Project 2 - Statistical learning

August Jonasson

December 17, 2024

Task 1

(a)

In this first task we want to show that the gradient of the loss function that uses deviance is given, component-wise, by

$$I(y_i = G_k) - p_k(x_i)$$

(see Hastie, Tibshirani, and Friedman 2009, Table 10.2), where I is the class G_k indicator variable and p_k is the probability of an observation belonging to class G_k.

Using the K-class multinomial deviance (Hastie, Tibshirani, and Friedman 2009, eq. (10.22)), we have that the loss function is given by

$$\begin{aligned} L(y, p(x_i)) &= -\sum_{k=1}^{K} I(y_i = G_k) \log p_k(x_i) \\ &= -\sum_{k=1}^{K} I(y_i = G_k) f_k(x_i) + \log\left(\sum_{l=1}^{K} e^{f_l(x_i)}\right). \end{aligned}$$

The component-wise gradient now evaluates to

$$-\frac{\partial L(y, p_k(x_i))}{\partial f_k(x_i)} = I(y_i = G_k) - \frac{e^{f_k(x_i)}}{\sum_{l=1}^{K} e^{f_l(x_i)}},$$

but the latter term of this right-hand side is exactly how we, under the task of classification, would model the probability $p_k(x_i)$. Hence, this expression matches what we wanted to show and we are done.

(b)

For this task, we want to show that the variance of the average of a sample of B identically distributed, but not necessarily independent, variables is given by

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2,$$

(see Hastie, Tibshirani, and Friedman 2009, eq. 15.1) where ρ denotes the pairwise *positive* correlation and σ^2 the variance of each variable. We then want to examine further why this appears to fail when the pairwise correlation is negative and diagnose the problem.

Let $(X_i, \ldots, X_B)$ be our (non-independent) sample. The variance of the sample average is then given by

$$\begin{aligned}
\text{Var}\left(\frac{1}{B^2}\sum_{i=1}^{B}\right) &= \frac{1}{B^2}\left(\sum_{i=1}^{B}\text{Var}(X_i) + \sum_{i\neq j}^{B}\text{Cov}(X_i, X_j)\right) \\
&= \frac{1}{B^2}\left(B\sigma^2 + (B^2 - B)\sigma^2\rho\right) \\
&= \frac{\sigma^2}{B}\left(1 + B\rho - \rho\right) \\
&= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2,
\end{aligned}$$

and we are done with the first part of the task. For the second part, we can start by stating that since the above is a variance, it holds that

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 > 0,$$

which for $B > 1$ implies that

$$\frac{B}{B(1-B)} - \rho < 0, \qquad \text{or} \qquad \frac{1}{1-B} < \rho.$$

It appears that as B grows larger, the negative lower bound on the correlation ρ comes closer and closer to zero. Hence, the variance expression above doesn't necessarily fail as soon as the correlation is negative, but as the sample size increases, only barely negative correlations are allowed (practically uncorrelated), and even for the smallest $B = 2$ the correlation cannot go lower than -0.5.

(c)

In this task, we want to show that the sampling correlation $corr(\bar{x}_1^*, \bar{x}_2^*)$ - where $\bar{x}_i^*$ denote bootstrap realizations of the sample mean of an iid sample $(x_1, \ldots, x_N)$ - is approximately equal to 50 %. We also want to derive $\text{Var}(\bar{x}_1^*)$ and

$$\text{Var}(\bar{x}_{bag}) = \text{Var}\left(\frac{1}{2}\left(\bar{x}_1^* + \bar{x}_2^*\right)\right)$$

along the way.

We start by denoting the two bootstrap samples as

$$\{x_1^{(i)} : i = 1, \ldots, n\} \qquad \text{and} \qquad \{x_2^{(i)} : i = 1, \ldots, n\},$$

where n is the size of the bootstrap samples. It follows from our initial sample, that

