EE769- Introduction to Machine Learning

Classification, Feature Engineering and Deployment of machine learning models

Instructor:

Prof. Amit Sethi

Submitted By:

Saurabh N. Pawar
Department of Aerospace Engineering
Roll Number: 23M0003

Indian Institute of Technology, Bombay

Code

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# I request the TA's to upload the .xls and .csv files for evaluation of the code

# Load the datasets
red_wine_data = pd.read_csv('winequality-red.csv', sep=';')  # Load red wine dataset
white_wine_data = pd.read_csv('winequality-white.csv', sep=';')  # Load white wine dataset

# Display basic information about the datasets
print("Red Wine Dataset:")
print(red_wine_data.info())  # Display information about red wine dataset

print("\nWhite Wine Dataset:")
print(white_wine_data.info())  # Display information about white wine dataset

# The syntax is taken from the ChatGPT and changes has been made
# Visualize the distributions of some key features

plt.figure(figsize=(12, 6))
plt.subplot(2, 2, 1)
sns.histplot(red_wine_data['fixed acidity'], color='red', kde=True)
plt.title('Fixed Acidity - Red Wine')
plt.subplot(2, 2, 2)
sns.histplot(white_wine_data['fixed acidity'], color='white', kde=True)
plt.title('Fixed Acidity - White Wine')
plt.subplot(2, 2, 3)
sns.histplot(red_wine_data['volatile acidity'], color='red', kde=True)
plt.title('Volatile Acidity - Red Wine')
plt.subplot(2, 2, 4)
sns.histplot(white_wine_data['volatile acidity'], color='white', kde=True)
plt.title('Volatile Acidity - White Wine')
plt.tight_layout()
plt.show()

# This part of the code is taken from the ChatGPT and changes has been made
# Pre-processing (if needed)
# For demonstration, let's concatenate the datasets
red_wine_data['wine_type'] = 'red'  # Create a new column indicating wine type (red)
white_wine_data['wine_type'] = 'white'  # Create a new column indicating wine type (white)
combined_data = pd.concat([red_wine_data, white_wine_data], ignore_index=True)  # Concatenate both
datasets

# Shuffle the combined dataset
combined_data = combined_data.sample(frac=1, random_state=42).reset_index(drop=True)
```

```
# The syntax is taken from the ChatGPT and changes has been made
# Splitting data into features and target
X = combined_data.drop(['quality', 'wine_type'], axis=1)  # Features
y = combined_data['quality']  # Target

# Splitting data into training and testing sets
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=42)

# Now you can use X_train, X_test, y_train, y_test for training and testing your regression mode

from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
# Define models
models = {
    'Random Forest': RandomForestRegressor(),
    'SVR': SVR(),
    'Neural Network': MLPRegressor(max_iter=1000)
}
# The syntax is taken from the ChatGPT and changes has been made
# Define hyperparameters for tuning
parameters = {
    'Random Forest': {'n_estimators': [10, 50, 100], 'max_depth': [None, 10, 20]},
    'SVR': {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 'scale']},
    'Neural Network': {'hidden_layer_sizes': [(50,), (100,), (50, 50)], 'alpha': [0.0001, 0.001, 0.01]}
}
# This part of the code is taken from the ChatGPT and changes has been made
# Train, validate, and test the models
for model_name, model in models.items():
    print(f"Training {model_name}...")
    grid_search = GridSearchCV(model, parameters[model_name], cv=3,
scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_

    # Validate the model
    print(f"Best parameters for {model_name}: {grid_search.best_params_}")
    print(f"Best mean squared error on validation set for {model_name}: {-grid_search.best_score_}")

    # Test the model
    y_pred_test = best_model.predict(X_test)
    mse_test = mean_squared_error(y_test, y_pred_test)
    print(f"Mean squared error on test set for {model_name}: {mse_test}\n")
```

```
nn_weights_input_hidden = best_model.coefs_[0]  # weights connecting input layer to hidden layer
nn_feature_importance = nn_weights_input_hidden.mean(axis=1)  # Taking mean of weights
# The syntax is taken from the ChatGPT and changes has been made
print("Neural Network Feature Importance:")
for i, (feature, importance) in enumerate(zip(X.columns, nn_feature_importance)):
    print(f"{feature}: {importance}")
print()

# Testing the model for red wine data using the model trained on white wine data
# This part of the code is taken from the ChatGPT and changes has been made
y_pred_red_from_white = best_model.predict(red_wine_data.drop(['quality', 'wine_type'], axis=1))
mse_red_from_white = mean_squared_error(red_wine_data['quality'], y_pred_red_from_white)
print(f"Mean squared error on red wine data using model trained on white wine data:
{mse_red_from_white}")
# Testing the model for white wine data using the model trained on red wine data
# The syntax is taken from the ChatGPT and changes has been made
y_pred_white_from_red = best_model.predict(white_wine_data.drop(['quality', 'wine_type'], axis=1))
mse_white_from_red = mean_squared_error(white_wine_data['quality'], y_pred_white_from_red)
print(f"Mean squared error on white wine data using model trained on red wine data:
{mse_white_from_red}")

# Import necessary libraries
import pandas as pd             # For data manipulation
import numpy as np              # For numerical computations
import matplotlib.pyplot as plt # For data visualization
import seaborn as sns           # For enhanced visualization
from sklearn.experimental import enable_iterative_imputer  # Enable IterativeImputer
from sklearn.impute import IterativeImputer  # For imputing missing values

# Load the data from an Excel file
data = pd.read_excel("Data_Cortex_Nuclear.xls")

# The syntax is taken from the ChatGPT and changes has been made
# Explore the data
print("Data Overview:")
print(data.head())  # Display the first few rows of the dataset
print("\nData Information:")
print(data.info())  # Display information about the dataset

# Summary statistics
print("\nSummary Statistics:")
print(data.describe())  # Display summary statistics of the numerical columns

# This part of the code is taken from the ChatGPT and changes has been made
# Check for missing values
print("\nMissing Values:")
print(data.isnull().sum())  # Count missing values in each column
```

```
# This part of the code is taken from the ChatGPT and changes has been made
# Visualize missing values
plt.figure(figsize=(10, 6))
sns.heatmap(data.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Values Heatmap')
plt.show()  # Plot a heatmap to visualize missing values in the dataset

# Select relevant columns for prediction
gene_expression_columns = data.columns[1:78]  # Select columns representing gene expression
selected_columns = ['Genotype'] + list(gene_expression_columns)  # Include 'Genotype' column
selected_data = data[selected_columns]  # Create a new DataFrame with selected columns

# Impute missing values using multivariate feature imputation
imputer = IterativeImputer(random_state=42)  # Initialize an IterativeImputer
imputed_data = imputer.fit_transform(selected_data.drop(columns=['Genotype']))  # Impute missing
values

# The syntax is taken from the ChatGPT and changes has been made
# Convert imputed data to DataFrame
imputed_df = pd.DataFrame(imputed_data, columns=gene_expression_columns)  # Create DataFrame
from imputed data

# Concatenate Genotype column with imputed DataFrame
imputed_df['Genotype'] = selected_data['Genotype']  # Add 'Genotype' column to the DataFrame

# Check for missing values after imputation
print("\nMissing Values After Imputation:")
print(imputed_df.isnull().sum())  # Count missing values in each column after imputation

# The syntax is taken from the ChatGPT and changes has been made
# Visualize the distribution of the selected variables
plt.figure(figsize=(20, 15))
for i, col in enumerate(gene_expression_columns, start=1):
    plt.subplot(10, 8, i)
    sns.histplot(imputed_df[col], kde=True)
    plt.title(col)
plt.tight_layout()
plt.show()  # Plot histograms to visualize the distribution of gene expression variables

# Import necessary libraries
import pandas as pd            # For data manipulation
import numpy as np             # For numerical computations
from sklearn.model_selection import train_test_split, GridSearchCV  # For splitting data and
hyperparameter tuning
from sklearn.ensemble import RandomForestClassifier  # For Random Forest classification
from sklearn.svm import SVC      # For Support Vector Classification
from sklearn.neural_network import MLPClassifier  # For Neural Network classification
from sklearn.metrics import accuracy_score  # For accuracy evaluation
```

