Capstone Project - Telecom Network Service Disruption

This is Part 2 of the project documentation. (File Name 02-FINAL-TelstraHyperparameters)

Please see Part 1 for Project Details and Executive Summary. (File Name 01-FINAL -TelstraEDA)

Generate Hyperparameters

Background

Hyper-parameters are parameters that are not directly learnt within estimators and are passed as arguments to the constructor of the estimator classes.
Python's *scikit learn* library, commonly referred to as *sklearn* provides methods that can be executed to get parameter values for optimised performance
for most estimators.

Process

The process for hyperparameter generation consists of the following steps:

- Instantiate the model for an estimator *eg: model = RandomForestClassifier()*
- Get the parameters for the model *eg: model.get_params()*
- Conduct research to determine the values to be provided to the optimiser
- Specify the parameters and the values in a dictionary
- Use either *sklearn.model_selection.GridSearchCV* or
  sklearn.model_selection.RandomizedSearchCV
  method to estimate hyperprameters.
  GridSeachCV is a an exhaustive method that uses each specified parameter value while *RandomizedSearchCV* randomly selects parameter value from the list provided
- Use *cross validation score* method to iterate through random instances of dataset

Method *RandomizedSearchCV* is used in this notebook for each of the 5 estimators. The above mentioned
process is repeated for each of the five estimators used.

In [1]:

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

Load Data

In [2]:

```
# Full dataset
train_id = pd.read_csv('./Data/train_id.csv')
test_id = pd.read_csv('./Data/test_id.csv')
```

In [3]:

```
print('Dataframe train - number of rows columns', train_id.shape)
print('Dataframe test - number of rows columns', test_id.shape)
```

```
Dataframe train - number of rows columns (7381, 434)
Dataframe test - number of rows columns (11171, 434)
```

In [4]:

```
test_id.head()
```

Out[4]:

	fault_severity	id	location_1008	location_1019	location_102	location_1042	location_105
0	NaN	11066	0	0	0	0	0
1	NaN	18000	0	0	0	0	0
2	NaN	16964	0	0	0	0	0
3	NaN	4795	0	0	0	0	0
4	NaN	3392	0	0	0	0	0

5 rows × 434 columns

Develop Models

In [5]:

```
#################################
# Import classifier model modules
#################################
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
#Gridsearch and scoring
from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
```

```
C:\Users\Vinita Auplish\Anaconda3\lib\site-packages\sklearn\cross_validatio
n.py:41: DeprecationWarning: This module was deprecated in version 0.18 in f
avor of the model_selection module into which all the refactored classes and
functions are moved. Also note that the interface of the new CV iterators ar
e different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\Vinita Auplish\Anaconda3\lib\site-packages\sklearn\grid_search.py:4
2: DeprecationWarning: This module was deprecated in version 0.18 in favor o
f the model_selection module into which all the refactored classes and funct
ions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [6]:

```
#################################
# Prepare data for modelling
#################################
# Create target dataframe
target = train_id['fault_severity']
# Drop fault_severity from train so we don't have to specify column names
train_id.drop(['fault_severity'], axis=1, inplace=True)
test_id.drop(['fault_severity'], axis=1, inplace=True)
```

In [7]:

```
#################################
# Assign values to lists
#################################
XTrain = train_id
yTrain = target
XTest = test_id
# Split training dataset into "train" and "validate" to compare model accuracy
X_train, X_test, y_train, y_test = train_test_split(XTrain,yTrain, test_size=0.3, random_st
```

Hyperparameter tuning

In [8]:

```
################################################################
# Get hyperparameters for estimators
################################################################
#RandomForestClassifier
model = RandomForestClassifier()
model.get_params()
# Create param grid
rf_pgrid = {'bootstrap': [True, False],
            'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
            'max_features': ['auto', 'sqrt'],
            'min_samples_leaf': [1, 2, 4],
            'min_samples_split': [2, 5, 10],
            'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]
           }
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_rsc = RandomizedSearchCV(estimator = model, param_distributions = rf_pgrid, n_iter = 100
                            verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
print('>> Running random search')
rf_rsc.fit(X_train,y_train)
print('Best parameters\n')
rf_rsc.best_params_
```

In [9]:

```
model = KNeighborsClassifier()
model.get_params()
# Create param grid
rf_pgrid = {'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
            'weights': ['uniform','distance'],
            'leaf_size': [5,10,15,20,25,30],
            'p': [1, 2, 4],
            'metric': ['minkowski'],
            'n_neighbors': [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
           }
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_rsc = RandomizedSearchCV(estimator = model, param_distributions = rf_pgrid, n_iter = 100
                            verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
print('>> Running random search')
rf_rsc.fit(X_train,y_train)
print('Best parameters\n')
rf_rsc.best_params_
```

In [10]:

```
model = DecisionTreeClassifier()
model.get_params()
# Create param grid
rf_pgrid = {'criterion': ['gini', 'entropy'],
            'class_weight': ['balanced'],
            'max_features': ['auto','sqrt','log2',None],
            'splitter': ['best','random'],
           }
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
Random Search for more parameters otherwose GridSearch
rf_rsc = RandomizedSearchCV(estimator = model, param_distributions = rf_pgrid, n_iter = 15,
                                verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
print('>> Running random search')
rf_rsc.fit(X_train,y_train)
print('Best parameters\n')
rf_rsc.best_params_
```

In [11]:

```
model = AdaBoostClassifier()
model.get_params()
# Create param grid
rf_pgrid = {'algorithm': ['SAMME', 'SAMME.R'],
            'base_estimator': [None],
            'learning_rate': [1.0],
            'n_estimators': [30,40,50,60,70,80,90],
            'random_state': [42,56,88,None]
           }
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
Random Search for more parameters otherwose GridSearch
rf_rsc = RandomizedSearchCV(estimator = model, param_distributions = rf_pgrid, n_iter = 15,
                                verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
print('>> Running random search')
rf_rsc.fit(X_train,y_train)
print('Best parameters\n')
rf_rsc.best_params_
```

In [12]:

```
model = GradientBoostingClassifier()
model.get_params()
# Create param grid
rf_pgrid = {'criterion': ['friedman_mse', 'mse','mae'],
            'init': [None],
            'learning_rate': [0.1,0.3,0.5,0.7,0.9,1.0],
            'n_estimators': [30,40,50,60,70,80,90],
            'random_state': [42,56,88,None],
            'max_depth': [1,2,3,4,5],
            'max_features': ['auto','sqrt','log2',None],
            'n_estimators': [80,90,100,130,150,190,200],
            'random_state': [42,56,88,None]
           }
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
Random Search for more parameters otherwose GridSearch
rf_rsc = RandomizedSearchCV(estimator = model, param_distributions = rf_pgrid, n_iter = 30,
                                verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
print('>> Running random search')
rf_rsc.fit(X_train,y_train)
print('Best parameters\n')
rf_rsc.best_params_
```

In [ ]:

```

```