Department of Computer Science and Engineering (Data Science)
Subject: Applied Data Science (DJ19DSL703)

Experiment - 7

(Modelling and Optimisation Trade-off)

Name: Keshav-O-Agarwal **SAPID: 60009200028**

Aim: To optimize the model.

Theory:

Hyper parameter and parameter tuning are crucial processes in machine learning that help optimize the performance of a model. They involve adjusting the settings of a machine learning algorithm to improve its ability to make accurate predictions. The ‘K’ in KNN can be regarded as the parameter whereas number of leaf nodes, number of branches, depth of a tree are hyper parameters of a Decision Tree or Random Forest.

1. Parameters:
Parameters are the internal settings or coefficients learned by the machine learning model during training. These values are adjusted automatically through the training process, and they determine how the model transforms input data into predictions. In a simple linear regression model, for example, the parameters are the slope and intercept of the regression line. Parameters are learned from the data and are specific to the model and its training data.

2. Hyper parameters:
Hyper parameters are settings or configurations that are not learned from the data but are set by the machine learning engineer or data scientist before training begins.
These settings control the behaviour of the learning algorithm itself and can significantly impact the model's performance.

Examples of hyper parameters include the learning rate in gradient descent, the number of hidden layers and neurons in a neural network, or the regularization strength in a support vector machine.
Hyper parameters are set by trial and error or by using techniques like grid search or random search to find the best values for a specific problem.

Parameter Tuning:
Parameter tuning involves adjusting the internal parameters of a machine learning model to optimize its performance on a specific dataset. This is typically done automatically during the training process, as the model learns the best values for its parameters to minimize a predefined loss function.
For example, in a neural network, the weights and biases of the neurons are adjusted during training to minimize the prediction error. Parameter tuning is specific to the model and is intrinsic to the training process.

Department of Computer Science and Engineering (Data Science)

Hyper parameter Tuning:
Hyper parameter tuning, also known as hyper parameter optimization, focuses on finding the best settings for hyper parameters to improve a model's generalization and performance. Hyper parameter tuning is an iterative process that typically involves searching through a range of hyper parameter values to find the combination that produces the best results on a validation dataset. Common techniques for hyper parameter tuning include grid search, random search, Bayesian optimization, and automated tools like scikit-learn's GridSearchCV or hyperopt.
Hyper parameter tuning is crucial for optimizing the performance of a machine learning model on a specific problem. In summary, parameters are internal settings learned by the model during training, while hyper parameters are external settings configured before training. Parameter tuning focuses on optimizing internal settings, while hyper parameter tuning is about finding the best external settings to improve model performance. Both processes are essential for developing effective machine learning models.

Lab Assignment:
- Students need to try various permutations and combinations for getting the right set of parameters and hyper parameter values.
- Post this your model is ready for deployment. Create a pickle file for your model such that it will take the input and produce the model response as the output. This should be then converted to an API using Flask/FastAPI frameworks.
- The API will have an endpoint (say /run-model) which will be a POST request. It will take the data records for which the prediction is to be made as input, run the model in the background and the API response will be the output from the model. For example, I can provide an email body as input to the API and it will return a response if the content can be regarded as spam or not. Or I can even provide multiple input fields for a house description and the endpoint will provide the estimated house price in response.

Flask server (app.py):

```
from flask import Flask, request, jsonify import pickle
 app = Flask(__name__) #
Load the model
model = pickle.load(open("trained_model.pkl", "rb"))
@app.route("/api", methods=["POST"]) def predict():
   # Get the data from the POST request.     data =
request.get_json(force=True)     row = [list(data.values())]
   # Make prediction using model loaded from disk as per the data.
   prediction = model.predict(row)

   # Take the first value of prediction     output =
prediction[0]     return jsonify(output)
 if __name__ == "__main__":
app.run(port=5000, debug=True)
```

Making request to model (req.py):

```
import requests

url = "http://localhost:5000/api"

test_data = {     "buying_low": 0.0,
   "buying_med": 1.0,
   "buying_vhigh": 0.0,
   "maint_low": 0.0,
   "maint_med": 0.0,
   "maint_vhigh": 1.0,
   "doors_3": 0.0,
   "doors_4": 1.0,
   "doors_5more": 0.0,
   "persons_4": 1.0,
   "persons_more": 0.0,
   "lug_boot_med": 0.0,
```

```
  "lug_boot_small": 0.0,
  "safety_low": 1.0,
  "safety_med": 0.0,
}

# 'unacc'  r = requests.post(url, json=test_data)
print("Predicted class: ",r.json())
```

output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   JUPYTER   COMMENTS

PS C:\Users\Jay\Desktop\DJSCE\Sem 7\Labs\ADS\Lab7 Optimaztion\flask> python req.py
Predicted class:  unacc
PS C:\Users\Jay\Desktop\DJSCE\Sem 7\Labs\ADS\Lab7 Optimaztion\flask>
```

In [2]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import pandas as pd

df = pd.read_csv('car_evaluation.csv')
df.head()
```