```
# Load the data
data = pd.read_excel("Data_Cortex_Nuclear.xls")
# Drop rows with missing values
# This part of the code is taken from the ChatGPT and changes has been made
data.dropna(inplace=True)
# Split data into features and target
X = data.iloc[:, 1:78]  # Features
y = data['Genotype']    # Target
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Model 1: Random Forest
# The syntax is taken from the ChatGPT and changes has been made
rf_model = RandomForestClassifier(random_state=42)
param_grid_rf = {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20]}
grid_search_rf = GridSearchCV(rf_model, param_grid_rf, cv=5)
grid_search_rf.fit(X_train, y_train)
# Print results for Random Forest
print("Random Forest:")
print("Best parameters:", grid_search_rf.best_params_)
rf_best_model = grid_search_rf.best_estimator_
rf_accuracy = accuracy_score(y_test, rf_best_model.predict(X_test))
print("Accuracy:", rf_accuracy)
```

Results

Red Wine Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):

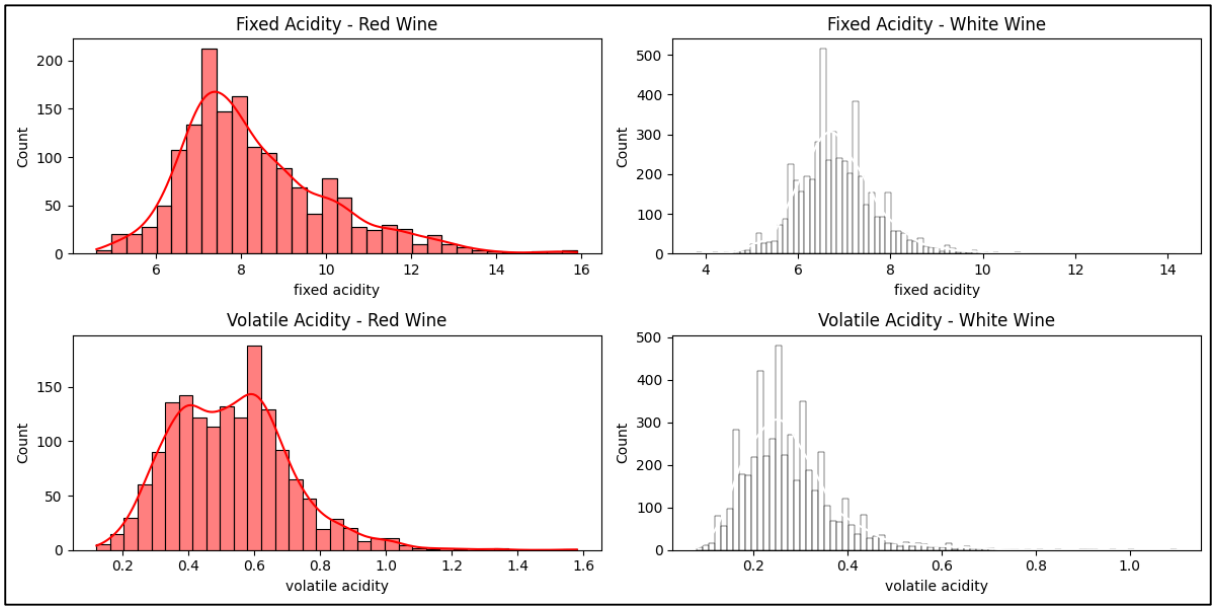
#	Column	Non-Null Count	Dtype
0	fixed acidity	1599 non-null	float64
1	volatile acidity	1599 non-null	float64
2	citric acid	1599 non-null	float64
3	residual sugar	1599 non-null	float64
4	chlorides	1599 non-null	float64
5	free sulfur dioxide	1599 non-null	float64
6	total sulfur dioxide	1599 non-null	float64
7	density	1599 non-null	float64
8	pH	1599 non-null	float64
9	sulphates	1599 non-null	float64
10	alcohol	1599 non-null	float64
11	quality	1599 non-null	int64

dtypes: float64(11), int64(1)
memory usage: 150.0 KB
None

White Wine Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):

#	Column	Non-Null Count	Dtype

