Grasping The Basics of Machine Learning

The technicalities of many machine learning techniques are profoundly difficult to understand , let alone to successfully implement. Sometimes implementation can be achieved without understanding the basics; other times the reverse is true. Either way, it is rarely smooth sailing to design a predictive model from start to finish as self-learning tutorials seem to imply.

I’ve studied the process a fair bit now and, having just completed [fast.ai’s practical deep learning for coders](https://course.fast.ai/), I thought this would be an ideal time to summarise my current knowledge of machine learning and how I've learnt it.

I've this blog down into the following short categories:

* Overall top down approach to learning
* Using simple models to clean data
* Base rates
* A general implementation strategy
* My next steps - Kaggle

**Overall Top Down Approach To Learning**

No doubt there will be many sharp minded non computer science graduates intrigued by the world of machine learning being met with a barrage of jargon and technical smoke and mirrors. For some they can push through. For most it’s enough to quit and leave feeling quite defeated.

Top down learning is a must. Starting with fairly basic Python skills and someone else's fully implemented model is the best way I've found to pick up skills in this field. Gradually, natural questing leads you beneath the black box to the [source code](http://neuralnetworksanddeeplearning.com/chap2.html), the [Arxiv papers](https://arxiv.org/abs/1604.06737) and the [Google dev talks](https://youtu.be/nQgUt_uADSE).

**Using Simple Models To Clean Data**

This is the first of the more technical chapters of this blog.

Below is an image which a galaxy classifier I made struggled to label. My calissifier was a restrained version of [Resnet 50](https://keras.io/api/applications/resnet/) and I built a [GUI](https://ipywidgets.readthedocs.io/en/latest/) to look at its least confident predictions and remove them from the training set if I thought they were inappropriate for the model.

With this output, it's easy to spot that one photo isn't an actual galaxy but a clipart-like drawing so it shouldn't be used for training my model.

This post-evaluation method sounds logical but wasn’t my immediate approach to cleaning up data.

I’ve applied it elsewhere with simple decision trees to see if a feature is important before passing the reduced data to a neural net. I think it's a good trick to have in the bag, and often is a quick way to get rid of dodgy data.



**Using Base Rates**

This chapter can be summarised in one question: is the model beating a rough and ready statistical/random guess approach?

This is akin to checking a velocity is less than the speed of light in physics. A vital sanity check which is often overlooked.

**My General Implement Strategy**

1. Load the data into a Jupyter notebook and state some exploratory data analysis to include as a minimum:
   * Box plots
   * Histograms
   * Correlation Matrices
2. Cut Cut Cut. Take away as much complexity as possible in the initial stages using the exploratory data analysis and the simplest model for the job. This idea is built around a framework outlined by [Elon Musk](https://modelthinkers.com/mental-model/musks-5-step-design-process#:~:text=Musk's%205%20Step%20Design%20Process%20consists%20of%20making%20the%20requirements,accelerate%20cycle%20time%3B%20and%20automate.). There’s no point optimising a complex data set/model if parts of it are unnecessary for the job in hand.
3. Does the simple model beat an appropriate base rate test ?
4. Feature engineering. Can any features be merged, can images be transformed etc. This is a good way of providing the model with more high quality data.
5. Test a range of models (random forest, neural net, gradient boosting) to get an idea for which one might be most appropriate.

Now for the parts I’m still working on…

1. Optimising the model. Use grid search to tune hyperparameters to get the most out of the model. Consider paying for a more powerful GPU.

1. Transfer ideas from recently published papers to your work. One example I have successfully implemented is the learning rate finder and [learning rate cycle](https://arxiv.org/abs/1506.01186).

This leads nicely onto my final paragraph.

**My next steps - Implementing state of the art approaches in Kaggle Competitions**

I'm aiming to score in the top half in [this football probability prediction](https://www.kaggle.com/competitions/football-match-probability-prediction) challenge on kaggle. I’ll write a new blog on that when I have progress.