Applications of ML HW 2

By Srilalith (nsrilalith)

Part 2:

Code:

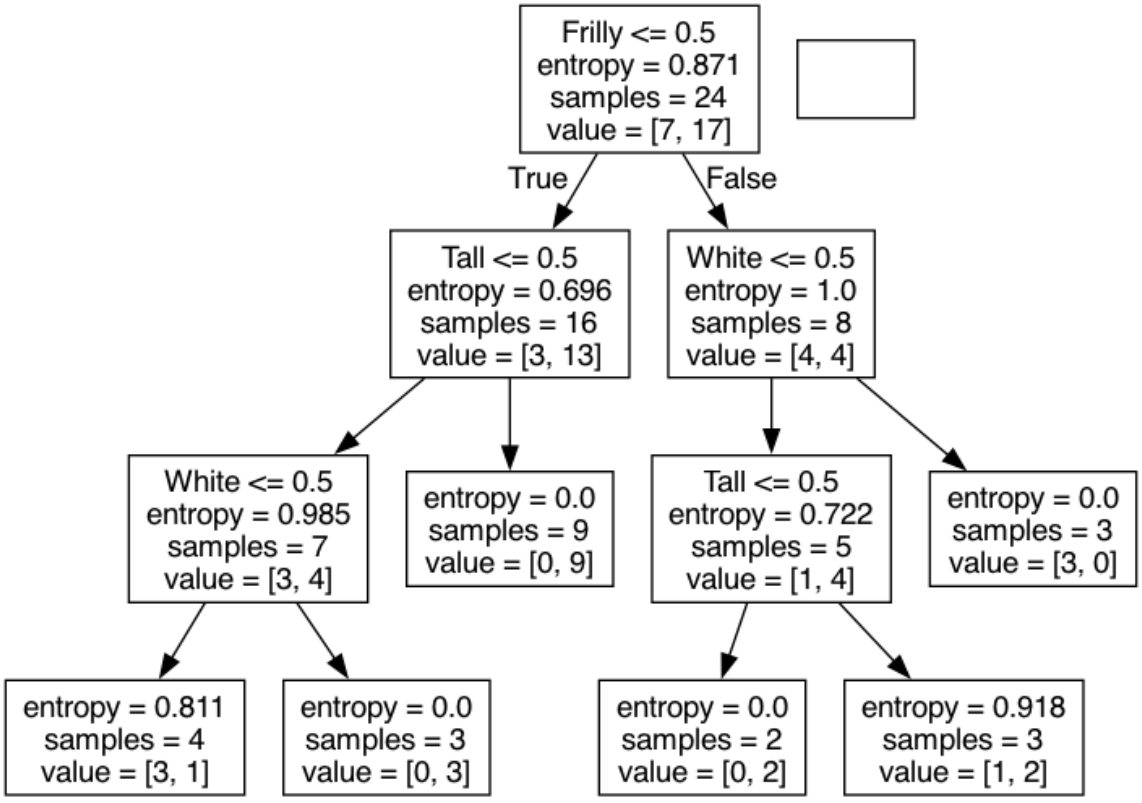
```
from sklearn import tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
import pandas as pd
import pydotplus


def writegraphtofile(clf, featurelabels, filename):
    dot_data = tree.export_graphviz(clf, feature_names=featurelabels, out_file=None)
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_png(filename)


data = pd.read_excel("AlienMushrooms.xlsx")

clf = tree.DecisionTreeClassifier(criterion="entropy")

feature_labels = data.columns[:-1]
features = data[feature_labels]

target_label = data.columns[-1]
target = data[target_label]

clf.fit(features, target)

writegraphtofile(clf, feature_labels, "Assignment_2" + "tree_pic.png")

predictions = clf.predict(features)

# Calculate metrics
accuracy = accuracy_score(target, predictions)
precision = precision_score(target, predictions, average='macro')  # Use 'macro' for multi-class
classification
recall = recall_score(target, predictions, average='macro')
f1 = f1_score(target, predictions, average='macro')
conf_matrix = confusion_matrix(target, predictions)
```

```
# Print the metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```

Console Output:

```
/usr/local/bin/python3.11
/Users/srilalithnampally/Classes/AppML_Assignments/Assignment_2/Q2.py
Accuracy: 0.9167
Precision: 0.8992
Recall: 0.8992
F1 Score: 0.8992
Confusion Matrix:
[[ 6  1]
 [ 1 16]]

Process finished with exit code 0
```

Image of Tree:



Discussion on Part 1 Tree vs Part 2 Tree:

The tree structure produced from manual calculations is identical to that generated by the software. However, there appears to be a labeling reversal in the code-generated tree, where 'Frilly=0' is labeled as True and 'Frilly=1' as False, which might make the manually calculated tree seem like a mirror image of the software-generated one. Upon examining the tree's leaves, it's evident that nodes with zero entropy are reached at the same decision level in both trees. Additionally, the manual calculations accurately reflect the count of samples filtered at each decision point.

Part 3:

Code:

```
import numpy as np
from sklearn import tree
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
import pydotplus


def loadData(filename):
    # Load data
    data = pd.read_excel(filename)
    return data


def writegraphtofile(clf, featurelabels, filename):
    dot_data = tree.export_graphviz(clf, feature_names=featurelabels, out_file=None)
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_png(filename)


def preprocessData(data):
    mix_col = []
    for col in data.columns:
        unique_types = data[col].apply(type).unique()
        if len(unique_types) > 1:
            mix_col.append(col)

    data[mix_col] = data[mix_col].astype(str)
    data.replace("?", np.nan, inplace=True)
    # Separate features and target
    X = data.iloc[:, :-1]  # All columns except the last one
    y = data.iloc[:, -1]  # The last column

    X = X.drop(['encounter_id', 'patient_nbr'], axis=1)

    # Identify categorical columns (modify this list based on your dataset)
```

```
   categorical_features = ["race", "gender", "age", "weight", "admission_type_id",
"discharge_disposition_id",
                        "admission_source_id", "payer_code", "medical_specialty", "diag_1", "diag_2",
"diag_3",
                        'max_glu_serum', 'A1Cresult', 'change', 'diabetesMed']  # Example column
names

   # Identify Ordinal columns (for medications)
   ordinal_features_medications = ['metformin', 'repaglinide', 'nateglinide', 'chlorpropamide',
'glimepiride',
                                   'acetohexamide', 'glipizide', 'glyburide', 'tolbutamide', 'pioglitazone',
                                   'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone', 'tolazamide', 'examide',
                                   'citoglipton', 'insulin', 'glyburide-metformin', 'glipizide-metformin',
                                   'glimepiride-pioglitazone', 'metformin-rosiglitazone', 'metformin-
pioglitazone', ]
   medication_order = [['No', 'Steady', 'Up', 'Down']]
   medication_order_for_all_columns = medication_order * len(ordinal_features_medications)

   # Imputers for categorical and ordinal features
   categorical_imputer = SimpleImputer(strategy='constant', fill_value='unknown')
   ordinal_imputer = SimpleImputer(strategy='most_frequent')

   # Identify numerical columns (assuming all other columns are numerical if not categorical or
ordinal)
   numerical_features = [col for col in X.columns if col not in categorical_features +
ordinal_features_medications]

   # Imputers
   numerical_imputer = SimpleImputer(strategy='mean')  # Impute numerical columns with their
mean

   # Encoder
   one_hot_encoder = OneHotEncoder()
   ordinal_encoder = OrdinalEncoder(categories=medication_order_for_all_columns)

   # Update ColumnTransformer to include numerical imputation
   preprocessor = ColumnTransformer(
      transformers=[
         ('cat', Pipeline([('imputer', categorical_imputer), ('encoder', one_hot_encoder)]),
categorical_features),
         ('ord_meds', Pipeline([('imputer', ordinal_imputer),
                                ('encoder', ordinal_encoder)]),
          ordinal_features_medications),
         ('num', numerical_imputer, numerical_features)
```

```
        ],
        remainder='passthrough'  # Ensure no column is left unprocessed
    )

    # Check for NaN values post-imputation

    print("Finished Pre-Processing, Starting Transformation")
    # Apply transformations
    X_encoded = preprocessor.fit_transform(X)

    # Extracting new feature names after preprocessing
    feature_names = []

    # Getting feature names for categorical features
    for cat_feature, one_hot in zip(categorical_features,
                                    preprocessor.named_transformers_['cat']['encoder'].categories_):
        feature_names.extend([f"{cat_feature}_{category}" for category in one_hot])