```
---  ------                --------------  -----
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
None
```



Training Random Forest...
Best parameters for Random Forest: {'max_depth': 20, 'n_estimators': 100}
Best mean squared error on validation set for Random Forest: 0.3960867539233737
Mean squared error on test set for Random Forest: 0.3863165608359199

Training SVR...
Best parameters for SVR: {'C': 10, 'gamma': 'scale'}
Best mean squared error on validation set for SVR: 0.56376278672802
Mean squared error on test set for SVR: 0.5707588235050213

Training Neural Network...
Best parameters for Neural Network: {'alpha': 0.0001, 'hidden_layer_sizes': (100,)}
Best mean squared error on validation set for Neural Network: 0.5505165437355222
Mean squared error on test set for Neural Network: 0.5245394760083404

Neural Network Feature Importance:
fixed acidity: 0.027361917832013007
volatile acidity: 0.025578833821419608
citric acid: 0.011977498016283178

residual sugar: 0.009382103588474973
chlorides: -0.018391339761916887
free sulfur dioxide: 0.0013897885289201216
total sulfur dioxide: 0.026680000896675538
density: -0.005316843391519408
pH: -0.005859664460694103
sulphates: 0.0037559344883925623
alcohol: 0.02988135552054656

Mean squared error on red wine data using model trained on white wine data: 0.43078027479404984
Mean squared error on white wine data using model trained on red wine data: 0.5390466038138811
Data Overview:

	MouseID	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	\
0	309_1	0.503644	0.747193	0.430175	2.816329	5.990152	0.218830	
1	309_2	0.514617	0.689064	0.411770	2.789514	5.685038	0.211636	
2	309_3	0.509183	0.730247	0.418309	2.687201	5.622059	0.209011	
3	309_4	0.442107	0.617076	0.358626	2.466947	4.979503	0.222886	
4	309_5	0.434940	0.617430	0.358802	2.365785	4.718679	0.213106	

	pBRAF_N	pCAMKII_N	pCREB_N	...	pCFOS_N	SYP_N	H3AcK18_N	\
0	0.177565	2.373744	0.232224	...	0.108336	0.427099	0.114783	
1	0.172817	2.292150	0.226972	...	0.104315	0.441581	0.111974	
2	0.175722	2.283337	0.230247	...	0.106219	0.435777	0.111883	
3	0.176463	2.152301	0.207004	...	0.111262	0.391691	0.130405	
4	0.173627	2.134014	0.192158	...	0.110694	0.434154	0.118481	

	EGR1_N	H3MeK4_N	CaNA_N	Genotype	Treatment	Behavior	class
0	0.131790	0.128186	1.675652	Control	Memantine	C/S	c-CS-m
1	0.135103	0.131119	1.743610	Control	Memantine	C/S	c-CS-m
2	0.133362	0.127431	1.926427	Control	Memantine	C/S	c-CS-m
3	0.147444	0.146901	1.700563	Control	Memantine	C/S	c-CS-m
4	0.140314	0.148380	1.839730	Control	Memantine	C/S	c-CS-m

[5 rows x 82 columns]

Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1080 entries, 0 to 1079
Data columns (total 82 columns):

#	Column	Non-Null Count	Dtype
0	MouseID	1080 non-null	object
1	DYRK1A_N	1077 non-null	float64
2	ITSN1_N	1077 non-null	float64
3	BDNF_N	1077 non-null	float64
4	NR1_N	1077 non-null	float64
5	NR2A_N	1077 non-null	float64
6	pAKT_N	1077 non-null	float64
7	pBRAF_N	1077 non-null	float64
8	pCAMKII_N	1077 non-null	float64
9	pCREB_N	1077 non-null	float64
10	pELK_N	1077 non-null	float64
11	pERK_N	1077 non-null	float64
12	pJNK_N	1077 non-null	float64
13	PKCA_N	1077 non-null	float64
14	pMEK_N	1077 non-null	float64
15	pNR1_N	1077 non-null	float64

16 pNR2A_N 1077 non-null float64
17 pNR2B_N 1077 non-null float64
18 pPKCAB_N 1077 non-null float64
19 pRSK_N 1077 non-null float64
20 AKT_N 1077 non-null float64
21 BRAF_N 1077 non-null float64
22 CAMKII_N 1077 non-null float64
23 CREB_N 1077 non-null float64
24 ELK_N 1062 non-null float64
25 ERK_N 1077 non-null float64
26 GSK3B_N 1077 non-null float64
27 JNK_N 1077 non-null float64
28 MEK_N 1073 non-null float64
29 TRKA_N 1077 non-null float64
30 RSK_N 1077 non-null float64
31 APP_N 1077 non-null float64
32 Bcatenin_N 1062 non-null float64
33 SOD1_N 1077 non-null float64
34 MTOR_N 1077 non-null float64
35 P38_N 1077 non-null float64
36 pMTOR_N 1077 non-null float64
37 DSCR1_N 1077 non-null float64
38 AMPKA_N 1077 non-null float64
39 NR2B_N 1077 non-null float64
40 pNUMB_N 1077 non-null float64
41 RAPTOR_N 1077 non-null float64
42 TIAM1_N 1077 non-null float64
43 pP70S6_N 1077 non-null float64
44 NUMB_N 1080 non-null float64
45 P70S6_N 1080 non-null float64
46 pGSK3B_N 1080 non-null float64
47 pPKCG_N 1080 non-null float64
48 CDK5_N 1080 non-null float64
49 S6_N 1080 non-null float64
50 ADARB1_N 1080 non-null float64
51 AcetylH3K9_N 1080 non-null float64
52 RRP1_N 1080 non-null float64
53 BAX_N 1080 non-null float64
54 ARC_N 1080 non-null float64
55 ERBB4_N 1080 non-null float64
56 nNOS_N 1080 non-null float64
57 Tau_N 1080 non-null float64
58 GFAP_N 1080 non-null float64
59 GluR3_N 1080 non-null float64
60 GluR4_N 1080 non-null float64
61 IL1B_N 1080 non-null float64
62 P3525_N 1080 non-null float64
63 pCASP9_N 1080 non-null float64
64 PSD95_N 1080 non-null float64
65 SNCA_N 1080 non-null float64
66 Ubiquitin_N 1080 non-null float64
67 pGSK3B_Tyr216_N 1080 non-null float64
68 SHH_N 1080 non-null float64
69 BAD_N 867 non-null float64
70 BCL2_N 795 non-null float64
71 pS6_N 1080 non-null float64
72 pCFOS_N 1005 non-null float64

