Agreement: This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

Q1

(a)

Proof:

$$\begin{aligned}&\mathbb{E}_{Y\sim D(x)} - \log\left(1+e^{-2Yf(x)}\right)\\ &\quad= P(Y=1|x)\left(-\log\left(1+e^{-2f(x)}\right)\right) + P(Y=-1|x)\left(-\log\left(1+e^{2f(x)}\right)\right)\\ &\quad= -P(Y=1|x)\log\left(1+e^{-2f(x)}\right) - P(Y=-1|x)\log\left(1+e^{2f(x)}\right)\end{aligned}$$

Where log is natural log (ln).

$$0 = \frac{d\mathbb{E}_{Y\sim D(x)} - \log\left(1+e^{-2Yf(x)}\right)}{df(x)} = -P(Y=1|x)\frac{-2e^{-2f(x)}}{1+e^{-2f(x)}} - P(Y=-1|x)\frac{2e^{2f(x)}}{1+e^{2f(x)}}$$

$$0 = -P(Y=1|x)\frac{-2e^{-2f(x)}}{1+e^{-2f(x)}} - P(Y=-1|x)\frac{2e^{2f(x)}}{1+e^{2f(x)}}$$

$$P(Y=1|x)\frac{2e^{-2f(x)}}{1+e^{-2f(x)}} = P(Y=-1|x)\frac{2e^{2f(x)}}{1+e^{2f(x)}}$$

$$\frac{P(Y=1|x)}{P(Y=-1|x)} = \frac{2e^{2f(x)}(1+e^{-2f(x)})}{2e^{-2f(x)}(1+e^{2f(x)})} = \frac{e^{2f(x)}(e^{2f(x)}+1)}{(1+e^{2f(x)})} = e^{2f(x)}$$

$$\boldsymbol{f^*(x) = \frac{1}{2}\ln\left(\frac{P(Y=1|x)}{P(Y=-1|x)}\right)}$$

(b)

Proof:

Given $Y_k^{(c)} = \begin{cases} 1 & c=k \\ -\frac{1}{K-1} & c\neq k \end{cases}$,

$$\mathbb{E}\left[\exp\left(-\frac{1}{K}\sum_{k=1}^{K} Y_k^{(c)} f_k\right)\right] = \mathbb{E}\left[\exp\left(-\frac{1}{K}\left(f_{k(k=c)} - \frac{\sum_{k\neq c} f_k}{K-1}\right)\right)\right]$$

Because $\sum_{k=1}^{K} f_k = 0$,

$$\sum_{k=1}^{K} f_k = f_{k(k=c)} + \sum_{k\neq c} f_k = 0 \Rightarrow f_{k(k=c)} = -\sum_{k\neq c} f_k$$

Therefore,

$$\mathbb{E}\left[\exp\left(-\frac{1}{K}\sum_{k=1}^{K} Y_k^{(c)} f_k\right)\right] = \mathbb{E}\left[\exp\left(-\frac{1}{K}\left(f_{k(k=c)} + \frac{f_{k(k=c)}}{K-1}\right)\right)\right]$$

$$\mathbb{E}\left[\exp\left(-\frac{1}{K}\sum_{k=1}^{K} Y_k^{(c)} f_k\right)\right] = \mathbb{E}\left[\exp\left(-\frac{1}{K} f_{k(k=c)} - \frac{1}{K}\frac{f_{k(k=c)}}{K-1}\right)\right]$$

$$\mathbb{E}\left[\exp\left(-\frac{1}{K}\sum_{k=1}^{K} Y_k^{(c)} f_k\right)\right] = \mathbb{E}\left[\exp\left(-\frac{1}{K} f_{k(k=c)} - \frac{f_{k(k=c)}}{K-1} + \frac{f_{k(k=c)}}{K}\right)\right]$$

$$\mathbb{E}\left[\exp\left(-\frac{1}{K}\sum_{k=1}^{K} Y_k^{(c)} f_k\right)\right] = \mathbb{E}\left[\exp\left(-\frac{f_{k(k=c)}}{K-1}\right)\right]$$

$$\mathbb{E}\left[\boldsymbol{\exp\left(-\frac{1}{K}\sum_{k=1}^{K} Y_k^{(c)} f_k\right)}\right] = \mathbb{E}\left[\boldsymbol{\exp\left(-\frac{1}{K-1} f_k\right)}\right]$$

(c)

The Lagrangian function is:

$$\mathcal{L} = \mathbb{E}\left[\mathrm{e}^{-\frac{1}{K-1}f_k(x)}\right] - \lambda\sum_{k'=1}^{K} f_{k'}(x)$$

Rewrite the Lagrangian function:

$$\mathcal{L} = P\left(Y = Y^{(1)}\middle|x\right)\mathrm{e}^{-\frac{1}{K-1}f_1(x)} + \cdots + P\left(Y = Y^{(K)}\middle|x\right)\mathrm{e}^{-\frac{1}{K-1}f_K(x)} - \lambda\left(f_1(x) + \cdots + f_K(x)\right)$$

Take derivatives:

$$\frac{d\mathcal{L}}{df_1(x)} = -\frac{1}{K-1}P\left(Y = Y^{(1)}\middle|x\right)e^{-\frac{1}{K-1}f_1(x)} - \lambda = 0 \qquad (1.1)$$

$$\vdots$$

$$\frac{d\mathcal{L}}{df_k(x)} = -\frac{1}{K-1}P\big(Y = Y^{(k)}\big|x\big)e^{-\frac{1}{K-1}f_k(x)} - \lambda = 0 \qquad (1.2)$$

$$\vdots$$

$$\frac{d\mathcal{L}}{df_K(x)} = -\frac{1}{K-1}P\big(Y = Y^{(K)}\big|x\big)e^{-\frac{1}{K-1}f_K(x)} - \lambda = 0 \qquad (1.3)$$

$$\frac{d\mathcal{L}}{d\lambda} = -\big(f_1(x) + \cdots + f_k(x) + \cdots + f_K(x)\big) = 0 \qquad (2)$$

