Assignment2

October 21, 2020

1 Preprocessing of Discrete Data 1 - Cancer Dataset

[1]:
```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# visualization
import seaborn as sns

from math import sqrt
```

[2]:
```
# reading data
data = pd.read_csv('dataDiscrete/data.csv')
```

[3]:
```
# first 5 entries of dataframe
data.head()
```

[3]:
```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
```

```
1          0.08474          0.07864          0.0869               0.07017
2          0.10960          0.15990          0.1974               0.12790
3          0.14250          0.28390          0.2414               0.10520
4          0.10030          0.13280          0.1980               0.10430

   …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  …          17.33           184.60      2019.0            0.1622
1  …          23.41           158.80      1956.0            0.1238
2  …          25.53           152.50      1709.0            0.1444
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractional_dimension_worst  Unnamed: 32
0                     0.11890          NaN
1                     0.08902          NaN
2                     0.08758          NaN
3                     0.17300          NaN
4                     0.07678          NaN

[5 rows x 33 columns]
```

[ ]:

[4]:

```
#data preprocessing
data_count=data.diagnosis.value_counts(normalize = True)
data_count = pd.Series(data_count)
data_count = pd.DataFrame(data_count)
data_count.index = ['Benign', 'Malignant']

data_count['Percent'] = 100*data_count['diagnosis']/sum(data_count['diagnosis'])
data_count['Percent'] = data_count['Percent'].round().astype('int')
data_count
```

[4]:

```
           diagnosis  Percent
Benign      0.627417       63
Malignant   0.372583       37
```

[5]:

```
# changing categorical data to numerical
data['diagnosis']= data['diagnosis'].map({'M':1,'B':0})
```

```
# checking the different values contained in the diagnosis column
#Benign : 0
#Malign : 1

data['diagnosis'].value_counts()

# it shows that all the data are unique that is.
data['id'].nunique()

# here data in last column is empty and id is unique, so removing this does not␣
↪affect data

data.drop(data.columns[[-1, 0]], axis=1, inplace=True)

data.shape
```

[5]: (569, 31)

[6]:
```
### Applying Dimensionality Reduction

from sklearn.preprocessing import StandardScaler  # Import
scaler = StandardScaler() # Instantiate
scaler.fit(data) # Fit
scaled_data = scaler.transform(data) # Transfor



# Applying PCA

from sklearn.decomposition import PCA  # Import

pca = PCA(n_components=2)  # Instantiate

pca.fit(scaled_data)  # Fit

X_pca = pca.transform(scaled_data)  # Transform

print(" Data dimensions before reduction:",scaled_data.shape)
print(" Data dimensions after reduction:",X_pca.shape)
```

```
 Data dimensions before reduction: (569, 31)
 Data dimensions after reduction: (569, 2)
```

[7]:
```
# Splitting the data set into train and test set
from sklearn.model_selection import train_test_split

features = data.drop(columns = ['diagnosis'])
```

```
target = data['diagnosis']
X_train1, X_test1, y_train1, y_test1 = train_test_split(features, target,␣
↪test_size = 0.3,random_state = 0)

print ("Train data set size : ", X_train1.shape)
print ("Test data set size : ", X_test1.shape)
```

```
Train data set size :  (398, 30)
Test data set size :  (171, 30)
```

[8]:
```
# applying 5NN , id3, Naive bayes
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# visualization
import seaborn as sns
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import  StratifiedKFold
from sklearn.metrics import precision_score, recall_score
from math import sqrt
# Spot-Check Algorithms
models = []

models.append(( 'KNN' , KNeighborsClassifier(n_neighbors = 5, metric =␣
↪'minkowski', p = 2)))
models.append(( 'ID3' , DecisionTreeClassifier(criterion = 'entropy',␣
↪random_state = 0)))
models.append(( 'NB' , GaussianNB()))


# Test options and evaluation metric
num_folds = 5
num_instances = len(X_train1)
seed = 7
scoring =  'accuracy'

# Test options and evaluation metric
num_folds = 5
num_instances = len(X_train1)
seed = 7
scoring =  'accuracy'
results = []
```

```
names = []
print("Cancer dataset ")

for name, model in models:
 #applying straified 5 fold technique
 cv_results = StratifiedKFold(n_splits=5, random_state=1)
   # calculating scores
 scores = cross_val_score(model, X_train1, y_train1, scoring='accuracy',␣
 ↪cv=cv_results, n_jobs=-1)
 model.fit(X_test1, y_test1)
 pred = model.predict(X_test1)

 results.append(model.score(X_test1, y_test1))
 names.append(name)
 msg = "%s: %f (%f)" % (name,scores.mean(), scores.std())
 print('\n')
#mean and standard deviation of prediction
 print('     Mean    Standard Deviation')
 print(msg)
#printing values for comparision ;ike confusion matrix and accuracy
 print('confusion matrix')
 print(confusion_matrix(y_test1,pred))

 # Print out confusion matrix
 cmat = confusion_matrix(y_test1, pred)
#print(cmat)
 print('TP - True Negative {}'.format(cmat[0,0]))
 print('FP - False Positive {}'.format(cmat[0,1]))
 print('FN - False Negative {}'.format(cmat[1,0]))
 print('TP - True Positive {}'.format(cmat[1,1]))
 Accuracy = (np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat)))
 print('Accuracy Rate:',Accuracy)
 #interval = z * sqrt( (accuracy * (1 - accuracy)) / n)
    #confidence interval with 80%
 confidence_interval = 1.282 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
 print("True error (with 80% Confidence Interval): ", confidence_interval)
 confidence_interval2 = 1.645 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
 print("True error (with 90% Confidence Interval): ", confidence_interval2)
    # printing pessimistic error of the
 print('Pessimistic error: {}'.format(np.divide(np.
 ↪sum([cmat[0,1],cmat[1,0]]),np.sum(cmat))))
 print('\n')

print('-> 5-Fold cross-validation accurcay score for the training data for␣
 ↪above classifiers')
print('test results',results)
```

Cancer dataset

```
     Mean    Standard Deviation
KNN: 0.919679 (0.018417)
confusion matrix
[[106   2]
 [  4  59]]
TP - True Negative 106
FP - False Positive 2
FN - False Negative 4
TP - True Positive 59
Accuracy Rate: 0.9649122807017544
True error (with 80% Confidence Interval):  0.011824096788044459
True error (with 90% Confidence Interval):  0.015172105472958762
Pessimistic error: 0.03508771929824561
```

```
     Mean    Standard Deviation
ID3: 0.916923 (0.032763)
confusion matrix
[[108   0]
 [  0  63]]
TP - True Negative 108
FP - False Positive 0
FN - False Negative 0
TP - True Positive 63
Accuracy Rate: 1.0
True error (with 80% Confidence Interval):  0.0
True error (with 90% Confidence Interval):  0.0
Pessimistic error: 0.0
```

```
     Mean    Standard Deviation
NB: 0.942244 (0.014880)
confusion matrix
[[104   4]
 [  6  57]]
TP - True Negative 104
FP - False Positive 4
FN - False Negative 6
TP - True Positive 57
Accuracy Rate: 0.9415204678362573
True error (with 80% Confidence Interval):  0.015078679724556699
```

```
True error (with 90% Confidence Interval):  0.019348227883694048
Pessimistic error: 0.05847953216374269


-> 5-Fold cross-validation accurcay score for the training data for above
classifiers
test results [0.9649122807017544, 1.0, 0.9415204678362573]
```

[9]:
```
# applying ripper algorithm
import wittgenstein as lw
ripper_clf = lw.RIPPER() # Or irep_clf = lw.IREP() to build a model using IREP
ripper_clf.fit( X_train1, y_train1) # Or pass X and y data to .fit
ripper_clf
```

[9]:
```
<RIPPER(dl_allowance=64, k=2, max_rules=None, prune_size=0.33,
random_state=None, verbosity=0, max_rule_conds=None, max_total_conds=None,
n_discretize_bins=10)>
```

[10]:
```
# ruelset of ripper algo
ripper_clf.ruleset_
```

[10]:
```
<Ruleset [radius_mean=17.93-20.26] V [perimeter_worst=119.4-140.9] V
[radius_mean=20.26-28.11] V [concavity_mean=0.15-0.22] V
[perimeter_worst=105.9-119.4^radius_mean=13.46-14.25] V
[texture_worst=30.96-34.85^radius_mean=15.34-17.93] V
[perimeter_worst=105.9-119.4^texture_worst=28.74-30.96] V
[texture_mean=21.38-22.55^radius_mean=14.25-15.34] V
[compactness_mean=0.19-0.31]>
```

[11]:
```
pred = ripper_clf.predict(X_test1)

print("Confusion Matrix")
print(confusion_matrix(y_test1,pred))
print('\n')
print("classification report")
print(classification_report(y_test1,pred))
# Print out confusion matrix
cmat = confusion_matrix(y_test1, pred)
#print(cmat)
print('TP - True Negative {}'.format(cmat[0,0]))
print('FP - False Positive {}'.format(cmat[0,1]))
print('FN - False Negative {}'.format(cmat[1,0]))
print('TP - True Positive {}'.format(cmat[1,1]))
Accuracy = (np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat)))
print('Accuracy Rate:',Accuracy)
 #interval = z * sqrt( (accuracy * (1 - accuracy)) / n)
    #confidence interval with 80%
```

```
confidence_interval = 1.282 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
print("True error (with 80% Confidence Interval): ", confidence_interval)
confidence_interval2 = 1.645 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
print("True error (with 90% Confidence Interval): ", confidence_interval2)
print('Pessimistic Error: {}'.format(np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.
↪sum(cmat))))
print('\n')
```

```
Confusion Matrix
[[102   6]
 [  9  54]]


classification report
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       108
           1       0.90      0.86      0.88        63

    accuracy                           0.91       171
   macro avg       0.91      0.90      0.90       171
weighted avg       0.91      0.91      0.91       171

TP - True Negative 102
FP - False Positive 6
FN - False Negative 9
TP - True Positive 54
Accuracy Rate: 0.9122807017543859
True error (with 80% Confidence Interval):  0.018178511005578008
True error (with 90% Confidence Interval):  0.02332578050247724
Pessimistic Error: 0.08771929824561403
```

