ExtraCredit

December 11, 2023

[1]:
```python
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```python
import pandas as pd
import numpy as np
import random as rn
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

[3]:
```python
random_state_num = 42
rn.seed(random_state_num)
```

[4]:
```python
# Import the data set and view 5 rows
df_master = pd.read_csv('survey_encoded_data.csv')
df_master.head()
```

[4]:
```
   0  1  2  3  4    5  6  7  8    9  …  3056  3057  3058  3059  3060  3061  \
0  0  0  0  0  0  0.0  0  0  0  0.0  …     0     0     0     0     0     0
1  0  0  0  0  0  0.0  0  0  0  0.0  …     0     0     0     0     0     0
2  0  0  0  0  0  0.0  0  0  0  0.0  …     0     0     0     0     0     0
3  0  0  0  0  0  0.0  0  0  0  0.0  …     0     1     0     0     0     0
4  0  0  0  0  0  0.0  0  0  0  0.0  …     0     0     0     0     0     0

   3062  3063  3064  labels
0     0     0     0       2
1     0     0     0       2
2     0     0     0       2
3     0     1     0       2
4     0     0     0       2
```

```
[5 rows x 3066 columns]
```

[5]:
```
# First explore any of the missing data
if len(df_master.isna().sum()[df_master.isna().sum()> 0]) > 0:
    print('There is some missing data that needs to be preprocessed\n')
else:
    print('No missing data is found\n')


# Check the unique label values
df_master['labels'].unique()
```

```
No missing data is found
```

[5]:
```
array([ 2,  8,  0, 10, 14,  1, 11, 12,  6, 13,  4,  3,  7,  5,  9])
```

[6]:
```
lable_counts = df_master['labels'].value_counts()
lable_counts = lable_counts[lable_counts == 1].index
df_master_filters = df_master[~df_master['labels'].isin(lable_counts)]
```

Since, we can't stratified split any label whose value just appears once, we can't predict/test-train split. Hence, I choose to drop such rows.

1. Since the data is already encoded, and column labels are not avialble, it's wise enough to proceed with Model Building as no real meaning for input variable relation can be drawn from encoded data set.

2. I will first split the data set into test,train,validation

[7]:
```
df_master_filters.groupby('labels').size().reset_index(name='count')
```

[7]:

	labels	count
0	0	7
1	1	10
2	2	17
3	4	3
4	6	7
5	8	8
6	9	6
7	10	12
8	11	8
9	12	7
10	13	6
11	14	18

[8]:
```
input_columns = list(df_master_filters.columns)
if 'labels' in input_columns:
    input_columns.remove('labels')
```

0.1 Test Train Validation Split

[9]:
```python
# Test Train Split
X_train, X_test, y_train, y_test =
 ↪train_test_split(df_master_filters[input_columns],
                                                                   df_master_filters['labels'],
                                                                   test_size=0.20,
                                                                   random_state=42,

 ↪stratify=df_master_filters['labels'])

# Further split the train set into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                                                   test_size=0.2,
                                                   random_state=42,
                                                   stratify=y_train)
```

[10]:
```python
model_scores = {}
```

0.2 Model 1: Logistic Regression

[11]:
```python
lr_model = LogisticRegression(random_state=random_state_num)
lr_model.fit(X_train, y_train)
y_pred = lr_model.predict(X_test)

lr_accuracy_before = accuracy_score(y_test, y_pred)
model_scores['LR-before'] = lr_accuracy_before

# Now let's tune the parameters
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga'],
    'max_iter': [100, 200, 300],
}


lr_model_tune = LogisticRegression(random_state= random_state_num)
grid_search = GridSearchCV(lr_model_tune, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_val, y_val)

print('Best tuned Logistic Regression Parameters: ', grid_search.best_params_)
best_model_lr = LogisticRegression(**grid_search.best_params_)
best_model_lr.fit(X_train, y_train)
y_pred_tune = best_model_lr.predict(X_test)
lr_accuracy_after = accuracy_score(y_test, y_pred_tune)
model_scores['LR-after'] = lr_accuracy_after
```

```
print(f'Logistics Regression Accuracy\n\tBefore Tunning ->␣
↪{lr_accuracy_before}\n\tAfter Tunning -> {lr_accuracy_after}')
```

