4-1

April 19, 2024

[15]:
```
# !pip install shap
```

[14]:
```
import numpy as np
import matplotlib.pyplot as plt
import shap
from sklearn.metrics import zero_one_loss, log_loss
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
```

[16]:
```
X, y = shap.datasets.adult()
X_display, y_display = shap.datasets.adult(display=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=7)
```

[62]:
```
X_train.head()
```

[62]:

	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
12011	51.0	4	10.0	0	6	0	4	0	0.0	0.0	40.0	21
23599	51.0	1	14.0	6	12	1	4	1	0.0	0.0	50.0	8
23603	21.0	4	11.0	4	3	3	2	1	0.0	0.0	40.0	39
6163	25.0	4	10.0	4	12	3	4	1	0.0	0.0	24.0	39
14883	48.0	4	13.0	0	1	3	4	1	0.0	0.0	38.0	39

0.1 4(a)

[18]:
```
clf = GradientBoostingClassifier(n_estimators=100 , random_state=10)
clf.fit(X_train.values, y_train)
```

[18]:
```
GradientBoostingClassifier(random_state=10)
```

[57]:
```
# Make predictions on train and test sets
y_pred = clf.predict(X_train.values)
y_test_pred = clf.predict(X_test.values)
```

[58]:
```
zero_one_loss(y_train, y_pred)
```

[58]:
```
0.13148802211302213
```

[59]:
```
zero_one_loss(y_test, y_test_pred)
```

[59]:
```
0.1337325349301397
```

[60]:
```
log_loss(y_train, y_pred)
```

[60]:
```
4.739308693862341
```

[61]:
```
log_loss(y_test, y_test_pred)
```

[61]:
```
4.820209135869959
```

0.2 4 (b)

[28]:
```
def permutation_importance(X_test_data, y_test_data, feature_index,␣
↪n_permutations=1,
                           loss_type="zero_one",␣
↪feature_removal_method="random"):
    original_error = 0
    y_pred = clf.predict(X_test_data.values)

    if loss_type == "zero_one":
        original_error = zero_one_loss(y_test_data, y_pred)
    elif loss_type == "log_loss":
        original_error = log_loss(y_test_data, y_pred)

    permutation_errors = np.zeros(n_permutations)

    for i in range(n_permutations):
        X_test_permuted = X_test_data.copy()
        column_values = X_test_permuted.iloc[:, feature_index]
        if feature_removal_method == "random":
```

```
            column_values = column_values.to_numpy()
            np.random.shuffle(column_values)
        elif feature_removal_method == "mean_impute":
            column_values = column_values.mean()
        X_test_permuted.iloc[:, feature_index] = column_values

        y_pred_permuted = clf.predict(X_test_permuted.values)

        if loss_type == "zero_one":
            permutation_errors[i] = zero_one_loss(y_test_data, y_pred_permuted)
        else:
            permutation_errors[i] = log_loss(y_test_data, y_pred_permuted)

    importance = permutation_errors - original_error
    return np.mean(importance), np.std(importance)
```

[29]:
```
n_features = X.shape[1]
feature_importances = np.zeros(n_features)
n_features
```

[29]: 12

[30]:
```
for i in range(n_features):
    feature_importances[i], _ = permutation_importance(X_test, y_test, i, 1,␣
 ↪"zero_one", "random")
```

[44]:
```
import plotly.graph_objects as go

feature_indices = list(range(n_features))
fig = go.Figure(data=[go.Bar(
    x=feature_indices,
    y=feature_importances,
)])

fig.update_layout(
    title="Finding Feature Importance using Permutation Test",
    xaxis_title="Feature Index",
    yaxis_title="Permutation Importance",
)

fig.update_xaxes(tickvals=feature_indices)
fig.show()
```

0.3 4(c)

[41]:
```
feature_importances_ten_times = np.zeros(n_features)
feature_importances_std_ten_times = np.zeros(n_features)
for i in range(n_features):
    feature_importances_ten_times[i], feature_importances_std_ten_times[i] =
    permutation_importance(X_test, y_test, i, 10, "zero_one", "random")
```

[49]:
```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Bar(
    x=list(range(n_features)),
    y=feature_importances_ten_times,
    error_y=dict(type='data', array=feature_importances_std_ten_times,␣
 ↪visible=True),
    name='Feature Importance',
))

# Update layout
fig.update_layout(
    title="Finding Feature Importance using Permutation Test",
    xaxis_title="Feature Index",
    yaxis_title="Permutation Importance",
    xaxis=dict(
        tickmode='array',
        tickvals=list(range(n_features)),
        ticktext=list(range(n_features)),
    ),
)

fig.show()
```

0.4 4 (d)

[47]:
```
feature_importances_mean = np.zeros(n_features)
for i in range(n_features):
    feature_importances_mean[i], _ = permutation_importance(X_test, y_test, i,␣
 ↪1, "zero_one", "mean_impute")
```

[50]:
```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Bar(
    x=list(range(n_features)),
```

```
    y=feature_importances_mean,
    name='Feature Importance',
))

fig.update_layout(
    title="Finding Feature Importance using Permutation Test",
    xaxis_title="Feature Index",
    yaxis_title="Permutation Importance",
    xaxis=dict(
        tickmode='array',
        tickvals=list(range(n_features)),
        ticktext=list(range(n_features)),
    ),
)

fig.show()
```

0.5 4(e)

[54]:
```
feature_importances_log_loss = np.zeros(n_features)
for i in range(n_features):
    feature_importances_log_loss[i], _ = permutation_importance(X_test, y_test,␣
 ↪i, 1, "log_loss", "random")
```

[64]:
```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Bar(
    x=list(range(n_features)),
    y=feature_importances_log_loss,
    name='Feature Importance',
))

fig.update_layout(
    title="Finding Feature Importance using Permutation Test",
    xaxis_title="Feature Index",
    yaxis_title="Permutation Importance",
    xaxis=dict(
        tickmode='array',
        tickvals=list(range(n_features)),
        ticktext=list(range(n_features)),
    ),
)

fig.show()
```

[ ]: