file: 01.CODE_TEMPLATE(ALL).ipynb

Table of Contents

TO DO:

https://www.analyticsvidhya.com/blog/2016/11/solution-for-skilltest-machine-learning-revealed/ (https://www.analyticsvidhya.com/blog/2016/11/solution-for-skilltest-machine-learning-revealed/) http://www.saedsayad.com/ (http://www.saedsayad.com/) Entropy & Information Gain: https://homes.cs.washington.edu/~shapiro/EE596/notes/InfoGain.pdf (https://homes.cs.washington.edu/~shapiro/EE596/notes/InfoGain.pdf) https://homes.cs.washington.edu/~shapiro/EE596/notes.html (https://homes.cs.washington.edu/~shapiro/EE596/notes.html) https://courses.cs.washington.edu/courses/cse416/18sp/lectures.html (https://courses.cs.washington.edu/courses/cse416/18sp/lectures.html)

https://courses.cs.washington.edu/courses/cse416/18sp/slides/ (https://courses.cs.washington.edu/courses/cse416/18sp/slides/)

https://homes.cs.washington.edu/~shapiro/ (https://homes.cs.washington.edu/~shapiro/)

GENERAL PYSPARK

In [ ]:
```
#NOTE: NOT ALL CODE BLOCKS WILL RUN PROPERLY
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("CODE TEMPLATE").getOrCreate()
spark
```

IMPORT IMAGE

In [ ]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(filename='images/standadizing_data.png', height=400, width=400)
```

In [ ]:
```
%%html
<table style="width:100%;height:100%" >
  <tr>
    <th><img src="images/f1_score.png", width=500, height=100></th>
    <th><img src="images/precision_recall.png", width=300, height=100></th>
  </tr>
</table>
```

SAVE CONTENTS OF THIS CELL

In [ ]:
```
import os
file_name = os.path.join(os.path.pardir, 'src', 'data', 'test.py')
```

In [ ]:
```
%%writefile $file_name
444 + 5555
```

RUN .PY SCRIPT FROM JUPYTER

import os get_data_processing_file = os.path.join(os.path.pardir, 'src', 'data', 'Titanic_processing_script.py') !python $get_data_processing_file # Output gets written below this cell

GIT

In [ ]:
```
STEPS TO PUSH YOUR LOCAL REPO TO GIT FOR THE FIRST TIME

1) Log into https://github.com/ using your credentials
2) Click Start a project (Repoistory name: Pluralsight_Python_data_science_Abhishek_kumar)
3) Add/Edit/Delete code/files
From your mac terminal window, execute below commands:
    a) git add .
    b) git commit -m 'logging, os info commands, read file environment variable, etc'
    c) GANESH-PRO:notebooks ganeshpillai$ pwd
        /Users/pinky/Downloads/LEARNING/PLURALSIGHT_Python_data_science_Abhishek_kumar/module2/titani
c
    d) GANESH-PRO:notebooks ganeshpillai$ git remote add origin https://github.com/ganesh33/Pluralsig
ht_Python_data_science_Abhishek_kumar.git
    e) GANESH-PRO:notebooks ganeshpillai$ git push -u origin master
        Counting objects: 38, done.
        Delta compression using up to 4 threads.
        Compressing objects: 100% (35/35), done.
        Writing objects: 100% (38/38), 1.09 MiB | 0 bytes/s, done.
        Total 38 (delta 4), reused 0 (delta 0)
        remote: Resolving deltas: 100% (4/4), done.
        To https://github.com/ganesh33/Pluralsight_Python_data_science_Abhishek_kumar.git
         * [new branch]      master -> master
        Branch master set up to track remote branch master from origin.
    f) GANESH-PRO:titanic ganeshpillai$ git log --oneline
        0bb1e7e logging, os info commands, read file environment variable, etc
        f128f7a initial commit#

4) Go to below github url to view your files:
https://github.com/ganesh33/Pluralsight_Python_data_science_Abhishek_kumar
```

In [ ]:
```
'''
GANESH-PRO:titanic ganeshpillai$ pwd
/Users/pinky/Downloads/LEARNING/PLURALSIGHT_Python_data_science_Abhishek_kumar/module2/titanic

GANESH-PRO:titanic ganeshpillai$ git init
Initialized empty Git repository in
/Users/pinky/Downloads/LEARNING/PLURALSIGHT_Python_data_science_Abhishek_kumar/module2/titanic/.git/

GANESH-PRO:titanic ganeshpillai$ git add .

GANESH-PRO:titanic ganeshpillai$ git commit -m "initial commit"
[master (root-commit) f128f7a] initial commit
 33 files changed, 998 insertions(+)
 create mode 100644 .gitignore
 ...etc...
 create mode 100644 src/visualization/visualize.py
 create mode 100644 test_environment.py
 create mode 100644 tox.ini

GANESH-PRO:titanic ganeshpillai$ git status
On branch master
nothing to commit, working directory clean

GANESH-PRO:titanic ganeshpillai$ git log --oneline
f128f7a initial commit

GANESH-PRO:titanic ganeshpillai$ ls -a
.                       .gitignore              data                    references              src
..                      LICENSE                 docs                    reports                 test_
environment.py
.env                    Makefile                models                  requirements.txt        tox.i
ni
.git                    README.md               notebooks               setup.py
'''
```

PUBLIC DATASETS

In [ ]:
```
https://www.data.gov/
http://archive.ics.uci.edu/ml/datasets.html
https://github.com/awesomedata/awesome-public-datasets


https://cloud.google.com/public-datasets/
AWS public datasets

#college-scorecard
https://catalog.data.gov/dataset/college-scorecard/resource/7b9f2bb7-21c2-4df0-9453-f332cddf61d6
```

OS INFO

In [ ]:
```
import os
os.curdir
```

In [ ]:
```
os.pardir
```

In [ ]:
```
os.path.dirname(os.pardir)
```

In [ ]:
```
os.getcwd()
```

In [ ]:
```
os.path.abspath(__name__)
```

In [ ]:
```
os.getegid()
```

In [ ]:
```
os.getenv('SPARK_HOME')
```

In [ ]:
```
os.environ
```

In [ ]:
```
my_folder= !pwd
my_folder
```

In [ ]:
```
!pwd
```

In [ ]:
```

```

In [ ]:
```

```

PYSPARK DATA SCIENCE:

GENERAL

In [ ]:
```
#GENERAL:
#adultDF.count()
#adultDF.describe()
#adultDF.printSchema()
#adultDF.stat.df
#tranformedDF.take(1)
#tranformedDF.columns
#all_names.tail().unstack()
```

SELECT

In [ ]:
```
#SELECT:
adultDF.select(adultDF['EducationNum'], adultDF['Age']).show(10)
adultDF.select('Age').show(5)
adultDF['Age'] # O:Column<b'Age'>
adultDF.select('Age', 'Age_int').show(10)

type(adultDF.Age.values[0])
```

READ FILE

In [ ]:
```
### import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("RANDOM FOREST CLASSIFICAION").getOrCreate()
#spark
%matplotlib inline


adult_file='/Users/pinky/Downloads/DATA/PLURALSIGHT_Spark_machine_learning_Janani_ravi/02/adult.csv'

rawData = spark.read.format('csv')\
                    .option('header', 'false')\
                    .option('ignoreLeadingWhiteSpace', 'true')\
                    .option("inferSchema", "true")\
                    .load(adult_file)
adultDF = adult_data.drop('FnlWgt')
```

SELECT DISTICT

In [ ]:
```
# SELECT DISTICT
adultDF.select('Label').distinct().show()
adultDF.select(adultDF['Education']).distinct().show()
adultDF.select(adultDF['WorkClass']).distinct().show()
adultDF.select(adultDF['Education'], adultDF['EducationNum']).distinct().sort(adultDF['EducationNum'
]).show()
adultDF.select('Age').filter(adultDF['Age'] > 85).distinct().show()
```

MAX

In [ ]:
```
# MAX
adultDF.select(max('Age')).show()
```

SUMMARY

In [ ]:
```
# SUMMARY
adultDF.select('Age', 'CapitalGain', 'HoursPerWeek').summary().show()
```

STAT

In [ ]:
```
# STAT:
adultDF.stat.freqItems(['Age']).show(truncate=False)
```

FILTER

In [ ]:
```
# FILTER
adultDF.filter('Gender="Male"').show(5)
adultDF.filter('Age =50').show(3)
adultDF.filter('Gender="Male"').select('Age', 'CapitalGain', 'HoursPerWeek').summary().show()
adultDF.filter('Age' == max('Age')).show()
adultDF.filter('Age == 90').show(5)
```

COUNT

In [ ]:
```
  trainingData2.select('Descript').distinct().count()
```

GROUP BY

In [ ]:
```
# GROUP BY:
adultDF.groupBy('NativeCountry').count().sort('count', ascending=False).show(10)
adultDF.select('EducationNum','Age').groupby(adultDF['EducationNum']).count().show()
adultDF.select('Gender').groupby('Gender').count().show()
```

FILTER + GROUP BY

In [ ]:
```
# FILTER + GROUP BY:
adultDF.filter('Age == 90').groupBy('NativeCountry').count().show()
adultDF.filter(adultDF['WorkClass'] == 'Without-pay').show(3)
```

SORT

In [ ]:
```
#SORT
df.sort("age", ascending=False).collect()
naiveBayesPredictions.orderBy('probability').sort('probability', ascenging=False).show(3, truncate=Fa
lse)
```

LAMBDA

In [ ]:
```
# LAMBDA
lambda x: x ** 2, list6
list(map(lambda x: x ** 2, list6))
list(map(lambda x: x **2, (filter(lambda x: x %2 == 0, range(1,20)))))
```

LIST COMPREHENSION

In [ ]:
```
# LIST COMPREHENSION
list(x ** 3 for x in list5) # list comprehension
list(x % 2 == 0 for x in list5) # filter using list comprehension
list(x for x in list5 if x % 2 == 0) # filter & map using list comprehension
```

PRINT

In [ ]:
```
# PRINT A LIST
print(*my_list)
```

TYPE CONVERSION

In [ ]:
```
from pyspark.sql.types import FloatType
adultDF = adultDF.withColumn('Age_int', adultDF['Age'].cast(FloatType()))


from pyspark.sql.types import DoubleType
changedTypedf = joindf.withColumn("label", joindf["show"].cast(DoubleType()))
#or short string:
changedTypedf = joindf.withColumn("label", joindf["show"].cast("double"))

#REPLACE EXISTING COLUMN
changedTypedf = joindf.withColumn("show", joindf["show"].cast(DoubleType()))
```

Batch convert DF column type

In [ ]:
```
#String Index all columns
indexers = [StringIndexer(inputCol=i, outputCol=i + '_indexed', handleInvalid='keep') for i in catego
ricalFeatures]
for indexer in indexers:
    print(indexer.getOutputCol())

#Covert to Double all columns
from pyspark.sql.types import FloatType, DoubleType
for col_name in columns_list:
    htrainDF =  htrainDF.withColumn(col_name + '_double', htrainDF[col_name].cast(DoubleType()))
    htrainDF =  htrainDF.drop(col_name)
```

DROP COLUMN

In [ ]:
```
htrainDF = htrainDF.drop('crim_float')

drop_list = ['a column', 'another column', ...]
df.select([column for column in df.columns if column not in drop_list])
```

UDF

In [ ]:
```
from pyspark.sql.types import StringType
from pyspark.sql.functions import udf

maturity_udf = udf(lambda age: "adult" if age >=18 else "child") #, StringType())
#maturity_udf = udf(lambda age: "adult" if age >=18 else "child") , StringType()) #same as above

#df = spark.createDataFrame([('Alice', 1)], ['name', 'age'])
df = spark.createDataFrame([('Alice', 1), ('Alice2', 50)], ['name', 'age'])
df.show()
df_new= df.withColumn("maturity", maturity_udf(df.age))
df_new.show()
```

SPARK 2.3 EVALUATION

In [ ]:
```
accuracy
weightedPrecision
weightedRecall
f1
#MulticlassClassificationEvaluator evaluator4 = new MulticlassClassificationEvaluator().setMetricName
("f1");
```

SPARK STATISTICAL FUNCTIONS

SPARK DF DETAILS

In [ ]:
```
htrainDF.describe().toPandas()
htrainDF.describe().toPandas().T

htrainDF.describe().show()
htrainDF.dtypes
```

In [ ]:

RANDOM

In [ ]:
```
#RANDOM
from pyspark.sql.functions import rand, randn
df1 = spark.range(1,10)
df1.toPandas().T

column1 = rand(seed=101)  #uniform distribution

column2 = randn(seed=202).alias('normal') # normal distribution
df2 = df1.withColumn('uniform', column1).withColumn('normal2', column2)
```

SUMMARY & DISCRIPTIVE STATISTICS

In [ ]:
```
#SUMMARY & DISCRIPTIVE STATISTICS

#mean, min, max, kurtosis, length
from pyspark.sql.functions import mean, min, max, stddev, variance, avg, kurtosis, length
df2.select(mean('normal2'), min('id'), max('uniform'), stddev('normal2'), variance('id'), avg('normal
2')).show()
df2.select(kurtosis('normal2')).show()
df2.select(length('id')).show()

#corr & covar
df3 = spark.range(0, 10).withColumn('rand1', rand(seed=10)).withColumn('rand2', rand(seed=27))
df4 = df3.withColumn('id2', df3['id'] * 2)

df4.stat.corr('rand1', 'rand2')
df4.stat.cov('id', 'id2')

#CROSS TABULATION
names = ["Alice", "Bob", "Mike"]
items = ["milk", "bread", "butter", "apples", "oranges"]
df6 = spark.createDataFrame([(names[i % 3], items[i % 5 ]) for i in range(10)], ['name', 'item'])
df6.stat.crosstab('name', 'item').show()

#FREQUENT ITEMS
df6.stat.freqItems(['name', 'item'], 0.5).show(truncate=False)
```

SPARK CONFIGURATION

In [ ]:
```
# Enable Arrow-based columnar data transfers (TO MAKE .toPandas() work)
spark.conf.set("spark.sql.execution.arrow.enabled", "true")
```

In [ ]:
```
##############################################################################################
```

GENERAL PYTHON

In [ ]:
```
name.split(',')[1]
title.strip().lower()
```

3.1.2 loc & iloc

In [ ]:
```
loc: (String based)
    --> is label based index as string
    --> takes slices based on labels
    --> includes last element
iloc: (Integer based)
    --> bases on index's position
    --> uses obersation's position
```

In [ ]:
```
loc gets rows (or columns) with particular labels from the index.
iloc gets rows (or columns) at particular positions in the index (so it only takes integers).
ix usually tries to behave like loc but falls back to behaving like iloc if a label is not present in
 the index.
```

Integer Index

In [ ]:
```
import pandas as pd
raw_data_path = os.path.join(os.path.pardir, 'data', 'raw') #'../data/raw

train_data_path = os.path.join(raw_data_path, 'train.csv') #'../data/raw/train.csv'
df1 = pd.read_csv(train_data_path)
```

In [ ]:
```
df2 = df1[['PassengerId', 'Pclass', 'Name']]
```

In [ ]:
```
df2.head()
```

In [ ]:
```
df2.loc[1:3]
```

In [ ]:
```
df2.iloc[1:3]
```

In [ ]:
```
df2.loc[2]
```

In [ ]:
```
df2.iloc[2]
```

String Index

In [ ]:
```
df3 = df2.copy()
df3.index = df3.Name
df3.head()
```

In [ ]:
```
#df3.loc[1:3] #TypeError: cannot do slice indexing
```

In [ ]:
```
df3.iloc[1:3]
```

In [ ]:
```
#df3.loc[2] #TypeError: cannot do label indexing
df3.loc['Heikkinen, Miss. Laina']
```

In [ ]:
```
df3.iloc[2]
```

In [ ]:
```
df4 = df2.copy()
df4.set_index("Name", inplace=True)
df4.head()
```

In [ ]:
```
df5 = df2.copy()
df5.set_index("Name", inplace=False)
df5.head()
```

In [ ]:
```
df5.index
```

PYTHON DATA SCIENCE

DICTIONARIES

In [ ]:
```
title_dictionary.keys()
title_dictionary.values()
title_dictionary.get('dona')
```

LIST COMPREHENSION

In [ ]:
```

```

LAMBDA / MAP

In [ ]:
```
combinedDF.Name.map(lambda x: getTitle(x)).head()
combinedDF.Name.map(lambda x: getTitle(x)).unique()
combinedDF.Name.map(lambda x: getTitle(x)).unique()
combinedDF.Name.map(lambda x: getTitle(x)).value_counts(ascending=False).head()
combinedDF.Name.map(lambda x: getTitle(x)).unique().tolist()
combinedDF['Title'] = combinedDF.Name.map(lambda x: getStandardTitle(getTitle(x)))
combinedDF['Deck'] = combinedDF.Cabin.map(lambda x: getDeck(x))
```

COUNT

In [ ]:
```
combinedDF.Age.value_counts(dropna=False).head()
len(male_passengers)
combinedDF.Embarked.value_counts()
df5.Age.value_counts(dropna=False)

trainDF.count().tolist()
```

TYPE CONVERSION

In [ ]:
```
X = trainDF.loc[:, 'Adult_No':].values.astype('float')
```

DATAFRAME MANIPULATIONS

GENERAL

In [ ]:
```
combinedDF.head()
type(combinedDF.Name)
combinedDF.index
combinedDF.columns
len(male_passengers)
combinedDF.Embarked.value_counts().to_frame()
combinedDF.groupby('Embarked').size()
df5 = combinedDF.copy()
df5.Age.value_counts(dropna=False).head()
trainDF.count().tolist()
trainDF.keys()

X_train.shape, X_test.shape
```

In [ ]:
```

```

SELECT

In [ ]:
```
testDF['Survived'] = -888
combinedDF.Name[0:5]  #select first 5 from series
combinedDF[['Name', 'Age']][6:8]  # select multiple columns
trainDF.loc[800:804] # select based on index values(rows)
trainDF.iloc[800:804]  #select based on index values(rows) & columns
trainDF.loc[2:5, ['Age', 'Pclass']]  #select based on index values(rows) & columns
trainDF.iloc[2:5,3:8] # use iloc for postion based indexing
trainDF.iloc[2:5,3:8] # use iloc for postion based indexing
combinedDF.groupby(['Survived', 'Embarked']).Embarked.count().to_frame().T
```

In [ ]:
```
combinedDF = pd.concat((trainDF, testDF), sort=True)  #acis=0 (default)
```

In [ ]:
```

```

WHERE

In [ ]:
```
combinedDF['Adult'] = np.where(combinedDF.Age >= 18, 'Yes', 'No')
combinedDF['IsMale'] = np.where(combinedDF.Sex == 'male', 1,0)
```

FILTER

In [ ]:
```
combinedDF.loc[combinedDF.Sex == 'male']
combinedDF.loc[(combinedDF.Sex == 'male') & (combinedDF.Pclass == 1)]
combinedDF[combinedDF.Survived != -888].Survived.value_counts()

combinedDF[combinedDF.Embarked.isnull()] # BEST RESULT for displaying obervations with nan's

combinedDF.Age.value_counts(dropna=False, ascending=False).head()

combinedDF[(combinedDF['Embarked'] == 'S') & (combinedDF['Pclass'] == 3)].Fare.value_counts(ascending
=False).head()

combinedDF.loc[combinedDF.Fare == combinedDF.Fare.max()]
combinedDF.loc[combinedDF.Cabin =='T']
```

STATS

In [ ]:
```
trainDF.info()
combinedDF.describe()
combinedDF.describe(include='all')

combinedDF.Sex.value_counts()
combinedDF.Age.mean()
combinedDF.Age.median()
combinedDF.Fare.min(), combinedDF.Fare.max()
combinedDF.Fare.quantile(0.25), combinedDF.Fare.quantile(0.5), combinedDF.Fare.quantile(0.75)
combinedDF.Fare.var(), combinedDF.Fare.std()

combinedDF.Embarked.mode()
combinedDF.groupby(['Pclass', 'Embarked']).Fare.median()

pd.crosstab(combinedDF.Sex, combinedDF.Pclass)
combinedDF.pivot_table(index='Sex', columns='Pclass', values='Age', aggfunc='mean')
combinedDF.groupby(['Sex', 'Pclass']).Age.mean()
combinedDF.groupby(['Sex', 'Pclass']).Age.mean().unstack()

combinedDF.loc[(combinedDF.Embarked =='S') & (combinedDF.Pclass == 3), 'Fare'].median()

np.log(combinedDF.Fare + 1.0)


np.mean(y_train), np.mean(y_test)
```

GROUP BY

In [ ]:
```
combinedDF.groupby(['Pclass','Sex'])['Fare', 'Age'].median() # display selected columns
combinedDF.groupby(['Pclass']).agg({'Fare':'mean', 'Age':'median'})  # using agg


aggregations = {
    'Fare': {
            'mean_fare' : 'mean' ,
            'max_fare' : max,
            'min_fare' : np.min
            },
    'Age': {
        'range_age' : lambda x: max(x) - min(x)
    }
}
combinedDF.groupby('Pclass').agg(aggregations)

combinedDF.groupby('Sex').Age.transform('median').head()
```

READ/WRITE File/DataFrame

trainDF = pd.read_csv(train_data_path, index_col='PassengerId') combinedDF.loc[combinedDF.Survived != -888].to_csv(write_train_data_path)

MISSING VALUES

In [ ]:
```
# Solution:
#       -> Deletion
#       -> Imputation
#          --> Mean imputation  (average) (result IS impacted by outliers)
#          --> Median imputation (middle value) (result NOT impacted by outliers)
#          --> Mode imputation (most occurring) (for categorical features)
#          --> Forward fill (replace with previous value)
#          --> Backward fill (replace with next value)
#          --> Predictive Model (predict using models)


combinedDF.Embarked.nunique(dropna=False)
combinedDF.Embarked.fillna('C', inplace=True)
combinedDF[combinedDF.Age.isna()]
df5.Age.fillna(combinedDF.groupby('Sex').Age.median(), inplace=True)
df5.Age.fillna(combinedDF.groupby('Sex').Age.transform('median'), inplace=True)
combinedDF.Age.fillna(combinedDF.groupby('Title').Age.transform('median'), inplace=True)
```

In [ ]:
```

```

UNIQUE

In [ ]:
```
combinedDF.Embarked.unique()
combinedDF.Cabin.unique()
```

DROP

In [ ]:
```
combinedDF.drop(['Cabin','Name','Ticket','Parch','SibSp','Sex'], axis=1, inplace=True)
```

DATA SCIENCE

GENERAL

In [ ]:
```
combinedDF = pd.get_dummies(combinedDF, columns=['Deck', 'Pclass', 'Title', 'Embarked', 'Adult'])

y = trainDF.Survived.ravel()


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

LOGISTIC REGRESSION

In [ ]:
```
from sklearn.linear_model import LogisticRegression
lrModel = LogisticRegression(random_state=0)
lrModel.fit(X_train, y_train)
lr_prediction = lrModel.predict(X_test)

lrModel.get_params()
lrModel.densify
lrModel.coef_
```

k-fold Cross validation

In [ ]:
```
from sklearn.model_selection import GridSearchCV
lrGridSearch = GridSearchCV(lrModel, param_grid=parameters, cv=3)
lrGridSearch.fit(X_train, y_train)
lrGridSearch.best_estimator_

lrGridSearch.best_params_
lrGridSearch.best_score_
bestModel = lrGridSearch.best_estimator_
best_predictions = bestModel.predict(X_test)


# EXAMPLE2
parameters = {'C':[1.0, 10.0, 50.0, 100.0, 1000.0]}
lrGridSearch = GridSearchCV(lrModel, param_grid=parameters, cv=3)
lrGridSearch.fit(X_train_scaled, y_train)
bestModel = lrGridSearch.best_estimator_
best_predictions = bestModel.predict(X_test)
```

BASELINE MODEL

In [ ]:
```
np.mean(y_train), np.mean(y_test)

# OPTIONS: "most_frequent", "stratified", "uniform",  "constant", "prior"
dummyModel = DummyClassifier(random_state=0, strategy='most_frequent')
dummyModel.fit(X=X_train, y = y_train)
prediction = dummyModel.predict(X=X_test)
```

EVALUATION

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
accuracy_score(y_test, prediction)
precision_score(y_test, prediction)
recall_score(y_test, prediction)
confusion_matrix(y_test, prediction)
```

FEATURE NORMALIZARION & STANDARDIZATION

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
minMaxScaler = MinMaxScaler()
X_train_scaled = minMaxScaler.fit_transform(X=X_train)
X_train_scaled.min(), X_train_scaled.max()
X_test_scaled = minMaxScaler.fit_transform(X=X_test)

#STANDARIZATION
stdScaler = StandardScaler()
X_train_scaled2 = stdScaler.fit_transform(X_train_scaled)
X_train_scaled2.min(), X_train_scaled2.max()
```

PERSISTING A MODEL

In [ ]:
```
# WRITE
mport pickle
lr_model_path = os.path.join(os.path.pardir, 'models', 'lrModel.pkl') #'../models/lrModel.pkl'
model_pickle = open(lr_model_path, 'wb')
pickle.dump(lrModel, model_pickle)
model_pickle.close()


# READ BACK
model_pickle_r = open(lr_model_path, 'rb')
lrModelLoaded = pickle.load(model_pickle_r)
model_pickle_r.close()
```

