Title: CSE469 Project

Date: Dec.6 – 2020

Name: Hon Ching Li, Yinuo Chen

UBIT: 50185646, 50220710

Introduction: We are trying to predict whether an individual income exceeds $50K per year based on census data. Our model is able to predict whether a person's income exceeds $50K per year based on age, workclass, fnlwgt, education, maritial-status, occupation, relationship, race, sex, capital-gain/loss, hours-per-week and native-country. Whenever there is a new census data come in but without their income's status, our model is able to predict whether each person within the census data earn more than $50K per year or not. Our correctness score is at least 83%.

Formulation: The data mining task can be formulated it into Decision Tree Classification. The inputs are age, workclass, education, maritial-status, occupation, relationship, race, sex, capital-gain/loss, hours-per-week and native-country of the adults. The expected output contains of 2 types of class label. Whether the person income <=$50K or >$50K.

Datasets: We got the dataset adult.data from https://archive.ics.uci.edu/ml/datasets/Adult. For example, the feature age, capital gain, capital lose, hours per week are the continuous data type, workclass, education, marital status, occupation, relationship, race, sex are the categorical type.

For preprocessing the data we had these tasks:

Pre-task: dummy up the class label.

Task1: Check to see how many categories within each feature and decide whether to 'dummy' the feature (convert to numeric).

Task2: If the categories within a feature not too much, then dive into the feature to see how 'convincing' of all the categories, if majority amount is within one category. Then we can ignore the rest categories by grouping them into one value/category.

Task3: If majority observations within a feature belong to one value/category, combine the other values/categories as one.

Task4: Create a dummy list, make all categorical features 'dummy', mean convert str to numeric. s.t. Machine Model can use. Need to drop the original column. It's redundant.

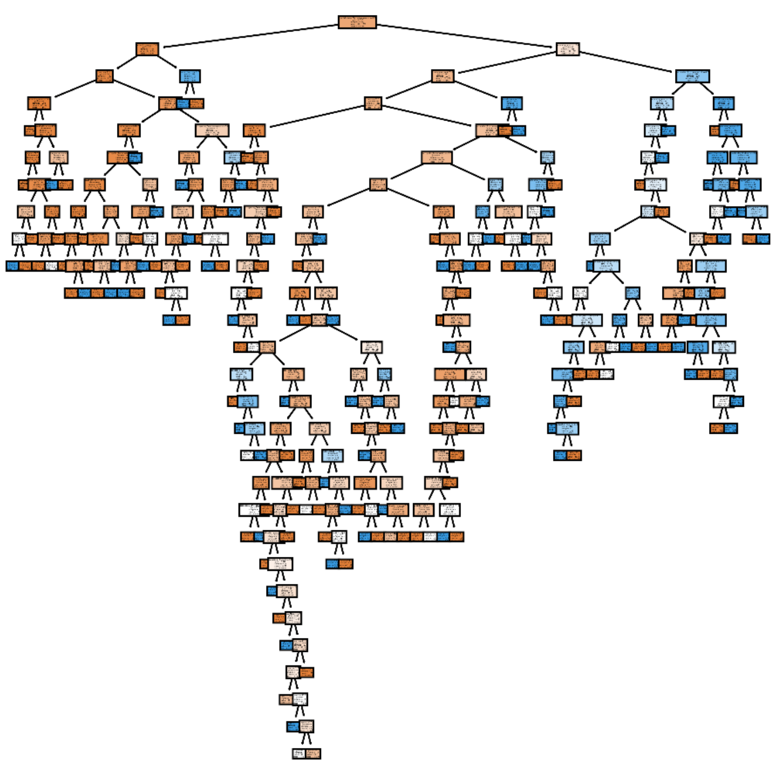
Task5: Handle with the missing value, replacing all the missing values with mean.

Algorithm: We applied decision tree classification algorithm to predict whether each person within the census data earn <=$50K or >$50K per year.

Experiments: Within 32562 observations of census dataset, we split 80% of all the observations as our training set and 20% as our testing set. We evaluate the output using sklearn.metrics import confusion_matrix, accuracy_score. We use supervised learning, we put in the actual test set with ground truth and the prediction from our model into the accurancy_score class. That will compute the correctness score for us, we also print out both as np array to visually compare the actual truth with our predictions.

```
confusion Matrix:
[[4444  475]
 [ 661  933]]
accurancy score: 0.8255796100107478
```

Decision Tree:



Challenge: our data type consists of various data types, in order to use the methods within sklearn library, we had to convert the types of our dataset to appropriate types that the classes of sklearn library can accept.

