NAME: Amish Faldu
NJIT UCID: af557
Email Address: af557@njit.edu
04/10/2024
Professor: Yasser Abduallah
CS 634 104 Data Mining

Final term Project Report

Implementation and Code Usage

Supervised Machine Learning (Classification)

Abstract:
In this project, I delve into the various machine learning algorithms for classifying the different dry bean. The algorithms are evaluated using F1-Score metric.

Introduction:
Supervised machine learning learns the patterns in the dataset and tries to best-fit the dataset so that it can accurately differentiate between items.

In this implementation, I've applied Random Forest, SVM, KNN and Conv1D algorithm to a dry bean dataset. Key steps in this process included:

- Loading the dataset from CSV files.
- Data analysis.
- Data pre-processing
- Model training

Core Concepts and Principles:

Supervised Machine Learning:
Supervised Machine learning is a part of Machine learning where the algorithm has access to the data points as well as label associated with that data point.

Neuron:
Neuron in neural network is machine representation of the biological neuron. The neuron accepts inputs, performs computation that is dot product of the input and weights, outputs activated computation.

Neural Network:
Neural network is an interconnection of the neurons.

F1-Score:
It is calculated based on model's precision and recall. The precision and recall are calculated on TP, FP, TN and TP.

K-Fold Cross Validation:
This is a technique to divide the training dataset into train and validation dataset. This way we can train the model as well as check how well it is performing on the unseen data without exposing it to the test data.

Project Workflow:

Our project follows a structured workflow involving various stages and the application of the various machine learning algorithm**s.**

Data Loading
We begin by loading csv dry bean dataset. Each item contains of 16 attributes and a label.

Data Analysis
We analyse data for any missing values, observe the attribute distribution, label distribution and correlations between different attributes.

Data Pre-processing:
To ensure clean and compatible data, we pre-process the dataset by converting string to numerical representation, normalizing dataset for SVM and Neural Networks.

Machine Learning Algorithms
- **Random Forest**
- **SVM**
- **KNN**
- **Conv1D Deep Learning**

Results and Evaluation:
The machine learning's accuracy and efficiency are evaluated based on performance measures such as F1-score, precision, recall and confusion metrics.

Conclusion:

In conclusion, this project demonstrates the application and evaluation of supervised data mining (Machine Learning) concepts and methods on dry bean classification dataset.

Directory Structure:

-------- dataset/ => This contains all the data files needed to run the code
-------- models/ => This is the saved neural net model.
-------- docs/ => This contains images for documentation
-------- .gitignore => Ignore files / folders to include in git history
-------- main.ipynb => This is the main Jupyter notebook that contains the code to run brute force and library's algorithm
-------- Finalterm-Project-Report.pdf => Obviously, project report
-------- README.md => This contains all the instructions to setup the project locally
-------- requirements.txt => This contains all the packages to run the code

---

Steps to run the program:

1. Make sure that you've python installed on your machine. To download and install Python, go to https://www.python.org/. To check if you've python installed, run `python3 --version`.
   NOTE - I have used python version 3.11.7 to run the program

2. Create a virtual environment using command `python3 -m venv .venv`.
3. Activate the virtual environment using command `source .venv/bin/activate`.
4. Install all the packages in **requirements.txt** using `pip install -r requirements.txt`.
5. Select existing `.venv` python environment as the kernel for the `main.ipynb` Jupyter notebook.
6. Run cells in `main.ipynb` Jupyter notebook.

Screenshots:

Pre-requisite steps

Step – 1

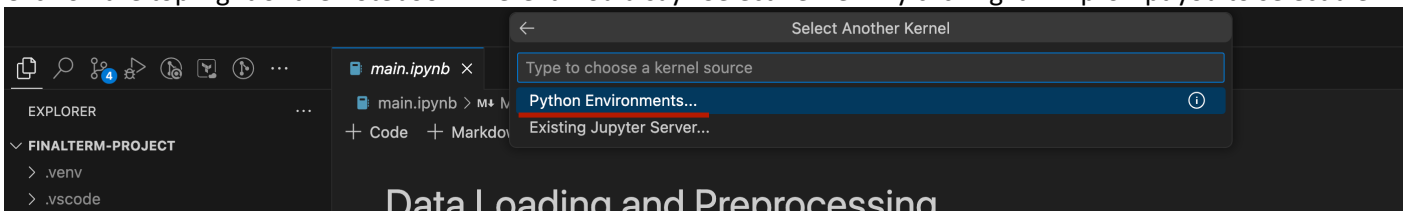
Step – 2

Step – 3

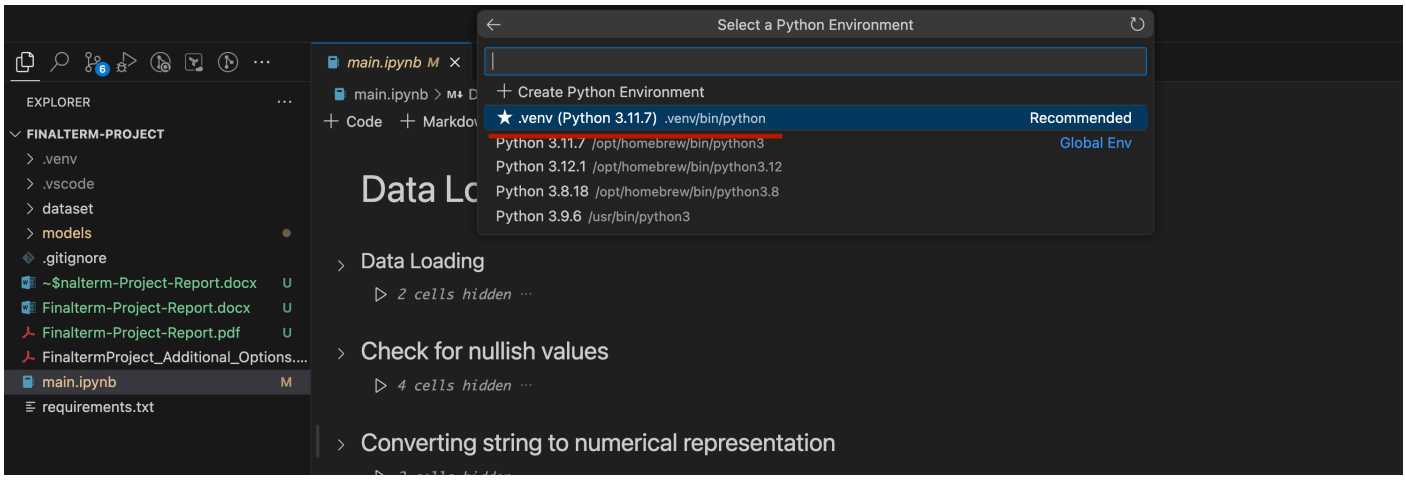
Step – 4

Step – 5

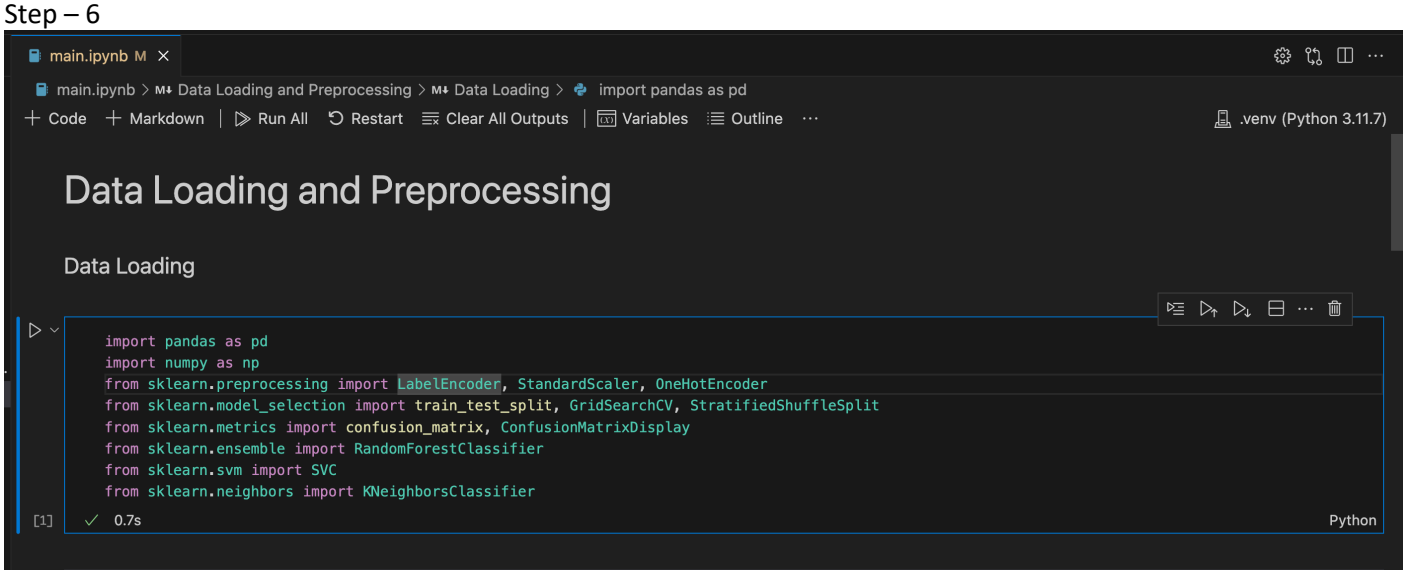
Click on the top-right of the notebook where it would say “Select Kernel”. By clicking it will prompt you to select a env.



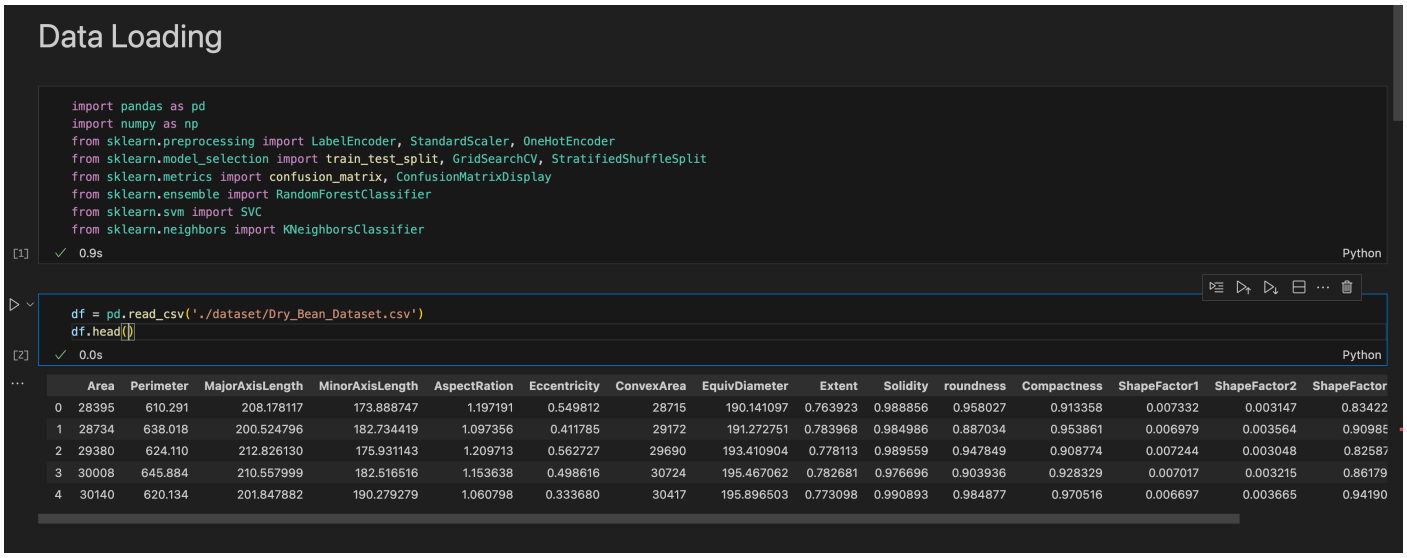
Now select the virtual we created in step – 2 as the environment. After that, it would show you the selected kernel in top-right section.