From (1.1), (1.2) and (1.3), we can derive that:

$$P\big(Y = Y^{(1)}\big|x\big)e^{-\frac{1}{K-1}f_1(x)} = \cdots = P\big(Y = Y^{(k)}\big|x\big)e^{-\frac{1}{K-1}f_k(x)} = \cdots = P\big(Y = Y^{(K)}\big|x\big)e^{-\frac{1}{K-1}f_K(x)}$$

$$lnP\big(Y = Y^{(1)}\big|x\big) - \frac{1}{K-1}f_1(x) = \cdots = lnP\big(Y = Y^{(k)}\big|x\big) - \frac{1}{K-1}f_k(x)$$

$$= \cdots = lnP\big(Y = Y^{(K)}\big|x\big) - \frac{1}{K-1}f_K(x)$$

Therefore,

$$f_1(x) = (K-1)\big[lnP\big(Y = Y^{(1)}\big|x\big) - lnP\big(Y = Y^{(k)}\big|x\big)\big] + f_k(x)$$

$$f_2(x) = (K-1)\big[lnP\big(Y = Y^{(2)}\big|x\big) - lnP\big(Y = Y^{(k)}\big|x\big)\big] + f_k(x)$$

$$\vdots$$

$$f_K(x) = (K-1)\big[lnP\big(Y = Y^{(K)}\big|x\big) - lnP\big(Y = Y^{(k)}\big|x\big)\big] + f_k(x)$$

Bring $f_1(x)$, $f_2(x)$, ..., $f_K(x)$ into (2), we get:

$$(K-1)\big[lnP\big(Y = Y^{(1)}\big|x\big) - lnP\big(Y = Y^{(k)}\big|x\big)\big] + f_k(x)$$

$$+ (K-1)\big[lnP\big(Y = Y^{(2)}\big|x\big) - lnP\big(Y = Y^{(k)}\big|x\big)\big] + f_k(x) + \cdots$$

$$+ (K-1)\big[lnP\big(Y = Y^{(K)}\big|x\big) - lnP\big(Y = Y^{(k)}\big|x\big)\big] + f_k(x) = 0$$

Simplify the equation above:

$$(K-1)\left[\sum_{k'=1}^{K} lnP\left(Y=Y^{(k')}\middle|x\right)-KlnP\left(Y=Y^{(k)}\middle|x\right)\right]+Kf_k(x)=0$$

Therefore,

$$\boldsymbol{f_k^*(x)=(K-1)\left[lnP\left(Y=Y^{(k)}\middle|x\right)-\frac{1}{K}\sum_{k'=1}^{K} lnP\left(Y=Y^{(k')}\middle|x\right)\right]} \quad (3)$$

Where $k=1,2,\dots,K$.

From (3), we can derive:

$$e^{\frac{f_k^*(x)}{K-1}}=\frac{P\left(Y=Y^{(k)}\middle|x\right)}{\sqrt[K]{\prod_{k'=1}^{K}P\left(Y=Y^{(k')}\middle|x\right)}}$$

Therefore,

$$\sqrt[K]{\prod_{k'=1}^{K}P\left(Y=Y^{(k')}\middle|x\right)}=\frac{P\left(Y=Y^{(1)}\middle|x\right)}{e^{\frac{f_1^*(x)}{K-1}}}=\cdots=\frac{P\left(Y=Y^{(k)}\middle|x\right)}{e^{\frac{f_k^*(x)}{K-1}}}=\cdots=\frac{P\left(Y=Y^{(K)}\middle|x\right)}{e^{\frac{f_K^*(x)}{K-1}}}$$

Therefore,

$$P\left(Y=Y^{(1)}\middle|x\right)=\frac{e^{\frac{f_1^*(x)}{K-1}}}{e^{\frac{f_k^*(x)}{K-1}}}P\left(Y=Y^{(k)}\middle|x\right)$$

$$P\left(Y=Y^{(2)}\middle|x\right)=\frac{e^{\frac{f_2^*(x)}{K-1}}}{e^{\frac{f_k^*(x)}{K-1}}}P\left(Y=Y^{(k)}\middle|x\right)$$

$$\vdots$$

$$P\left(Y=Y^{(K)}\middle|x\right)=\frac{e^{\frac{f_K^*(x)}{K-1}}}{e^{\frac{f_k^*(x)}{K-1}}}P\left(Y=Y^{(k)}\middle|x\right)$$

Because $P\left(Y=Y^{(1)}\middle|x\right)+P\left(Y=Y^{(2)}\middle|x\right)+\cdots+P\left(Y=Y^{(K)}\middle|x\right)=1$,

$$\frac{e^{\frac{f_1^*(x)}{K-1}}}{e^{\frac{f_k^*(x)}{K-1}}} P\left(Y = Y^{(k)}\middle|x\right) + \frac{e^{\frac{f_2^*(x)}{K-1}}}{e^{\frac{f_k^*(x)}{K-1}}} P\left(Y = Y^{(k)}\middle|x\right) + \cdots + \frac{e^{\frac{f_K^*(x)}{K-1}}}{e^{\frac{f_k^*(x)}{K-1}}} P\left(Y = Y^{(k)}\middle|x\right) = 1$$

Simplify the equation above:

$$\frac{\sum_{k'=1}^{K} e^{\frac{f_{k'}^*(x)}{K-1}}}{e^{\frac{f_k^*(x)}{K-1}}} P\left(Y = Y^{(k)}\middle|x\right) = 1$$

Therefore,

$$\boldsymbol{P\left(Y = Y^{(k)}\middle|x\right) = \frac{e^{\frac{f_k^*(x)}{K-1}}}{\sum_{k'=1}^{K} e^{\frac{f_{k'}^*(x)}{K-1}}}}$$

Where $k = 1, 2, \ldots, K$.