```
Best tuned Logistic Regression Parameters:  {'C': 1, 'max_iter': 100, 'penalty':
'l1', 'solver': 'liblinear'}
Logistics Regression Accuracy
        Before Tunning -> 0.2727272727272727
        After Tunning -> 0.36363636363636365
```

0.3 Model 2: Decision Trees

[12]:
```
dt_model = DecisionTreeClassifier(random_state= random_state_num)
dt_model.fit(X_train, y_train)
y_pred = dt_model.predict(X_test)

dt_accuracy_before = accuracy_score(y_test, y_pred)
model_scores['DT-before'] = dt_accuracy_before

# Now let's tune the parameters
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2'],
}

dt_model_tune = DecisionTreeClassifier(random_state= random_state_num)
grid_search = GridSearchCV(dt_model_tune, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_val, y_val)

print('Best tuned Decision Tree Parameters: ', grid_search.best_params_)
best_model_dt = DecisionTreeClassifier(**grid_search.best_params_)
best_model_dt.fit(X_train, y_train)
y_pred_tune = best_model_dt.predict(X_test)
dt_accuracy_after = accuracy_score(y_test, y_pred_tune)
model_scores['DT-after'] = dt_accuracy_after

print(f'Decision Tree Accuracy\n\tBefore Tunning ->␣
↪{dt_accuracy_before}\n\tAfter Tunning -> {dt_accuracy_after}')
```

```
Best tuned Decision Tree Parameters:  {'criterion': 'gini', 'max_depth': None,
'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 2}
Decision Tree Accuracy
        Before Tunning -> 0.36363636363636365
        After Tunning -> 0.2727272727272727
```

0.4 Model 3: Random Forest

[14]:
```
rf_model = RandomForestClassifier(random_state= random_state_num)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

rf_accuracy_before = accuracy_score(y_test, y_pred)
model_scores['RF-before'] = rf_accuracy_before

# Now let's tune the parameters
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False],
}


rf_model_tune = RandomForestClassifier(random_state= random_state_num)
grid_search = GridSearchCV(rf_model_tune, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_val, y_val)

print('Best tuned Random Forest Parameters: ', grid_search.best_params_)
best_model_rf = RandomForestClassifier(**grid_search.best_params_)
best_model_rf.fit(X_train, y_train)
y_pred_tune = best_model_rf.predict(X_test)
rf_accuracy_after = accuracy_score(y_test, y_pred_tune)
model_scores['RF-after'] = rf_accuracy_after

print(f'Random Forest Accuracy\n\tBefore Tunning ->␣
↪{rf_accuracy_before}\n\tAfter Tunning -> {rf_accuracy_after}')
```

```
Best tuned Random Forest Parameters:  {'bootstrap': True, 'max_depth': None,
'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 50}
Random Forest Accuracy
        Before Tunning -> 0.3181818181818182
        After Tunning -> 0.45454545454545453
```

0.5 Model 4: K-Nearest Neighbors (KNN)

[15]:
```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)

knn_accuracy_before = accuracy_score(y_test, y_pred)
```

```
model_scores['KNN-before'] = knn_accuracy_before

# Now let's tune the parameters
param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2],
    'leaf_size': [10, 20, 30, 40],
}


knn_model_tune = KNeighborsClassifier()
grid_search = GridSearchCV(knn_model_tune, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_val, y_val)

print('Best tuned Random Forest Parameters: ', grid_search.best_params_)
best_model_knn = KNeighborsClassifier(**grid_search.best_params_)
best_model_knn.fit(X_train, y_train)
y_pred_tune = best_model_knn.predict(X_test)
knn_accuracy_after = accuracy_score(y_test, y_pred_tune)
model_scores['KNN-after'] = knn_accuracy_after

print(f'Random Forest Accuracy\n\tBefore Tunning ->␣
↪{knn_accuracy_before}\n\tAfter Tunning -> {knn_accuracy_after}')
```

```
Best tuned Random Forest Parameters:  {'algorithm': 'auto', 'leaf_size': 10,
'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
Random Forest Accuracy
        Before Tunning -> 0.13636363636363635
        After Tunning -> 0.09090909090909091
```

0.6 Model 5: Support Vector Machines

[16]:
```
svc_model = SVC()
svc_model.fit(X_train, y_train)
y_pred = svc_model.predict(X_test)

svc_accuracy_before = accuracy_score(y_test, y_pred)
model_scores['SVC-before'] = svc_accuracy_before

