**Feasibility of Utilizing Machine Learning Models for Cancer Rate Prediction**

Shan Chen 1, Yuanzhao Ding 2,\*

1 Department of Applied Social Sciences, The Hong Kong Polytechnic University, 11 Yuk Choi Rd, Hung Hom, Hong Kong, China. ORCID: 0000-0002-5434-4883. [chenshan1893@gmail.com](mailto:chenshan1893@gmail.com)

2 Social Sciences Division, University of Oxford, South Parks Road, Oxford, OX1 3QY, United Kingdom. ORCID: 0000-0003-0116-3648. [armstrongding@163.com](mailto:armstrongding@163.com)

\*Correspondence: [armstrongding@163.com](mailto:armstrongding@163.com)

**Table S1**. Coding for cancer incidence prediction

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

data = pd.read\_csv('20230319cancerincidence.csv')

data.head(5)

sns.set(font\_scale=2)

plt.hist(x = data.AGE, bins = 10, color = 'green')

plt.title('Age')

plt.xlabel('Score')

plt.ylabel('Number')

plt.show()

plt.hist(x = data.SEX, bins = 10, color = 'green')

plt.title('PercentPercentBelowBasic')

plt.xlabel('Score')

plt.ylabel('Number')

plt.show()

colors = iter(['Uncertain:red purple'])

with sns.axes\_style('white'):

g = sns.factorplot("AGE", data=data, aspect=2,

kind="count", color='green')

g.set\_xticklabels(step=5)

df = data.copy()

df.head()

x = data.iloc[:,0:7]

y = data.iloc[:,7]

print(x.shape)

print(y.shape)

print(x.columns)

df = data.copy()

df.head()

plt.figure(figsize = (15,12), dpi = 600)

sns.set(font\_scale=1.2)

sns.heatmap(data.corr(),annot = True, cmap='Greens',annot\_kws={"fontsize":20})

plt.figure(figsize=(15, 8))

sns.distplot(df.RATEROUND)

plt.ylabel('Frequency', fontsize=15)

plt.xlabel('Incidence Rate', fontsize=15)

plt.title('Incidence Rate Distribution', fontsize=15)

plt.show()

plt.figure(figsize=(15, 8))

sns.distplot(df.AGE)

plt.ylabel('Incidence Rate Frequency', fontsize=15)

plt.xlabel('Age', fontsize=15)

plt.title('Incidence Rate Distribution', fontsize=15)

plt.show()

x = pd.get\_dummies(x)

x.head()

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, shuffle=False)

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

print(y\_test)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.fit\_transform(x\_test)

x\_train = pd.DataFrame(x\_train)

x\_train.head()

print(x\_train)

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion\_matrix

model = DecisionTreeClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuaracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.model\_selection import cross\_val\_score

cvs = cross\_val\_score(estimator = model, X = x\_train, y = y\_train, cv = 10)

print(cvs)

print(y\_pred)

print("Mean Accuracy :", cvs.mean())

print("Variance :", cvs.std())

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.svm import SVC

model = SVC()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.neural\_network import MLPClassifier

model = MLPClassifier(hidden\_layer\_sizes = (100, 100), activation ='relu',

solver = 'adam', max\_iter = 50)

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000, linewidth=1)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

df = pd.read\_csv("20230319cancerincidence.csv", index\_col=0)

ncol = len(df.axes[1])

nrow=len(df.axes[0])

print("%s x %s" % (nrow, ncol)) #check dimension

print (df.dtypes) #check data types

df.groupby('RATEROUND').count()

n = ncol

for i in range(0,n):

if (df.iloc[:,i].dtype==object):

A= df.iloc[:,i].unique()

map\_to\_int = {name: n for n, name in enumerate(A)}

df.iloc[:,i] = df.iloc[:,i].replace(map\_to\_int)

print (df.dtypes)

df1=df.iloc[:,0:n]

features = list(df1.columns[:(n-2)])

print("\* features:", features, sep="\n")

df1.rename(columns={'y':'Target'}, inplace=True)

y = df1["RATEROUND"]

