Double-click (or enter) to edit

Project: Machine Learning of Salary and Demographic Factors Name: Shaohua Feng Supervisor:

Double-click (or enter) to edit

```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import confusion_matrix
#from sklearn.metrics import plot_confusion_matrix
```

```
# from sklearn.metrics import plot_confusion_matrix doesn't work, so
!pip install plot_confusion_matrix
```

```
Collecting plot_confusion_matrix
  Downloading plot_confusion_matrix-0.0.2-py3-none-any.whl (3.6 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from plot_confusion_matrix) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from plot_confusion_matrix) (1.23.5)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (4.44.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->plot_confusion_matrix) (2.8.2
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->plot_confusion_matr
Installing collected packages: plot_confusion_matrix
Successfully installed plot_confusion_matrix-0.0.2
```



```
########################################
## Read data and data wrangling
########################################
# read in data loaded in google drive
file_path_1 = '/content/drive/My Drive/adult.data'
adult_1= pd.read_csv(file_path_1,header=None)
file_path_2 = '/content/drive/My Drive/adult.test.txt'
adult_2= pd.read_csv(file_path_2,header=None)
adult=pd.concat([adult_1, adult_2], ignore_index=True)
```

```
# add column names
cols=['age','workclass','fnlwgt','education','education-num','marital-status','occupation','relationship','race','sex','capital-gain','capital-
adult.columns=cols

# add y column to data frame. target=1 for label '>50k' and y=0 for label '<=50k'
adult['target']=np.where(adult['label']==' >50K',1,0)
#adult['target'] = adult['target'].astype(bool)
#
print(adult.describe())
adult.dtypes
adult.info()
```

```
                age        fnlwgt  education-num  capital-gain  capital-loss  \
count  48842.000000  4.884200e+04   48842.000000  48842.000000  48842.000000
mean      38.643585  1.896641e+05      10.078089   1079.067626     87.502314
std       13.710510  1.056040e+05       2.570973   7452.019058    403.004552
min       17.000000  1.228500e+04       1.000000      0.000000      0.000000
25%       28.000000  1.175505e+05       9.000000      0.000000      0.000000
50%       37.000000  1.781445e+05      10.000000      0.000000      0.000000
75%       48.000000  2.376420e+05      12.000000      0.000000      0.000000
max       90.000000  1.490400e+06      16.000000  99999.000000   4356.000000

       hours-per-week        target
count    48842.000000  48842.000000
mean        40.422382      0.160538
std         12.391444      0.367108
min          1.000000      0.000000
25%         40.000000      0.000000
50%         40.000000      0.000000
75%         45.000000      0.000000
max         99.000000      1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             48842 non-null  int64
```

```
 1   workclass       48842 non-null  object
 2   fnlwgt          48842 non-null  int64
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64
 5   marital-status  48842 non-null  object
 6   occupation      48842 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  int64
 11  capital-loss    48842 non-null  int64
 12  hours-per-week  48842 non-null  int64
 13  native-country  48842 non-null  object
 14  label           48842 non-null  object
 15  target          48842 non-null  int64
dtypes: int64(7), object(9)
memory usage: 6.0+ MB
```

```
# Data Manipulation: replace '?' with None
adult['workclass']=adult['workclass'].replace(' ?',None)
adult['occupation']=adult['occupation'].replace(' ?',None)
adult['native-country']=adult['native-country'].replace(' ?',None)
```

```
# charactegorical columns
cols_cat=['workclass','education','marital-status','occupation','relationship','race','sex','native-country']

for x in cols_cat:
  adult[x] = adult[x].astype('category')
  #print(x)

adult.dtypes
```

```
age                  int64
workclass         category
fnlwgt               int64
education         category
education-num        int64
marital-status    category
occupation        category
relationship      category
race              category
sex               category
capital-gain         int64
capital-loss         int64
hours-per-week       int64
native-country    category
label               object
```



```
target               int64
dtype: object
```

```
# delete missing value
adult_cleaned=adult.dropna()
print(len(adult_cleaned))
```

```
45222
```

```
# drop original label
del adult_cleaned['label']
adult_cleaned.head(5)
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0

