STAT154 HW6
Bagging, Boosting, and Statistical Learning Theory

Seth Metcalf

14 April 2024

1 Theoretical Exercises

Q1 (Bagging)

- Step 1: Describe how to generate the Bootstrapped dataset $(\mathcal{D}^{(b)})_{b\in[B]}$.

Bootstrapping is random sampling with replacement from the original dataset to create new datasets similar to the original, but randomized. For each iteration, denoted by b, there are randomly sampled examples with replacement from the original dataset which results in B different bootstrapped datasets.

- Step 2: Describe how to define the bagging function $\bar{f}_\lambda^{(B)}$ by aggregation.

For each dataset that was bootstrapped in Step 1:, a classifier, denoted as $\hat{f}_\lambda$, is trained on it to create predictions for unknown data points. An aggregation method is to average the predicted probabilities obtained from each $\hat{f}_\lambda$ so that you could create $\bar{f}_\lambda^{(B)}(\boldsymbol{x})$ which is the average probability of all B classifiers.

- Step 3: Describe how to define the OOB error, and how to use it for model selection (choosing parameter λ).

The OOB (Out of bag) error $\bar{f}_\lambda^{(B)}$ is calculated as the error of the bagging model, the model is the predictions from the classifiers trained on bootstrapped datasets and OOB refers to the data points in the original model that are not included in the bootstrap sample. This error can be used as model selection by trying different values for the λ parameter and using the value which minimizes the OOB error as it provides an estimate of the bootstraps performance compared to the original dataset.

Q2 (Combinatorics in OOB error)

Calculate the following quantities:

- For fixed $i \in [n]$ and $b \in [B]$, the probability that $\boldsymbol{z}_i$ is not contained in $\mathcal{D}^{(b)}$ (denoted $\mathbb{P}(\boldsymbol{z}_i \notin \mathcal{D}^{(b)})$).

Each data point, denoted as $\boldsymbol{z}_i$ has prob $(1-\frac{1}{n})$ of not being selected (prob $\frac{1}{n}$ of being selected). Since the probability is done with replacement, and there are B different iterations, the prob $\mathbb{P}(\boldsymbol{z}_i \notin \mathcal{D}^{(b)}) = (1-\frac{1}{n})^B$

- For fixed $b \in [B]$, the expected size of $\mathcal{D}^{(b)}$ (denoted $n_b = \mathbb{E}[\#\{i \in [n] : \boldsymbol{z}_i \in \mathcal{D}^{(b)}\}]$).

Since there are n elements in each b data, there are B different datasets. Therefore, the size of the total data, $\mathcal{D}$ is the size of n elements $\times$ the amount B, so $n_b = n \times B$.

- For fixed $i \in [n]$, the expected number of datasets containing $\boldsymbol{z}_i$ (denoted $m_i = \mathbb{E}[\#\{b \in [B] : \boldsymbol{z}_i \in \mathcal{D}^{(b)}\}]$).

Since the sets B are drawn with replacement and are independent of one another, the chance that $\boldsymbol{z}_i$ is in any given dataset is $\frac{1}{n}$, the expected number of datasets containing $\boldsymbol{z}_i$ is just as simple as: $m_i = B \times \frac{1}{n}$.

Q3 (Statistical Learning Theory)

- The population risk minimizer $\boldsymbol{\beta}_\star$ (assuming it is unique).

In order to find the population risk minimizer, we derive the population risk function and set it equal to zero. Since the population risk function can be expressed as

$$\boldsymbol{\beta}_\star = \arg\min_{\boldsymbol{\beta}} \mathbb{E}[(y - \boldsymbol{\beta}^T \boldsymbol{\psi}(\boldsymbol{x}))^2]$$

Given the quantities given: $\boldsymbol{\beta}_\star$ can be expressed as:

$$\frac{\partial}{\partial \boldsymbol{\beta}} \mathbb{E}[(y - \boldsymbol{\beta}^T \boldsymbol{\psi}(\boldsymbol{x}))^2] = -2\mathbb{E}[(y - \boldsymbol{\beta}^T \boldsymbol{\psi}(\boldsymbol{x}))\boldsymbol{\psi}(\boldsymbol{x})] = 0$$

Solving for $\boldsymbol{\beta}_\star$:

$$\boldsymbol{\beta}_\star = \frac{\mathbb{E}[\boldsymbol{\psi}(\boldsymbol{x})y]}{\mathbb{E}[\boldsymbol{\psi}(\boldsymbol{x})\boldsymbol{\psi}(\boldsymbol{x})^T]} = K^{-1}h$$

- The empirical risk minimizer $\hat{\boldsymbol{\beta}}_n$ (assuming it is unique).

The empirical risk minimizer is the sol to minimization for this problem, so in this case it is:

$$\hat{\boldsymbol{\beta}_n} = \arg\min_{\boldsymbol{\beta}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{\beta}^T \boldsymbol{\psi}(\boldsymbol{x}_i))^2$$

Similarly as the first problem, if we take the partial derivative with respect to $\boldsymbol{\beta}$:

$$\frac{\partial}{\partial \boldsymbol{\beta}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{\beta}^T \boldsymbol{\psi}(\boldsymbol{x}_i))^2 = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{\beta}^T \boldsymbol{\psi}(\boldsymbol{x}_i))\boldsymbol{\psi}(\boldsymbol{x}_i) = 0$$

Solving for $\boldsymbol{\beta}$:

$$\hat{\boldsymbol{\beta}}_n = \frac{(\frac{1}{n})\sum_{i=1}^{n}\boldsymbol{\psi}(\boldsymbol{x}_i)y_i}{(\frac{1}{n})\sum_{i=1}^{n}\boldsymbol{\psi}(\boldsymbol{x}_i)\boldsymbol{\psi}(\boldsymbol{x}_i)^\mathsf{T}} = \hat{K}^{-1}\hat{h}$$

- The approximation error $R(\langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\beta}_\star \rangle)$.

The approximation error in this case is the expected loss when using the population risk minimizer, therefore,

$$R(\langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\beta}_\star \rangle) = \mathbb{E}[(y - \boldsymbol{\beta}_\star^T \boldsymbol{\psi}(\boldsymbol{x}))^2]$$

If you substitute in for $\boldsymbol{\beta}_\star^T$, you get:

$$= \mathbb{E}[(y - K^{-1}h^T \boldsymbol{\psi}(\boldsymbol{x}))^2]$$

By expanding, you get:

$$= \mathbb{E}[y^2] - 2\mathbb{E}[yK^{-1}h^t \boldsymbol{\psi}(\boldsymbol{x})] + \mathbb{E}[K^{-1}h^t \boldsymbol{\psi}(\boldsymbol{x})K^{-1}h^t \boldsymbol{\psi}(\boldsymbol{x})]$$

Given this, we can now substitute in other quantities and accounting for the fact that K is symmetric:

$$M - 2h^T K^{-1}h + h^T K^{-1}KK^{-1}h = M - h^T K^{-1}h$$

Therefore,

$$R(\langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\beta}_\star \rangle) = M - h^T K^{-1}h$$

- The generalization error $R(\langle \boldsymbol{\psi}(\boldsymbol{x}), \hat{\boldsymbol{\beta}}_n \rangle) - \widehat{R}_n(\langle \boldsymbol{\psi}(\boldsymbol{x}), \hat{\boldsymbol{\beta}}_n \rangle)$.

The generalization error is the diff between the expected loss and the empirical risk, knowing this and based off of our previous calculations we can substitute in:

$$R(\langle \boldsymbol{\psi}(\boldsymbol{x}), \hat{\boldsymbol{\beta}}_n \rangle) - \widehat{R}_n(\langle \boldsymbol{\psi}(\boldsymbol{x}), \hat{\boldsymbol{\beta}}_n \rangle) = \mathbb{E}[(y - (\widehat{K})^{-1} \hat{h}^T \boldsymbol{\psi}(\boldsymbol{x}))^2] - \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{\boldsymbol{\beta}}_n^T \boldsymbol{\psi}(\boldsymbol{x}_i))^2$$

Expanding more:

$$= \mathbb{E}[y^2 - 2y((\widehat{K})^{-1} \hat{h}^T \boldsymbol{\psi}(\boldsymbol{x})) + ((\widehat{K})^{-1} \hat{h}^T \boldsymbol{\psi}(\boldsymbol{x}))^2] - \frac{1}{n} \sum_{i=1}^{n} (y_i ((\widehat{K})^{-1} \hat{h})^T \boldsymbol{\psi}(\boldsymbol{x}))^2$$

$$= M - \frac{2}{n} \sum_{i=1}^{n} \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{x}_i) (\widehat{K})^{-1} h + \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{x}_i) (\widehat{K})^{-1} M (\widehat{K})^{-1} \boldsymbol{\psi}(bx_i)^T \boldsymbol{\psi}(\boldsymbol{x}) - \frac{1}{n} \sum_{i=1}^{n} (y_i - \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{\psi}(\boldsymbol{x}_i)^T (\widehat{K})^{-1} \boldsymbol{\psi}(\boldsymbol{x}_j) y_j)^2$$

Simplifying further we finally get the generalization term:

$$R(\langle \boldsymbol{\psi}(\boldsymbol{x}), \hat{\boldsymbol{\beta}}_n \rangle) - \widehat{R}_n(\langle \boldsymbol{\psi}(\boldsymbol{x}), \hat{\boldsymbol{\beta}}_n \rangle) = M - \frac{2}{n} \sum_{i=1}^{n} \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{x}_i) (\widehat{K})^{-1} h + \frac{1}{n^2} \sum_{i=1}^{n} \sum_{i=j}^{n} (y_i - \boldsymbol{\psi}(\boldsymbol{x}_i)^T (\widehat{K})^{-} 1 \boldsymbol{\psi}(\boldsymbol{x}_j) y_j)^2$$

2 Computational Exercises

Q1 (Implementing Boosting)

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier

# number of models T, dataset X, label y, maximum depth D
def boosted_tree_classifier(T, X, y, D):
    weights = np.ones(len(y)) / len(y)
    classifiers = []
    alphas = []

    # calculating the error, alpha, weights for each model
    for num_models in range(T):
        tree = DecisionTreeClassifier(max_depth=D)
        tree.fit(X, y, sample_weight=weights)
        pred = tree.predict(X)
        error = np.sum(weights * (pred != y)) / np.sum(weights)
        alpha = 0.5 * np.log((1 - error) / error)

        weights *= np.exp(alpha * (pred != y))
        weights /= np.sum(weights)

        classifiers.append(tree)
        alphas.append(alpha)

    def boosted_model(X):
        pred = np.zeros(len(X))

        for i in range(T):
            pred += alphas[i] * classifiers[i].predict(X)
        return np.sign(pred)

    return boosted_model
```

Q2 (Testing your implementation)

As T grows in this case, there is a decrease in training error and a decrease in bias. However, because the model becomes more complex, this ill lead to an increase in variance due to the model overfitting the training data.

Furthermore, after a certain point, there are no improvements in performance, it jkust begins to overfit.