Q2

(a)

First,

$$Var(\bar{X}) = Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2} Var\left(\sum_{i=1}^{n} X_i\right)$$

$$Var(\bar{X}) = \frac{1}{n^2}\sum_{i=1}^{n} Var(X_i) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j\neq i}^{n} Cov(X_i, X_j)$$

Given that all decision trees have a binary output variance of σ^2, the correlation ρ_{ij} between two different trees can be derived as below.

$$\rho_{ij} = \frac{Cov(X_i, X_j)}{\sqrt{Var(X_i)Var(X_j)}} = \frac{Cov(X_i, X_j)}{\sigma^2}$$

Therefore,

$$Cov(X_i, X_j) = \sigma^2 \rho_{ij}$$

$$Var(\bar{X}) = \frac{1}{n^2} n\sigma^2 + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j\neq i}^{n} \sigma^2 \rho_{ij}$$

$$Var(\bar{X}) = \frac{\sigma^2}{n} + \frac{\sigma^2}{n^2}\sum_{i=1}^{n}\sum_{j\neq i}^{n} \rho_{ij}$$

The average correlation $\bar{\rho}$ is defined:

$$\bar{\rho} = \frac{1}{n(n-1)}\sum_{i=1}^{n}\sum_{j\neq i}^{n} \rho_{ij}$$

Therefore, $Var(\bar{X})$ can be rewritten as:

$$Var(\bar{X}) = \frac{\sigma^2}{n} + \frac{\sigma^2}{n^2} n(n-1)\bar{\rho}$$

$$Var(\bar{X}) = \frac{\sigma^2}{n} + \frac{\sigma^2}{n} (n-1)\bar{\rho}$$

For Geoff, $\boldsymbol{\bar{\rho} = \rho_1,\ Var_{Geoff}(\bar{X}) = \frac{\sigma^2}{n} + \frac{\sigma^2}{n}(n-1)\rho_1.}$

For Carina, $\boldsymbol{\bar{\rho} = \rho_2,\ Var_{\text{Carina}}(\bar{X}) = \frac{\sigma^2}{n} + \frac{\sigma^2}{n}(n-1)\rho_2.}$

Because $\boldsymbol{\rho_1 < \rho_2,\ Var_{Geoff}(\bar{X}) < Var_{\text{Carina}}(\bar{X}).}$

Second,

Applying Chebyshev's Inequality, we get:

$$P(|\bar{X} - p| \geq \epsilon) \leq \frac{Var(\bar{X})}{\epsilon^2}$$

$$P(|\bar{X} - p| \geq \epsilon) \leq \frac{\sigma^2 + \sigma^2(n-1)\bar{\rho}}{n\epsilon^2}$$

For Geoff,

$$\boldsymbol{P_{\text{Geoff}}(|\bar{X} - p| \geq \epsilon) \leq \frac{\sigma^2 + \sigma^2(n-1)\rho_1}{n\epsilon^2}}$$

For Carina,

$$\boldsymbol{P_{\text{Carina}}(|\bar{X} - p| \geq \epsilon) \leq \frac{\sigma^2 + \sigma^2(n-1)\rho_2}{n\epsilon^2}}$$

When $n \to \infty$,

$$P_{\text{Geoff}}(|\bar{X} - p| \geq \epsilon) \leq \frac{\sigma^2\rho_1}{\epsilon^2}$$

$$P_{\text{Carina}}(|\bar{X} - p| \geq \epsilon) \leq \frac{\sigma^2\rho_2}{\epsilon^2}$$

Because $\boldsymbol{\rho_1}$ **is much smaller than** $\boldsymbol{\rho_2,\ P_{\text{Geoff}}(|\bar{X} - p| \geq \epsilon)}$ **will be**

smaller than $P_{\text{Carina}}(|\bar{X} - p| \geq \epsilon)$, which means Geoff's model is more stable than Carina's. Therefore, Geoff will be more likely to have a better random forest model.

When having a model with near-zero correlation, $\bar{\rho} \approx 0$, and $n \to \infty$, the probability that the aggregated result of the forest deviates from the expected value by more than a small positive value ϵ become:

$$P(|\bar{X} - p| \geq \epsilon) \leq \frac{\sigma^2 \bar{\rho}}{\epsilon^2} \approx 0$$

Thus, $P(|\bar{X} - p| < \epsilon) \approx 1$, which is exactly what we want. Therefore, having a model with near-zero correlation will increase the stability of the model.

In addition, according to the law of large numbers (LLN), the average output of the trees $\bar{X}$ converge to the expected accuracy p when n becomes very big. Therefore, the larger n is, the more stable the model is.

(b)

1. For KNN:

The volume of a d-dimensional unit hypercube is $V_{total} = 1$, and the volume of a hypersphere within r_k is $V_k = C_d r_k^d$. Therefore,

$$k = N \frac{V_k}{V_{total}} = N C_d r_k^d$$

$$r_k = \sqrt[d]{\frac{k}{NC_d}}$$

2. For Random Forests:

For a single draw, the probability that a given point is chosen is $\frac{1}{N}$. Therefore, the probability that a given point is not chosen in a single draw is $1 - \frac{1}{N}$. Because the size of the bootstrapped sample is the same size as the original dataset, N, the probability that a given point is not chosen in N draws is $(1 - \frac{1}{N})^N$. In contrast, the probability that a given point is chosen at least once is $1 - (1 - \frac{1}{N})^N$. Therefore, the expected number of unique samples $E[unique]$ can be derived as below:

$$\boldsymbol{E[unique] = N\left(1 - \left(1 - \frac{1}{N}\right)^N\right)}$$

3. Direct Comparison:

When the dimension d grows to big numbers, holding all else constant and $NC_d > k$, the expected radius r_k will increase, approaching the edge of the d-dimensional unit hypercube. This indicates that in KNN, all points in high dimensional space tend to become equidistant, which reduces the discriminative power of KNN. Therefore, KNN is less effective when it is under high-dimensional cases.