```
 73  SYP_N           1080 non-null   float64
 74  H3AcK18_N       900 non-null    float64
 75  EGR1_N          870 non-null    float64
 76  H3MeK4_N        810 non-null    float64
 77  CaNA_N          1080 non-null   float64
 78  Genotype        1080 non-null   object
 79  Treatment       1080 non-null   object
 80  Behavior        1080 non-null   object
 81  class           1080 non-null   object
dtypes: float64(77), object(5)
memory usage: 692.0+ KB
None

Summary Statistics:
        DYRK1A_N     ITSN1_N      BDNF_N       NR1_N      NR2A_N  \
count  1077.000000  1077.000000  1077.000000  1077.000000  1077.000000
mean      0.425810     0.617102     0.319088     2.297269     3.843934
std       0.249362     0.251640     0.049383     0.347293     0.933100
min       0.145327     0.245359     0.115181     1.330831     1.737540
25%       0.288121     0.473361     0.287444     2.057411     3.155678
50%       0.366378     0.565782     0.316564     2.296546     3.760855
75%       0.487711     0.698032     0.348197     2.528481     4.440011
max       2.516367     2.602662     0.497160     3.757641     8.482553

          pAKT_N      pBRAF_N    pCAMKII_N      pCREB_N       pELK_N  ...  \
count  1077.000000  1077.000000  1077.000000  1077.000000  1077.000000  ...
mean      0.233168     0.181846     3.537109     0.212574     1.428682  ...
std       0.041634     0.027042     1.295169     0.032587     0.466904  ...
min       0.063236     0.064043     1.343998     0.112812     0.429032  ...
25%       0.205755     0.164595     2.479834     0.190823     1.203665  ...
50%       0.231177     0.182302     3.326520     0.210594     1.355846  ...
75%       0.257261     0.197418     4.481940     0.234595     1.561316  ...
max       0.539050     0.317066     7.464070     0.306247     6.113347  ...

           SHH_N       BAD_N      BCL2_N       pS6_N      pCFOS_N  \
count  1080.000000  867.000000  795.000000  1080.000000  1005.000000
mean      0.226676    0.157914    0.134762     0.121521     0.131053
std       0.028989    0.029537    0.027417     0.014276     0.023863
min       0.155869    0.088305    0.080657     0.067254     0.085419
25%       0.206395    0.136424    0.115554     0.110839     0.113506
50%       0.224000    0.152313    0.129468     0.121626     0.126523
75%       0.241655    0.174017    0.148235     0.131955     0.143652
max       0.358289    0.282016    0.261506     0.158748     0.256529

           SYP_N   H3AcK18_N      EGR1_N    H3MeK4_N       CaNA_N
count  1080.000000  900.000000  870.000000  810.000000  1080.000000
mean      0.446073    0.169609    0.183135    0.205440     1.337784
std       0.066432    0.059402    0.040406    0.055514     0.317126
min       0.258626    0.079691    0.105537    0.101787     0.586479
25%       0.398082    0.125848    0.155121    0.165143     1.081423
50%       0.448459    0.158240    0.174935    0.193994     1.317441
75%       0.490773    0.197876    0.204542    0.235215     1.585824
max       0.759588    0.479763    0.360692    0.413903     2.129791

[8 rows x 77 columns]

Missing Values:
```

MouseID 0
DYRK1A_N 3
ITSN1_N 3
BDNF_N 3
NR1_N 3
..
CaNA_N 0
Genotype 0
Treatment 0
Behavior 0
class 0
Length: 82, dtype: int64