practical considerations

- ▶ Let's take a closer look at a few practical aspects of ...
  - ▶ data collection and features
  - ▶ models and training
  - ▶ **improving your model**

# Underfitting and overfitting

- ▶ We want models that generalize well to new (previously unseen) inputs.
- ▶ Reasons for under-performance include **underfitting** and **overfitting**.  
*Example: polynomial regression*



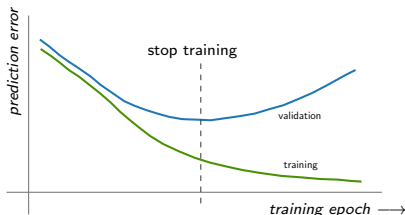
- ▶ **Underfitting** (high model bias): the model is too simple for the underlying structure in the data.
- ▶ **Overfitting** (high model variance): the model is too complex, and learns the noise in the data instead of ignoring it.

# The bias-variance tradeoff

- ▶ How can we find the sweet spot between underfitting and overfitting?  
*Use the validation set!*

	<b>Underfit</b>	<b>Good fit</b>	<b>Overfit</b>
performance on training set	bad	good	very good
performance on validation set	bad	good	bad

- ▶ We want to keep training (keep decreasing the training error) only while the validation error is also decreasing.



## A few more potential remedies

- ▶ To avoid **overfitting**, try ...
  - ▶ a simpler model structure
  - ▶ a larger training set (makes the model less sensitive to noise)
  - ▶ regularization (penalize model complexity during optimization)
  - ▶ bagging (train different models on random subsets of the data, and aggregate)
  
- ▶ To avoid **underfitting**, try ...
  - ▶ a more complex model structure
  - ▶ more features (more information about the underlying structure)
  - ▶ boosting (sequentially train models on random subsets of the data, focussing attention on areas where the previous model struggled)
  
- ▶ Note: it seems as if, no matter what, **more data** is better!

# Summary

- ▶ We touched on what ML is and why it exists, and listed a few examples of typical ML tasks.
- ▶ We viewed the learning process as an optimization problem, and mentioned the importance of considering ML in terms of probability distributions.
- ▶ We ran through a few practical considerations concerning data cleanup, feature selection, model selection and training, measuring performance, and improving performance.

# Take-home messages

- ▶ It can be extremely useful to think in terms of probability distributions, and also that training is essentially an optimization process.
- ▶ Components of an ML solution, in decreasing order of importance:
  - ▶ data
  - ▶ features
  - ▶ the model/algorithm
- ▶ Of course, **deep learning** blurs the boundary between data and features.