Out[2]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

In [3]:
```
df.shape
```

Out[3]: (1727, 7)

In [4]:
```
df.describe()
```

Out[4]:

	vhigh	vhigh.1	2	2.1	small	low	unacc
count	1727	1727	1727	1727	1727	1727	1727
unique	4	4	4	3	3	3	4
top	high	high	3	4	med	med	unacc
freq	432	432	432	576	576	576	1209

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
  0vhigh     1727 non-null   object
  1vhigh.1   1727 non-null   object
  22         1727 non-null   object
  32.1       1727 non-null   object
  4small     1727 non-null   object
  5low       1727 non-null   object
  6unacc     1727 non-null   objectdtypes: object(7) memory usage: 94.6+ KB
```

In [6]:
```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety',
'class']

df.columns = col_names

col_names
```

Out[6]:
```
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
buying      0
maint       0
doors       0
persons     0
lug_boot    0
safety      0
class       0
dtype: int64
```

In [8]:
```
df['class'].value_counts()
```

Out[8]:
```
unacc    1209
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

In [9]:
```
X=df.drop(['class'],axis=1)
```

In [10]:
```
y=df['class']
```

In [11]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [12]:
```
encoder = OneHotEncoder(sparse=False, drop='first')
categorical_columns = X.select_dtypes(include=['object']).columns
X_encoded = encoder.fit_transform(X[categorical_columns])
column_names = encoder.get_feature_names_out(categorical_columns)
X_encoded = pd.DataFrame(X_encoded, columns=column_names)
X = pd.concat([X.drop(columns=categorical_columns), X_encoded], axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
ndom_state=42)

# Define your machine learning model
model = RandomForestClassifier()

# Define a grid of hyperparameters to search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4]
}

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, sc
oring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters from the grid search
best_params = grid_search.best_params_ print("Best
Hyperparameters:", best_params)

# Train the model with the best hyperparameters on the entire training set
best_model = RandomForestClassifier(**best_params) best_model.fit(X_train,
y_train)

# Evaluate the model's performance on the test set
accuracy = best_model.score(X_test, y_test)
print("Model Accuracy:", accuracy)
```

```
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples
_split': 2, 'n_estimators': 50}
Model Accuracy: 0.8728323699421965
```

In [13]:
```
import pickle
```

In [14]:
```
model = RandomForestClassifier(n_estimators=100, max_depth=10)
model.fit(X, y)

# Save the trained model as a pickle file
with open('trained_model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)
```

In [16]:
```
X_test.head()
```

Out[16]:

	buying_low	buying_med	buying_vhigh	maint_low	maint_med	maint_vhigh	doors_3	d
599	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
932	0.0	1.0	0.0	0.0	0.0	1.0	0.0	
628	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1497	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
1262	0.0	1.0	0.0	1.0	0.0	0.0	0.0	

In [17]:
```
y_test[:5]
```

Out[17]:
```
599     unacc
932     unacc
628     unacc
1497      acc
1262    unacc
Name: class, dtype: object
```

In [26]:
```
def get_Xy(i, data, target):
    print(data.iloc[i], "\n", target.iloc[i])

    return (list(data.iloc[i]), target.iloc[i])
```

In [27]:
```
x1, y1 = get_Xy(1,X_test,y_test)
x1, y1
```

```
buying_low        0.0
buying_med        1.0
buying_vhigh      0.0
maint_low         0.0
maint_med         0.0
maint_vhigh       1.0
doors_3           0.0
doors_4           1.0
doors_5more       0.0
persons_4         1.0
persons_more      0.0
lug_boot_med      0.0
lug_boot_small    0.0
safety_low        1.0
safety_med        0.0
Name: 932, dtype: float64 unacc
```

Out[27]:
```
([0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.
0],
 'unacc')
```

In [28]:
```
best_model.predict([x1])
```

```
c:\Users\Jay\AppData\Local\Programs\Python\Python39\lib\site-packages\sklea
rn\base.py:450: UserWarning: X does not have valid feature names, but Rando
mForestClassifier was fitted with feature names
  warnings.warn(
```

Out[28]:
```
array(['unacc'], dtype=object)
```

In [ ]: