CAP_5610_Assignment_1_Solution_Arman_Sayan

January 26, 2025

CAP 5610 Assignment #1: Decision Tree and Naive Bayes Classifier

This source code is written by Arman Sayan.

Last Edit: January 26, 2024

1 Q1 - Decision Tree Basics

1.1 Part (3):

[1]:
```
import pandas as pd
import numpy as np
from math import log2
from collections import Counter
from graphviz import Digraph

# Data setup
data = [
    [1, 1, 0, 0, 1, 1],
    [1, 1, 1, 0, 1, 1],
    [0, 0, 1, 0, 0, 0],
    [0, 1, 1, 0, 1, 0],
    [0, 1, 1, 0, 0, 1],
    [0, 0, 1, 1, 1, 1],
    [1, 0, 0, 0, 1, 0],
    [0, 1, 0, 1, 1, 1],
    [0, 0, 1, 0, 1, 1],
    [1, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0, 1],
    [0, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 0],
    [1, 0, 0, 1, 0, 1],
]
columns = ["Early", "Finished HMK", "Senior", "Likes Coffee", "Liked The Last␣
↪Jedi", "A"]
df = pd.DataFrame(data, columns=columns)

# Entropy calculation
```

```python
def entropy(labels):
    total = len(labels)
    counts = Counter(labels)
    return -sum((count / total) * log2(count / total) for count in counts.values() if count > 0)

# Information gain calculation
def information_gain(df, attribute, target="A"):
    total_entropy = entropy(df[target])
    values = df[attribute].unique()
    weighted_entropy = sum(
        (len(subset) / len(df)) * entropy(subset[target])
        for value in values
        if (subset := df[df[attribute] == value]) is not None
    )
    return total_entropy - weighted_entropy

# ID3 algorithm
def id3(df, attributes, target="A", depth=1):
    node_entropy = entropy(df[target])
    if depth == 0 or node_entropy == 0 or len(attributes) == 0:
        most_common_label = Counter(df[target]).most_common(1)[0][0]
        return {"label": most_common_label, "count": len(df), "entropy": node_entropy}

    best_attribute = max(attributes, key=lambda attr: information_gain(df, attr, target))
    tree = {"attribute": best_attribute, "entropy": node_entropy, "children": {}}

    for value in df[best_attribute].unique():
        subset = df[df[best_attribute] == value]
        if len(subset[target].unique()) == 1:
            label = subset[target].iloc[0]
            tree["children"][value] = {"label": label, "count": len(subset), "entropy": entropy(subset[target])}
        else:
            remaining_attributes = [attr for attr in attributes if attr != best_attribute]
            tree["children"][value] = id3(subset, remaining_attributes, target, depth - 1)

    return tree

# Visualize the decision tree
def visualize_tree(tree, graph=None, parent=None, edge_label=None):
```

```python
    if graph is None:
        graph = Digraph(format="png")
        graph.attr("node", shape="box")

    if "label" in tree:
        node_label = f"Leaf: {tree['label']}\nCount: {tree['count']}\nEntropy:␣
↪{tree['entropy']:.4f}"
        node_id = str(id(tree))
        graph.node(node_id, label=node_label)
        if parent:
            graph.edge(parent, node_id, label=edge_label)
    else:
        node_label = f"{tree['attribute']}\nEntropy: {tree['entropy']:.4f}"
        node_id = str(id(tree))
        graph.node(node_id, label=node_label)
        if parent:
            graph.edge(parent, node_id, label=edge_label)

        for value, child in tree["children"].items():
            visualize_tree(child, graph, parent=node_id, edge_label=str(value))

    return graph

# Build trees
depth_1_tree = id3(df, columns[:-1], depth=1)
depth_2_tree = id3(df, columns[:-1], depth=2)

# Save visualizations
visualize_tree(depth_1_tree).render("depth_1_tree", cleanup=True)
visualize_tree(depth_2_tree).render("depth_2_tree", cleanup=True)
```

[1]: 'depth_2_tree.png'

1.2 Part (4):

[2]:
```python
# Build a depth-3 tree
depth_3_tree = id3(df, columns[:-1], depth=3)

# Save visualization
visualize_tree(depth_3_tree).render("depth_3_tree", cleanup=True)
```

[2]: 'depth_3_tree.png'

[3]:
```python
def calculate_average_leaf_entropy(tree):
    def collect_leaf_entropies(node):
        if "label" in node:  # Leaf node
```

```
            return [node["entropy"]]
        # Recursive collection of entropies from children
        entropies = []
        for child in node["children"].values():
            entropies.extend(collect_leaf_entropies(child))
        return entropies

    leaf_entropies = collect_leaf_entropies(tree)
    return sum(leaf_entropies) / len(leaf_entropies) if leaf_entropies else 0
```

[4]:
```
avg_entropy_depth_1 = calculate_average_leaf_entropy(depth_1_tree)
avg_entropy_depth_2 = calculate_average_leaf_entropy(depth_2_tree)

print(f"Average leaf entropy for depth-1 tree: {avg_entropy_depth_1:.4f}")
print(f"Average leaf entropy for depth-2 tree: {avg_entropy_depth_2:.4f}")
```

```
Average leaf entropy for depth-1 tree: 0.9242
Average leaf entropy for depth-2 tree: 0.4305
```

[5]:
```
# Build and visualize depth-3 tree
depth_3_tree = id3(df, columns[:-1], depth=3)
visualize_tree(depth_3_tree).render("depth_3_tree", cleanup=True)

# Calculate average leaf entropy for depth-3 tree
avg_entropy_depth_3 = calculate_average_leaf_entropy(depth_3_tree)
print(f"Average leaf entropy for depth-3 tree: {avg_entropy_depth_3:.4f}")
```

```
Average leaf entropy for depth-3 tree: 0.3197
```

2 Q2 - Application of Decision Tree on Real-Word Data-set

2.1 Check Statistics for the Census-Income Data Set:

[6]:
```
import pandas as pd

# Load the dataset

column_names = [
"AAGE",
"ACLSWKR",
"ADTIND",
"ADTOCC",
"AHGA",
"AHRSPAY",
"AHSCOL",
"AMARITL",
"AMJIND",
```

```
"AMJOCC",
"ARACE",
"AREORGN",
"ASEX",
"AUNMEM",
"AUNTYPE",
"AWKSTAT",
"CAPGAIN",
"CAPLOSS",
"DIVVAL",
"FILESTAT",
"GRINREG",
"GRINST",
"HHDFMX",
"HHDREL",
"MARSUPWT",
"MIGMTR1",
"MIGMTR3",
"MIGMTR4",
"MIGSAME",
"MIGSUN",
"NOEMP",
"PARENT",
"PEFNTVTY",
"PEMNTVTY",
"PENATVTY",
"PRCITSHP",
"SEOTR",
"VETQVA",
"VETYN",
"WKSWORK",
"YEAR",
"INCCLS"]

data = pd.read_csv("census-income.data", header=None, names=column_names)
test = pd.read_csv("census-income.test", header=None, names=column_names)
```

[7]:
```
data.head(5)
```

[7]:
```
   AAGE                      ACLSWKR  ADTIND  ADTOCC  \
0    73              Not in universe       0       0
1    58   Self-employed-not incorporated       4      34
2    18              Not in universe       0       0
3     9              Not in universe       0       0
4    10              Not in universe       0       0

                          AHGA  AHRSPAY            AHSCOL         AMARITL  \
```

```
0        High school graduate        0   Not in universe        Widowed
1   Some college but no degree        0   Not in universe       Divorced
2                  10th grade        0       High school   Never married
3                    Children        0   Not in universe   Never married
4                    Children        0   Not in universe   Never married

                         AMJIND                                  AMJOCC  …  \
0  Not in universe or children                          Not in universe  …
1                  Construction   Precision production craft & repair  …
2  Not in universe or children                          Not in universe  …
3  Not in universe or children                          Not in universe  …
4  Not in universe or children                          Not in universe  …

        PEFNTVTY       PEMNTVTY       PENATVTY  \
0  United-States  United-States  United-States
1  United-States  United-States  United-States
2        Vietnam        Vietnam        Vietnam
3  United-States  United-States  United-States
4  United-States  United-States  United-States

                                PRCITSHP SEOTR           VETQVA  VETYN  \
0      Native- Born in the United States     0  Not in universe      2
1      Native- Born in the United States     0  Not in universe      2
2    Foreign born- Not a citizen of U S      0  Not in universe      2
3      Native- Born in the United States     0  Not in universe      0
4      Native- Born in the United States     0  Not in universe      0

  WKSWORK  YEAR     INCCLS
0       0    95   - 50000.
1      52    94   - 50000.
2       0    95   - 50000.
3       0    94   - 50000.
4       0    94   - 50000.

[5 rows x 42 columns]
```