2 Preprocessing of Continuous Data - Census/Adult Income dataset

[12]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
sns.set(style="darkgrid")
from time import time
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder
import scipy.stats as stats
from scipy.stats import kurtosistest

# displaying for notebooks
%matplotlib inline

columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',␣
↪'marital-status', 'occupation',
           'relationship', 'race','sex', 'capital-gain', 'capital-loss',␣
↪'hours-per-week', 'native-country', 'income']

# Load the Census dataset
census = pd.read_csv('dataConti/adult.data', header=None, names=columns,␣
↪skipinitialspace=True)


# Success - Display the first 5 record
display(census.head())
```

```
   age         workclass  fnlwgt  education  education-num  \
0   39         State-gov   77516  Bachelors             13
1   50  Self-emp-not-inc   83311  Bachelors             13
2   38           Private  215646    HS-grad              9
3   53           Private  234721       11th              7
4   28           Private  338409  Bachelors             13

       marital-status         occupation   relationship   race     sex  \
0       Never-married       Adm-clerical  Not-in-family  White    Male
1  Married-civ-spouse    Exec-managerial        Husband  White    Male
2            Divorced  Handlers-cleaners  Not-in-family  White    Male
3  Married-civ-spouse  Handlers-cleaners        Husband  Black    Male
4  Married-civ-spouse     Prof-specialty           Wife  Black  Female

   capital-gain  capital-loss  hours-per-week native-country income
0          2174             0              40  United-States  <=50K
1             0             0              13  United-States  <=50K
2             0             0              40  United-States  <=50K
3             0             0              40  United-States  <=50K
4             0             0              40           Cuba  <=50K
```

[13]:
```
import warnings
warnings.filterwarnings("ignore")

# Drop the fnlwgt column which is useless for later analysis
```

```
census = census.drop('fnlwgt', axis=1)

# Read in test data
census_test = pd.read_csv('dataConti/adult.test', header=None, skiprows=1,␣
 ↪names=columns, skipinitialspace=True)

# Drop the fnlwgt column which is useless for later analysis
census_test = census_test.drop('fnlwgt', axis=1)

# Remove '.' in income column
census_test['income'] = census_test['income'].apply(lambda x: '>50K' if␣
 ↪x=='>50K.' else '<=50K')

# Review several rows and shape of data set
display(census_test.head())
display(census_test.shape)
```

```
   age  workclass     education  education-num      marital-status  \
0   25    Private          11th              7       Never-married
1   38    Private       HS-grad              9  Married-civ-spouse
2   28  Local-gov    Assoc-acdm             12  Married-civ-spouse
3   44    Private  Some-college             10  Married-civ-spouse
4   18          ?  Some-college             10       Never-married

          occupation relationship   race     sex  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4                  ?    Own-child  White  Female             0             0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

```
(16281, 14)
```

[14]:
```
# Convert '?' to NaNs and remove the entries with NaN value
object_col = census.select_dtypes(include=object).columns.tolist()
for col in object_col:
    census.loc[census[col]=='?', col] = np.nan
    census_test.loc[census_test[col]=='?', col] = np.nan
```

```
# Perform an mssing assessment in each column of the dataset.
col_missing_pct = census.isna().sum()/census.shape[0]
col_missing_pct.sort_values(ascending=False)
```

[14]:
```
occupation        0.056601
workclass         0.056386
native-country    0.017905
income            0.000000
hours-per-week    0.000000
capital-loss      0.000000
capital-gain      0.000000
sex               0.000000
race              0.000000
relationship      0.000000
marital-status    0.000000
education-num     0.000000
education         0.000000
age               0.000000
dtype: float64
```

[15]:
```
# Removing data entries with missing value
adult_train = census.dropna(axis=0, how='any')
adult_test = census_test.dropna(axis=0, how='any')

# Show the results of the split
print("After removing the missing value:")
print("Training set has {} samples.".format(adult_train.shape[0]))
print("Testing set has {} samples.".format(adult_test.shape[0]))
```

```
After removing the missing value:
Training set has 30162 samples.
Testing set has 15060 samples.
```

[16]:
```
num_col = adult_train.dtypes[adult_train.dtypes != 'object'].index

# Split the data into features and target label
income_raw = adult_train['income']
feature_raw = adult_train.drop('income', axis=1)

income_raw_test = adult_test['income']
feature_raw_test = adult_test.drop('income', axis=1)

# Log transform the skewed feature highly-skewed feature 'capital-gain' and␣
↪'capital-loss'.
skewed = ['capital-gain', 'capital-loss']
census_log = pd.DataFrame(data=feature_raw)
census_log[skewed] = feature_raw[skewed].apply(lambda x: np.log(x + 1))
```

```
census_log_test = pd.DataFrame(data=feature_raw_test)
census_log_test[skewed] = feature_raw_test[skewed].apply(lambda x: np.log(x +␣
↪1))


# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)

features_log_minmax_transform = pd.DataFrame(data = census_log)
features_log_minmax_transform[num_col] = scaler.
↪fit_transform(census_log[num_col])

# Transform the test data set
features_log_minmax_transform_test = pd.DataFrame(data = census_log_test)
features_log_minmax_transform_test[num_col] = scaler.
↪transform(census_log_test[num_col])

# Show an example of a record with scaling applied
display(features_log_minmax_transform.head())
display(features_log_minmax_transform_test.head())
```

```
        age         workclass  education  education-num      marital-status  \
0  0.301370         State-gov  Bachelors       0.800000       Never-married
1  0.452055  Self-emp-not-inc  Bachelors       0.800000  Married-civ-spouse
2  0.287671           Private    HS-grad       0.533333            Divorced
3  0.493151           Private       11th       0.400000  Married-civ-spouse
4  0.150685           Private  Bachelors       0.800000  Married-civ-spouse

          occupation   relationship   race     sex  capital-gain  \
0       Adm-clerical  Not-in-family  White    Male      0.667492
1    Exec-managerial        Husband  White    Male      0.000000
2  Handlers-cleaners  Not-in-family  White    Male      0.000000
3  Handlers-cleaners        Husband  Black    Male      0.000000
4     Prof-specialty           Wife  Black  Female      0.000000

   capital-loss  hours-per-week native-country
0           0.0        0.397959  United-States
1           0.0        0.122449  United-States
2           0.0        0.397959  United-States
3           0.0        0.397959  United-States
4           0.0        0.397959           Cuba


        age  workclass     education  education-num      marital-status  \
0  0.109589    Private          11th       0.400000       Never-married
1  0.287671    Private       HS-grad       0.533333  Married-civ-spouse
2  0.150685  Local-gov    Assoc-acdm       0.733333  Married-civ-spouse
```

```
3  0.369863    Private  Some-college       0.600000  Married-civ-spouse
5  0.232877    Private          10th       0.333333       Never-married

          occupation   relationship   race   sex  capital-gain  capital-loss  \
0  Machine-op-inspct      Own-child  Black  Male      0.000000           0.0
1    Farming-fishing        Husband  White  Male      0.000000           0.0
2    Protective-serv        Husband  White  Male      0.000000           0.0
3  Machine-op-inspct        Husband  Black  Male      0.777174           0.0
5      Other-service  Not-in-family  White  Male      0.000000           0.0

   hours-per-week native-country
0        0.397959  United-States
1        0.500000  United-States
2        0.397959  United-States
3        0.397959  United-States
5        0.295918  United-States
```

[17]:
```
X = feature_raw

y =income_raw

# splitting data to train and test for further preprocessing

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state = 0)

from sklearn import preprocessing

categorical = ['workclass', 'education', 'marital-status', 'occupation',␣
↪'relationship', 'race', 'sex', 'native-country']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])

# Changing the income column into Numerical Value
y_train = y_train.map({'<=50K':0, '>50K':1})
```

