WEEK 3

GENERALIZATION

in machine learning

→ Generalisation is about how well our classifier performs on testing data (and data that our classifier has never seen before)

UNDERFITTING

1) is a result from a predictor being not flexible enough

2) is a result from a predictor tht. cannot capture the relevant patterns in the data

3) Predictor F under-fits the data if: we can find another predictor F' w/ smaller E_{train} and E_{gen}

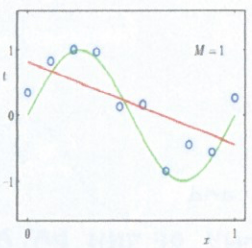
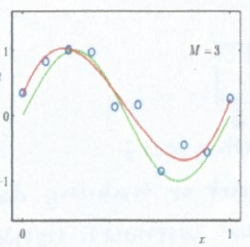
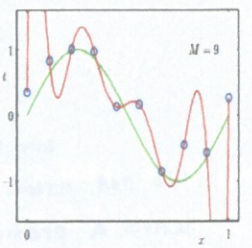
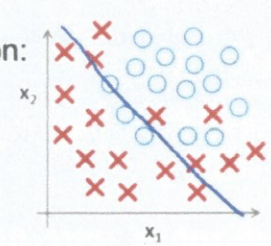
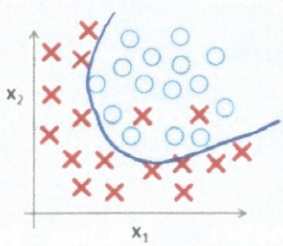
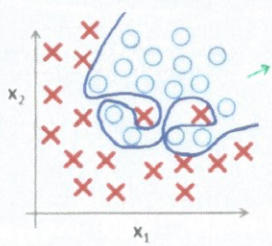
error rate on training data — error rate on unseen data

OVERFITTING

1) is a result from a predictor being too flexible

2) is a result from a predictor tht. can capture every detail of the data.

3) Predictor F over-fits the data if we can find another predictor F' which makes more mistakes on training data: $E_{train}(F') > E_{train}(F)$ but fewer mistakes on unseen future data: $E_{gen}(F') < E_{gen}(F)$

	UNDERFITTING		OVERFITTING
Regression:			
	predictor too inflexible: cannot capture pattern		predictor too flexible: fits noise in the data
Classification:			 → separates data perfectly

HOW OVERFITTING IS CONTROLLED

→ Each dataset needs diff. level of 'flexibility'

→ We can control this by:

1) Regression — order of polynomial / the no. of attributes
2) Naive Bayes — no. of attributes
   \ the limits on the distribution parameters (σ^2, ε)
3) Decision Tree — no. of nodes in the tree / pruning confidence
4) kNN ——— no. of nearest neighbors
5) SVM ——— kernel type and cost parameter

TRAINING vs. GENERALISATION ERROR

→ TRAINING ERROR:

$$E_{train} = \frac{1}{n} \sum_{i=1}^{n} error\left(f_D(x_i), y_i \right)$$

(y_i: labels; x_i: training instances; error: fn tht. computes the error for the instances; f_D: fn that predicts the label for an instance)

↳ We minimise this to build the best predictor

→ Generalisation error cannot be computed, but it can be estimated frm. testing error

→ TESTING ERROR

$$E_{test} = \frac{1}{n} \sum_{i=1}^{n} error\left(f_D(x_i), y_i \right)$$

(x_i: our testing set; We set aside part of training data and learn a predictor WITHOUT USING ANY OF THIS DATA!)

↳ Predict values for testing set and compute error

↳ This gives an estimate of true generalisation error

- If testing set is unbiased sample frm. $p(x,y)$: $\lim_{n \to \infty} E_{test} = E_{gen}$

→ i.e. Binary classification, 100 instances: Assume 75 classified correctly, 25 incorrectly...

- $E_{test} = 0.25$, this means E_{gen} will be around 0.25 but how close?