[8]:
```
test.head(5)
```

[8]:
```
  AAGE                         ACLSWKR  ADTIND  ADTOCC  \
0   38                         Private       6      36
1   44  Self-employed-not incorporated      37      12
2    2                 Not in universe       0       0
3   35                         Private      29       3
4   49                         Private       4      34

                          AHGA  AHRSPAY           AHSCOL  \
0        1st 2nd 3rd or 4th grade        0  Not in universe
```

```
1   Associates degree-occup /vocational        0   Not in universe
2                             Children        0   Not in universe
3                 High school graduate        0   Not in universe
4                 High school graduate        0   Not in universe

                             AMARITL                             AMJIND  \
0   Married-civilian spouse present    Manufacturing-durable goods
1   Married-civilian spouse present   Business and repair services
2                     Never married    Not in universe or children
3                          Divorced                 Transportation
4                          Divorced                   Construction

                                   AMJOCC  …        PEFNTVTY  \
0   Machine operators assmblrs & inspctrs  …          Mexico
1                  Professional specialty  …   United-States
2                         Not in universe  …   United-States
3          Executive admin and managerial  …   United-States
4      Precision production craft & repair  …   United-States

         PEMNTVTY        PENATVTY                            PRCITSHP SEOTR  \
0          Mexico          Mexico   Foreign born- Not a citizen of U S     0
1   United-States   United-States     Native- Born in the United States     0
2   United-States   United-States     Native- Born in the United States     0
3   United-States   United-States     Native- Born in the United States     2
4   United-States   United-States     Native- Born in the United States     0

             VETQVA  VETYN  WKSWORK  YEAR     INCCLS
0   Not in universe      2       12    95   - 50000.
1   Not in universe      2       26    95   - 50000.
2   Not in universe      0        0    95   - 50000.
3   Not in universe      2       52    94   - 50000.
4   Not in universe      2       50    95   - 50000.

[5 rows x 42 columns]
```

[9]:
```python
# Print the number of instances
num_instances_data = data.shape[0]
print(f"Number of instances in data: {num_instances_data}")

num_instances_test = test.shape[0]
print(f"Number of instances in test: {num_instances_test}")
```

```
Number of instances in data: 199523
Number of instances in test: 99762
```

[10]:
```python
# Calculate the probability distribution of the 'income' column
income_probabilities = test["INCCLS"].value_counts(normalize=True)
```

```python
# Print the probabilities
print("Class probabilities for income-projected.test file:")
print(income_probabilities)
```

```
Class probabilities for income-projected.test file:
INCCLS
- 50000.    0.937992
50000+.     0.062008
Name: proportion, dtype: float64
```

[11]:

```python
print("Information about .data file:")
# Calculate the number of distinct values for each column
distinct_values_count = data.nunique()

# Display the results
print("Number of distinct values for each column:")
print(distinct_values_count)
```

```
Information about .data file:
Number of distinct values for each column:
AAGE           91
ACLSWKR         9
ADTIND         52
ADTOCC         47
AHGA           17
AHRSPAY      1240
AHSCOL          3
AMARITL         7
AMJIND         24
AMJOCC         15
ARACE           5
AREORGN        10
ASEX            2
AUNMEM          3
AUNTYPE         6
AWKSTAT         8
CAPGAIN       132
CAPLOSS       113
DIVVAL       1478
FILESTAT        6
GRINREG         6
GRINST         51
HHDFMX         38
HHDREL          8
MARSUPWT    99800
MIGMTR1        10
MIGMTR3         9
```

```
MIGMTR4       10
MIGSAME        3
MIGSUN         4
NOEMP          7
PARENT         5
PEFNTVTY      43
PEMNTVTY      43
PENATVTY      43
PRCITSHP       5
SEOTR          3
VETQVA         3
VETYN          3
WKSWORK       53
YEAR           2
INCCLS         2
dtype: int64
```

2.2 Task (a):

Train a decision tree classifier using the data file. Vary the cut-off depth from 2 to 10 and report the training accuracy for each cut-off depth k. Based on your results, select an optimal k.