Select a Python Environment
Create Python Environment
.venv (Python 3.11.7) .venv/bin/python
Recommended
Python 3.11.7 /opt/homebrew/bin/python3
Global Env
Python 3.12.1 /opt/homebrew/bin/python3.12
Python 3.8.18 /opt/homebrew/bin/python3.8
Python 3.9.6 /usr/bin/python3
main.ipynb M
EXPLORER
FINALTERM-PROJECT
.venv
.vscode
dataset
models
.gitignore
~$nalterm-Project-Report.docx
Finalterm-Project-Report.docx
Finalterm-Project-Report.pdf
FinaltermProject_Additional_Options....
main.ipynb
requirements.txt
Code
Data Loading
2 cells hidden
Check for nullish values
4 cells hidden
Converting string to numerical representation

Step – 6


main.ipynb M
main.ipynb > Data Loading and Preprocessing > Data Loading > import pandas as pd
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
.venv (Python 3.11.7)
Data Loading and Preprocessing
Data Loading
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedShuffleSplit
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
[1]
0.7s
Python

Below are screenshots of the running code from Jupyter notebook:
Data loading



Data Loading

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedShuffleSplit
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

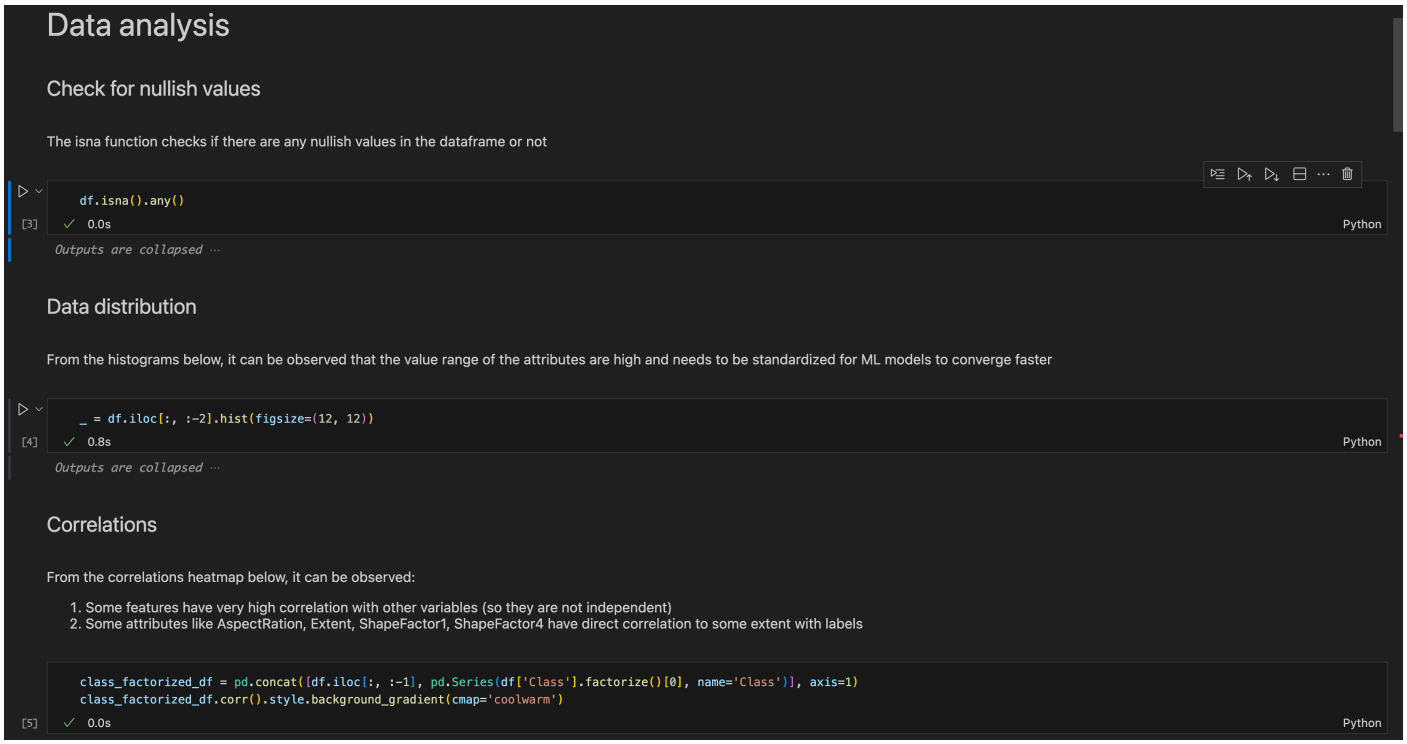
[1] ✓ 0.9s Python

```
df = pd.read_csv('./dataset/Dry_Bean_Dataset.csv')
df.head()
```

