Advanced Machine Learning Subsidary Notes
Lecture 4: Ensemble Methods

Adam Prügel-Bennett

April 21, 2020

1 Keywords

- Decision Trees, Bagging, Boosting

2 Main Points

2.1 Decision Trees

- Decision trees are binary tree where a set of data is partitioned into two at each node of the tree

- Rule for partition depends on a single feature

- Trees are grown greedily

- All possible rules are tried to find the one that maximise the purity of the leaves

- Gini, entropy or variance used to measure purity

- Decision trees can handle categorical or numerical data

- They can handle missing data

  - after deciding on a new rule (ignoring missing data), a rule to send the missing data right or left is selected to maximise purity

- Decision trees are useful for understanding new data sets

  - By examining the rules at the top of the tree we get to see the most important variables

- Decisions trees are often not competitive

  - they can have high bias because they can only split the data on single variables (limits their expressiveness)

  - they can also have high variance because a small change to the data set that changes a rule at the top of the tree can lead to very different predictions

2.2 Ensembling

- This is used to reduces variance by averaging many different machines

- This only works if the machines make weakly correlated classifications

- We ensemble machines by coming to a consensus

  - Vote on a class in classification

  - Average for regression

2.2.1 Bagging

- Uses bootstrap aggregation

  - that is we sample the training set with replacement to obtain slightly different datasets

- Random Forest

  - Uses decision trees with bootstrap aggregation
  - But also use different random subsets of features
  - Often gives state-of-the-art performance

2.2.2 Boosting

- Boosting constructs a *strong learner* as a weighted sum of *weak learners*

- Adaboost

  - Used for binary decisions
  - Start with a set of weak learners, $\mathcal{W}$
  - Each weak learner $h_i(x)$ outputs ± 1
  - Greedily build the strong learner by adding $\alpha_t \, h_t(x)$ at iteration t
  - Uses an exponential "error" to choose the weak learner and α_t
  - Algorithm does the following
    * Define a weight, w_t^μ, for each training example (x^μ, y^μ)
      · initially these are set to 1
      · Large weight implies the training example is poorly predicted
    * Choose the weak learner, h_t that fails only where prediction is good
      · it decides this by summing the weights of training examples where the weak learner makes an error
      · it choose the weak learner with the smallest sum
    * Choose the parameter α_t to minimises the error
  - Need to understand derivation and resulting algorithm (this is complicated)

- Gradient Boosting

  - Used on regression problems
  - Iterative algorithm where we learn a new weak learner that minimises the residual errors
  - Uses very small decision trees for regression

- Performance of Boosting

  - Can over-fit (use early stopping)
  - Only works for very simple weak-learners (strong learners will over-fit)
  - Can give state-of-the-art performance

3 Exercises

3.1 Compute expected error in mean for correlated variables

- We look at estimating the mean by sampling n random variables

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

- In expectation $\mathbb{E}[X_i] = \mu$

- Thus in expectation $\mathbb{E}[\hat{\mu}] = \mu$

- But there will typically be fluctuations from the mean which we can compute using

$$\sigma_{\hat{\mu}}^2 = \mathbb{E}\left[\left(\hat{\mu} - \mu\right)^2\right]$$

- **Compute this** using

$$\mathbb{E}[(X_i - \mu)^2] = \sigma^2 \qquad\qquad \mathbb{E}[(X_i - \mu)(X_j - \mu)] = \rho\,\sigma^2$$

- Answer given in lecture notes

- Note that the estimated error in the mean is $\sigma_{\hat{\mu}}$ (this is what you use when you compute error-bars)

4 Experiments

4.1 Visualise a decision tree

```
from sklearn.datasets import load_iris
from sklearn import tree
X, y = load_iris(return_X_y=True)
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)

tree.plot_tree(clf.fit(iris.data, iris.target))
```