# Now let's tune the parameters
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3, 4],
```

```
    'coef0': [0.0, 0.1, 1.0],
}



svc_model_tune = SVC()
grid_search = GridSearchCV(svc_model_tune, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_val, y_val)

print('Best tuned Random Forest Parameters: ', grid_search.best_params_)

best_model_svc = SVC(**grid_search.best_params_)
best_model_svc.fit(X_train, y_train)
y_pred_tune = best_model_svc.predict(X_test)
svc_accuracy_after = accuracy_score(y_test, y_pred_tune)
model_scores['SVC-after'] = svc_accuracy_after

print(f'SVC Accuracy\n\tBefore Tunning -> {svc_accuracy_before}\n\tAfter␣
↪Tunning -> {svc_accuracy_after}')
```

```
Best tuned Random Forest Parameters:  {'C': 0.1, 'coef0': 0.0, 'degree': 3,
'gamma': 'scale', 'kernel': 'poly'}
SVC Accuracy
        Before Tunning -> 0.18181818181818182
        After Tunning -> 0.18181818181818182
```

0.7 Model Comparisons

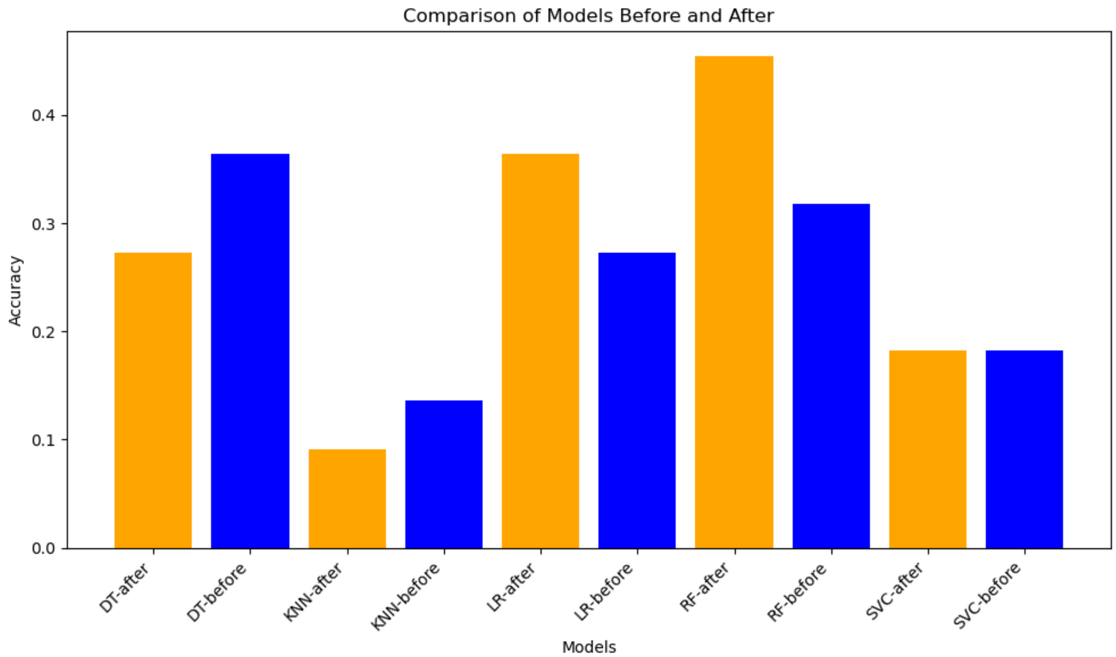
[18]:
```
model_scores
sorted_data = dict(sorted(model_scores.items()))

# Extract keys and values
labels = list(sorted_data.keys())
values = list(sorted_data.values())

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(labels, values, color=['blue' if 'before' in label else 'orange' for␣
↪label in labels])
plt.title('Comparison of Models Before and After')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
↪readability
plt.tight_layout()

# Show the plot
```

```
plt.show()
```



0.8 Observations

1. Based on the plot above, the Random Forest Classifier seems to be the best suited modle for this data set considering accuracy as the model comparison criteria.
2. The accuracy of RF classifier improved to 0.45 from 0.31 and best tuned hyper-parameters are identified as: 'bootstrap': True, 'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50