[18]:
```
# applying 5NN , id3, Naive bayes



# Spot-Check Algorithms
models = []
```

```
models.append(( 'KNN' , KNeighborsClassifier(n_neighbors = 5, metric =␣
↪'minkowski', p = 2)))
models.append(( 'ID3' , DecisionTreeClassifier(criterion = 'entropy',␣
↪random_state = 0)))
models.append(( 'NB' , GaussianNB()))


# Test options and evaluation metric
num_folds = 5
num_instances = len(X_train.dtypes[X_train.dtypes != 'object'].index)
seed = 7
scoring =  'accuracy'

# Test options and evaluation metric
num_folds = 5
num_instances = len(X_train)
seed = 7
scoring =  'accuracy'
results = []
names = []
print("For Census dataset ")

for name, model in models:

 cv_results = StratifiedKFold(n_splits=5, random_state=1)
 scores = cross_val_score(model, X_train, y_train, scoring='accuracy',␣
↪cv=cv_results, n_jobs=-1)

 model.fit(X_test, y_test)
 pred = model.predict(X_test)

 results.append(model.score(X_test, y_test))
 names.append(name)
 msg = "%s: %f (%f)" % (name,scores.mean(), scores.std())
 print('     Mean    Standard Deviation')
 print(msg)
 print('confusion matrix')
 print(confusion_matrix(y_test,pred))
  # Print out confusion matrix
 cmat = confusion_matrix(y_test, pred)
#print(cmat)
 print('TP - True Negative {}'.format(cmat[0,0]))
 print('FP - False Positive {}'.format(cmat[0,1]))
 print('FN - False Negative {}'.format(cmat[1,0]))
 print('TP - True Positive {}'.format(cmat[1,1]))
#Accuracy Rate
```

```
    Accuracy = (np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat)))
    print('Accuracy Rate:',Accuracy)
    #interval = z * sqrt( (accuracy * (1 - accuracy)) / n)
       #confidence interval with 80%
    confidence_interval = 1.282 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
    print("True error (with 80% Confidence Interval): ", confidence_interval)
    confidence_interval2 = 1.645 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
    print("True error (with 90% Confidence Interval): ", confidence_interval2)
       # printing pessimistic error of the
       #pessimistic error
    print('Pessimistic Error: {}'.format(np.divide(np.
 ↪sum([cmat[0,1],cmat[1,0]]),np.sum(cmat))))
    print('\n')

print('-> 5-Fold cross-validation accurcay score for the training data for␣
 ↪above classifiers')
print('test results',results)
```

```
For Census dataset
     Mean    Standard Deviation
KNN: 0.818216 (0.004867)
confusion matrix
[[6307  457]
 [ 792 1493]]
TP - True Negative 6307
FP - False Positive 457
FN - False Negative 792
TP - True Positive 1493
Accuracy Rate: 0.8619736987512432
True error (with 80% Confidence Interval):  0.0030432734954399443
True error (with 90% Confidence Interval):  0.003904980421215841
Pessimistic Error: 0.13802630124875676


     Mean    Standard Deviation
ID3: 0.810969 (0.009602)
confusion matrix
[[6756    8]
 [ 103 2182]]
TP - True Negative 6756
FP - False Positive 8
FN - False Negative 103
TP - True Positive 2182
Accuracy Rate: 0.9877334512100785
True error (with 80% Confidence Interval):  0.0009711674716237582
True error (with 90% Confidence Interval):  0.0012461548290336054
Pessimistic Error: 0.012266548789921538
```

```
     Mean    Standard Deviation
NB: 0.804481 (0.006848)
confusion matrix
[[5886  878]
 [ 935 1350]]
TP - True Negative 5886
FP - False Positive 878
FN - False Negative 935
TP - True Positive 1350
Accuracy Rate: 0.7996463697646149
True error (with 80% Confidence Interval):  0.003531511658043248
True error (with 90% Confidence Interval):  0.004531463869899715
Pessimistic Error: 0.20035363023538513


-> 5-Fold cross-validation accurcay score for the training data for above
classifiers
test results [0.8619736987512432, 0.9877334512100785, 0.7996463697646149]
```

[19]:
```
# applying ripper algorithm
import wittgenstein as lw
ripper_clf = lw.RIPPER() # Or irep_clf = lw.IREP() to build a model using IREP
ripper_clf.fit( X_train, y_train) # Or pass X and y data to .fit
ripper_clf
```

[19]:
```
<RIPPER(dl_allowance=64, k=2, max_rules=None, prune_size=0.33,
random_state=None, verbosity=0, max_rule_conds=None, max_total_conds=None,
n_discretize_bins=10)>
```

[20]:
```
# ruelset of ripper algo
ripper_clf.ruleset_
```

[20]:
```
<Ruleset [marital-status=2^education-num=0.8-1.0^capital-gain=0.0-1.0^hours-per-
week=0.5-0.7] V [marital-status=2^education-num=0.6-0.8^capital-
gain=0.0-1.0^occupation=2.0-3.0] V [marital-status=2^education-
num=0.8-1.0^capital-gain=0.0-1.0^hours-per-week=0.4-0.5^occupation=7.0-9.0] V
[marital-status=2^education-num=0.8-1.0^capital-loss=0.0-0.98^hours-per-
week=0.4-0.5] V [marital-status=2^education-
num=0.6-0.8^occupation=2.0-3.0^hours-per-week=0.4-0.5^capital-loss=0.0-0.98] V
[marital-status=2^education-num=0.8-1.0^capital-
gain=0.0-1.0^education=11.0-15.0^hours-per-week=0.24-0.38] V [marital-
status=2^education-num=0.6-0.8^hours-per-
week=0.4-0.5^occupation=2.0-3.0^age=0.41-0.49] V [marital-status=2^education-
num=0.8-1.0^race=4^capital-loss=0.0-0.98] V [marital-status=2^education-
num=0.6-0.8^occupation=2.0-3.0^workclass=2^age=0.34-0.41^race=4^hours-per-
```

