***Machine Learning Yearning***

***1. Why Machine Learning Strategy***

* Machine learning is the foundation of countless important applications.
* This book will help my team make rapid progress
* If your ML model’s accuracy sucks you can try many things such as:
  + Collecting more data
  + Diversifying data
  + Hyperparameter tuning (more layers, hidden units, parameters, network size)
* If you choose well among the possible options you will do good if not you waste a lot of time
* ML problems leave clues in what’s useful to try… Learning to read these clues will save you months or years in development time

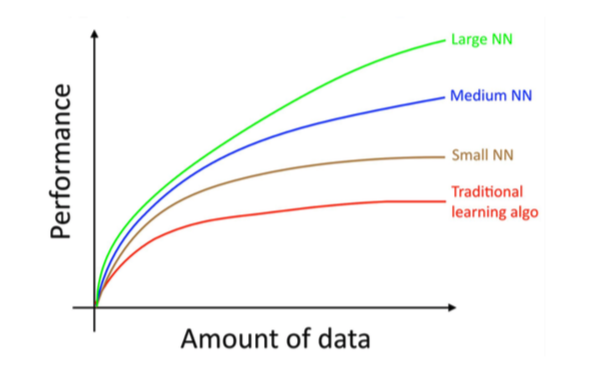
***2. How to use this book to help your team***

* After this book I will have a deep understanding of the technical direction for a machine learning project
* A few changes in prioritization can have a huge effect on your team’s productivity

***3. Prerequisites and Notation***

* Supervised learning: Learning functions that map from x to y using labeled training examples of (x,y)

***4. Scale drives machine learning progress***

* Two of the biggest drivers of recent progress:
  + Data availability: People are spending so much time on digital devices in this era. Their digital activities generate huge amounts of data that we can feed to our AI.
  + Computational scale: Since only a few years ago we now have the ability to train neural networks that are big enough to take advantage of the huge datasets we now have
* To get the best performance these days, the process is still:
  + Train a bigger network (the plateaue is higher)
  + Get more data

***5. Your development and test sets***

* Usually define 3 test sets:
  + **Training set**: Which you run your learning algorithm on
  + **Dev (development) set**: Which you use to tune your parameters, select features, and make other decisions regarding the learning algorithm
  + **Test set**: Which we use to evaluate the performance of the algorithm, but not make any decisions regarding what learning algorithm or parameters to use
* The purpose of the dev and test sets are to direct your team toward the most important changes to make to the machine learning system
* Make sure to choose dev and test sets to reflect data you expect to get in the future

***6. Your dev and test sets should come from the same distribution***

* + There is a chance that your team will build something that works well on the dev set, only to find out it does poorly on the test set. That means make sure your Dev and Test sets are picked out of the same hat of data. And this hat should contain data that we expect to see in production
  + Ex) Suppose we have a system that works well on the dev set but not the test set. If the dev and test sets came from the same distribution, then we know you overfit the dev set (so get more data). But if the dev and test sets came from different distributions. There can be several issues:
    - You had overfit to the Dev-set
    - The test set is harder than the dev set. So you algorithm might be doing as well as it could but it’s just not strong enough
    - The test set is not harder.. its just different from the dev set. What worked on the dev set doesn’t work on your test set because it is so different (ex. Different angles, resolutions). In this case we wasted effort improving the dev set performance.
  + Having mismatched dev and test sets introduces uncertainty for no reason
  + Having mismatched dev and test sets make it harder to figure out what working and what isn’t

***7. How large do the dev/test sets need to be ?***

* + The dev set should be large enough to detect the differences between the algorithms that you are trying out.
    - ex) if classifier A has an accuracy of 90.0% and classifier B has an accuracy of 90.1% then a dev set of 100 examples would not be able to detect this 0.1% difference
  + Dev sets with sizes 1,000 to 10,000 are common
  + Test set should be large enough to give high confidence in the overall performance of the system. (People say 30% of your data but you might not need to feed in 30% just to gain good confidence)
  + There is also no need to have excessively large dev/test sets beyond what is needed to evaluate the performance of your algorithm

***8. Establish a single-number evaluation metric for your team to optimize ?***

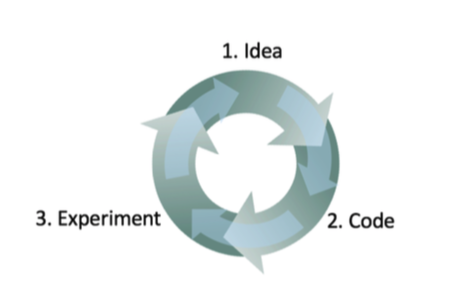
* + Classification accuracy is an example of a **single-number evaluation metric:** You run your classifier on the dev set (or test set), and get back a single number about what fraction of examples it classified correctly.
  + Precision and Recall is not a single-number evaluation metric: It gives to numbers for assessing your classifier. **This makes it harder to compare algorithms**
  + During development, your team will try a lot of ideas about algorithm architecture, model parameters, choice of features, etc. Having a single-number evaluation metric will allow us to sort out each variation.
  + Having a single-number evaluation metric speeds up your ability to make a decision when you are selecting among a number of classifiers.

***9. Optimizing and satisficing metrics***

* + Suppose you care about the accuracy and running time of a learning algorithm:
    - First define what is an “acceptable” running time
      * This is called the “satisficing metric” – you just need to be good enough on this metric
      * You simply only want to ensure that these metrics meet a certain value.
    - Your accuracy is what you will be optimizing for.
      * Call this the “optimizing metric”
      * This metric you will try to optimize within the constraints of the satisficing metric
    - Another examples: A “Hi Siri” system with false positives and false negatives
    - Once your team is aligned on the evaluation metric to optimize, they will be able to make faster progress

***10. Having a dev set and metric speeds up iterations***

* + - It is very difficult to know in advance what approach will work best for a new problem:
    - The process usually follows this:



* The faster you can go round this loop, the faster you will make progress. This is why having dev/test sets and a metric are important
* In contrast imagine you develop a new classifier and to test it you integrate it into your app and play with it for a few hours to see if it is better. This is incredibly slow.

***11. When to change dev/test sets and metrics***

* When starting a new project, try to quickly choose dev/test sets
* Try to come up with an initial dev/test set and an initial metric in less than one week
* If you realize that your dev/test set or metric missed the mark, by all means change them quickly.
* Three main causes to pivot on your dev set/metric:

1. The actual distribution you need to do well on is different from the dev/test sets:
   1. Ex) Your cat dev/test sets are mainly adult pictures of cats. You ship your cat app and find that users are uploading a lot more kitten images then expected. This means your test set distribution does not represent the actual distribution. In this case pivot.
2. You have over-fit to the dev set
   1. Repeatedly evaluating your ideas on the dev set causes your algorithm to leak to much information about your dev set.
   2. If your dev accuracy is extremely higher than your test set that is a big sign of overfitting. If this is the case get a fresh dev set.
3. The metric is measuring something other than what the project needs to optimize.
   1. Example of the cat classifier accidently letting through bad photos.

* Its quite common to change dev/test sets or evaluation metrics during a project. Having an initial dev/test set and metric helps us to iterate quickly.

***12. Takeaways: Setting up development and test sets***

* Choose dev and test sets from a distribution that reflects what data you expect to get in the future and want to do well on. This may not be the same as your training data’s distribution.
* Choose dev and test sets from the same distribution if possible.
* Choose a single-number evaluation metric for your team to optimize. If there are multiple goals that you care about, consider combining them into a single formula (such as averaging multiple error metrics) or defining satisficing and optimizing metrics.
* Machine learning is a highly iterative process: You may try many dozens of ideas before finding one that you’re satisfied with.
  + Having dev/test sets and a single-number evaluation metric helps you quickly evaluate algorithms, and therefore iterate faster.
* When starting out on a brand new application, try to establish dev/test sets and a metric quickly, say in less than a week. It might be okay to take longer on mature applications.
* The old heuristic of a 70%/30% train/test split does not apply for problems where you have a lot of data; the dev and test sets can be much less than 30% of the data.
* Your dev set should be large enough to detect meaningful changes in the accuracy of your algorithm, but not necessarily much larger. Your test set should be big enough to give you a confident estimate of the final performance of your system.
* If your dev set and metric are no longer pointing your team in the right direction, quickly change them: (i) If you had overfit the dev set, get more dev set data. (ii) If the actual distribution you care about is different from the dev/test set distribution, get new dev/test set data. (iii) If your metric is no longer measuring what is most important to you, change the metric.