When N is very large, given that $\left(1-\frac{1}{N}\right)^N \approx \frac{1}{e}$, the percent of data will be represented in a bootstrapped sample can be derived as below:

$$\boldsymbol{percent} = \frac{\boldsymbol{E[unique]}}{\boldsymbol{N}} \times \mathbf{100\%} = \frac{\boldsymbol{N}\left(\mathbf{1}-\left(\mathbf{1}-\frac{\mathbf{1}}{\boldsymbol{N}}\right)^{\boldsymbol{N}}\right)}{\boldsymbol{N}} \times \mathbf{100\%}$$

$$= \left(\mathbf{1}-\frac{\mathbf{1}}{\boldsymbol{e}}\right) \times \mathbf{100\%} = \mathbf{63.2\%}$$

When N is very large, the proportion of the entire data space N that is covered by $E[unique]$ points $V_{RF} = \frac{E[unique]}{N} \approx 1 - \frac{1}{e}$. When d is large, the volume covered by KNN, V_k will approach the volume of the d-dimensional unit hypercube. So, in this case, $V_k \approx 1$. Therefore, the ratio between V_{RF} and V_k can be derived as:

$$\frac{\boldsymbol{V_{RF}}}{\boldsymbol{V_k}} = \frac{\mathbf{1}-\frac{\mathbf{1}}{\boldsymbol{e}}}{\mathbf{1}} = \mathbf{1}-\frac{\mathbf{1}}{\boldsymbol{e}} < \mathbf{1}$$

Therefore, when both the number of samples N and the dimension d are very large, V_{RF} will be smaller than V_k. This indicates that the random forest is more robust than KNN in high-dimensional situations.

HW3_Q3

October 24, 2023

1 Q3 Variable Importance for Trees and Random Forests

1.1 Initialization

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.utils import resample
import collections

# read data from csv file
train_data = pd.read_csv("Variable Importance Data/train.csv")
test_data = pd.read_csv("Variable Importance Data/test.csv")
```

1.2 (a)

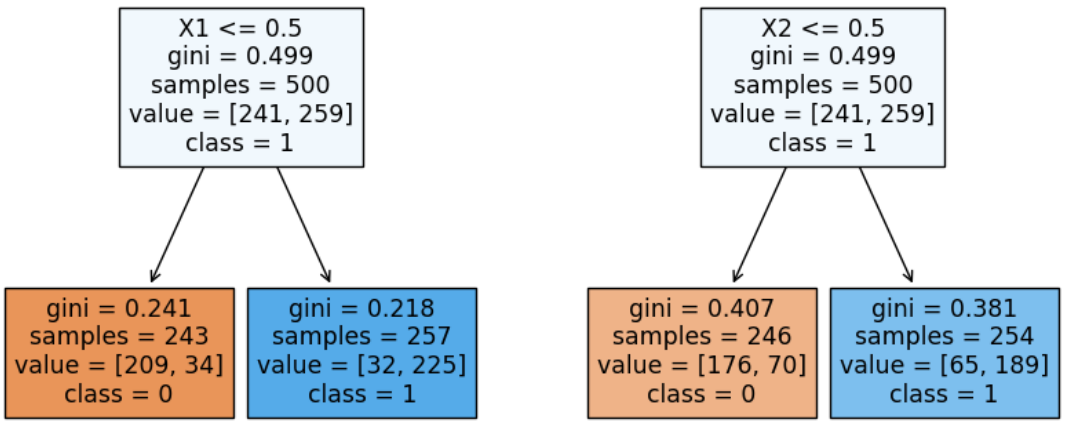
[2]:
```
features = ['X1', 'X2', 'X3', 'X4', 'X5']
train_x = train_data[features]
train_y = train_data.Y

# build the decision stump based on the best split
model_split = DecisionTreeClassifier(criterion='gini', splitter='best',␣
 ↪max_depth=1)
model_split.fit(train_x, train_y)
best_split_feature = train_x.columns[model_split.tree_.feature[0]]

# find the best surrogate split feature
max_lamb = -100
best_surrogate_split_feature = ''
for feature in features:
    if feature == best_split_feature:
        continue
    # calculate the lambda
    pL, pR, pLL, pRR = 0, 0, 0, 0
    for i in range(len(train_x)):
        if train_x[best_split_feature][i] == 0: pL += 1 / len(train_x)
        if train_x[best_split_feature][i] == 1: pR += 1 / len(train_x)
```

```
        if train_x[best_split_feature][i] == 0 and train_x[feature][i] == 0:␣
 ↪pLL += 1 / len(train_x)
        if train_x[best_split_feature][i] == 1 and train_x[feature][i] == 1:␣
 ↪pRR += 1 / len(train_x)
    lamb = (min(pL,pR) - (1 - pLL - pRR)) / min(pL,pR)
    # print(lamb)
    if max_lamb < lamb:
        max_lamb = lamb
        best_surrogate_split_feature = feature

# build the decision stump based on the best surrogate split
model_surrogate_split = DecisionTreeClassifier(criterion='gini',␣
 ↪splitter='best', max_depth=1)
model_surrogate_split.fit(train_x[[best_surrogate_split_feature]], train_y);
```

1.2.1 (i)

[3]:
```
print("The left one is the decision stump based on the best split",␣
 ↪best_split_feature)
print("The right one is the decision stump based on the best surrogate split",␣
 ↪best_surrogate_split_feature)

# draw the decision stump based on the best split
plt.figure(figsize=(10,5))
plt.subplot(121)
_ = plot_tree(
model_split,
feature_names=[best_split_feature],
class_names=['0', '1'], # Y
filled=True
)

# draw the decision stump based on the best surrogate split
plt.subplot(122)
_ = plot_tree(
model_surrogate_split,
feature_names=[best_surrogate_split_feature],
class_names=['0', '1'], # Y
filled=True
)
```

```
The left one is the decision stump based on the best split X1
The right one is the decision stump based on the best surrogate split X2
```



1.2.2 (ii)

[4]:
```
# calculate the gini impurity reduction
def impurity_reduction(model):
    root_impurity = model.tree_.impurity[0]
    child_impurity_l = model.tree_.impurity[1]
    child_impurity_r = model.tree_.impurity[2]
    root_sample_num = np.sum(model.tree_.value, -1)[0][0]
    child_sample_num_l = np.sum(model.tree_.value, -1)[1][0]
    child_sample_num_r = np.sum(model.tree_.value, -1)[2][0]
    return root_impurity - child_sample_num_l / root_sample_num *␣
 ↪child_impurity_l - child_sample_num_r / root_sample_num * child_impurity_r

imp_equ2 = impurity_reduction(model_split)
imp_equ3 = impurity_reduction(model_split) +␣
 ↪impurity_reduction(model_surrogate_split)

print("The variable importance measurements from Equations (2) for this stump␣
 ↪is:", imp_equ2)
print("The variable importance measurements from Equations (3) for this stump␣
 ↪is:", imp_equ3)
print("This suggests that the feature", best_split_feature, "is more important␣
 ↪than the others.")
```