X = df1[features]

%matplotlib

import matplotlib.pyplot as plt

fig = plt.figure(figsize=(20, 20))

df1.hist(bins=20)

plt.show()

corr\_df = df1.corr()

%matplotlib inline

import seaborn

import matplotlib.pyplot as plt

print(" CorrelationMatrix")

mask = np.zeros\_like(corr\_df)

mask[np.triu\_indices\_from(mask)] = True

seaborn.heatmap(corr\_df, cmap='RdYlGn\_r', vmax=1.0, vmin=-1 ,mask = mask, linewidths=3,fmt='.1f')

plt.yticks(rotation=0,fontsize=10)

plt.xticks(rotation=90,fontsize=10)

plt.show()

print(\_\_doc\_\_)

import numpy as np

from time import time

from operator import itemgetter

from scipy.stats import randint as sp\_randint

from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

from sklearn.datasets import load\_digits

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n\_estimators=20)

def report(grid\_scores, n\_top=3):

top\_scores = sorted(grid\_scores, key=itemgetter(1), reverse=True)[:n\_top]

for i, score in enumerate(top\_scores):

print("Model with rank: {0}".format(i + 1))

print("Mean validation score: {0:.3f} (std: {1:.3f})".format(

score.mean\_validation\_score,

np.std(score.cv\_validation\_scores)))

print("Parameters: {0}".format(score.parameters))

print("")

param\_dist = {"max\_depth": [3, None],

"max\_features": sp\_randint(1, 11),

"min\_samples\_split": sp\_randint(1, 11),

"min\_samples\_leaf": sp\_randint(1, 11),

"bootstrap": [True, False],

"criterion": ["gini", "entropy"]}

n\_iter\_search = 20

random\_search = RandomizedSearchCV(clf, param\_distributions=param\_dist,

n\_iter=n\_iter\_search)

start = time()

random\_search.fit(X, y)

print("RandomizedSearchCV took %.2f seconds for %d candidates"

" parameter settings." % ((time() - start), n\_iter\_search))

pd.DataFrame(random\_search.cv\_results\_)

**Table S2**. Coding for cancer incidence prediction

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

data = pd.read\_csv('20230319cancermortility.csv')

data.head(5)

sns.set(font\_scale=2)

plt.hist(x = data.AGE, bins = 10, color = 'red')

plt.title('Age')

plt.xlabel('Score')

plt.ylabel('Number')

plt.show()

plt.hist(x = data.SEX, bins = 10, color = 'red')

plt.title('PercentPercentBelowBasic')

plt.xlabel('Score')

plt.ylabel('Number')

plt.show()

colors = iter(['Uncertain:red purple'])

with sns.axes\_style('white'):

g = sns.factorplot("AGE", data=data, aspect=2,

kind="count", color='red')

g.set\_xticklabels(step=5)

df = data.copy()

df.head()

x = data.iloc[:,0:7]

y = data.iloc[:,7]

print(x.shape)

print(y.shape)

print(x.columns)

df = data.copy()

df.head()

plt.figure(figsize = (15,12), dpi = 600)

sns.set(font\_scale=1.2)

sns.heatmap(data.corr(),annot = True, cmap='Reds',annot\_kws={"fontsize":20})

plt.figure(figsize=(15, 8))

sns.distplot(df.RATEROUND)

plt.ylabel('Frequency', fontsize=15)

plt.xlabel('Mortility Rate', fontsize=15)

plt.title('Mortility Rate Distribution', fontsize=15)

plt.show()

plt.figure(figsize=(15, 8))

sns.distplot(df.AGE)

plt.ylabel('Mortility Rate Frequency', fontsize=15)

plt.xlabel('Age', fontsize=15)

plt.title('Mortility Rate Distribution', fontsize=15)

plt.show()

x = pd.get\_dummies(x)

x.head()

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, shuffle=False)

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

print(y\_test)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.fit\_transform(x\_test)

x\_train = pd.DataFrame(x\_train)

x\_train.head()

print(x\_train)

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion\_matrix

model = DecisionTreeClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuaracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.model\_selection import cross\_val\_score

cvs = cross\_val\_score(estimator = model, X = x\_train, y = y\_train, cv = 10)

print(cvs)

print(y\_pred)

print("Mean Accuracy :", cvs.mean())

print("Variance :", cvs.std())

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.svm import SVC

model = SVC()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

from sklearn.neural\_network import MLPClassifier

model = MLPClassifier(hidden\_layer\_sizes = (100, 100), activation ='relu',

solver = 'adam', max\_iter = 50)

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

print("Training Accuracy :", model.score(x\_train, y\_train))

print("Testing Accuracy :", model.score(x\_test, y\_test))

cm = confusion\_matrix(y\_test, y\_pred)

np.set\_printoptions(threshold=50000, linewidth=1)

pd.set\_option('max\_colwidth',1)

print(cm)

print(y\_pred)

df = pd.read\_csv("20230319cancermortility.csv", index\_col=0)

ncol = len(df.axes[1])

nrow=len(df.axes[0])

print("%s x %s" % (nrow, ncol)) #check dimension

print (df.dtypes) #check data types

df.groupby('RATEROUND').count()

n = ncol

for i in range(0,n):

if (df.iloc[:,i].dtype==object):

A= df.iloc[:,i].unique()

map\_to\_int = {name: n for n, name in enumerate(A)}

df.iloc[:,i] = df.iloc[:,i].replace(map\_to\_int)

print (df.dtypes)

df1=df.iloc[:,0:n]

features = list(df1.columns[:(n-2)])

print("\* features:", features, sep="\n")

df1.rename(columns={'y':'Target'}, inplace=True)

y = df1["RATEROUND"]

X = df1[features]

%matplotlib

import matplotlib.pyplot as plt

fig = plt.figure(figsize=(20, 20))

df1.hist(bins=20)

plt.show()

corr\_df = df1.corr()

%matplotlib inline

import seaborn

import matplotlib.pyplot as plt

print(" CorrelationMatrix")

mask = np.zeros\_like(corr\_df)

mask[np.triu\_indices\_from(mask)] = True

seaborn.heatmap(corr\_df, cmap='RdYlGn\_r', vmax=1.0, vmin=-1 ,mask = mask, linewidths=3,fmt='.1f')

plt.yticks(rotation=0,fontsize=10)

plt.xticks(rotation=90,fontsize=10)

plt.show()

print(\_\_doc\_\_)

import numpy as np

from time import time

from operator import itemgetter

from scipy.stats import randint as sp\_randint

from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

from sklearn.datasets import load\_digits

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n\_estimators=20)

def report(grid\_scores, n\_top=3):

top\_scores = sorted(grid\_scores, key=itemgetter(1), reverse=True)[:n\_top]

for i, score in enumerate(top\_scores):

print("Model with rank: {0}".format(i + 1))

print("Mean validation score: {0:.3f} (std: {1:.3f})".format(

score.mean\_validation\_score,

np.std(score.cv\_validation\_scores)))

print("Parameters: {0}".format(score.parameters))

print("")

param\_dist = {"max\_depth": [3, None],

"max\_features": sp\_randint(1, 11),

"min\_samples\_split": sp\_randint(1, 11),

"min\_samples\_leaf": sp\_randint(1, 11),

"bootstrap": [True, False],

"criterion": ["gini", "entropy"]}

n\_iter\_search = 20

random\_search = RandomizedSearchCV(clf, param\_distributions=param\_dist,

n\_iter=n\_iter\_search)

start = time()

random\_search.fit(X, y)

print("RandomizedSearchCV took %.2f seconds for %d candidates"