[12]:
```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Preprocess the data
# Encode categorical variables
label_encoders = {}
for column in data.select_dtypes(include=["object"]).columns:
    label_encoders[column] = LabelEncoder()
    data[column] = label_encoders[column].fit_transform(data[column])
```

[13]:
```
# Separate features and target variable
X = data.drop(["INCCLS", "MARSUPWT"], axis=1)
y = data["INCCLS"]
```

[14]:
```
# Train decision trees with varying depths
training_accuracies = []
depths = range(2, 11)

for depth in depths:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X, y)

    # Compute training accuracy
```

```
    y_train_pred = clf.predict(X)
    train_acc = accuracy_score(y, y_train_pred)
    training_accuracies.append(train_acc)
```

[15]:
```
# Report results
for depth, train_acc in zip(depths, training_accuracies):
    print(f"Depth: {depth}, Training Accuracy: {train_acc:.4f}")

# Select the optimal depth based on testing accuracy
optimal_depth = depths[np.argmax(training_accuracies)]
print(f"\nOptimal Depth: {optimal_depth}")
```

```
Depth: 2, Training Accuracy: 0.9445
Depth: 3, Training Accuracy: 0.9449
Depth: 4, Training Accuracy: 0.9455
Depth: 5, Training Accuracy: 0.9492
Depth: 6, Training Accuracy: 0.9498
Depth: 7, Training Accuracy: 0.9520
Depth: 8, Training Accuracy: 0.9525
Depth: 9, Training Accuracy: 0.9535
Depth: 10, Training Accuracy: 0.9544

Optimal Depth: 10
```

2.3 Task (b)

Using the trained classifier with optimal cut-off depth k, classify the 99,762 instances from the test file and report the testing accuracy (the portion of testing instances classified correctly).

[16]:
```
# Preprocess the test data
# Apply the same LabelEncoders used for training
label_encoders_test = {}
for column in test.select_dtypes(include=["object"]).columns:
    label_encoders_test[column] = LabelEncoder()
    test[column] = label_encoders_test[column].fit_transform(test[column])
```

[17]:
```
# Separate features and target variable in the test data
X_test = test.drop(["INCCLS", "MARSUPWT"], axis=1)
y_test = test["INCCLS"]
```

[18]:
```
# Train a classifier with the optimal depth
clf_optimal = DecisionTreeClassifier(max_depth=optimal_depth, random_state=42)
clf_optimal.fit(X, y)

# Classify the test instances
y_pred_final = clf_optimal.predict(X_test)

# Compute testing accuracy
```

```
final_accuracy = accuracy_score(y_test, y_pred_final)

# Report the results
print(f"Testing Accuracy on the test file: {final_accuracy:.4f}")
```

```
Testing Accuracy on the test file: 0.9509
```

2.4 Task (c)

Do you see any over-fitting issues for this experiment? Report your observations.

[19]:
```
# Train decision trees with varying depths
training_accuracies = []
testing_accuracies = []
depths = range(2, 11)

for depth in depths:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X, y)

    # Compute training accuracy
    y_train_pred = clf.predict(X)
    train_acc = accuracy_score(y, y_train_pred)
    training_accuracies.append(train_acc)

    # Compute testing accuracy
    y_test_pred = clf.predict(X_test)
    test_acc = accuracy_score(y_test, y_test_pred)
    testing_accuracies.append(test_acc)

# Report results
for depth, train_acc, test_acc in zip(depths, training_accuracies,␣
 ↪testing_accuracies):
    print(f"Depth: {depth}, Training Accuracy: {train_acc:.4f}, Testing␣
 ↪Accuracy: {test_acc:.4f}")

# Select the optimal depth based on testing accuracy
optimal_depth = depths[np.argmax(testing_accuracies)]
print(f"\nOptimal Depth: {optimal_depth}")
```

```
Depth: 2, Training Accuracy: 0.9445, Testing Accuracy: 0.9442
Depth: 3, Training Accuracy: 0.9449, Testing Accuracy: 0.9447
Depth: 4, Training Accuracy: 0.9455, Testing Accuracy: 0.9447
Depth: 5, Training Accuracy: 0.9492, Testing Accuracy: 0.9485
Depth: 6, Training Accuracy: 0.9498, Testing Accuracy: 0.9486
Depth: 7, Training Accuracy: 0.9520, Testing Accuracy: 0.9505
Depth: 8, Training Accuracy: 0.9525, Testing Accuracy: 0.9507
Depth: 9, Training Accuracy: 0.9535, Testing Accuracy: 0.9505
```

```
Depth: 10, Training Accuracy: 0.9544, Testing Accuracy: 0.9509

Optimal Depth: 10
```