```
############################
# Functions used by the machine learning algorithms
#Normalizing numeric data
def normalize(x):
  if x.dtype == 'int' or x.dtype == 'float':
    return ((x - min(x)) / (max(x) - min(x)))
  else:
    return x
```

```
#Converting categorical data to dummmy/one-hot variables
def dummy(x):
  cat_col=['workclass','education','marital-status','occupation','relationship','race','sex','native-country']
  x = pd.get_dummies(x, columns=cat_col, prefix = cat_col)
  return x
#print the dataset
#adult_new.head(5)

```

```
# Print out Accuracy
def printAcc(y_test,y_pred):
  from sklearn.metrics import accuracy_score

  accuracy = accuracy_score(y_test, y_pred)
  print(f"Accuracy: {accuracy}")
```

```
# print out confusion matrix
def printConfusion(y_test, y_pred):

  from sklearn.metrics import confusion_matrix

  cf=confusion_matrix(y_test, y_pred)
  print(cf)
  tn, fp, fn, tp=cf.ravel()
  print ("TP: ", tp,", FP: ", fp,", TN: ", tn,", FN:", fn)
```

```
#print precision, recall, and accuracy from the perspective of each of the class (0 and 1 for German dataset)
def printReport(y_test,y_pred):

  from sklearn.metrics import classification_report
  from sklearn import metrics

  print(classification_report(y_test, y_pred))
```



```
##################################################
# Decision tree
##################################################

# Decision tree without oversampling and normalization
# I want to use it as baseline to proof that oversampling and normalization improves decision tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Create a deep copy of the data frame adult_cleaned and name it adult_new
# adult_new is not normalized or oversampled
# I will use adult_new as the base line
adult_new=adult_cleaned.copy(deep=True)

#X = list(set(list(adult_cleaned)) - set(['target']))
X = adult_new.drop('target', axis=1)
y = adult_new['target']

# dummy variable
X=dummy(X)

# split train and test. test size is 0.35
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)

# create decision tree classifier
dtc = DecisionTreeClassifier(random_state=52)
# Train the model on the training data
dtc.fit(X_train, y_train)
# Make predictions on the test data
y_pred = dtc.predict(X_test)

print("Decision Tree for data without normalization and oversampling")
# Print Accuracy
printAcc(y_test,y_pred)

# Print Confusion Matrix
print("")
print("Confustion Matrix")
printConfusion(y_test, y_pred)

# Print Diagnosis
print("")
printReport(y_test,y_pred)


```

```
Decision Tree for data without normalization and oversampling
Accuracy: 0.7998483699772555

Confustion Matrix
[[11507  1697]
 [ 1471  1153]]
TP:  1153 , FP:  1697 , TN:  11507 , FN: 1471

              precision    recall  f1-score   support

           0       0.89      0.87      0.88     13204
           1       0.40      0.44      0.42      2624

    accuracy                           0.80     15828
   macro avg       0.65      0.66      0.65     15828
weighted avg       0.81      0.80      0.80     15828
```



```
# Decision tree with normalization but no oversampling

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Create a deep copy of the data frame adult_cleaned and name it adult_new
# adult_new is not normalized or oversampled
# I will use adult_new as the base line
adult_dt_norm = adult_cleaned.copy(deep=True)
X = adult_dt_norm.drop('target', axis=1)
y = adult_dt_norm['target']

# Normalization the numerical columns
num_cols = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
# why this code doesn't work?
#X[num_cols] = X[num_cols].apply(lambda x:normalize(x), axis=0)
X_num = X[num_cols]
X_num_normalized = X_num.apply(normalize, axis=0)
# combine the normalized numerical columns with the categorical columns
X = pd.concat([X_num_normalized, X.drop(num_cols, axis=1)], axis=1)