In [ ]:
```

```

In [ ]:
```
#PYTHON DATA SCIENCE
adultDF = adultDF.replace('?', None)
adultDF.dropna(how='any')


adultDF = adultDF.replace('?', None)
adultDF.dropna(how='any')

adultDF.filter(adultDF['WorkClass'] == 'Without-pay').show(3)
'''
String values --> numerical values:       <column>.cast(FloatType())
Categorical Variables --> numeric         String Indexer
Indexed columns --> Vector                OneHotEncoderEstimator
Create feature vector(of input columns)   VectorAssembler
'''

from pyspark.sql.functions import min, max, mean

#STEP1: Convert string values into numeric values (not needed for my dataset)
from pyspark.sql.types import FloatType
from pyspark.sql.functions import col
adultDF = adultDF.withColumn('Age_int', adultDF['Age'].cast(FloatType()))

#STEP2: Convert Categorical Variables into numeric
#WorkClass --> WorkClass_index
#'State-gov' -->  3.0
from pyspark.ml.feature import StringIndexer
stringIndexer = StringIndexer(inputCol='WorkClass', outputCol='WorkClass_index')
indexedDF = stringIndexer.fit(adultDF).transform(adultDF)
#indexedDF.select('WorkClass', 'WorkClass_index').distinct().show(10)

#STEP3: Convert Indexed Categorical Variables to vectors
#WorkClassindex -->WorkClass_encoded
#2.0 -->   (6,[2],[1.0])
from pyspark.ml.feature import OneHotEncoderEstimator
oneHotEncoderEstimator = OneHotEncoderEstimator(inputCols=['WorkClass_index'], outputCols=['WorkClass
_encoded'])
ncodedDF = oneHotEncoderEstimator.fit(indexedDF).transform(indexedDF)
encodedDF.select('WorkClass', 'WorkClass_index', 'WorkClass_encoded').distinct().show(10)

encodedDF = encodedDF.withColumnRenamed('WorkClass_index','WorkClass_index_one_off')\
                     .withColumnRenamed('WorkClass_encoded','WorkClass_encoded_one_off')
```

```
#STEP4: String Indexing all other categorical features
categoricalFeatures = [  'WorkClass',
                         'Education',
                         'MaritalStatus',
                         'Occupation',
                         'Relationship',
                         'Race',
                         'Gender',
                         'NativeCountry']
indexers = [StringIndexer(inputCol=i, outputCol=i + '_indexed', handleInvalid='keep') for i in catego
ricalFeatures]
#for indexer in indexers:
  #print(indexer.getOutputCol())

encoders = [OneHotEncoderEstimator(inputCols=[i + '_indexed'], outputCols=[i + '_encoded']) for i in
categoricalFeatures]
#for encoder in encoders:
   #print(encoder.getInputCols(), '\t', encoder.getOutputCols())

#STEP5: Convert label column into index
labelIndexer = StringIndexer(inputCol='Label', outputCol='Label_indexed')

#STEP6: USE PIPELINE
##import pyspark.ml.Pipeline
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=indexers + encoders + [labelIndexer])
tranformedDF = pipeline.fit(encodedDF).transform(encodedDF)
#tranformedDF.take(1)
#tranformedDF.select('Label', 'Label_indexed').distinct().show()
#tranformedDF.columns

#STEP7: Vectorize all feature columns together
requiredFeatures = ['Age',
 'EducationNum',
 'CapitalGain',
 'CapitalLoss',
 'HoursPerWeek',
 'WorkClass_encoded',
 'Education_encoded',
 'MaritalStatus_encoded',
 'Occupation_encoded',
 'Relationship_encoded',
```

```
 'Race_encoded',
 'Gender_encoded',
 'NativeCountry_encoded']
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=requiredFeatures, outputCol='features')
vectorizedDF = assembler.transform(tranformedDF)
#vectorizedDF.select('features').show(5, truncate=False)
#vectorizedDF.select('WorkClass_index_one_off', 'WorkClass_encoded_one_off',
          #'WorkClass_indexed','WorkClass_encoded').show(5, truncate=False)

#STEP8: Split the data
trainingDataDF, testDataDF = vectorizedDF.randomSplit([0.8,0.2])
#trainingDataDF.count(), testDataDF.count()

#STEP9: Create RANDOMFOREST Classifier
from pyspark.ml.classification import RandomForestClassifier
randomForestClassifier = RandomForestClassifier(maxDepth=5,
                                                labelCol='Label_indexed',
                                                featuresCol= 'features')

#NOTE: BELOW STEP DOESN'T WORK BEACUSE ALL THE stages ARE ALREADY APPLIED.
#EITHER DROP ALL THSOSE NEW COLUMNS OR
#RECREATE THE ORIGINAL DATASET OR
#APPLY ONLY THE classification STEPstep
'''
pipelineAll  = Pipeline(stages = indexers +
                                 encoders +
                                [labelIndexer] +
                                [assembler] +
                                [randomForestClassifier])
model = pipelineAll.fit(trainingDataDF)
'''
model = randomForestClassifier.fit(trainingDataDF)

#{param[0].name: param[1] for param in model.extractParamMap().items()}
#model.explainParam('maxDepth')

#dict_values = model.extractParamMap().values()
#list(dict_values)
#list(model.extractParamMap().values())[7]
#model.getNumTrees
#model.featuresCol
#model.impurity  # SAME AS model.getParam('impurity')
```

```
#model.numClasses, model.numFeatures
#model.labelCol, model.featuresCol

#STEP10: PREDICT ON TEST DATA
predictedDF = model.transform(testDataDF)
#predictedDF.select('Label_indexed', 'prediction').show(10)
#PLOT
#predictedDF.select('Label_indexed', 'prediction').toPandas().plot(kind='scatter', x='Label_indexed',
 y='prediction')

#STEP11 Evaluate the model(evaluate the predictions)

from pyspark.ml.evaluation import BinaryClassificationEvaluator
binaryClassificationEvaluator = BinaryClassificationEvaluator(
                                        labelCol='Label_indexed',
                                        rawPredictionCol='prediction',
                                        metricName='areaUnderROC')
accuracy = binaryClassificationEvaluator.evaluate(predictedOnlyDF)
#accuracy
#predictedOnlyPandasDF = predictedOnlyDF.toPandas()
#predictedOnlyPandasDF.loc[predictedOnlyPandasDF['Label_indexed']
                     #  != predictedOnlyPandasDF['prediction']][0:5]


###STEP12: Try again with different depth
randomForestClassifier10 = RandomForestClassifier(maxDepth=10,
                                                  labelCol='Label_indexed',
                                                  featuresCol= 'features')
model10 = randomForestClassifier10.fit(trainingDataDF)
preditions10 = model10.transform(testDataDF)
accuracy10 = binaryClassificationEvaluator.evaluate(preditions10)
#accuracy10

maxBins   = range(10, 33, 11)  # max value = 32
maxBins   = range(10, 33, 11)  # max value = 32
my_parameter_list = list(zip(maxDepths,maxBins)) #[(10, 10), (20, 21), (30, 32)]
#for x in my_parameter_list
  #print(x[0],x[1])

randomForestClassifierList = []

randomForestClassifierList.append(
                            RandomForestClassifier(maxDepth=x[0], maxBins=x[1],
```

```
                                                    labelCol='Label_indexed',
                                                    featuresCol= 'features'))

#len(randomForestClassifierList

def classify(classifier):
    model = classifier.fit(trainingDataDF)
    prediction = model.transform(testDataDF)
    accuracy = binaryClassificationEvaluator.evaluate(prediction)
    print('MAXDEPTH:', list(model.extractParamMap().values())[7],
          'MAXBINS:', list(model.extractParamMap().values())[6],
          'ACCURACY:', accuracy)

# USING LIST COMPREHENSION
[classify(r) for r in randomForestClassifierList]
#O:
#MAXDEPTH: 10 MAXBINS: 10 ACCURACY: 0.7187703161570407
#MAXDEPTH: 20 MAXBINS: 21 ACCURACY: 0.7622451075477894
#MAXDEPTH: 30 MAXBINS: 32 ACCURACY: 0.7754354575437111

#STEP13:
'''
Two Types of Hyper Parameter Tuning (aka Model Selection)
    1) Cross Validator
    2) TrainValidationSplit
'''
#CROSS VALIDATOR
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.tuning import ParamGridBuilder
paramGrid =   ParamGridBuilder().addGrid(randomForestClassifier10.maxDepth, [4,6]) \
                                .addGrid(randomForestClassifier10.maxBins, [20,60]) \
                                .addGrid(randomForestClassifier10.numTrees, [5,20]) \
                                .build()

crossValidator = CrossValidator(estimator=randomForestClassifier10,
                                evaluator=binaryClassificationEvaluator,
                                estimatorParamMaps=paramGrid,
                                numFolds=5)



STEP14: Fit training data to all the models
cvModel = crossValidator.fit(trainingDataDF) # THIS STEP WILL TAKE A LONG TIME
```

```
#cvModel.avgMetrics   # (2 x 2 x 2 iterations)

#STEP15: GET THE BEST MODEL

bestModel = cvModel.bestModel
list(bestModel.extractParamMap().values())
{param[0].name: param[1] for param in bestModel.extractParamMap().items()}

#STEP16: Predict using the test data
bestModelPredictions = bestModel.transform(testDataDF)
bestModelAccuracy = binaryClassificationEvaluator.evaluate(bestModelPredictions)
#bestModelAccuracy #0.7170562491880605

#STEP17: Save the model for future use
bestModel.write().overwrite().save("random_forest_CrossValidator_best_model")

#STEP 17.1: READ BACK
from pyspark.ml.classification import RandomForestClassificationModel
saved_model = RandomForestClassificationModel.load("random_forest_CrossValidator_best_model")
_, testDataDF2 = vectorizedDF.randomSplit([0.9,0.1])
saved_model_predictions = saved_model.transform(testDataDF2)
saved_model_accuracy = binaryClassificationEvaluator.evaluate(saved_model_predictions)
#saved_model_accuracy

#STEP18: Hyper Parameter Tuning (aka Model Selection) (TrainValidationSplit)
#Evaluates each combination of parameters once
from pyspark.ml.tuning import TrainValidationSplit
paramGrid2 = ParamGridBuilder().addGrid(randomForestClassifier10.maxDepth, [4,6])
                               .addGrid(randomForestClassifier10.numTrees, [5,20])
                               .build()
trainingData90, testData10 = vectorizedDF.randomSplit([0.9, 0.1], seed=12345)
tvs = TrainValidationSplit(estimator=bestModel, estimatorParamMaps=paramGrid2,
                           evaluator=BinaryClassificationEvaluator, trainRatio=0.9)

tvsModel = tvs.fit(trainingData90)
#ERROR: AttributeError: 'RandomForestClassificationModel' object has no attribute 'fitMultiple'
#TO DO: FIX THIS LATER

#STEP19: EVALUATION using BinaryClassificationMetrics
predictedOnlyDF2 = predictedOnlyDF.withColumnRenamed('Label_indexed', 'label')
#predictedOnlyDF2.columns  O:['label', 'prediction']
predictionAndLabels = predictedOnlyDF2.rdd.map(lambda row: (float(row.label), float(row.prediction)))
binaryClassificationMetrics = BinaryClassificationMetrics(predictionAndLabels)
```

```
binaryClassificationMetrics.areaUnderPR #0.4040690039202276
binaryClassificationMetrics.areaUnderROC #0.8158521147102044

#STEP20: TO DO: EVALUATION using MulticlassMetrics
#TO DO: Select the best model and do the evaluation in it( same as for CrossValidator)
```

PARAMS PRINT

In [ ]:
```
for key, value in bestModel.extractParamMap().items():
    print (key.name, ":", value)
```

1. GBT (GRADENT BOOSTING TREE) CLASSIFIER

In [ ]:
```
# 1. GBT (GRADENT BOOSTING TREE) CLASSIFIER
from pyspark.ml.classification import GBTClassifier
gbtClassifier = GBTClassifier(labelCol='Label_indexed',  featuresCol = 'features',  maxIter=30)
gbtModel = gbtClassifier.fit(trainingDataDF)
gbtPredictions = gbtModel.transform(testDataDF)
gbtAccuracy = binaryClassificationEvaluator.evaluate(gbtPredictions)
#gbtAccuracy #0.7647927357765553
```

PYTHON

LOGGING

In [ ]:
```
# Levels: DEBUG, INFO, WARNING, ERROR, CRITICAL or NOTSET
# NOTE: NOTSET means all messages will be logged
```

In [ ]:
```
import logging

'''
# create file handler which logs even debug messages
fileLogger = logging.FileHandler('pluralsight_python_datascience_abhishek.log')
fileLogger.setLevel(logging.DEBUG)

# create console handler with a higher log level
consoleStreamingLogger = logging.StreamHandler()
consoleStreamingLogger.setLevel(logging.ERROR)

# create formatter and add it to the handlers
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
consoleStreamingLogger.setFormatter(formatter)
consoleStreamingLogger.setFormatter(formatter)

# add the handlers to the logger
logger.addHandler(fileLogger)
logger.addHandler(consoleStreamingLogger)

logger = logging.getLogger(name=__name__)
logger.setLevel(logging.DEBUG)

print(logger.level)
logger.level = logging.DEBUG

logger.debug('This is debug message')
logger.info('This is info message')
# logger.warn('This is warn message') # DeprecationWarning: The 'warn' method is deprecated
logger.warning('This is warning message')
logger.error('This is error message')  # 2018-09-11 19:29:27,348 - __main__ - ERROR - This is error m
essage
logger.fatal('This is fatal message') # 2018-09-11 19:29:27,362 - __main__ - CRITICAL - This is fatal
 message
logger.critical('This is critical message') # 2018-09-11 19:29:27,381 - __main__ - CRITICAL - This is
 critical message
'''
```

In [ ]:
```
# WORKS
import logging
from logging.config import fileConfig

#logging.config.fileConfig('titanic/logging/logging.conf')
fileConfig('titanic/logging/logging.conf')

# create logger
logger = logging.getLogger()

# 'application' code
logger.debug('debug message')
logger.info('info message')
logger.warning('warn message')
logger.error('error message')
logger.critical('critical message')
```

In [ ]:
```
#### TO DO: LOGGING USING YAML config file
```

PYTHON DATA FRAME

In [ ]:
```

```

In [ ]:
```
# PYTHON DATA FRAME
pd.DataFrame(data = [list(ucase), list(lcase)], index=("upper", "lower"), columns = list(ucase))
pd.DataFrame(np.random.randint(1,100,12).reshape(3,4))
forecast = pd.DataFrame({"high": high_temps, "low": low_temps}, index=["Mon", "Tue", "Wed", "Thu", "F
ri"])

df1.head()
df1.index= np.arange(1,27)

df1.dtypes  # column types
df1.index = df1['lower'] # set index
df1.sort_values('numbers').head() # sort
df1['numbers'].head() # select column
df1[['numbers', 'lower']].head(3) # select columns
df1.iloc[20] # select row
df1.iloc[15:18] # select rows

df1.describe() # stats

df1['numbers'] = np.random.randint(1,100,26) # adding new column
forecast['difference']= forecast['high'] - forecast['low'] # adding new column
df['three']=pd.Series([10,20,30],index=['a','b','c']) # add series as a new column

df2 = pd.read_csv(s2, parse_dates=['Date'])   # Date format gets changed
df2.set_index('Date', inplace=True) # df2 gets changed

df2.plot(kind='area')


del df['one']  # delete a column
df.pop('two')  # delete a column
```

RANGE & RANDOM

In [ ]:
```
#RANGE & RANDOM
A1 = range(1,10) #O: 1 2 3 4 5 6 7 8 9
A = np.arange(10) #O: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
B = np.random.randn(10)
#O: Array([-1.49553106, -0.84957435,  0.63583913, ...,  1.17658258,  2.43626867, -1.50656064])
colors = np.random.randint(0,10,10) #array([0, 9, 8, 8, 8, 5, 7, 2, 2, 3])
D = np.random.random(10)
#O: array([ 0.55010528,  0.57947712,  0.23147827, ..., 0.01487446,  0.12264088,  0.34392944])
E = np.random.random_sample(10)
#O: array([ 0.55010528,  0.57947712,  0.23147827, ..., 0.01487446,  0.12264088,  0.34392944])
```

READ FILE

In [ ]:
```
pandasDF = pd.read_csv(airpassengers_file)
```

GENERAL

In [ ]:
```
pandasDF.dtypes
type(pandasDF)
pandasDF.index
pandasDF.head(2)
```

JUPYTER NOTEBOOK RELATED

In [ ]:
```
!jupyter nbconvert --to script Pluralsight_Python_data_science_Abhishek_kumar_2.ipynb
```