```
The variable importance measurements from Equations (2) for this stump is:
0.2703185497750236
The variable importance measurements from Equations (3) for this stump is:
0.37588077959385724
```

This suggests that the feature X1 is more important than the others.

1.2.3 (iii)

[5]:
```
test_x = test_data[features]
test_y = test_data.Y

# predict on test dataset
split_predict_y = model_split.predict(test_x)
surrogate_split_predict_y = model_surrogate_split.
 ↪predict(test_x[[best_surrogate_split_feature]])

# get misclassification error of predictions on the test dataset
split_err, surrogate_split_err = 0, 0
for i in range(len(test_x)):
    if split_predict_y[i] != test_y[i]:
        split_err += 1
    if surrogate_split_predict_y[i] != test_y[i]:
        surrogate_split_err += 1

# get mean misclassification error
mean_split_err = split_err / len(test_x)
mean_surrogate_split_err = surrogate_split_err / len(test_x)

print("The mean misclassification error when using the best split feature:",␣
 ↪mean_split_err)
print("The mean misclassification error when using the best surrogate split␣
 ↪feature:", mean_surrogate_split_err)
```

```
The mean misclassification error when using the best split feature: 0.1
The mean misclassification error when using the best surrogate split feature:
0.27
```

1.3 (b)

[6]:
```
# set random seed
np.random.seed(42)

M = 1000
n = 500
B = int(0.8 * n)

count_best_splits, count_best_surrogate_splits, tree_arrs, best_splits_arr =␣
 ↪[], [], [], []
oob_data_arrs = []
for K in range(1, 6):
    best_splits, best_surrogate_splits, tree_arr, oob_data_arr = [], [], [], []
```

```
    for _ in range(M):
        # randomly select K variables available for each stump
        feature_selected = np.random.choice(features, size=K, replace=False)
        # get bootstrap samples
        bootstrap_data = resample(train_data, n_samples=B, replace=True)

        # get OOB samples
        oob_index = [i for i in range(n) if i not in bootstrap_data.index]
        oob_data = train_data.iloc[oob_index]
        oob_data_arr.append(oob_data[[*feature_selected, *['Y']]])

        bootstrap_data = bootstrap_data.reset_index(drop=True)
        train_x = bootstrap_data[feature_selected]
        train_y = bootstrap_data.Y
        # build each decision stump
        tree = DecisionTreeClassifier(criterion='gini', splitter='best',␣
↪max_depth=1)
        tree.fit(train_x, train_y)
        # get best split feature
        best_split = train_x.columns[tree.tree_.feature[0]]
        best_splits.append(best_split)
        tree_arr.append(tree)

        # get the best surrogate split feature
        max_lamb = -100
        best_surrogate_split = ""
        for feature in feature_selected:
            if feature == best_split:
                continue
            # calculate the lambda
            pL, pR, pLL, pRR = 0, 0, 0, 0
            for i in range(B):
                if train_x[best_split][i] == 0: pL += 1 / B
                if train_x[best_split][i] == 1: pR += 1 / B
                if train_x[best_split][i] == 0 and train_x[feature][i] == 0:␣
↪pLL += 1 / B
                if train_x[best_split][i] == 1 and train_x[feature][i] == 1:␣
↪pRR += 1 / B
            lamb = (min(pL,pR) - (1 - pLL - pRR)) / min(pL,pR)
            # print(lamb)
            if max_lamb < lamb:
                max_lamb = lamb
                best_surrogate_split = feature
        best_surrogate_splits.append(best_surrogate_split)

    # get the number of each feature used for the best split and the best␣
↪surrogate split
```

```
    count_best_split = collections.Counter(best_splits)
    count_best_surrogate_split = collections.Counter(best_surrogate_splits)
    count_best_splits.append(count_best_split)
    count_best_surrogate_splits.append(count_best_surrogate_split)

    tree_arrs.append(tree_arr)
    best_splits_arr.append(best_splits)
    oob_data_arrs.append(oob_data_arr)
```

1.3.1 (i)

[7]:
```
# build tables
table_data_best_splits = {
    'X1' : [],
    'X2' : [],
    'X3' : [],
    'X4' : [],
    'X5' : []
}
table_data_best_surrogate_splits = {
    'X1' : [],
    'X2' : [],
    'X3' : [],
    'X4' : [],
    'X5' : []
}
table_index = ['K=1', 'K=2', 'K=3', 'K=4', 'K=5']
for K in range(5):
    for f in features:
        table_data_best_splits[f].append(count_best_splits[K][f])
        table_data_best_surrogate_splits[f].
 ↪append(count_best_surrogate_splits[K][f])

table_best_splits = pd.DataFrame(table_data_best_splits, index=table_index)
table_best_surrogate_splits = pd.DataFrame(table_data_best_surrogate_splits,␣
 ↪index=table_index)
print("     Best Splits Numbers")
print(table_best_splits)
print("\nBest Surrogate Splits Numbers")
print(table_best_surrogate_splits)
print("\n")

# determine the most important variable for each K
table_sum = table_best_splits + table_best_surrogate_splits
for i in range(5):
    print("When", table_index[i], "The most important variable is", features[np.
 ↪argmax(table_sum.iloc[i])])
```

```
     Best Splits Numbers
       X1   X2   X3   X4   X5
K=1   196  205  208  205  186
K=2   393  283  113  114   97
K=3   582  308   36   38   36
K=4   805  195    0    0    0
K=5  1000    0    0    0    0

Best Surrogate Splits Numbers
     X1    X2   X3   X4   X5
K=1   0     0    0    0    0
K=2   0    92  299  305  304
K=3   0   308  235  172  285
K=4   0   595  138   64  203
K=5   0  1000    0    0    0


