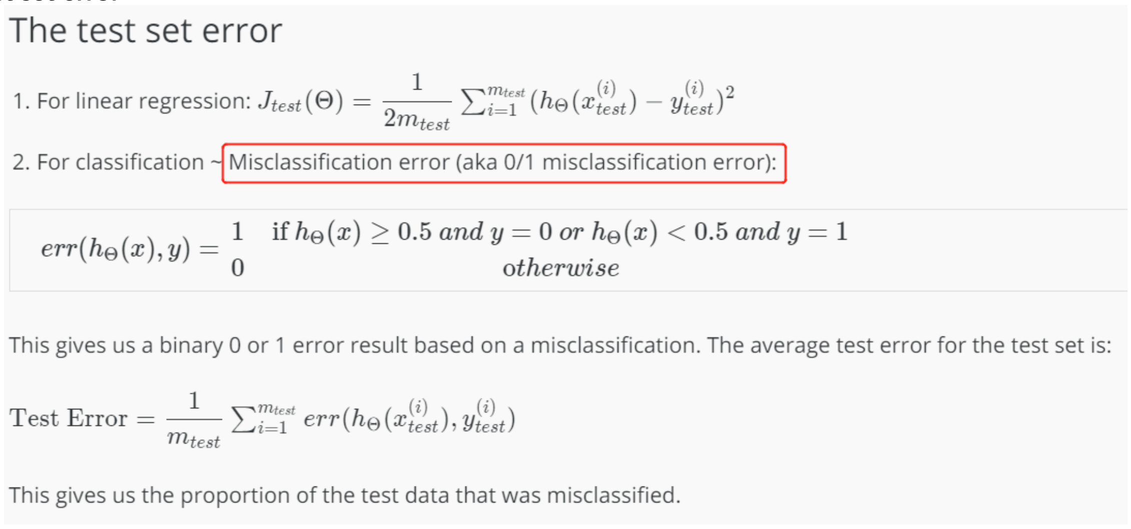
machine learning_week6

Created:	18/11/20 09:33	**Updated:**	18/11/22 17:55

Advice for Applying Machine Learning

Evaluating algorithm:

- split data set_training/testing
  - make sure the data set is random. If not, **shuffle** it.
  - usually, training set 70% and testing set 30
  - **the test set error**
    - 

      The test set error

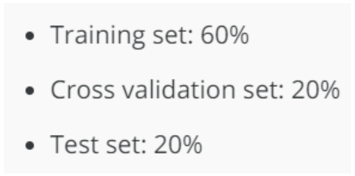
      1. For linear regression: $J_{test}(\Theta) = \dfrac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$

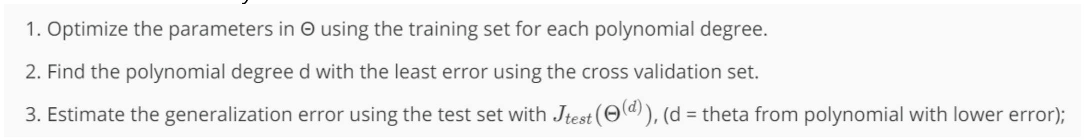
      2. For classification ~ Misclassification error (aka 0/1 misclassification error):

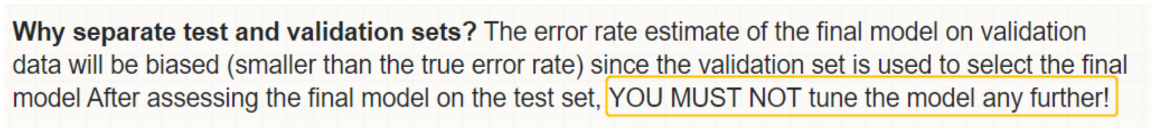
      $$err(h_\Theta(x), y) = \begin{matrix} 1 & \text{if } h_\Theta(x) \geq 0.5\ and\ y = 0\ or\ h_\Theta(x) < 0.5\ and\ y = 1 \\ 0 & otherwise \end{matrix}$$

      This gives us a binary 0 or 1 error result based on a misclassification. The average test error for the test set is:

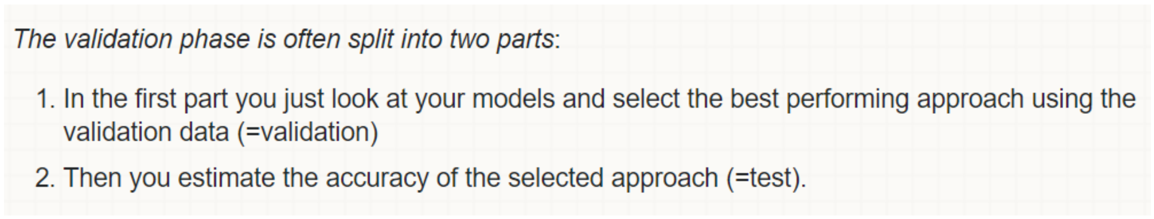
      $$\text{Test Error} = \dfrac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\Theta(x_{test}^{(i)}), y_{test}^{(i)})$$