    # Adding ordinal feature names as is
    feature_names.extend(ordinal_features_medications)

    # Adding numerical feature names as is
    feature_names.extend(numerical_features)

    print("Finished Transformation of dataset")
    return X_encoded, y, feature_names


def MultiClassPrediction(X_encoded, y):
    # Split data for the multiclass classification model
    X_train_multi, X_test_multi, y_train_multi, y_test_multi = train_test_split(X_encoded, y,
test_size=0.2,
                                                                                random_state=42)

    # Initialize and train the multiclass classifier
    clf_multiclass = tree.DecisionTreeClassifier(criterion="entropy", max_depth=4)
    clf_multiclass.fit(X_train_multi, y_train_multi)

    # Predictions for multiclass classification
    predictions_multi = clf_multiclass.predict(X_test_multi)
    predictions_multi_train = clf_multiclass.predict(X_train_multi)

    # Evaluate the multiclass model
    accuracy_multi = accuracy_score(y_test_multi, predictions_multi)
```

```
    print(f"Multiclass Classification Accuracy with Test Set: {accuracy_multi:.4f}")

    accuracy_multi_train = accuracy_score(y_train_multi, predictions_multi_train)
    print(f"Multiclass Classification Accuracy with Train Set: {accuracy_multi_train:.4f}")
    # Plotting the multiclass classification tree
    return clf_multiclass


def BinaryClassification(X_encoded, y):
    # Binary target preprocessing
    y_binary = y.replace({'NO': 0, '<30': 1, '>30': 1})

    # Split data for the binary classification model
    X_train_bin, X_test_bin, y_train_bin, y_test_bin = train_test_split(X_encoded, y_binary,
test_size=0.2,
                                                        random_state=42)

    # Initialize and train the binary classifier
    clf_binary = tree.DecisionTreeClassifier(criterion="entropy", max_depth=4)
    clf_binary.fit(X_train_bin, y_train_bin)

    # Predictions for binary classification
    predictions_bin_test = clf_binary.predict(X_test_bin)
    predictions_bin_train = clf_binary.predict(X_train_bin)

    # Evaluate the binary model
    accuracy_bin_test = accuracy_score(y_test_bin, predictions_bin_test)
    print(f"Binary Classification Accuracy with Test Set: {accuracy_bin_test:.4f}")

    accuracy_bin_train = accuracy_score(y_train_bin, predictions_bin_train)
    print(f"Binary Classification Accuracy with Train Set: {accuracy_bin_train:.4f}")

    # Plotting the binary classification tree
    return clf_binary


if __name__ == "__main__":
    file_location = "diabetic_data.xlsx"
    dataframe = loadData(file_location)  # created a dataframe
    feature_labels = dataframe.columns[:-1].tolist()

    X, Y, feature_names = preprocessData(dataframe)  # Preprocessing the dataframe, and
obtaining (features, target)
    print("Dataset Is Ready \n")
```

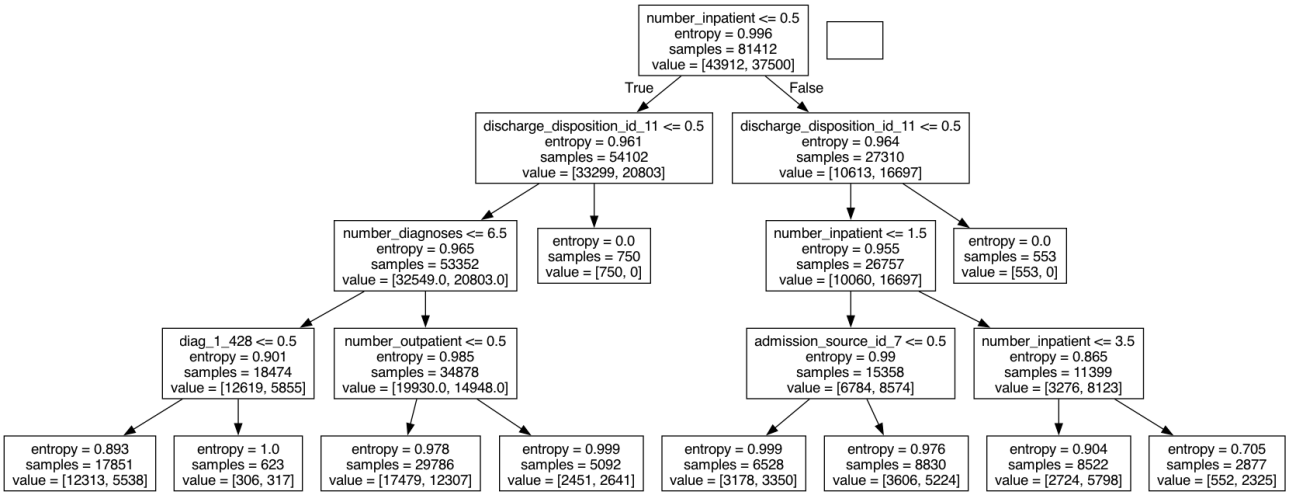
```
print("Training Dataset")