When K=1 The most important variable is X3
When K=2 The most important variable is X4
When K=3 The most important variable is X2
When K=4 The most important variable is X1
When K=5 The most important variable is X1
```

Discuss the dependence with K As K grows, the importance of variables varies. When K is small, the importance is more evenly distributed. When K is large, X1 become dominant. This may because that when the number of selectable variables increases, the variables that provide better splits will naturally be chosen more often.

1.3.2 (ii)

[8]:
```python
# set random seed
np.random.seed(42)

# compute the variable importance measures in Equations (5)
imp_equ5 = []
for i in range(5):
    imp_mean = []
    imps = {'X1' : [], 'X2' : [], 'X3' : [], 'X4' : [], 'X5' : []}
    for j in range(M):
        imp = impurity_reduction(tree_arrs[i][j])
        imps[best_splits_arr[i][j]].append(imp)
    # calculate variable importance for each variable
    for f in features:
        if len(imps[f]) == 0:
            imp_mean.append(0)
        else:
            imp_mean.append(np.mean(imps[f]))
```

```
    imp_equ5.append(imp_mean)

# compute the variable importance measures in Equations (6)
imp_equ6 = []
for i in range(5):
    imp_mean = []
    imps = {'X1' : [], 'X2' : [], 'X3' : [], 'X4' : [], 'X5' : []}
    for j in range(M):
        oob_data = oob_data_arrs[i][j].reset_index(drop=True)
        val_x = oob_data.drop(['Y'], axis=1)
        val_x_perm = val_x.copy()
        val_x_perm[best_splits_arr[i][j]] = np.random.
 ↪permutation(val_x_perm[best_splits_arr[i][j]])
        val_y = oob_data.Y
        predict_y = tree_arrs[i][j].predict(val_x)
        predict_y_perm = tree_arrs[i][j].predict(val_x_perm)

        err, err_perm = 0, 0
        for a in range(len(val_x)):
            if predict_y[a] != val_y[a]: err += 1
            if predict_y_perm[a] != val_y[a]: err_perm += 1
        mean_err = err / len(val_x)
        mean_err_perm = err_perm / len(val_x)
        imp = mean_err_perm - mean_err
        imps[best_splits_arr[i][j]].append(imp)
    # calculate variable importance for each variable
    for f in features:
        if len(imps[f]) == 0:
            imp_mean.append(0)
        else:
            imp_mean.append(np.mean(imps[f]))
    imp_equ6.append(imp_mean)

# build tables for variable importance measures in Equations (5)
table_imp_equ5 = pd.DataFrame(imp_equ5, index=table_index, columns=features)
print("  The variable importance measures in Equations (5)")
print(table_imp_equ5)

# build tables for variable importance measures in Equations (6)
table_imp_equ6 = pd.DataFrame(imp_equ6, index=table_index, columns=features)
print("\n  The variable importance measures in Equations (6)")
print(table_imp_equ6)
```

```
  The variable importance measures in Equations (5)
          X1        X2        X3        X4        X5
K=1  0.267123  0.106929  0.001575  0.001522  0.001602
K=2  0.268810  0.107413  0.002851  0.002735  0.002655
K=3  0.269235  0.105722  0.004103  0.003128  0.003880
```

```
K=4  0.270631  0.106916  0.000000  0.000000  0.000000
K=5  0.269995  0.000000  0.000000  0.000000  0.000000

  The variable importance measures in Equations (6)
          X1        X2        X3        X4        X5
K=1  0.369197  0.229397 -0.004149 -0.005417 -0.009207
K=2  0.367252  0.233123 -0.009750  0.000026 -0.015606
K=3  0.367740  0.227779 -0.011884 -0.003682 -0.013046
K=4  0.364562  0.233485  0.000000  0.000000  0.000000
K=5  0.367577  0.000000  0.000000  0.000000  0.000000
```

The Most important variable The tables above suggest that for any K = 1,...,5, X1 is the most important variable, and X2 is the second most important variable.

Masking Compared with Decision Stumps, Random forests can reduce the masking effect to some degree because each tree in the forest may be splitted based on different variables. By averaging the outcomes from every trees in the forest, the phenomenon of masking will be much less likely to occur.

1.3.3 (iii)

[9]:
```
test_x = test_data[features]
test_y = test_data.Y

# method 1
mean_err_m1 = []
for i in range(5):
    predict_y_m1 = []
    for j in range(M):
        predict_y = tree_arrs[i][j].predict(test_x[oob_data_arrs[i][j].
 ↪drop(['Y'], axis=1).columns])
        predict_y_m1.append(predict_y)
    predict_y_m1 = sum(predict_y_m1)
    predict_y_m1[predict_y_m1 < 0.5 * M] = 0
    predict_y_m1[predict_y_m1 >= 0.5 * M] = 1
    # get mean misclassification error
    err = 0
    for a in range(len(test_y)):
        if predict_y_m1[a] != test_y[a]: err += 1
    mean_err = err / len(test_y)
    mean_err_m1.append(mean_err)

# method 2
mean_err_m2 = []
for i in range(5):
    mean_errs = []
    for j in range(M):
```

```
        predict_y = tree_arrs[i][j].predict(test_x[oob_data_arrs[i][j].
 ↪drop(['Y'], axis=1).columns])
        # get mean misclassification error of each tree
        err = 0
        for a in range(len(test_y)):
            if predict_y[a] != test_y[a]: err += 1
        mean_err = err / len(test_y)
        mean_errs.append(mean_err)
    mean_err_m2.append(np.mean(mean_errs))

# build tables for method1 and method2
table_mean_err = pd.DataFrame(np.transpose(np.array([mean_err_m1,␣
 ↪mean_err_m2])), index=table_index, columns=['method1', 'method2'])
print("The mean misclassification error using two methods")
print(table_mean_err)
```

```
The mean misclassification error using two methods
     method1  method2
K=1     0.14  0.37842
K=2     0.13  0.27589
K=3     0.10  0.19550
K=4     0.10  0.13315
K=5     0.10  0.10000
```

The Majority Vote Method (Method 1) is correct. Because the final prediction of RF is based on the collective decision of all its trees, rather than an average of individual trees. The second method does not capture the ensemble nature of Random Forest.

1.4 (c)

[10]:
```
# set random seed
np.random.seed(42)