From the data provided, there **does not appear to be a significant overfitting issue**, as both the **training accuracy** and **testing accuracy** are increasing or remaining stable as the tree depth increases.

2.4.1 Observations:

1. **Consistency between Training and Testing Accuracy**:
   - The testing accuracy does not decrease as the depth increases, which is a hallmark of overfitting. Instead, the testing accuracy either slightly increases or plateaus, indicating that the model is still generalizing well even at higher depths.
   - The gap between training and testing accuracy remains small (less than ~0.004), which is minimal.
2. **Optimal Depth**:
   - The testing accuracy peaks at **Depth = 10** (Testing Accuracy = 0.9509). However, the improvement from Depth = 8 to Depth = 10 is very marginal (0.9507 → 0.9509), and increasing the depth further may result in diminishing returns.
   - Depth = 8 or Depth = 10 can be considered the optimal depth based on the goal (e.g., achieving maximum accuracy or reducing computational complexity).
3. **Overfitting Behavior**:
   - While training accuracy increases more quickly with depth, this is expected for deeper decision trees as they capture more details in the data. However, the testing accuracy keeps up, suggesting the deeper trees are still generalizing well.
   - There is no clear overfitting in this experiment.

---

3 Q4 - Implementing Naive Bayes

Implement Naive Bayes Algorithm. Train your classifier on the training set that is given and report training accuracy, testing accuracy, and the amount of time spent training the classifier.

[20]:
```python
import pandas as pd
import numpy as np
import time
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Load data
def load_data(train_file, train_labels_file, test_file, test_labels_file):
    X_train = pd.read_csv(train_file, header=None).values
    y_train = pd.read_csv(train_labels_file, header=None).values.ravel()
    X_test = pd.read_csv(test_file, header=None).values
    y_test = pd.read_csv(test_labels_file, header=None).values.ravel()
    return X_train, y_train, X_test, y_test
```

```python
# Train and evaluate Naive Bayes classifier
def train_naive_bayes(X_train, y_train, X_test, y_test):
    print(f"Missing values in X_train: {np.isnan(X_train).sum()}")
    print(f"Missing values in X_test: {np.isnan(X_test).sum()}")

    start_time = time.time()

    # Initialize and train the classifier
    nb_classifier = MultinomialNB()
    nb_classifier.fit(X_train, y_train)

    # Measure training time
    training_time = time.time() - start_time

    # Predict on training and test sets
    y_train_pred = nb_classifier.predict(X_train)
    y_test_pred = nb_classifier.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    return train_accuracy, test_accuracy, training_time

# Main function to load data, train classifier, and report results
def main():
    # File paths (update these paths if necessary)
    train_file = "train.csv"
    train_labels_file = "train_labels.txt"
    test_file = "test.csv"
    test_labels_file = "test_labels.txt"

    # Load the data
    X_train, y_train, X_test, y_test = load_data(train_file, train_labels_file, test_file, test_labels_file)

    # Train Naive Bayes and get metrics
    train_accuracy, test_accuracy, training_time = train_naive_bayes(X_train, y_train, X_test, y_test)

    # Print results
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Testing Accuracy: {test_accuracy:.4f}")
    print(f"Training Time: {training_time:.4f} seconds")

if __name__ == "__main__":
    main()
```

```
Missing values in X_train: 0
Missing values in X_test: 0
Training Accuracy: 0.9693
Testing Accuracy: 0.9823
Training Time: 0.0835 seconds
```