clf_binary = BinaryClassification(X, Y)
writegraphtofile(clf_binary, feature_names, "Assignment_2" + "tree_pic_binary.png")
clf_multiclass = MultiClassPrediction(X, Y)
writegraphtofile(clf_multiclass, feature_names, "Assignment_2" + "tree_pic_multiclass.png")
```

Console Output:

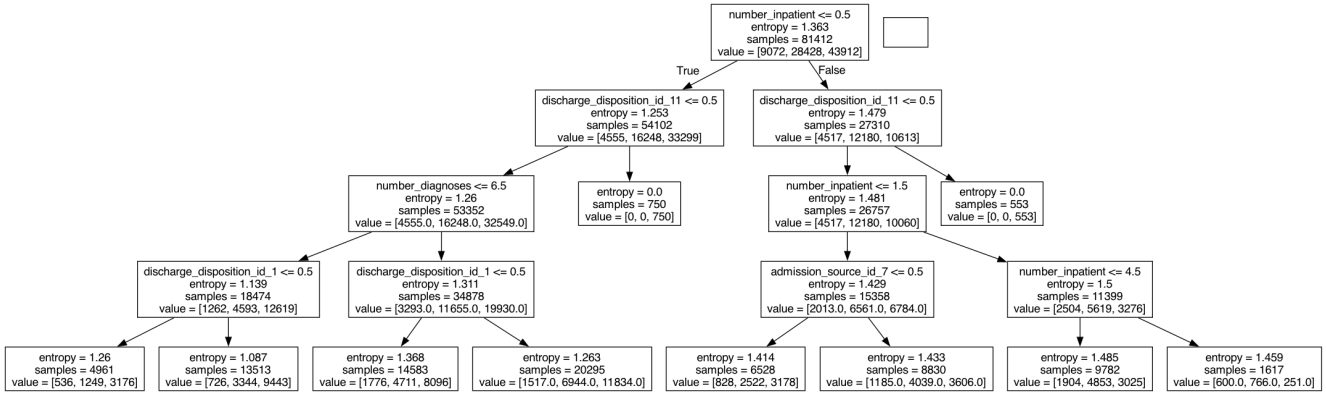
```
/usr/local/bin/python3.11
/Users/srilalithnampally/Classes/AppML_Assignments/Assignment_2/Q3.py
Finished Pre-Processing, Starting Transformation
Finished Transformation of dataset
Dataset Is Ready

Training Dataset
/Users/srilalithnampally/Classes/AppML_Assignments/Assignment_2/Q3.py:130:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future
version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to
the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  y_binary = y.replace({'NO': 0, '<30': 1, '>30': 1})
Binary Classification Accuracy with Test Set: 0.6209
Binary Classification Accuracy with Train Set: 0.6234
Multiclass Classification Accuracy with Test Set: 0.5748
Multiclass Classification Accuracy with Train Set: 0.5735

Process finished with exit code 0
```

Images Generated:

Binary Classification:



Multiclass Classification:



Discussion on the differences between the Trees:

- Entropy: The multi-class tree has higher entropy values, indicating more complexity and less purity at each node compared to the binary tree.
- Leaf Purity: The binary tree has leaves with zero entropy, showing perfect class separation, while the multi-class tree does not, suggesting some classes remain mixed at the leaves.
- Class Counts in Leaves: The binary tree's leaves show counts for two classes, whereas the multi-class tree's leaves have counts for multiple classes, reflecting the complexity of separating more classes.
- Tree Structure: The binary tree might be simpler with fewer splits to achieve class separation, while the multi-class tree may require more splits to handle additional classes.
- Decision Boundaries: The binary tree suggests clearer decision boundaries with leaves often fully belonging to one class, whereas the multi-class tree indicates less distinct boundaries due to the presence of multiple classes.