M = 1000
n = 500
K = 2
Q = [0.4, 0.5, 0.6, 0.7, 0.8]

tree_arrs, best_splits_arr, oob_data_arrs = [], [], []
for q in Q:
    B = int(q * n)
    best_splits, tree_arr, oob_data_arr = [], [], []
    for _ in range(M):
        # randomly select K variables available for each stump
        feature_selected = np.random.choice(features, size=K, replace=False)
        # get bootstrap samples
        bootstrap_data = resample(train_data, n_samples=B, replace=True)
```

```
        # get OOB samples
        oob_index = [i for i in range(n) if i not in bootstrap_data.index]
        oob_data = train_data.iloc[oob_index]
        oob_data_arr.append(oob_data[[*feature_selected, *['Y']]])

        bootstrap_data = bootstrap_data.reset_index(drop=True)
        train_x = bootstrap_data[feature_selected]
        train_y = bootstrap_data.Y
        # build each decision stump
        tree = DecisionTreeClassifier(criterion='gini', splitter='best',␣
↪max_depth=1)
        tree.fit(train_x, train_y)
        # get best split feature
        best_split = train_x.columns[tree.tree_.feature[0]]
        best_splits.append(best_split)
        tree_arr.append(tree)

        # get the best surrogate split feature
        max_lamb = -100
        best_surrogate_split = ""
        for feature in feature_selected:
            if feature == best_split:
                continue
            # calculate the lambda
            pL, pR, pLL, pRR = 0, 0, 0, 0
            for i in range(B):
                if train_x[best_split][i] == 0: pL += 1 / B
                if train_x[best_split][i] == 1: pR += 1 / B
                if train_x[best_split][i] == 0 and train_x[feature][i] == 0:␣
↪pLL += 1 / B
                if train_x[best_split][i] == 1 and train_x[feature][i] == 1:␣
↪pRR += 1 / B
            lamb = (min(pL,pR) - (1 - pLL - pRR)) / min(pL,pR)
            # print(lamb)
            if max_lamb < lamb:
                max_lamb = lamb
                best_surrogate_split = feature

    tree_arrs.append(tree_arr)
    best_splits_arr.append(best_splits)
    oob_data_arrs.append(oob_data_arr)
```

1.4.1 (i)

[11]:
```
# set random seed
np.random.seed(42)

# compute the variable importance measures in Equations (5)
imp_equ5 = []
for i in range(5):
    imp_mean = []
    imps = {'X1' : [], 'X2' : [], 'X3' : [], 'X4' : [], 'X5' : []}
    for j in range(M):
        imp = impurity_reduction(tree_arrs[i][j])
        imps[best_splits_arr[i][j]].append(imp)
    # calculate variable importance for each variable
    for f in features:
        if len(imps[f]) == 0:
            imp_mean.append(0)
        else:
            imp_mean.append(np.mean(imps[f]))
    imp_equ5.append(imp_mean)

# compute the variable importance measures in Equations (6)
imp_equ6 = []
for i in range(5):
    imp_mean = []
    imps = {'X1' : [], 'X2' : [], 'X3' : [], 'X4' : [], 'X5' : []}
    for j in range(M):
        oob_data = oob_data_arrs[i][j].reset_index(drop=True)
        val_x = oob_data.drop(['Y'], axis=1)
        val_x_perm = val_x.copy()
        val_x_perm[best_splits_arr[i][j]] = np.random.
 ↪permutation(val_x_perm[best_splits_arr[i][j]])
        val_y = oob_data.Y
        predict_y = tree_arrs[i][j].predict(val_x)
        predict_y_perm = tree_arrs[i][j].predict(val_x_perm)

        err, err_perm = 0, 0
        for a in range(len(val_x)):
            if predict_y[a] != val_y[a]: err += 1
            if predict_y_perm[a] != val_y[a]: err_perm += 1
        mean_err = err / len(val_x)
        mean_err_perm = err_perm / len(val_x)
        imp = mean_err_perm - mean_err
        imps[best_splits_arr[i][j]].append(imp)
    # calculate variable importance for each variable
    for f in features:
        if len(imps[f]) == 0:
```

```
            imp_mean.append(0)
        else:
            imp_mean.append(np.mean(imps[f]))
    imp_equ6.append(imp_mean)

table_index = ['B=200', 'B=250', 'B=300', 'B=350', 'B=400']
# build tables for variable importance measures in Equations (5)
table_imp_equ5 = pd.DataFrame(imp_equ5, index=table_index, columns=features)
print("  The variable importance measures in Equations (5)")
print(table_imp_equ5)

# build tables for variable importance measures in Equations (6)
table_imp_equ6 = pd.DataFrame(imp_equ6, index=table_index, columns=features)
print("\n  The variable importance measures in Equations (6)")
print(table_imp_equ6)
```

```
  The variable importance measures in Equations (5)
            X1        X2        X3        X4        X5
B=200  0.269323  0.105718  0.005587  0.004807  0.004481
B=250  0.271203  0.105047  0.003282  0.004408  0.003750
B=300  0.271034  0.107022  0.003696  0.004209  0.003124
B=350  0.269317  0.106197  0.003435  0.003576  0.003111
B=400  0.270094  0.106553  0.002759  0.003109  0.002605