      This gives us the proportion of the test data that was misclassified.
- **model selection:train/validation/test set**
  - task:
    1. find a suitable model
    2. test it's accuracy in real world performance.
  - method:
    - split data-set into 3 sets:
      - 

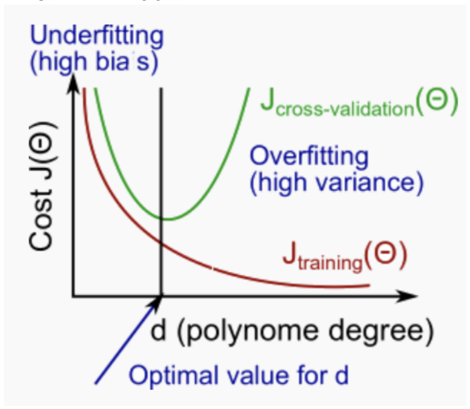
        - Training set: 60%
        - Cross validation set: 20%
        - Test set: 20%
    - then choose model by:
      - 

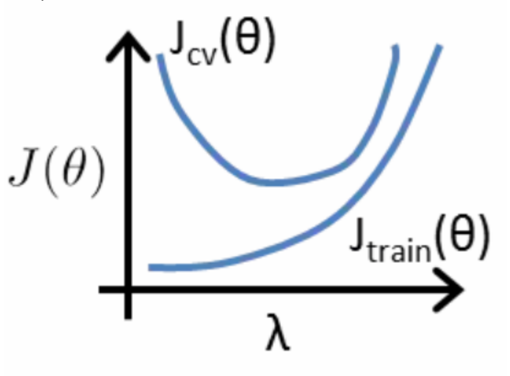
        1. Optimize the parameters in Θ using the training set for each polynomial degree.
        2. Find the polynomial degree d with the least error using the cross validation set.
        3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);
  - **why 3 sets? Can training/testing sets work?**
    - Because 3 sets can let us not only choose suitable model but also test its accuracy (recall our task).
    - and we use 3 sets to fulfill task by 2 steps. First using validation set to choose model, then using test set to predict how the model work in real circumstances.
      - crucial point: we need a set to simulate **real world circumstances.**
      - **Before implement, always keep a pure set for testing.**
    - We can't simply choose model and predict its accuracy at the same time because when choosing model there are still some factors like different lambdas may affects model selection. Hence the accuracy we got by this way would lower than actual implementing.
    - reference:
      - 

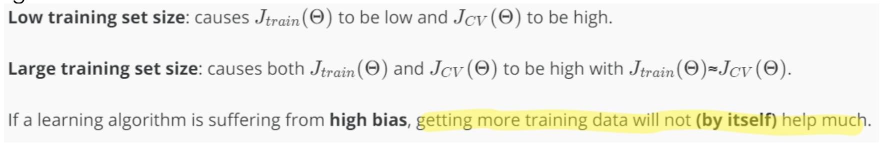
        Why separate test and validation sets? The error rate estimate of the final model on validation data will be biased (smaller than the true error rate) since the validation set is used to select the final model After assessing the final model on the test set, YOU MUST NOT tune the model any further!

        - 

          The validation phase is often split into two parts:

          1. In the first part you just look at your models and select the best performing approach using the validation data (=validation)
          2. Then you estimate the accuracy of the selected approach (=test).
        - https://stats.stackexchange.com/questions/19048/what-is-the-difference-between-test-set-and-validation-set
        - https://stats.stackexchange.com/questions/9357/why-only-three-partitions-training-validation-test
- diagnose **bias(under-fitting) or variance(over-fitting)** problem
    - compare $J_{CV}(\theta)$ with $J_{train}(\theta)$
        - if $J_{CV}(\theta) >> J_{train}(\theta)$, that may be a variance problem
        - if $J_{CV}(\theta) \approx J_{train}(\theta)$, that may be a bias problem
        - 

    - **choose lambda**
        - create a list of lambda like {0,0.01,0.02,0.04······10.24}
        - like model selection, we use loop to implement the list of lambda and compute it on cross validation. Then choose the min cost-function one.
        - The ideal plot would like this:
            - 

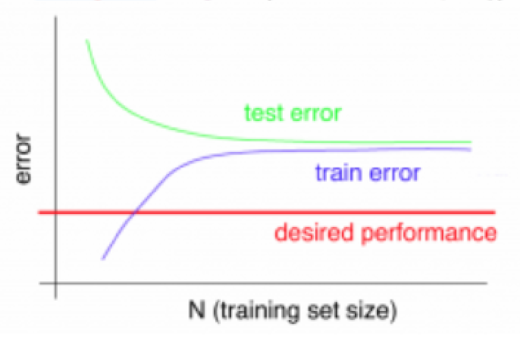
            - when lambda closes to 0, the model may be over-fitting; when lambda grows to large, it led theta approximately equal to 0, so the model we get would be a straight line, that is under-fitting.
    - **diagnose by plotting learning curves**
        - plot cost-function as a function of **m (examples)**
        - when high bias:
            - 

              Low training set size: causes $J_{train}(\Theta)$ to be low and $J_{CV}(\Theta)$ to be high.

              Large training set size: causes both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ to be high with $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

              If a learning algorithm is suffering from **high bias**, getting more training data will not **(by itself)** help much.

Typical learning curvefor high bias(at fixed model complexity):



- when high variance

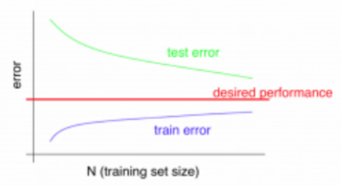
  Low training set size: $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be high.

  Large training set size: $J_{train}(\Theta)$ increases with training set size and $J_{CV}(\Theta)$ continues to decrease without leveling off. Also, $J_{train}(\Theta) < J_{CV}(\Theta)$ but the difference between them remains significant.

  If a learning algorithm is suffering from **high variance**, getting more training data is likely to help.

  More on Bias vs. Variance

  Typical learning curve for high variance(at fixed model complexity):



- **sum up**

  Our decision process can be broken down as follows:

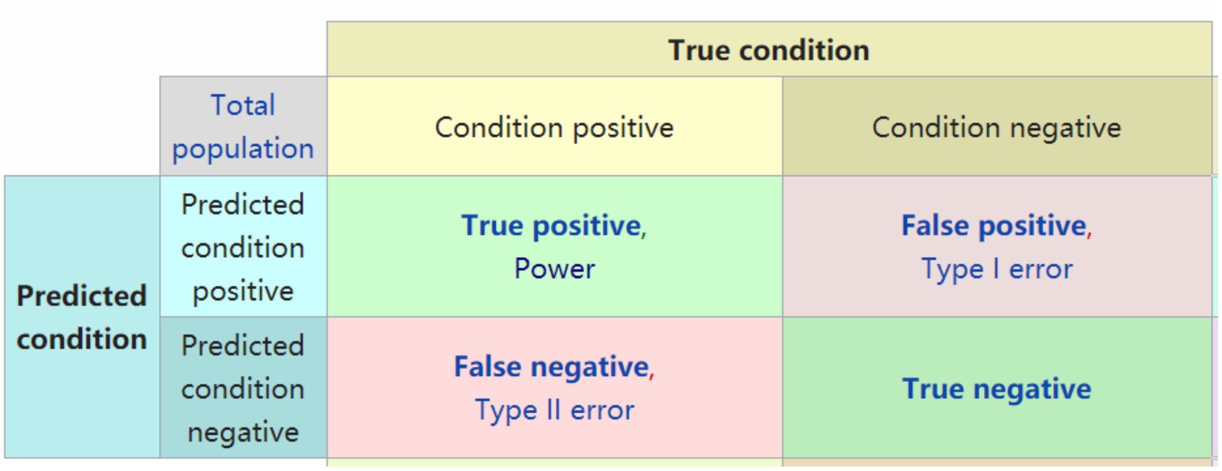
  - **Getting more training examples:** Fixes high variance
  - **Trying smaller sets of features:** Fixes high variance
  - **Adding features:** Fixes high bias
  - **Adding polynomial features:** Fixes high bias
  - **Decreasing λ:** Fixes high bias
  - **Increasing λ:** Fixes high variance.

  - **when in neural network:**
    - high bias: little units, little layers
    - high variance: much units, much layers.

System design:

- **spam classifier**
  - features: lots of words
  - improve accuracy?
    - collect more data
    - Develop sophisticated features
    - algorithm to detect mispellings
  - **implement**
    - **fast in.** start with simple model, build up it quickly. Test it
    - plot learning curves to debug
    - manually examine errors on examples.

    - Build up evaluation to have numerical metric.

- Error Metric_Confusion Matrix
  - skewed class:
    - one class is over-represented in the data set. Like in one set 99% is A while only 1% is B class.
    - **Accuracy can't be a metric!**
      - consider this situation: when we have a skewed class (99%_A,1%_B), just simply let model always give prediction with 0, which means the test example is A, will gain a accuracy of 99%!
  - so we use Confusion Matrix
    - 

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

    - We usually denote the rare one as 1.
    - By setting different threshold, we could get different recall and precision. But it is trade off between precision and recall.
    - Because TP+FN is constant. So when we set a high threshold (e.g. if h≥0.9, y=1)
    - precision:
      - **precision or positive predictive value (PPV)**

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

    - recall
      - **sensitivity, recall, hit rate, or true positive rate (TPR)**

$$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

    - F1 Scores:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precison * recall}{precison + recall} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

    - best prediction:
      - only in diagonal have value.