Lecture 12: Ensemble methods

Combining models (Ensembling)

- Construct a set of base models (learners) from given training set and aggregates the outputs into a single meta-model (ensemble)
  - Classification via (weighted) majority vote
  - Regression vis (weighted) averaging
  - More generally: meta-model = $f(\text{base model})$
- Recall bias-variance trade-off:
  - $E[l(y,\hat{f}(x_0))] = (E[y]-E[\hat{f}])^2 + Var[\hat{f}] + Var[y]$
  - Averaging k independent and identically distributed predictions **reduces variance**: $Var[\hat{f}_{avg}] = \frac{1}{k}Var[\hat{f}]$
- Three methods:
  - Bagging and random forests
  - Boosting
  - Stacking

Bagging (bootstrap aggregating)

- Method: construct "near-indendent" datasets via **sampling with replacement**
  - Generate k datasets, each size n
  - Build base classifiers on each constructed dataset
  - Aggregate predictions via voting / averaging
- Bagging example: Random Forest
  - Select random subset of l of the m features
  - Train decision tree on bootstrap sample using the l feature
  - Works extremely well in many practical settings
- Reflections
  - Simple method based on sampling and voting
  - Possibility to parallelise computation of individual base classifiers
  - Highly effective over noisy datasets
  - Performance is often significantly better than (simple) base classifiers, never substantially worse
  - Improve unstable classifiers by reducing variance

Using out-of-sample data

- For each round, a particular sample has probability of $(1-\frac{1}{n})$ of not being selected
  - Probability of being left out is $(1-\frac{1}{n})^n$
  - For large n, $e^{-1} \approx 0.368$
  - On average, only 63.2% of data included per bootstrap sample
- Can use this for independent error estimate of ensemble
  - OOB (Out-Of-Bag) Error
  - Safe like CV, but on overlapping sub0samples
  - Evaluate each base classifier on its out-of-sample 36.8%
  - Average these evaluation $\rightarrow$ Evaluation of ensemble

Boosting

- Intuition:
  - Focus attention of base classifiers on examples "hard to classify"
- Method: iteratively change the distribution on examples to reflect performance of the classifier on the previous iteration
  - Start with each training instance having $1/n$ probability of being included in the sample
  - Over k iterations, train a classifier and **update the weight of each instance** according to classifier's ability to classify it
    - Misclassified -> give more weight to that instance
  - Combine the base classifiers via **weighted voting**

Adaboost

1. Initialise example distribution $P_1(i) = 1/n$
2. For $c = 1...k$
   1. Train base classifier A_c on sample with replacement from P_c
   2. Set (classifier) confidence $\alpha_c = \frac{1}{2}\ln(\frac{1-\epsilon_c}{\epsilon_c})$ for classifier's error rate ϵ_c
   3. Update example distribution to be normalised of:
      - $P_{c+1}(i) \propto P_c(i) \times \exp(-\alpha_c)$ if $A_c(i) = y_i$ (correct prediction)
      - $P_{c+1}(i) \propto P_c(i) \times \exp(\alpha_c)$ if otherwise (wrong prediction)
   4. Classify as majority vote weighted by confidences $\arg\max_y \Sigma_{c=1}^{k} \alpha_t \delta(A_c(x) = y)$

- Technically: Reinitialise example distribution whenever $\epsilon_c > 0.5$
- Base learners: often decision stumps or trees, anything "weak"
- Reflections
  - Method based on **iterative sampling** and **weighted voting**
  - More computationally expansive than bagging
  - The method has **guaranteed performance** in the form of error bounds over the training data
  - In practical applications, boosting can overfit
    - (Can do hybrid of bagging and boosting, but if using too many base classifiers -> overfit)

Bagging v.s. Boosting

- Bagging
  - Parallel sampling
  - Minimise variance
  - Simple voting
  - Classification or regression
  - Not prone to overfitting
- Boosting
  - Iterative sampling
  - Target "hard" instances
  - Weighted voting
  - Classification or regression

- Prone to overfitting (unless base learners are simple)

Stacking

- Intuition: "smooth" errors over a range of algorithms with different biases
- Method: train a meta-model over the outputs of the base learners
    - Train base- and meta-learners using CV
    - Simple meta-classifier: logistic regression
- Generalisation of bagging and boosting
- Reflections:
    - Compare this to ANNs and basis expansion
        - Mathematically simple but computationally expansive method
        - Able to combine heterogeneous classifiers with varying performance
        - With care, stacking results in as good or better results **than the best of the base classifier**

Lecture 13: Multi-armed bandits

Stochastic multi-armed bandits

- Learn to take actions
    - Receive only indirect supervision in the form of **rewards**
    - Only observe rewards for actions taken
    - Simplest setting with an **explore-exploit trade-off**

Exploration v.s. Exploitation

- "Multi-armed" bandit (MAB)
    - Simplest setting for balancing **exploration, exploitation**
    - Same family of ML tasks as reinforcement learning
- Numerous applications
    - Online advertising
    - Stochastic search in games
    - Adaptive A/B testing
    - ...

