Structuring Machine Learning Projects

Orthogonalization

In Orthogonalization, each operation does only thing. The operations are independent of each other. In theory, although we can model systems which have coupled (or interdependent) variables it becomes tedious. In ML, there are 4 assumptions that need to be true and orthogonal.

- Fit train set well
- Fit dev set well
- Perform well on test set
- Maximize practical application's robustness.

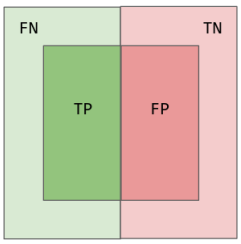
- `Train` issues → Bigger network, different optimizer
- `Dev` issues → Regularization, Bigger `train` set
- `Test` issues → Bigger `dev`
- `Application` issues → Examine distribution shift, change cost function etc.

Single number evaluation metric

Single metrics give better idea of performance than multiple competing metrics.

Item	Precision	Recall	F1
A	95%	90%	92.4
B	98%	85%	91.0

- Precision: True Positives in the sample recognized i.e. $\frac{TP}{TP+FP}$
- Recall: True Positives out of actual positives i.e. $\frac{TP}{TP+FN}$



Schema of TP,FN,FP, FN

Often there could be a trade-off between precision and recall. In such a case it is useful to consider `F1` metric.

$$
\begin{aligned}
F1 &= \frac{2}{precision^{-1} + recall^{-1}} \\
&= \frac{precision \times recall}{precision + recall}
\end{aligned}
$$

Satisficing & Optimizing Metric

Item	Acc.	Time
A	90%	80ms
B	92%	90ms
C	96%	1500ms

We could, in theory combine accuracy and time by some hybrid metric, but it will not turn out to be robust and direct. In the case of `N` metrics, one metric is kept as optimizing metric and all other metrics become the satisficing metrics.

- Max. $\underbrace{Accuracy}_{Optimizing}$ such that $\underbrace{T}_{Satisficing} \leq 100ms$
- Max. $\underbrace{Accuracy}_{Optimizing}$ such that $\underbrace{FP}_{Satisficing} < 1$ and $\underbrace{T}_{Satisficing} < 24h$

Data Distribution

`dev` and `test` should always be from the same distribution. If they happen to follow different distributions, then we would have validated our model on one, and expecting results on another. In such cases, it is advisable to randomly reshuffle the data.
We should `dev` and `test` sets to reflect the data we expect in application. `test` should be just big enough to give high confidence in model performance. Including usual cases and few edge cases are also advisable.
It is okay to not have test set at times. Practitioners report the `train-dev` performance in such cases instead. If it can be assured that model has taken care of bias and over-fitting, it is alright to skip the `test` set.

Human Factors

Choosing a classifier depends on human factors as well, such as if the model returns any junk or potentially harmful results despite of high accuracy. We can measure such performance with a designed metric such as,

$$E = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} w^{(i)} \, I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

$w^{(i)}$ is some kind of weight to rank the severity of the bad results. One needs to go through the dataset to be able to design such a metric.

Why human level performance expectations?

Bayes Optimal Error is the best theoretical ceiling limit. Progress in ML often slows down after reaching this limit. Bayes Error is often very close to human error level. Surpassing this level it is harder to verify the correctness of the solution.

Avoidable Bias

Human level error is a proxy for `Bayes Error`. We can look for reduction in bias or variance metrics.

Item	Err.	
Human	1%	
Train	8%	↑
Dev	10%	↓

Item	Err.	
Human	7.5%	
Train	8%	↑
Dev	10%	↓

Difference between Bayes Error and `train` error is the Avoidable Bias. In the left table, there is a possible under-fit which can be treated with avoidable bias reduction. In the other table, variance reduction by regularization can make the performance better.

Transfer Learning

> `"Taking knowledge learnt in one task, and apply part or whole of that knowledge to another task, usually via minor modifications"`

In CNN, we delete the last layer or last few layers and retrain with new data. The first few layers have fundamental attributes such as edges and corners, which require no change. The initial training is called Pre-training, whereas the secondary, newer task is called Fine-tuning. Transfer learning can save a lot of GPU time. Usually, pre-training is done via large data sets such as ImageNet. For Transfer learning to work well,

- Inputs should be of the same type
- Pre-training task should be done on much larger data
- Low level features from pre-training should be useful for fine-tuning.

Multi-task Learning

In multi-task learning, we start several tasks simultaneously with the objective of increasing overall accuracy in the end. The ground truth $[Y]$ could be a multi-value vector instead of a single number. These vectors could represent the ground truth of constituent tasks e.g.,

$$y^{(i)} = \begin{bmatrix} 2 \\ 0 \\ \text{True} \end{bmatrix} = \begin{bmatrix} \text{Number of Cars} \\ \text{Traffic Light} \\ \text{Day} \end{bmatrix}$$

The loss function for multitask problem with N tasks over m-samples can represented as,

$$\mathbb{L}_{\gg} = \frac{1}{m} \sum_{m} \sum_{N} L(\hat{y}_j^{(i)}, y_j^{(i)})$$

The minimization is harder because we are trying to solve N tasks on the data. Multi-task learning makes sense when,

- Training on set that could benefit from shared low-level features.
- When amount of data is similar to solve any of the task.
- We can train a large neural network for each task.

End-to-End Deep learning

When we could remove one or more pre-processing steps and rely fully on the neural network to do the task, we call such a setup End-to-end learning