Code:

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import  confusion_matrix, accuracy_score
import sklearn.feature_selection


def decisionTreeClassification():
    ds = pd.read_csv('adult.csv')
    ds = ds.iloc[0:, :]
    X = ds.iloc[:, :-1]

    # print(type(X))
    # print(X)
    #
    # print("X info ===")
    # print(X.info())
    #
    # print("==========preprocess_phase===============")

    '''
    All the parameters, attr and methods of pandas dataframe -
        https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.DataFrame.html
    Pandas Series attr, methods -
        https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.Series.html
    Sklearn confusion matrix explain
        https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
    This is the preprocessing phase.
        Pre-task: dummy up the class label.
        Task1: Check to see how many categories within each feature and
decide whether to 'dummy' the feature (convert
        to numeric).
        Task2: If the categories within a feature not too much, then dive
into the feature to see how 'convincing' of
        all the categories, if majority amount is within one category. Then
we can ignore the rest categories by
        grouping them into one categories.
        Task3: If majority observations within a feature belong to one
value/category, combine the other
        values/categories as one.
```

```
        Task4: Create a dummy list, make all categorical features 'dummy',
mean convert str to numeric. s.t. Machine
        Model can use. Need to drop the original column. It's redundant.
        Task5: Handle with the missing value, replace with mean or whatever.
        Useful tool - https://scikit-
learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html
        Task6: Implement a histogram function to show distribution of
features.
        Task7: Split the dataset into training_set and test_set
        #Optional Task: choose the best 18 features. S.t. the decision tree
can be visualize. Also able to get
        their colnames which used for visualization later.
        Task8: apply training_set to the decision tree classification model.
        Task9: Predict the test_set in the model, and compare the correctness
of predicted test_set with ground truth.
        Task10: Visualization the model.
    '''
    # TODO Pre-task
    ds['label'] = [0 if x=='<=50K' else 1 for x in ds['label']]
    y = ds.label # object
    #print(y) # pandas.core.series.Series

    # TODO Task1
    for colnames in X.columns:
        #print(type(colnames))
        # print(type(X[colnames]))
        # print(X[colnames])
        if X[colnames].dtypes == 'object':
            #print(X[colnames].unique())
            unique_cat = len(X[colnames].unique())
            print(f"feature: {colnames}, has {unique_cat} unique categories")

    # TODO Task2
    # able to show each value/category counts within feature 'native_country'

print(pd.DataFrame(data=X['native_country']).value_counts().sort_values(ascen
ding=False).head(10))

    # TODO Task3
    X['native_country'] = ['United-States' if x == 'United-States' else
'Other' for x in X['native_country']]
    print(pd.DataFrame(data=X['native_country']).value_counts())

    # TODO Task4
    desire_dummy = ['workclass', 'education', 'marital_status', 'occupation',
'relationship', 'race',
                    'sex', 'native_country']

    # it will drop the original columns, no need to worried.
    X = pd.get_dummies(data=X, columns=desire_dummy)
    print(X.info()) # able to display the dtype of all features and count of
them

    # TODO Task5
    # check null value status in dataframe
    print(pd.DataFrame(data=X).isnull().sum().sort_values(ascending =
False).head())
```

```
    impute = SimpleImputer(missing_values=np.nan, strategy='median')
    impute.fit(X)
    X = pd.DataFrame(data=impute.transform(X), columns= X.columns)
    #print(X)

print(pd.DataFrame(data=X).isnull().sum().sort_values(ascending=False).head()
)

    # TODO Task6
    #histogram(X['age'])




    #TODO Task7
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    # print(type(X_train))
    # print(X_train)
    # print(X_test)
    print(y_train)
    print(y_test.value_counts())



    # TODO Optional Task
    #print(X.shape)

    select = sklearn.feature_selection.SelectKBest(k = 18)
    feature_selected = select.fit(X_train, y_train)
    index_selected = feature_selected.get_support(indices=True)
    print(index_selected)

    colnames_selected = []
    for i in index_selected:
        colnames_selected.append(X.columns[i])
    print(colnames_selected)

    X_train_selected = X_train[colnames_selected]
    X_test_selected = X_test[colnames_selected]
    print(X_train_selected)


    # TODO Task8
    #print("Task8 ========================================")
    classifier = DecisionTreeClassifier(random_state=1)
    classifier.fit(X_train_selected, y_train)


    # TODO Task9
    print("Task9 ========================================")
    y_pred = classifier.predict(X_test_selected)

    #print(type(y_test))
    #print(y_test)
    y_test = np.array(y_test)
    #print(y_test.reshape(len(y_test), 1))
```

```
    #print(type(y_pred))
    #print(y_pred.reshape(len(y_pred), 1))

    print(np.concatenate([y_test.reshape(len(y_test), 1),
y_pred.reshape(len(y_pred), 1)], axis=1))

    confusionM = confusion_matrix(y_test, y_pred)
    print("confusion Matrix: ")
    print(confusionM)

    print(f"accurancy score: {accuracy_score(y_test, y_pred)}")


    # TODO Task10

    className = ['<=50K', '>50K']

    fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(8, 8), dpi=600)

    #tree.plot_tree(classifier, feature_names=colnames_selected,
class_names=className, filled=True)

    #fig.savefig('DecisionTree.png')

    # tree.export_graphviz(classifier,
    #                       out_file="tree.dot",
    #                       feature_names=colnames_selected,
    #                       class_names=className,
    #                       filled=True)








def histogram(series_column):
    plt.hist(x=series_column, edgecolor = 'black', linewidth = 1)
    plt.title(label=f"{series_column.name} histogram distribution")
    plt.xlabel(xlabel=series_column.name)
    plt.ylabel(ylabel="Frequence")
    plt.show()



if __name__ == '__main__':
    #testing()
    decisionTreeClassification()
```