Stochastic MAB setting

- Possible actions $\{1, ..., k\}$ called **"arms"**
    - Arm i has distribution P_i on bounded rewards with mean μ_i
- In round $t = 1..T$
    - Play action $i_t \in \{1, ..., k\}$ (possibly randomly)
    - Receive reward $X_{i_t}(t) \sim P_{i_t}$
- Goal: miniise cumulative **regret**
    - $\mu^* T - \Sigma_{t=1}^T E[X_{i_t}(t)]$
    - Where $\mu^* = \max_i \mu_i$

Greedy

- At round t
  - Estimate value of each arm i as **average reward** observed
    - $Q_{t-1}(i) = \frac{\Sigma_{s=1}^{t-1} X_i(s) I(i_s=i)}{\Sigma_{s=1}^{t-1} I(i_s=i)}$, if $\Sigma_{s=1}^{t-1} I[i_s = i] > 0$
    - $Q_{t-1}(i) = Q_0$, otherwise
    - Init constant: $Q_0(i) = Q_0$ used until arm i has been pulled
  - Exploit
    - $i_t \in \arg\max_{i \le i \le k} Q_{t-1}(i)$
  - Tie breaking randomly

ϵ-Greedy

- At round t
  - Estimate value of each arm i as average reward observed
    - $Q_{t-1}(i) = \frac{\Sigma_{s=1}^{t-1} X_i(s) I(i_s=i)}{\Sigma_{s=1}^{t-1} I(i_s=i)}$, if $\Sigma_{s=1}^{t-1} I[i_s = i] > 0$
    - $Q_{t-1}(i) = Q_0$, otherwise
    - Init constant: $Q_0(i) = Q_0$ used until arm i has been pulled
  - Exploit
    - $i_t \in \arg\max_{i \le i \le k} Q_{t-1}(i)$ w.p. $1 - \epsilon$
    - $i_t \in$ Unif($\{1, ..., k\}$) w.p. ϵ
  - Tie breaking randomly
- Hyperparameter ϵ controls exploration v.s. exploitation
- Does better long-term (than Greedy) by exploring
- Pessimism v.s. Optimism:
  - Pessimism: Init Q's below observable reward -> Only try one arm (E.g. $Q_0 = -10$)
  - Optimism: Init Q's above observable rewards -> Explore arms at least once (E.g. $Q_0 = 10$)
  - Middle-ground init Q -> Explore arms at most once
  - Pure greedy never **explores** an arm more than once
- Limitations:
  - Exploration and exploitation are too distinct
    - Exploration actions completely blind to promising arms
    - Initialisation tricks only help with "cold start"
  - Exploitation is blind to **confidence** of estimates

Upper Confidence Bound (UCB)

- At round t
  - Estimate value of each arm i as average reward observed
    - $Q_{t-1}(i) = \hat{\mu}_{t-1}(i) + \sqrt{\frac{2\log(t)}{N_{t-1}(i)}}$, if $\Sigma_{s=1}^{t-1} I[i_s = i] > 0$
    - $Q_{t-1}(i) = Q_0$, otherwise
    - Init constant: $Q_0(i) = Q_0$ used until arm i has been pulled
  - $\hat{\mu}_{t-1}(i) = \frac{\Sigma_{s=1}^{t-1} X_i(s) I(i_s=i)}{\Sigma_{s=1}^{t-1} I(i_s=i)}$
  - $N_{t-1}(i) = \Sigma_{s=1}^{t-1} I[i_s = i]$
  - Exploit
    - $i_t \in \arg\max_{i \le i \le k} Q_{t-1}(i)$

- Tie breaking randomly
- (upper bound for **explore** boost)
- Addresses several limitation of ϵ-Greedy
- Can "pause" in a bad arm for a while, but eventually find best
- Results:
  - Quickly overtakes the ϵ-Greedy approaches
  - Continues to outspace on per round rewards for some time
  - More striking when viewed as mean cumulative rewards
- Notes:
  - Theoretical **regret bounds**, optimal up to multiplicative constant
  - Tunable $\rho > 0$ exploration hyperparam, can replace "2"
  - Captures different ϵ rates & bounded rewards outside [0,1]
  - Many variations e.g. different confidence bounds