week=0.38-0.4] V [marital-status=2^education-num=0.6-0.8^hours-per-
week=0.4-0.5^occupation=2.0-3.0] V [marital-status=2^education-
num=0.6-0.8^capital-gain=0.0-1.0^hours-per-
week=0.4-0.5^workclass=2^occupation=7.0-9.0] V [marital-status=2^education-
num=0.8-1.0^occupation=2.0-3.0^hours-per-week=0.4-0.5] V [marital-
status=2^occupation=7.0-9.0^education-num=0.8-1.0^hours-per-week=0.38-0.4] V
[marital-status=2^education-num=0.6-0.8^capital-
gain=0.0-1.0^occupation=9.0-11.0^age=0.49-0.63] V [marital-status=2^education-
num=0.6-0.8^capital-gain=0.0-1.0^hours-per-week=0.5-0.7] V [marital-
status=2^education-num=0.6-0.8^capital-loss=0.0-0.98] V [marital-
status=2^education-num=0.6-0.8^capital-gain=0.0-1.0] V [marital-
status=2^education=11.0-15.0^education-
num=0.8-1.0^occupation=2.0-3.0^age=0.29-0.34] V [marital-
status=2^occupation=7.0-9.0^education-
num=0.8-1.0^workclass=4^education=11.0-15.0^hours-per-week=0.5-0.7] V [marital-
status=2^education=11.0-15.0^education-
num=0.8-1.0^occupation=2.0-3.0^relationship=0^workclass=2] V [marital-
status=2^education-num=0.6-0.8^hours-per-week=0.4-0.5^workclass=3] V [marital-
status=2^occupation=2.0-3.0^capital-
gain=0.0-1.0^workclass=3^education=11.0-15.0] V [marital-
status=2^occupation=7.0-9.0^education-num=0.8-1.0^workclass=3] V [marital-
status=2^education-num=0.6-0.8^occupation=7.0-9.0^workclass=2^hours-per-
week=0.4-0.5^age=0.29-0.34] V [marital-status=2^education-
num=0.6-0.8^occupation=7.0-9.0^relationship=5^age=0.23-0.29] V [marital-
status=2^occupation=2.0-3.0^capital-gain=0.0-1.0^age=0.41-0.49] V [marital-
status=2^education-num=0.6-0.8^occupation=2.0-3.0^workclass=2^hours-per-
week=0.5-0.7^age=0.18-0.23] V [marital-status=2^education=11.0-15.0^education-
num=0.8-1.0^occupation=9.0-11.0^workclass=3] V [marital-status=2^education-
num=0.6-0.8^occupation=7.0-9.0] V [marital-
status=2^education=11.0-15.0^age=0.41-0.49^hours-per-week=0.4-0.5^workclass=2] V
[marital-status=2^education=11.0-15.0^education-
num=0.8-1.0^occupation=7.0-9.0^hours-per-week=0.4-0.5^age=0.29-0.34] V [marital-
status=2^education=11.0-15.0^capital-gain=0.0-1.0^age=0.34-0.41] V [marital-
status=2^occupation=2.0-3.0^capital-gain=0.0-1.0] V [marital-
status=2^occupation=2.0-3.0^capital-loss=0.0-0.98^education=9.0-11.0] V
[marital-status=2^education=11.0-15.0^education-
num=0.8-1.0^occupation=2.0-3.0^workclass=1] V [marital-
status=2^education=11.0-15.0^occupation=2.0-3.0^workclass=0] V [marital-
status=2^education=11.0-15.0^education-
num=0.8-1.0^occupation=7.0-9.0^race=4^relationship=5] V [marital-
status=2^education=11.0-15.0^age=0.41-0.49^workclass=2^occupation=0.0-2.0^hours-
per-week=0.38-0.4] V [marital-status=2^education=11.0-15.0^education-
num=0.8-1.0] V [marital-status=2^occupation=2.0-3.0^education-
num=0.6-0.8^workclass=2^age=0.29-0.34] V [marital-status=2^education-
num=0.6-0.8^occupation=9.0-11.0] V [marital-
status=2^occupation=2.0-3.0^workclass=2^age=0.49-0.63^education=11.0-15.0] V
[marital-status=2^education-

```
num=0.53-0.6^occupation=2.0-3.0^age=0.41-0.49^workclass=3] V [marital-
status=2^education-num=0.53-0.6^capital-loss=0.0-0.98^age=0.34-0.41^hours-per-
week=0.38-0.4] V [marital-status=2^education-num=0.53-0.6^capital-
gain=0.0-1.0^race=2] V [marital-status=2^occupation=2.0-3.0^workclass=2] V
[marital-status=2^capital-gain=0.0-1.0^occupation=9.0-11.0]>
```

[21]:
```
print('Accuracy:',ripper_clf.score(X_test, y_test))
print('Pessimistic error: {}'.format(np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.
 ↪sum(cmat))))
```

```
Accuracy: 0.8872803624709913
Pessimistic error: 0.20035363023538513
```