↓ compute confidence interval

CONFIDENCE INTERVAL

→ We use this to describe the range of error rates we'd expect to see when testing (future) unseen sets of instances

→ E_{test} is an unbiased estimate of E (True Error Rate)
Probability our system will misclassify a random instance

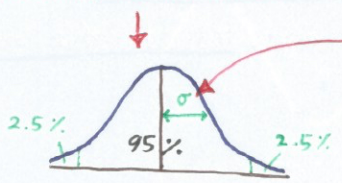
→ Take a random set of n instances, how many will be misclassified?

- This is simple to estimate bc. our classifier operater independently for each one of the instances
- No. of misclassification follows a binomial distribution w/ mean = nE and variance = $nE(1-E)$
- Future error rate E_{future} follows a gaussian distribution w/ mean = E and variance = $\frac{E(1-E)}{n}$

$$E_{future} = \frac{\text{no. of misclassifications}}{n}$$

→ Confidence Interval (CI) is specified fully by a confidence level, a mean and an interval either side of the mean → CI varies roughly w/ the $\sqrt{\text{no. of samples}}$

$$CI = \underbrace{E}_{\text{Expected Error (Mean of the Gaussian)}} \pm \underbrace{\sqrt{\frac{E(1-E)}{n}}}_{\text{Std. deviation } \sigma \text{ of error for } n \text{ instances}} \cdot \underbrace{\Phi^{-1}\left(\frac{1-p}{2}\right)}_{\text{how many deviations around mean needed to get } p\% \text{ confidence interval}}$$



If $n = 1000$ and $p = 0.95$,
$CI = 0.25 \pm 0.008$
Hence, big training sets give tighter confidence interval

→ i.e. If we have future test set of $n = 100$ instances, error rate $E = 0.25$ and we want 95% confidence interval → $p = 0.95$

$$\sigma = \sqrt{\frac{0.25(1-0.25)}{100}} = 0.043, \quad CI = 0.25 \pm 1.96\sigma$$

$$= 0.25 \pm 0.08$$

The 95% confidence interval for the error rate in a random sample of size N is about 2 std. deviations either side of the mean

If we want 99% confidence interval → $p = 0.99$

$$CI = 0.25 \pm 2.58\sigma$$

$$= 0.25 \pm 0.11$$

→ Higher confidence means more of the future test sets have to fall within the range, so we have to expand the range hence higher std. deviation to the left & right

VALIDATION SETS

→ In pratice, training and testing set is not enough!

→ We use validation set to:

1) pick best-performing algorithms to use
2) fine-tune knobs

(1) Divide N instances into 3 sets: T for training, V for validation, S for test
(2) Train each model on T and test on V and pick the one tht. performs best on V
(3) Test the chosen model **once** on S and use the error rate & size of S to generate a CI

→ We need a separate testing set bc. our classifier will be biased towards our validation set, and use it to estimate future error rate

(We run testing set **only once**)

→ Note tht. we need to split these sets randomly to avoid bias

However, we have **conflicting priorities** when splitting the dataset:

1) We need large testing set [big n_{test} → tight confidence interval]
   - to estimate future error as accurately as possible
2) We need large training set [big n_{train} → better estimates]
   - to learn classifier as accurately as possible
3) Training and testing sets cannot overlap

CROSS-VALIDATION

→ In Cross-Validation, every instance will be used for testing & training.

→ We pick the subsets in turn, train on the other subsets and test on the one we picked then average the results.

↳ Reduces the chances of getting a biased testing set

LEAVE-ONE OUT

→ is a type of cross validation in which all but one of the instances are used for training
↳ k-fold cross-validation (where k = total no. of instances)

→ A problem tht. arises is tht. the training/testing sets have classes in different proportions

SOLN → STRATIFICATION → Simple way to guard against unlucky splits
- Split dataset into classes and split each class into K parts randomly
- Assemble i^{th} part from each of the classes to make the i^{th} fold