#print(X.head(5))
#print(X.tail(5))
# dummy variable
X=dummy(X)
print(X.head(5))
# split train and test. test size is 0.35
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=52)

# create decision tree classifier
dtc_norm = DecisionTreeClassifier(random_state=52)
# Train the model on the training data
dtc_norm.fit(X_train, y_train)
# Make predictions on the test data
y_pred = dtc_norm.predict(X_test)

print("Decision Tree for data with normalization but no oversampling")
# Print Accuracy
printAcc(y_test,y_pred)

# Print Confusion Matrix
print("")
print("Confustion Matrix")
printConfusion(y_test, y_pred)

# Print Diagnosis
```

```
print("")
printReport(y_test,y_pred)
```

```
         age    fnlwgt  education-num  capital-gain  capital-loss  \
0  0.301370  0.043350       0.800000       0.02174           0.0
1  0.452055  0.047274       0.800000       0.00000           0.0
2  0.287671  0.136877       0.533333       0.00000           0.0
3  0.493151  0.149792       0.400000       0.00000           0.0
4  0.150685  0.219998       0.800000       0.00000           0.0

   hours-per-week  workclass_ Federal-gov  workclass_ Local-gov  \
0        0.397959                       0                     0
1        0.122449                       0                     0
2        0.397959                       0                     0
3        0.397959                       0                     0
4        0.397959                       0                     0

   workclass_ Never-worked  workclass_ Private  ...  native-country_ Portugal  \
0                        0                   0  ...                         0
1                        0                   0  ...                         0
2                        0                   1  ...                         0
3                        0                   1  ...                         0
4                        0                   1  ...                         0

   native-country_ Puerto-Rico  native-country_ Scotland  \
0                            0                         0
1                            0                         0
2                            0                         0
3                            0                         0
4                            0                         0

   native-country_ South  native-country_ Taiwan  native-country_ Thailand  \
0                      0                       0                         0
1                      0                       0                         0
2                      0                       0                         0
3                      0                       0                         0
4                      0                       0                         0

   native-country_ Trinadad&Tobago  native-country_ United-States  \
0                                0                              1
1                                0                              1
2                                0                              1
3                                0                              1
4                                0                              0
```



```
   native-country_ Vietnam  native-country_ Yugoslavia
0                        0                           0
1                        0                           0
2                        0                           0
3                        0                           0
4                        0                           0

[5 rows x 105 columns]
Decision Tree for data with normalization but no oversampling
Accuracy: 0.8035759413697245

Confustion Matrix
[[11590  1636]
 [ 1473  1129]]
TP:  1129 , FP:  1636 , TN:  11590 , FN: 1473
```

```
# Decision tree with normalization and random oversampling
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler

# Create a deep copy of the data frame adult_dt_norm_ros
# X is normalized and random oversampled
adult_dt_norm_ros = adult_cleaned.copy(deep=True)
X = adult_dt_norm_ros.drop('target', axis=1)
y = adult_dt_norm_ros['target']

# Normalization the numerical columns
num_cols = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
X_num = X[num_cols]
X_num_normalized = X_num.apply(normalize, axis=0)
# combine the normalized numerical columns with the categorical columns
X = pd.concat([X_num_normalized, X.drop(num_cols, axis=1)], axis=1)

# dummy variable
X=dummy(X)

# split train and test. test size is 0.35
X_train_norm, X_test_norm, y_train_norm, y_test_norm = train_test_split(X, y, test_size=0.35, random_state=52)

#Random Oversampling
ros = RandomOverSampler(sampling_strategy='auto', random_state=52)
X_train_ros, y_train_ros = ros.fit_resample(X_train_norm, y_train_norm)