[22]:
```
#performance of ripper
from sklearn.metrics import precision_score, recall_score
precision = ripper_clf.score(X_test, y_test, precision_score)
recall = ripper_clf.score(X_test, y_test, recall_score)
print(f'precision ripper: {precision} recall ripper: {recall}')
```

```
precision ripper: 0.0 recall ripper: 0.0
```

3 Clustering data1 - titanic

[23]:
```
#importing data

df_train= pd.read_csv("dataClustering/train.csv")
df_test= pd.read_csv("dataClustering/test.csv")




#preprocessing
df_train = df_train.drop('Name', axis=1,)
df_train = df_train.drop('Ticket', axis=1,)
df_train = df_train.drop('Fare', axis=1,)
df_train = df_train.drop('Cabin', axis=1,)

df_train['Family'] = df_train['SibSp'] + df_train['Parch'] + 1

df_train = df_train.drop('SibSp', axis=1,)
df_train = df_train.drop('Parch', axis=1,)

df_train["Age"] = df_train["Age"].fillna(df_train["Age"].median())
df_train["Embarked"] = df_train["Embarked"].fillna("S")
df_train = df_train.drop('Age', axis=1,)
df_train.head()
```

[23]:

	PassengerId	Survived	Pclass	Sex	Embarked	Family
0	1	0	3	male	S	2
1	2	1	1	female	C	2
2	3	1	3	female	S	1
3	4	1	1	female	S	2
4	5	0	3	male	S	1

[24]:
```
df1 = df_train.filter(['Pclass','Sex','Embarked','Family','Adult'], axis=1)

X = df1

df2 = df_train['Survived']

y = df2
```

[25]:
```
### Dropping the Embarked and Family column

X = X.drop('Embarked', axis=1,)
X = X.drop('Family', axis=1,)
```

[26]:
```
#testing and spliting data
features_train, features_test, labels_train, labels_test = \
    train_test_split(X, y, test_size=0.3, random_state=42)
```

[27]:
```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
features_train['Sex'] = lb_make.fit_transform(features_train['Sex'])
```

[28]:
```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
features_test['Sex'] = lb_make.fit_transform(features_test
                                              ['Sex'])
```

[29]:
```
# applying 5NN , id3, Naive bayes
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold, KFold
from statistics import mean, stdev
# visualization
import seaborn as sns
```

```python
# Spot-Check Algorithms
models = []

models.append(( 'KNN' , KNeighborsClassifier(n_neighbors = 5, metric =␣
↪'minkowski', p = 2)))
models.append(( 'ID3' , DecisionTreeClassifier(criterion = 'entropy',␣
↪random_state = 0)))
models.append(( 'NB' , GaussianNB()))


# Test options and evaluation metric
num_folds = 5
num_instances = len(features_train)
seed = 7
scoring =  'accuracy'

# Test options and evaluation metric
num_folds = 5
num_instances = len(features_train)
seed = 7
scoring =  'accuracy'
results = []
names = []
print("For Titanic dataset ")
print('model mean standard deviation')
for name, model in models:

 cv_results = StratifiedKFold(n_splits=5, random_state=1)
 scores = cross_val_score(model, X_train, y_train, scoring='accuracy',␣
↪cv=cv_results, n_jobs=-1)
 model.fit(features_train,labels_train)

 pred = model.predict(features_test)

 results.append(model.score(features_test, labels_test))
 names.append(name)
 msg = "%s: %f (%f)" % (name,scores.mean(), scores.std())
 print('     Mean    Standard Deviation')
 print(msg)
 print('confusion matrix')
 print(confusion_matrix(labels_test,pred))
  # Print out confusion matrix
 cmat = confusion_matrix(labels_test, pred)

 print('TP - True Negative {}'.format(cmat[0,0]))
 print('FP - False Positive {}'.format(cmat[0,1]))
```

```
    print('FN - False Negative {}'.format(cmat[1,0]))
    print('TP - True Positive {}'.format(cmat[1,1]))
        #accuracy
    Accuracy = (np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat)))
    print('Accuracy Rate:',Accuracy)
    #interval = z * sqrt( (accuracy * (1 - accuracy)) / n)
        #confidence interval with 80%
    confidence_interval = 1.282 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
    print("True error (with 80% Confidence Interval): ", confidence_interval)
    confidence_interval2 = 1.645 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
    print("True error (with 90% Confidence Interval): ", confidence_interval2)
        #pessimistic error
    print('Pessimistic error: {}'.format(np.divide(np.
 ↪sum([cmat[0,1],cmat[1,0]]),np.sum(cmat))))
    print('\n')
    print()

print('-> 5-Fold cross-validation accurcay score for the training data for␣
 ↪above classifiers')
print('test results',results)
```

```
For Titanic dataset
model mean standard deviation
     Mean    Standard Deviation
KNN: 0.818216 (0.004867)
confusion matrix
[[152   5]
 [ 56  55]]
TP - True Negative 152
FP - False Positive 5
FN - False Negative 56
TP - True Positive 55
Accuracy Rate: 0.7723880597014925
True error (with 80% Confidence Interval):  0.02153571514161661
True error (with 90% Confidence Interval):  0.02763358144146593
Pessimistic error: 0.22761194029850745



     Mean    Standard Deviation
ID3: 0.810969 (0.009602)
confusion matrix
[[152   5]
 [ 56  55]]
TP - True Negative 152
FP - False Positive 5
FN - False Negative 56
```

```
TP - True Positive 55
Accuracy Rate: 0.7723880597014925
True error (with 80% Confidence Interval):  0.021535715141616612
True error (with 90% Confidence Interval):  0.02763358144146593
Pessimistic error: 0.22761194029850745



     Mean    Standard Deviation
NB: 0.804481 (0.006848)
confusion matrix
[[134  23]
 [ 33  78]]
TP - True Negative 134
FP - False Positive 23
FN - False Negative 33
TP - True Positive 78
Accuracy Rate: 0.7910447761194029
True error (with 80% Confidence Interval):  0.020881954819997876
True error (with 90% Confidence Interval):  0.026794708017859988
Pessimistic error: 0.208955223880597



-> 5-Fold cross-validation accurcay score for the training data for above
classifiers
test results [0.7723880597014925, 0.7723880597014925, 0.7910447761194029]
```