" parameter settings." % ((time() - start), n\_iter\_search))

pd.DataFrame(random\_search.cv\_results\_)

**Table S3**. Random search for random forest (incidence rate)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **rank\_test\_score** | **mean\_test\_score** | **param\_bootstrap** | **param\_criterion** | **param\_max\_depth** | **param\_max\_features** | **param\_min\_samples\_leaf** | **param\_min\_samples\_split** |
| 1 | 0.848598 | True | gini | None | 5 | 7 | 2 |
| 2 | 0.847897 | True | entropy | None | 5 | 7 | 10 |
| 3 | 0.842017 | True | gini | None | 3 | 4 | 3 |
| 4 | 0.840660 | False | gini | None | 3 | 6 | 6 |
| 5 | 0.427748 | False | gini | 3 | 2 | 10 | 6 |
| 6 | 0.382632 | False | gini | 3 | 1 | 10 | 2 |
| 7 | 0.370059 | False | entropy | 3 | 5 | 8 | 2 |
| 8 | 0.359000 | True | gini | 3 | 1 | 1 | 7 |
| 9 | 0.325554 | True | entropy | 3 | 1 | 5 | 4 |
| 10 | NaN | True | gini | None | 9 | 2 | 1 |
| 11 | NaN | True | entropy | 3 | 10 | 2 | 7 |
| 12 | NaN | False | gini | 3 | 6 | 5 | 9 |
| 13 | NaN | True | entropy | 3 | 10 | 2 | 7 |
| 14 | NaN | False | entropy | None | 8 | 6 | 9 |
| 15 | NaN | False | entropy | None | 10 | 4 | 9 |
| 16 | NaN | False | entropy | 3 | 9 | 2 | 6 |
| 17 | NaN | False | entropy | None | 8 | 10 | 1 |
| 18 | NaN | True | entropy | 3 | 10 | 9 | 10 |
| 19 | NaN | True | entropy | None | 7 | 4 | 6 |
| 20 | NaN | False | gini | 3 | 10 | 10 | 9 |

**Table S4**. Random search for random forest (mortality rate)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **rank\_test\_score** | **mean\_test\_score** | **param\_bootstrap** | **param\_criterion** | **param\_max\_depth** | **param\_max\_features** | **param\_min\_samples\_leaf** | **param\_min\_samples\_split** |
| 1 | 0.879490 | False | entropy | None | 4 | 5 | 5 |
| 2 | 0.873357 | True | entropy | None | 5 | 7 | 5 |
| 3 | 0.859751 | True | entropy | None | 4 | 10 | 6 |
| 4 | 0.836136 | False | entropy | None | 2 | 6 | 3 |
| 5 | 0.493426 | False | entropy | 3 | 5 | 8 | 10 |
| 6 | 0.487435 | True | gini | 3 | 3 | 7 | 6 |
| 7 | 0.426492 | True | entropy | 3 | 1 | 3 | 5 |
| 8 | 0.417081 | False | entropy | 3 | 1 | 5 | 4 |
| 9 | NaN | False | entropy | None | 9 | 2 | 6 |
| 10 | NaN | True | gini | 3 | 7 | 6 | 7 |
| 11 | NaN | True | entropy | 3 | 9 | 10 | 6 |
| 12 | NaN | False | entropy | None | 10 | 6 | 9 |
| 13 | NaN | True | gini | 3 | 8 | 4 | 8 |
| 14 | NaN | True | gini | 3 | 8 | 1 | 3 |
| 15 | NaN | True | entropy | None | 7 | 10 | 4 |
| 16 | NaN | False | entropy | 3 | 9 | 1 | 3 |
| 17 | NaN | True | gini | None | 7 | 10 | 9 |
| 18 | NaN | False | entropy | None | 4 | 10 | 1 |
| 19 | NaN | False | gini | None | 9 | 5 | 9 |
| 20 | NaN | True | entropy | None | 7 | 9 | 4 |