# create decision tree classifier for normalized data with random oversampling
dtc_norm_ros = DecisionTreeClassifier(random_state=52)
# Train the model on the training data
dtc_norm.fit(X_train_ros, y_train_ros)
# Make predictions on the test data
y_pred_norm_ros = dtc_norm.predict(X_test_norm)

print("Decision Tree for data with normalization and random oversampling")
# Print Accuracy
printAcc(y_test_norm,y_pred_norm_ros)

# Print Confusion Matrix
print("")
print("Confustion Matrix")
printConfusion(y_test_norm, y_pred_norm_ros)

# Print Diagnosis
```



```
print("")
printReport(y_test_norm,y_pred_norm_ros)
```

```
Decision Tree for data with normalization and random oversampling
Accuracy: 0.8037022997220116

Confustion Matrix
[[11656  1570]
 [ 1537  1065]]
TP:  1065 , FP:  1570 , TN:  11656 , FN: 1537

              precision    recall  f1-score   support

           0       0.88      0.88      0.88     13226
           1       0.40      0.41      0.41      2602

    accuracy                           0.80     15828
   macro avg       0.64      0.65      0.64     15828
weighted avg       0.80      0.80      0.80     15828
```

Double-click (or enter) to edit

```
# Cross validated decision tree normalized no over sampling
from sklearn.model_selection import cross_val_score, cross_val_predict, KFold
# StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from imblearn.pipeline import Pipeline
from sklearn.metrics import classification_report

# Create a deep copy of the data frame adult_cleaned and name it adult_c_dt_norm_ros
# X is normalized and random oversampled
adult_c_dt = adult_cleaned.copy(deep=True)
X = adult_c_dt.drop('target', axis=1)
y = adult_c_dt['target']

# Normalization the numerical columns
num_cols = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
X_num = X[num_cols]
X_num_normalized = X_num.apply(normalize, axis=0)
# combine the normalized numerical columns with the categorical columns
X = pd.concat([X_num_normalized, X.drop(num_cols, axis=1)], axis=1)

# dummy variable
X=dummy(X)

# Create a Decision Tree classifier
c_dt= DecisionTreeClassifier()

# Create a pipeline with oversampling and the decision tree classifier
model = Pipeline([('ros', ros), ('dt', c_dt_rom)])

# Set up cross-validation using StratifiedKFold
cv = KFold(n_splits=10, shuffle=True, random_state=52)

# Perform cross-validation and obtain predicted labels
y_pred = cross_val_predict(c_dt,X,y, cv=cv)

# Calculate and print classification report
print("Classification Report:\n", classification_report(y, y_pred))

# Perform cross-validation
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv)

# Print the mean accuracy across all folds
print("Mean Accuracy:", scores.mean())

```



```
Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.88      0.88     37714
           1       0.41      0.43      0.42      7508

    accuracy                           0.80     45222
   macro avg       0.65      0.65      0.65     45222
weighted avg       0.81      0.80      0.81     45222

Mean Accuracy: 0.8046967325976435
```

```
# cross validated decision tree normalized random oversampling
# I use pipeline!
from sklearn.model_selection import cross_val_score, cross_val_predict, KFold
# StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import RandomOverSampler
from imblearn.pipeline import Pipeline
from sklearn.metrics import classification_report
import pandas as pd

# Create a deep copy of the data frame adult_cleaned and name it adult_c_dt_norm_ros
# X is normalized and random oversampled
adult_c_dt_norm_ros = adult_cleaned.copy(deep=True)
X = adult_c_dt_norm_ros.drop('target', axis=1)
y = adult_c_dt_norm_ros['target']

# Normalization the numerical columns
num_cols = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
X_num = X[num_cols]
X_num_normalized = X_num.apply(normalize, axis=0)
# combine the normalized numerical columns with the categorical columns
X = pd.concat([X_num_normalized, X.drop(num_cols, axis=1)], axis=1)

# dummy variable
X=dummy(X)

# Create a RandomOverSampler
ros = RandomOverSampler()

# Create a Decision Tree classifier
c_dt_rom = DecisionTreeClassifier()

# Apply oversampling to X and y
X_resampled, y_resampled = ros.fit_resample(X, y)
```

```
# Create a pipeline with oversampling and the decision tree classifier
model = Pipeline([('ros', ros), ('dt', c_dt_rom)])

# Set up cross-validation using StratifiedKFold
cv = KFold(n_splits=10, shuffle=True, random_state=52)

# Perform cross-validation and obtain predicted labels
y_pred = cross_val_predict(model, X_resampled,y_resampled, cv=cv)

# Calculate and print classification report
print("Classification Report:\n", classification_report(y_resampled, y_pred))

# Perform cross-validation
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv)

# Print the mean accuracy across all folds
print("Mean Accuracy:", scores.mean())

################### comments: how to print out all those stuff?
```

```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.88      0.93     37714
           1       0.89      1.00      0.94     37714

    accuracy                           0.94     75428
   macro avg       0.94      0.94      0.94     75428
weighted avg       0.94      0.94      0.94     75428

Mean Accuracy: 0.8071071215644292
```



```
##########################################
# Logistic regression
##########################################

# Logistic regression with cross validation and no oversampling
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, KFold

# Create a deep copy of the data frame adult_cleaned and name it adult_c_dt_norm_ros
# X is normalized and random oversampled
adult_c_log = adult_cleaned.copy(deep=True)
X = adult_c_log.drop('target', axis=1)
y = adult_c_log['target']

# Normalization the numerical columns
num_cols = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
X_num = X[num_cols]
X_num_normalized = X_num.apply(normalize, axis=0)
# combine the normalized numerical columns with the categorical columns
X = pd.concat([X_num_normalized, X.drop(num_cols, axis=1)], axis=1)

# dummy variable
X=dummy(X)

# create a logistic model
model = LogisticRegression(max_iter=1000)  # Increase max_iter if needed for convergence

# Set up 10-fold cross-validation
kfold = KFold(n_splits=10, shuffle=True, random_state=52)

# Perform cross-validation and get the accuracy scores for each fold
scores = cross_val_score(model, X, y, cv=kfold)

# Print the accuracy for each fold and the mean accuracy
for i, score in enumerate(scores, 1):
    print(f'Fold {i}: {score}')

print(f'Mean Accuracy: {scores.mean()}')
```

```
Fold 1: 0.8425823568428035
Fold 2: 0.853415874419633
Fold 3: 0.8423264042459089
Fold 4: 0.8491817779743477
Fold 5: 0.8524988942945599
Fold 6: 0.8429898275099513
Fold 7: 0.8516143299425033
```

```
Fold 8: 0.8398938522777533
Fold 9: 0.8403361344537815
Fold 10: 0.8423264042459089
Mean Accuracy: 0.8457165856207152
```



```
# Logistic regression with cross validation and random oversampling

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, KFold
from imblearn.over_sampling import RandomOverSampler

# Create a deep copy of the data frame adult_cleaned and name it adult_c_dt_norm_ros
# X is normalized and random oversampled
adult_c_log_ros = adult_cleaned.copy(deep=True)
X = adult_c_log_ros.drop('target', axis=1)
y = adult_c_log_ros['target']

# Normalization the numerical columns
```

```
##################################################
# KNN
##################################################
# KNN with cross validation no oversampling
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, KFold

# Create a deep copy of the data frame adult_cleaned and name it adult_c_knn
adult_c_knn = adult_cleaned.copy(deep=True)
X = adult_c_knn.drop('target', axis=1)
y = adult_c_knn['target']

# Normalization the numerical columns
num_cols = ['age','fnlwgt','education-num','capital-gain','capital-loss','hours-per-week']
X_num = X[num_cols]
X_num_normalized = X_num.apply(normalize, axis=0)
# combine the normalized numerical columns with the categorical columns
X = pd.concat([X_num_normalized, X.drop(num_cols, axis=1)], axis=1)

# dummy
X=dummy(X)

```