KNN

```
# -*- coding: utf-8 -*-
"""Untitled3.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1ERktupahigX3qs3_ToYFLKOBEXviHxtd
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

from sklearn.model_selection import cross_val_score

data = pd.read_csv("adult.csv")
data.head(5)

data.info()

data.columns

data.isnull().sum()

data.shape

data['workclass'].unique()

data['education'].unique()

data['native-country'].unique()
```

```
del data['fnlwgt']
del data['educational-num']
del data['race']
del data['capital-gain']
del data['capital-loss']

data.columns

label_encoder = preprocessing.LabelEncoder()

data['gender'] = label_encoder.fit_transform(data['gender'])
data['workclass'] = label_encoder.fit_transform(data['workclass'])
data['education'] = label_encoder.fit_transform(data['education'])
data['marital-status'] = label_encoder.fit_transform(data['marital-status'])
data['occupation'] = label_encoder.fit_transform(data['occupation'])
data['relationship'] = label_encoder.fit_transform(data['relationship'])
data['native-country'] = label_encoder.fit_transform(data['native-country'])
data['income'] = label_encoder.fit_transform(data['income'])

data['workclass'].unique()

data.head()

X_train, X_test, y_train, y_test = train_test_split(data[['age', 'gender', 'workclass', 'hours-per-week',
                                                          'education', 'native-country', 'income']],
                                                    data.age, test_size=0.3,
                                                    random_state=0)
X_train.shape, X_test.shape

scaler = RobustScaler()

X_train_scaled_ro = scaler.fit_transform(X_train)
X_test_scaled_ro = scaler.transform(X_test)

print('Mean value of age, gender, workclass, hours-per-week, education, native-country, income features:
', X_train_scaled_ro.mean(axis=0))
print('Std value of age, gender, workclass, hours-per-week, education, native-country, income features: ',
X_test_scaled_ro.std(axis=0))

plt.hist(X_train_scaled_ro[:,1], bins=8)

plt.hist(X_train_scaled_ro[:,2], bins=20)

sns.pairplot(data)

plt.figure(figsize=(10,5))
```

```
total = float(len(data['income']))

a = sns.countplot(x = 'workclass',data = data)

for f in a.patches:
    height = f.get_height()
    a.text(f.get_x() + f.get_width()/2., height+3, '{:1.2f}'.format((height/total)*100),ha="center")
plt.show()

data['workclass'].unique()

data['workclass'].value_counts()

plt.figure(figsize=(10,5))

a = float(len(['income']))

a = sns.countplot(x='education',data = data)
for s in a.patches:
    height = s.get_height()
    a.text(s.get_x()+s.get_width()/2.,height+3,'{:1.2f}'.format((height/total)*100),ha='center')
plt.show()

plt.figure(figsize=(15,8))
total = float(len(data) )

ax = sns.countplot(x = "occupation", data = data)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}'.format((height/total)*100),
            ha="center")
plt.show()

data['occupation'].unique()

plt.figure(figsize=(5,5))
total = float(len(data) )

ax = sns.countplot(x = "income", data = data)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,
            height + 3,
            '{:1.2f}'.format((height/total)*100),
            ha="center")
plt.show()

data['income'].value_counts()

fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(data.corr(), annot = True, linewidths = .5, fmt = '.1f',ax = ax, cmap = 'Blues')
plt.show()

data.columns

x = data[['age', 'workclass', 'education', 'marital-status', 'occupation',
       'relationship', 'gender', 'hours-per-week', 'native-country']]
y = data['income']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)

x.shape, y.shape

print('x_train: ', x_train.shape)
print('x_test: ', x_test.shape)
print('y_train: ', y_train.shape)
print('y_test: ', y_test.shape)

knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
predict = knn.predict(x_test)
predict
knn.score(x_test, y_test)*100

print('Accuracy Score: ', accuracy_score(y_test, predict))

print('Precision Score: ', precision_score(y_test, predict))

print('Recall Score: ', recall_score(y_test, predict))

print('F1 Score: ', f1_score(y_test, predict))

print(classification_report(y_test, predict))

cm = confusion_matrix(y_test, predict)
cm

ax = sns.heatmap(cm/np.sum(cm), annot=True,  fmt='.2%', cmap='Blues')

ax.set_title('Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

plt.show()

accuracy_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score = cross_val_score(knn, x, y, cv=10)
    accuracy_rate.append(score.mean())

error_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    score = cross_val_score(knn, x, y, cv=10)
    error_rate.append(1-score.mean())

plt.figure(figsize=(10,6))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker='o', markerfacecolor='red',
markersize=10)

plt.title('Error rate VS K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')

knn = KNeighborsClassifier(n_neighbors = 10)

knn.fit(x_train, y_train)
pred = knn.predict(x_test)

print('WITH K = 10')
print('\n')
print(confusion_matrix(y_test, pred))
print('\n')
print(classification_report(y_test, pred))

knn = KNeighborsClassifier(n_neighbors = 1)

knn.fit(x_train, y_train)
pred = knn.predict(x_test)

print('WITH K = 1')
print('\n')
print(confusion_matrix(y_test, pred))
print('\n')
print(classification_report(y_test, pred))
```

Decision Tree

```
# -*- coding: utf-8 -*-
"""Untitled4.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/19CiwX3HwvDXMQsgDiJyUfCA_jQ6JWKZv
"""

import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.decomposition import PCA
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import fbeta_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv('adult.csv')

sns.heatmap(df.isnull())

cat_df=df.select_dtypes('object')
cat_df.head()

arr1=[]
```

```
for item in cat_df['workclass']:
    if (item == '?'):
        arr1.append(item)
print('Length of missing vals in workclass column:')
print(len(arr1))
print('\n')
arr2=[]
for item in cat_df['occupation']:
    if (item == '?'):
        arr2.append(item)
print('Length of missing vals in occupation column:')
print(len(arr2))

null_data=((2809+2799)/(48842-(2809+2799)))*100
print(null_data)

x=df.select_dtypes(object)

oe=OrdinalEncoder()
cat_df=oe.fit_transform(cat_df)

cat_df

cat_df1=pd.DataFrame(data=cat_df,columns=x.columns)
cat_df1

num_df1=df.select_dtypes(int)
num_df1

final_df=pd.concat([num_df1,cat_df1],axis=1)
final_df

X=final_df.drop('income',axis=1)
y=final_df['income']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=50)

tree=DecisionTreeRegressor(max_depth=7)

tree.fit(X_train,y_train)

predictions=tree.predict(X_test)
print(predictions)

pred2=pd.DataFrame(data=predictions,columns=['predictions'])
pred2['predictions']
```

```
def num(n):
    if(n < 0.5):
        return 0
    else:
        return 1

x=pred2['predictions'].apply(num)
x.unique()

result2=classification_report(x,y_test)
print(result2)

cm = confusion_matrix(x, y_test)
cm
```