[30]:
```python
# applying ripper algorithm
import wittgenstein as lw
ripper_clf = lw.RIPPER() # Or irep_clf = lw.IREP() to build a model using IREP
ripper_clf.fit( features_train,labels_train) # Or pass X and y data to .fit
ripper_clf
```

[30]:
```
<RIPPER(dl_allowance=64, k=2, max_rules=None, prune_size=0.33,
random_state=None, verbosity=0, max_rule_conds=None, max_total_conds=None,
n_discretize_bins=10)>
```

[31]:
```python
# ruelset of ripper algo
ripper_clf.ruleset_
```

[31]:
```
<Ruleset [Sex=0^Pclass=1] V [Sex=0^Pclass=2]>
```

[32]:
```python
# ripper test performance

print("Accuracy:",ripper_clf.score(features_train,labels_train))
```

```
print('Pessimistic error: {}'.format(np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.
↪sum(cmat))))
```

```
Accuracy: 0.6324237560192616
Pessimistic error: 0.208955223880597
```

[33]:
```
#performance of ripper
from sklearn.metrics import precision_score, recall_score
precision = ripper_clf.score(features_train,labels_train, precision_score)
recall = ripper_clf.score(features_train,labels_train, recall_score)
print(f'precision ripper: {precision}, recall ripper: {recall}')
```

```
precision ripper: 1.0, recall ripper: 0.008658008658008658
```

[34]:
```
pred1 = ripper_clf.predict(features_test)

print('Confusion Matrix')
print(confusion_matrix(labels_test,pred1))
print('\n')
print("Classification Report")
print(classification_report(labels_test,pred1))
```

```
Confusion Matrix
[[157   0]
 [109   2]]



Classification Report
              precision    recall  f1-score   support

           0       0.59      1.00      0.74       157
           1       1.00      0.02      0.04       111

    accuracy                           0.59       268
   macro avg       0.80      0.51      0.39       268
weighted avg       0.76      0.59      0.45       268
```

[35]:
```
# Print out confusion matrix
cmat = confusion_matrix(labels_test, pred1)
#print(cmat)
print('TP - True Negative {}'.format(cmat[0,0]))
print('FP - False Positive {}'.format(cmat[0,1]))
print('FN - False Negative {}'.format(cmat[1,0]))
print('TP - True Positive {}'.format(cmat[1,1]))
Accuracy = (np.divide(np.sum([cmat[0,0],cmat[1,1]]),np.sum(cmat)))
print('Accuracy Rate:',Accuracy)
```

```python
 #interval = z * sqrt( (accuracy * (1 - accuracy)) / n)
    #confidence interval with 80%
confidence_interval = 1.282 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
print("True error (with 80% Confidence Interval): ", confidence_interval)
confidence_interval2 = 1.645 * sqrt( Accuracy * (1 - Accuracy)/num_instances)
print("True error (with 90% Confidence Interval): ", confidence_interval2)
print('Pessimistic Error: {}'.format(np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.
 ↪sum(cmat))))
print('\n')
# Pessimistic error
print('Pessimistic Error: {}'.format(np.divide(np.sum([cmat[0,1],cmat[1,0]]),np.
 ↪sum(cmat))))
```

```
TP - True Negative 157
FP - False Positive 0
FN - False Negative 109
TP - True Positive 2
Accuracy Rate: 0.5932835820895522
True error (with 80% Confidence Interval):  0.02523021892538764
True error (with 90% Confidence Interval):  0.03237418887071971
Pessimistic Error: 0.40671641791044777


Pessimistic Error: 0.40671641791044777
```