$$\text{Var}(x_1^{(i)}) = \text{Var}(x_2^{(i)}) = \sigma^2,$$

and that

$$\text{Cov}(x_1^{(i)}, x_2^{(j)}) = \frac{\sigma^2}{n},$$

for all i, j. We can now calculate one of the variances as

$$\begin{aligned}
\text{Var}(\bar{x}_1^*) &= \frac{1}{n^2}\left(\sum_{i=1}^{n}\text{Var}(x_1^{(i)}) + \sum_{i\neq j}^{n}\text{Cov}(x_1^{(i)}, x_1^{(j)})\right) \\
&= \frac{1}{n^2}\left(n\sigma^2 + (n^2-n)\frac{\sigma^2}{n}\right) = \ldots = \\
&= \frac{\sigma^2(2n-1)}{n^2},
\end{aligned}$$

and for the covariance between the two bootstrap averages, it follows rather simply that

$$\text{Cov}(\bar{x}_1^*, \bar{x}_2^*) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\text{Cov}(x_1^{(i)}, x_2^{(j)}) = \frac{\sigma^2}{n},$$

as the n^2 cancels out. The desired correlation between the two bootstrap averages can now be calculated as

$$\begin{aligned}
\text{Corr}(\bar{x}_1^*, \bar{x}_2^*) &= \frac{\text{Cov}(\bar{x}_1^*, \bar{x}_2^*)}{\text{Var}(\bar{x}_1^*)} \\
&= \frac{\sigma^2}{n}\frac{n^2}{\sigma^2(2n-1)} \\
&= \frac{n}{2n-1} \to \frac{1}{2},
\end{aligned}$$

as $n \to \infty$, which is what we wanted to show. Finally, the variance of the bagged mean is

$$\begin{aligned}
\text{Var}(\bar{x}_{bag}) &= \frac{1}{4}\left(\sum_{i=1}^{2}\text{Var}(\bar{x}_i^*) + \sum_{i\neq j}\text{Cov}(\bar{x}_i^*, \bar{x}_j^*)\right) \\
&= \frac{2}{4}\left(\frac{\sigma^2(2n-1)}{n^2} + \frac{\sigma^2}{n}\right) \\
&= \frac{\sigma^2(3n-1)}{2n^2},
\end{aligned}$$

and we are done.

(d)

In the construction of random forests, the hyper-parameter m, indicating how many of the features to randomly sample in each split, has to be fixed. In this task we will give a brief answer of how selecting m big or small affects the bias and/or the variance.

Smaller m reasonably leads to lower correlation (as fewer features overlap), which, according to the equation we showed in task **(b)** lowers the total variance as well. Larger m also very reasonably leads to lower bias as the chance of selecting the truly important features becomes higher. Our aim is to choose m sufficiently small such that the correlation is decreased, without the bias increasing too much.

Task 2

This task is about exploring the use and performance of gradient boosting trees for classification in practice. We are going to do this using Python, and the spam email data set[1].

(a)

We will now give a brief overview of the python function of our choice, which loss function we are going to use and why it is suitable, and list and explain the choices of all of the parameters and settings required to train the boosting trees.

Function of our choice: `GradientBoostingClassifier` from the `sklearn.ensemble` library [2]. In Table 1 are all of the settings and parameters we have used, directly reflected by the code in the appendix.

Table 1: Explanation of GradientBoostingClassifier Parameters

Parameter	**Value**	**Explanation**
`loss`	`'log_loss'`	Specifies the loss function. For binary classification, log-loss (binomial deviance) minimizes classification errors by aligning with logistic regression.
`n_estimators`	`default:100`	Number of boosting iterations (trees). Higher values increase accuracy but also computational cost. This is what is referred to as M in later tasks.
`max_depth`	`default:10`	Limits the maximum depth of each tree, controlling complexity and reducing overfitting. Larger values allow deeper, more complex splits.
`max_leaf_nodes`	`default:5`	Restricts the number of leaf nodes per tree, helping maintain simple trees and prevent overfitting. Referred to as m in later tasks (not to be confused with m from previous tasks).
`max_features`	`'sqrt'`	The number of features considered at each split is the square root of the total features. Refer to Hastie, Tibshirani, and Friedman 2009 ch. 15.2 for more info.
`learning_rate`	`default:0.1`	Shrinks the contribution of each tree. A lower value combined with a larger `n_estimators` often improves performance.
`subsample`	`1.0`	Fraction of samples used to fit each tree. Setting values like `0.8` can help prevent overfitting by introducing randomness.
`verbose`	`0`	Controls the verbosity during training. `0` means silent, `1` or `2` displays progress during training.
`random_state`	`0`	Ensures reproducibility of the results by fixing the random seed. Useful for consistent performance evaluation.

[1] https://hastie.su.domains/ElemStatLearn/datasets/spam.data
[2] https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

(b)

This second task is purely about writing the code to construct some simple gradient boosted trees. See code appendix at the end of the document.

(c)

This task is all about creating the plot that can be seen in Figure 1. Refer to the next task for interpretations.

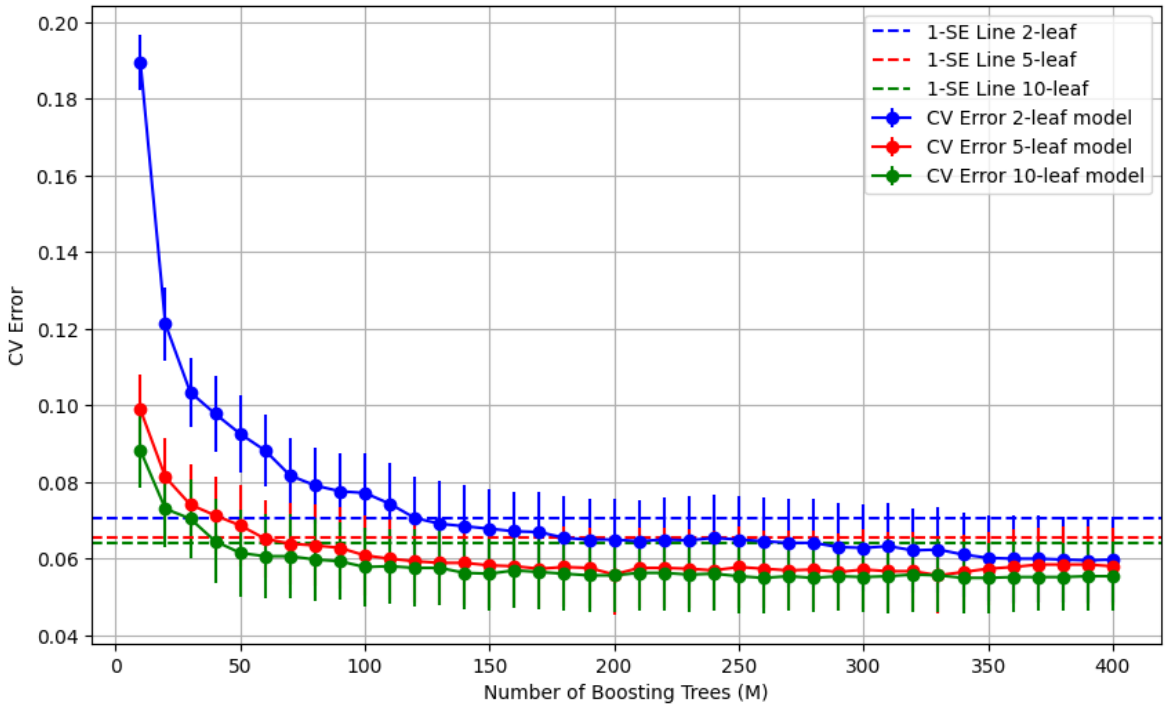

Figure 1: Cross-validation error as a function of number of boosting trees for three different configurations of maximal number of leaves (terminal nodes).

(d)

Finally, we are to round off this project by discussing the performance of the boosting trees in relation to different values on m (maximum number of terminal nodes) and M (number of boosting trees) - especially on the connection to the bias-variance trade-off.

As can be seen from the blue line in Figure 1, the model using the lowest m also takes the longest time (highest M) to converge in its training, i.e. it has the highest bias overall. Equivalently, the green line, which has the highest m seems to converge the fastest - its bias is lower. On the other hand, higher m also means higher complexity, which means more prone to overfitting - high variance. In all

cases, having M too large would risk leading to overfitting. As such it is very important to monitor this training curve.

In summary, m and M have to be balanced together.

- Small m and large M: may take the longest to train but less prone to overfitting.
- Large m and small M: converges fast but prone to overfitting.

Python Code

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import GradientBoostingClassifier

# loading the data
spam_df = pd.read_csv("spam_data.txt", header=None, sep=' ')

# separating data into predictors X and labels y
X, y = np.array(spam_df.iloc[:, :-1]), np.array(spam_df.iloc[:, -1])

# function for constructing gradient boosted trees
def gradientBoostingClassifier(max_leaf_nodes = 5, max_depth = 10, verbose = 0, n_
    estimators = 100):
    """
    Gradient boosted tree classifier using the scikit-learn function
    GradientBoostingClassifier.

    The log_loss (binomial deviance) function is used as loss function.
    The number of features randomly selected in each split is sqrt(total # features)

    n_estimators is the effective number of trees and this is the parameter that we
    are going to tune, primarily.
    """
    model = GradientBoostingClassifier(
        loss = 'log_loss',
        max_features = 'sqrt',
        max_depth = max_depth,
        max_leaf_nodes =  max_leaf_nodes,
        n_estimators = n_estimators,
        verbose = verbose,
        random_state=0
    )
    return model


# examining three models where max #leaf nodes = 2, 5, 10

# specifying the range of number of trees to examine
M_values = range(10, 410, 10)
cv_errors_1 = []  # Store mean CV errors for first model
std_errors_1 = []  # Store CV standard errors of first model

cv_errors_2 = [] # second model ...
std_errors_2 = []

cv_errors_3 = [] # third model ...
std_errors_3 = []

# Perform 10-fold CV for each M
for M in M_values:

    # first model cross validation errors
    model_1 = gradientBoostingClassifier(n_estimators=M, max_leaf_nodes=2) # 2-leaf
    model
```

```
    scores = cross_val_score(model_1, X, y, cv=10, scoring='accuracy')
    mean_error = 1 - np.mean(scores)
    std_error = np.std(scores) / np.sqrt(len(scores))
    cv_errors_1.append(mean_error)
    std_errors_1.append(std_error)

    # second model ...
    model_2 = gradientBoostingClassifier(n_estimators=M, max_leaf_nodes=5) # 5-leaf
    model
    scores = cross_val_score(model_2, X, y, cv=10, scoring='accuracy')
    mean_error = 1 - np.mean(scores)  # CV error
    std_error = np.std(scores) / np.sqrt(len(scores))
    cv_errors_2.append(mean_error)
    std_errors_2.append(std_error)

    # third model ...
    model_3 = gradientBoostingClassifier(n_estimators=M, max_leaf_nodes=10) # 10-leaf
    model
    scores = cross_val_score(model_3, X, y, cv=10, scoring='accuracy')
    mean_error = 1 - np.mean(scores)  # CV error
    std_error = np.std(scores) / np.sqrt(len(scores))
    cv_errors_3.append(mean_error)
    std_errors_3.append(std_error)


# and plotting the results
# Plot CV error versus number of boosting trees M
plt.figure(figsize=(10, 6))
plt.errorbar(M_values, cv_errors_1, yerr=std_errors_1, fmt='-o', label='CV Error 2-
    leaf model', color='blue')
plt.errorbar(M_values, cv_errors_2, yerr=std_errors_2, fmt='-o', label='CV Error 5-
    leaf model', color = 'red')
plt.errorbar(M_values, cv_errors_3, yerr=std_errors_3, fmt='-o', label='CV Error 10-
    leaf model', color = 'green')
plt.xlabel('Number of Boosting Trees (M)')
plt.ylabel('CV Error')
# plt.title('CV Error vs. Number of Boosting Trees')
plt.axhline(min(cv_errors_1) + std_errors_1[np.argmin(cv_errors_1)], color='blue',
    linestyle='--', label='1-SE Line 2-leaf')
plt.axhline(min(cv_errors_2) + std_errors_2[np.argmin(cv_errors_2)], color='red',
    linestyle='--', label='1-SE Line 5-leaf')
plt.axhline(min(cv_errors_3) + std_errors_3[np.argmin(cv_errors_3)], color='green',
    linestyle='--', label='1-SE Line 10-leaf')
plt.legend()
plt.grid()
plt.show()
```

References

Hastie, T., R. Tibshirani, and J.H. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer. ISBN: 9780387848846. URL: https://books.google.se/books?id=eBSgoAEACAAJ.