CSCI - 6409 - The Process of Data Science - Fall 2022

</center>

Assignment 2

</center>

Meagan Sinclair
B00737317

Samarth Jariwala
B00899380

1. Task Explanation

In order to fulfill the goal of predicting the average rating of a restaurant, the final model must be supervised during training as it will predict a specific target. The average rating for the restaurants are a set of [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0. 4.5, 5.0]. These can be considered the class labels for the output target which the model will predict. Therefore, the model type will be classification. Note for better performance, the ratings will be rounded and the predicted classes will be [1.0, 2.0, 3.0, 4.0, 5.0].

2. Evaluation Metric

2-fold cross validation will be used to assess the the training performance and avoid over or under fitting. To evaluate the model's performance on unseen test data, a confusion matrix will be generated to visualize the classification performance. The precision, recall, F1 score, and overall accuracy will be determined. These metrics suit our task as they provide insight to the model's classification abilities.

3. Feature Selection

Entropy will be assessed for the potential features.

In [ ]:
```
import pandas as pd
from google.colab import drive
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import entropy
```

In [ ]:
```
# code for regenerating the features dataframe (final_features) from A1 was omitted here
```

In [ ]:
```
# drop any rows left with nan
final_features.dropna(inplace=True)
```

In [ ]:
```
final_features = final_features.convert_dtypes()
final_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 985117 entries, 0 to 1083396
Data columns (total 18 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   name          985117 non-null  string 
 1   country       985117 non-null  string 
 2   claimed       985117 non-null  UInt8  
 3   veg           985117 non-null  UInt8  
 4   vegan         985117 non-null  UInt8  
 5   gf            985117 non-null  UInt8  
 6   awards        985117 non-null  UInt8  
 7   pop_score     985117 non-null  Float64
 8   top_tag       985117 non-null  string 
 9   top_cuisine   985117 non-null  string 
 10  num_features  985117 non-null  Int64  
 11  num_meals     985117 non-null  Int64  
 12  p_excellent   985117 non-null  Float64
 13  p_vgood       985117 non-null  Float64
 14  p_average     985117 non-null  Float64
 15  p_poor        985117 non-null  Float64
 16  p_terrible    985117 non-null  Float64
 17  ave_rating    985117 non-null  Float64
dtypes: Float64(7), Int64(2), UInt8(5), string(4)
memory usage: 123.1 MB
```

In [ ]:

```python
# compute entropy of whole data set

class_proportions = final_features["ave_rating"].value_counts(normalize=True)

classes = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]

class_proportions = class_proportions[classes].to_numpy(dtype='float')

S = entropy(class_proportions)
```

In [ ]:

```python
S
```

Out[ ]:

```
1.6695032630908475
```

In [ ]:

```python
# compute entropy after splitting on specific feature
# Reference: [5]

#claimed portion
claimed = final_features[final_features["claimed"] == 1]
weight = len(claimed)/len(final_features)

class_proportions = claimed["ave_rating"].value_counts(normalize=True)
class_proportions = class_proportions[classes].to_numpy(dtype='float')

rem_claimed = entropy(class_proportions)*weight

# unclaimed portion
unclaimed = final_features[final_features["claimed"] == 0 ]
weight = len(unclaimed)/len(final_features)

class_proportions = unclaimed["ave_rating"].value_counts(normalize=True)
class_proportions = class_proportions[classes].to_numpy(dtype='float')

rem_unclaimed = entropy(class_proportions)*weight

rem = rem_claimed + rem_unclaimed
```

In [ ]:

```
# information gain calculation
IG = S - rem
IG
```

Out[ ]:

```
0.024573239732237795
```

4. Model Development

4.1 Model Justification

The model chosen is decision tree classifier. The dataset is made up of information based attributes so this classifier is well suited. The decision tree classifier is simple to implement and use so it is an ideal first attempt at a model to satisfy our business problem. Our data is all labeled with a specific class label so the model's requirments are met.

In [ ]:

```
#Importing Libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier #Decision Tree Classifier
from sklearn.model_selection import train_test_split #train_test_split function
from sklearn.metrics import classification_report, confusion_matrix #scikit-learn metrics
from sklearn.model_selection import RandomizedSearchCV #To perform Hyper-Parameter Tuning
from scipy.stats import randint #To perform Hyper-Parameter Tuning
```

In [ ]:

```
#Loading the data
final_features.head()
```

Out[ ]:

	name	country	claimed	veg	vegan	gf	awards	pop_score	top_tag	top_cuisine	num_features	num_me
0	Le 147	France	1	0	0	0	0	2.0	Cheap Eats	French	6	
1	Le Saint Jouvent	France	0	0	0	0	0	1.0	Cheap Eats		0	
2	Au Bout du Pont	France	1	0	0	0	0	1.0	Cheap Eats	French	4	
3	Le Relais de Naiade	France	1	0	0	0	0	1.0	Cheap Eats	French	5	
4	Relais Du MontSeigne	France	0	0	0	0	0	1.0	Mid-range	French	4	

In [ ]:

```
features_cols=['claimed', 'veg', 'vegan', 'gf', 'awards',
        'pop_score', 'num_features', 'num_meals',
        'p_excellent', 'p_vgood', 'p_average', 'p_poor', 'p_terrible']

X = final_features[features_cols] # Features
y = final_features.ave_rating # Target variable
y=y.astype('int')
```

In [ ]:

```
#Splitting the data into 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

4.2 Hyperparameter Tuning

In [ ]:
```
# Code source: [2]

# Creating the hyperparameter grid
param_dist = {"max_depth": [3, None],
                        "max_features": randint(1, 14),
                        "min_samples_leaf": randint(1, 9),
                        "criterion": ["gini", "entropy"]}

# Instantiating Decision Tree classifier
tree_param = DecisionTreeClassifier()

# Instantiating RandomizedSearchCV object
tree_cv = RandomizedSearchCV(tree_param, param_dist, cv = 5)

tree_cv.fit(X_train, y_train)

# Print the tuned parameters and score
print("Best Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Score: {}".format(tree_cv.best_score_))
```

```
Best Decision Tree Parameters: {'criterion': 'entropy', 'max_depth': None, 'max_features':
7, 'min_samples_leaf': 8}
Score: 0.8420703380169359
```

In [ ]:
```
tree_cv.best_estimator_
```

Out[ ]:
```
DecisionTreeClassifier(criterion='entropy', max_features=7, min_samples_leaf=8)
```

4.3 Training and Evaluation

In [ ]:
```
#Building Decision Tree Model
# code reference: [1]
# Decision Tree Classifer model

clf = DecisionTreeClassifier(criterion='entropy', max_features=7, min_samples_leaf=8)

# fitting the model
clf.fit(X_train, y_train)

#predicting the model
y_pred = clf.predict(X_test)
```

In [ ]:
```
confusion_matrix(y_test, y_pred)
```

Out[ ]:
```
array([[  2984,    283,    130,     17,      4],
       [   451,   9925,   2108,    489,    153],
       [   273,   2094,  51915,  15677,   1708],
       [    31,    235,  10370, 175133,   8927],
       [     0,      6,    229,   8606,  33341]])
```

In [ ]:
```
#Checking the accuracy for test set
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.80      0.87      0.83      3418
           2       0.79      0.76      0.77     13126
           3       0.80      0.72      0.76     71667
           4       0.88      0.90      0.89    194696
           5       0.76      0.79      0.77     42182
```

```
    accuracy                           0.84    325089
   macro avg       0.80      0.81      0.81    325089
weighted avg       0.84      0.84      0.84    325089
```

In [ ]:

```
#Checking for overfitting or underfitting

print('Training score:',(clf.score(X_train, y_train)))

print('Test score:',(clf.score(X_test, y_test)))
```

```
Training score: 0.8759582926784925
Test score: 0.8406867042563728
```

Training and testing scores are close in value and therefore there is no under or overfitting.

4.4 Learning Curve Analysis

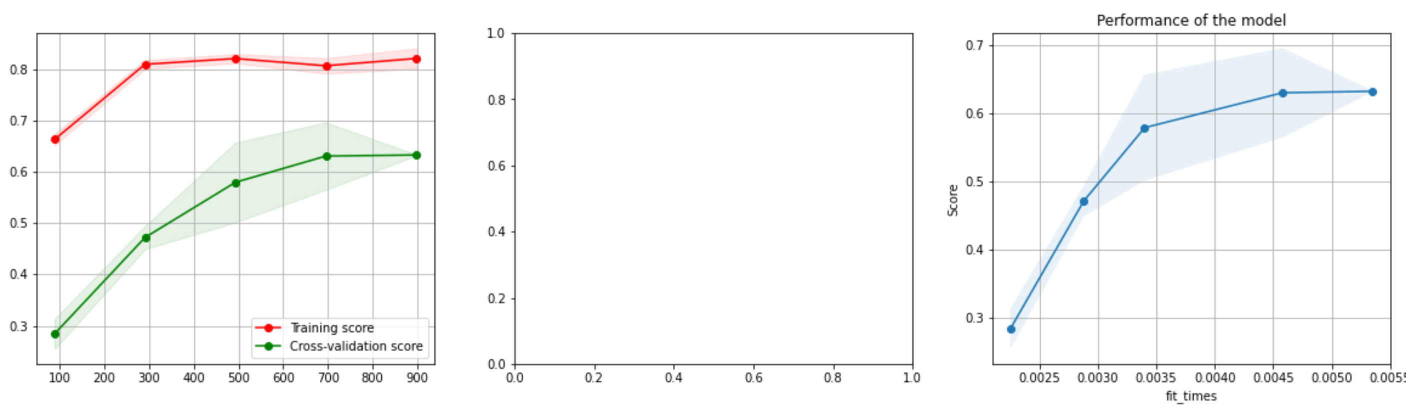
In [ ]:

```
# Code source: [4]
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

fig, axes = plt.subplots(1, 3, figsize=(20, 5))

train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(
    clf,
    X,
    y,
    cv=2,
    n_jobs=4,
    return_times=True,
)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
fit_times_mean = np.mean(fit_times, axis=1)
fit_times_std = np.std(fit_times, axis=1)

# Plot learning curve
axes[0].grid()
axes[0].fill_between(
    train_sizes,
    train_scores_mean - train_scores_std,
    train_scores_mean + train_scores_std,
    alpha=0.1,
    color="r",
)
axes[0].fill_between(
    train_sizes,
    test_scores_mean - test_scores_std,
    test_scores_mean + test_scores_std,
    alpha=0.1,
    color="g",
)
axes[0].plot(
    train_sizes, train_scores_mean, "o-", color="r", label="Training score"
)
axes[0].plot(
    train_sizes, test_scores_mean, "o-", color="g", label="Cross-validation score"
```

```
)
axes[0].legend(loc="best")


# Plot fit_time vs score
fit_time_argsort = fit_times_mean.argsort()
fit_time_sorted = fit_times_mean[fit_time_argsort]
test_scores_mean_sorted = test_scores_mean[fit_time_argsort]
test_scores_std_sorted = test_scores_std[fit_time_argsort]
axes[2].grid()
axes[2].plot(fit_time_sorted, test_scores_mean_sorted, "o-")
axes[2].fill_between(
    fit_time_sorted,
    test_scores_mean_sorted - test_scores_std_sorted,
    test_scores_mean_sorted + test_scores_std_sorted,
    alpha=0.1,
)
axes[2].set_xlabel("fit_times")
axes[2].set_ylabel("Score")
axes[2].set_title("Performance of the model")
```

Out[ ]:

```
Text(0.5, 1.0, 'Performance of the model')
```



The learning curve shows that the model performance during training. Using 2-fold cross validation, the validiation score plateaus around 6.3 around epoch 900. The training score plateaus around 8.1 around epoch 400. Training can be stopped and the model is likely well fitted.

5. Performance analysis:

The model's performance shows an accuracy of 84\% and therefore the model can be used to predict a restaurant's average rating with a reasonable amount of confidence. By inspecting the confusion matrix, the incorrect classifications are most often in the ranking directly above or below the correct class. Therefore, even incorrect predictions will likely give a close average rating for the restaurant. This model can be used to predict average rating in order to select a high quality restaurants to feature, and therefore fulfils the business problem.

References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[2] https://www.geeksforgeeks.org/hyperparameter-tuning/

[3] Class tutorial

[4] https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

[5] Course text (FUNDAMENTALS OF MACHINE LEARNING FOR PREDICTIVE DATA ANALYTICS, Algorithms, Worked Examples, and Case Studies by John D. Kelleher, Brian Mac Namee, Aoife D’Arcy)