  The variable importance measures in Equations (6)
            X1        X2        X3        X4        X5
B=200  0.367505  0.231299 -0.006850 -0.004162 -0.011580
B=250  0.368079  0.232027 -0.004622 -0.001926 -0.011512
B=300  0.367227  0.230447 -0.007934 -0.007099 -0.014836
B=350  0.369944  0.230538 -0.012056 -0.004971 -0.013559
B=400  0.367324  0.228894 -0.010838 -0.004668 -0.016738
```

The Most important variable The tables above suggest that for any B = 200,250,300,350,400, X1 is the most important variable.

Discuss the dependence with B The choice of B affects the stability of these variable importance measures. The larger B is, the more stable the variable importance measures are.

HW3_Q4

October 8, 2023

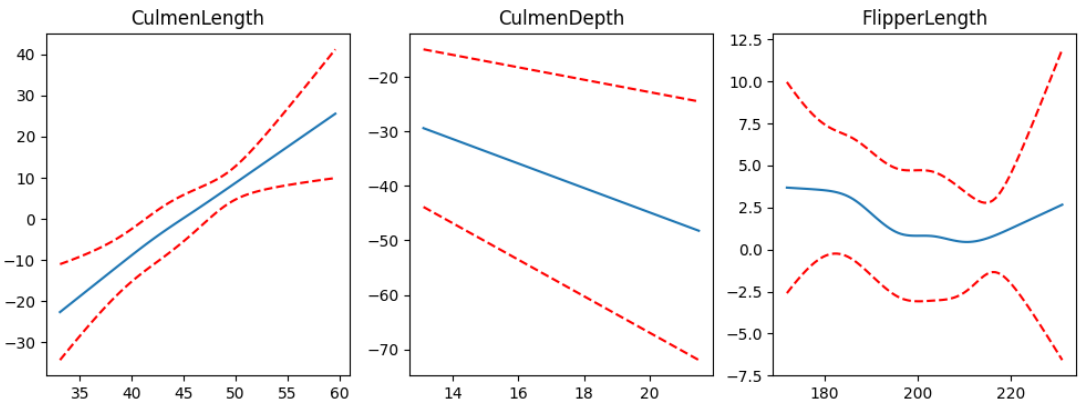
1 Q4 GAM

[1]:
```
# pyGAM
from pygam import LogisticGAM, s, l
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# read data from csv file
data = pd.read_csv("penguins_trunc.csv")

# split train and test dataset
x = data[['CulmenLength', 'CulmenDepth', 'FlipperLength']]
y = data['Species']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪train_size=0.8, random_state=42, shuffle=True, stratify=y)

# fit a GAM
gam = LogisticGAM(s(0) + l(1) + s(2)).fit(x_train, y_train)

# predict
pred_y_train = gam.predict(x_train)
pred_y_test = gam.predict(x_test)

# get training and test accuracy
train_acc = gam.accuracy(x_train, y_train)
test_acc = gam.accuracy(x_test, y_test)

print("training accuracy:", train_acc)
print("test accuracy:", test_acc)

# plot shape function of each feature
print("\nThe shape functions for each feature are shown below.")
plt.figure(figsize=(12,4))
for i in range(len(x_train.columns)):
    XX = gam.generate_X_grid(term=i)
```

```
    output, confidence = gam.partial_dependence(term=i, X=XX, width=0.95)
    plt.subplot(131 + i)
    plt.plot(XX[:, i], output)
    plt.plot(XX[:, i], confidence, c='r', ls='--')
    plt.title(x_train.columns[i])
plt.show()
```

```
training accuracy: 0.9853479853479854
test accuracy: 1.0
```

The shape functions for each feature are shown below.



HW3_Q5

October 8, 2023

1 Q5 Boosting Algorithm Practice

1.1 (a)

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, roc_auc_score, accuracy_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt

# read data from csv file
data = pd.read_csv("Titanic.csv")
data = data.dropna(axis='columns')
data.loc[data['Sex'] == 'female', 'Sex'] = 1
data.loc[data['Sex'] == 'male', 'Sex'] = 0
data['Sex'] = pd.to_numeric(data['Sex'])

# split train and test dataset
x = data.drop("Survived", axis=1)
y = data['Survived']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪train_size=0.8, random_state=42, shuffle=True, stratify=y)
```

1.2 (b)

[2]:
```
# sklearn's Adaboost
# tune parameters of the Adaboost using cv
adaboost_hyperparam = {
    'base_estimator': [DecisionTreeClassifier(max_depth=2),␣
↪DecisionTreeClassifier(max_depth=3)],
    'n_estimators': [50, 75, 100],
    'learning_rate': [0.01, 0.05, 0.1, 0.5, 1],
    'algorithm': ['SAMME', 'SAMME.R']
}
```

```
adaboost_model = AdaBoostClassifier()
adaboost_grid = GridSearchCV(adaboost_model, adaboost_hyperparam, cv=5,␣
 ↪scoring='accuracy')
adaboost_grid.fit(x_train, y_train)

best_adaboost_model = adaboost_grid.best_estimator_
# print(adaboost_grid.best_score_)
# print(adaboost_grid.best_params_)
# print(best_adaboost_model)
```

[3]:
```
# XGBoost
# tune parameters of the XGBoost using cv
xgboost_hyperparam = {
    'booster': ['gbtree'],
    'tree_method': ['auto', 'exact', 'approx'],
    'n_estimators': [30, 50, 75, 100],
    'eta': [0.01, 0.1, 0.5, 1],
    'max_depth': [2, 3, 4],
    'gamma': [0, 0.1, 0.2]
}
xgboost_model = XGBClassifier()
xgboost_grid = GridSearchCV(xgboost_model, xgboost_hyperparam, cv=5,␣
 ↪scoring='accuracy')
xgboost_grid.fit(x_train, y_train)

best_xgboost_model = xgboost_grid.best_estimator_
# print(xgboost_grid.best_score_)
# print(xgboost_grid.best_params_)
# print(best_xgboost_model)
```

[4]:
```
# performance on the test set
y_pred_ada = best_adaboost_model.predict(x_test)
y_pred_xgb = best_xgboost_model.predict(x_test)

table_data = [
    [f1_score(y_test, y_pred_ada), roc_auc_score(y_test, y_pred_ada),␣
 ↪accuracy_score(y_test, y_pred_ada)],
    [f1_score(y_test, y_pred_xgb), roc_auc_score(y_test, y_pred_xgb),␣
 ↪accuracy_score(y_test, y_pred_xgb)]
]
table = pd.DataFrame(table_data, index=['Adaboost', 'XGBoost'], columns=['f1',␣
 ↪'roc_auc', 'accuracy'])
print(table)
```

```
                f1   roc_auc  accuracy
Adaboost  0.671642  0.735178  0.754190
XGBoost   0.715328  0.768709  0.782123
```

1.2.1 Comment on performance difference

When random_state of train_test_split is 42, XGBoost outperforms Adaboost in f1 score, auc, and accuracy.