[2] ✓ 0.0s Python

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor
0	28395	610.291	208.178117	173.888747	1.197191	0.549812	28715	190.141097	0.763923	0.988856	0.958027	0.913358	0.007332	0.003147	0.83422
1	28734	638.018	200.524796	182.734419	1.097356	0.411785	29172	191.272751	0.783968	0.984986	0.887034	0.953861	0.006979	0.003564	0.90985
2	29380	624.110	212.826130	175.931143	1.209713	0.562727	29690	193.410904	0.778113	0.989559	0.947849	0.908774	0.007244	0.003048	0.82587
3	30008	645.884	210.557999	182.516516	1.153638	0.498616	30724	195.467062	0.782681	0.976696	0.903936	0.928329	0.007017	0.003215	0.86179
4	30140	620.134	201.847882	190.279279	1.060798	0.333680	30417	195.896503	0.773098	0.990893	0.984877	0.970516	0.006697	0.003665	0.94190

Data analysis



Data analysis

Check for nullish values

The isna function checks if there are any nullish values in the dataframe or not

```
df.isna().any()
```

[3] ✓ 0.0s Python

Outputs are collapsed ···

Data distribution

From the histograms below, it can be observed that the value range of the attributes are high and needs to be standardized for ML models to converge faster

```
_ = df.iloc[:, :-2].hist(figsize=(12, 12))
```

[4] ✓ 0.8s Python

Outputs are collapsed ···

Correlations

From the correlations heatmap below, it can be observed:

1. Some features have very high correlation with other variables (so they are not independent)
2. Some attributes like AspectRation, Extent, ShapeFactor1, ShapeFactor4 have direct correlation to some extent with labels

```
class_factorized_df = pd.concat([df.iloc[:, :-1], pd.Series(df['Class'].factorize()[0], name='Class')], axis=1)
class_factorized_df.corr().style.background_gradient(cmap='coolwarm')
```

[5] ✓ 0.0s Python

Correlations

From the correlations heatmap below, it can be observed:

1. Some features have very high correlation with other variables (so they are not independent)
2. Some attributes like AspectRation, Extent, ShapeFactor1, ShapeFactor4 have direct correlation to some extent with labels

```
class_factorized_df = pd.concat([df.iloc[:, :-1], pd.Series(df['Class'].factorize()[0], name='Class')], axis=1)
class_factorized_df.corr().style.background_gradient(cmap='coolwarm')
```

[5] ✓ 0.0s Python

Outputs are collapsed ···

Label distribution

From the below bar graph, we can infer following points:

1. There aren't any out-of-place/nullish class values
2. We need to convert the string values of class to numerical representation
3. The class distribution is unbalanced, so the training data needs to be split with stratification.

+ Code + Markdown

```
df['Class'].value_counts().plot(kind='bar')
```

[6] ✓ 0.0s Python

Outputs are collapsed ···

Data pre-processing

Data Preprocessing

Converting string to numerical representation

```
encoder = LabelEncoder()
df['Class_numerical'] = encoder.fit_transform(df['Class'])
```

[7] ✓ 0.0s Python

```
encoder.inverse_transform([0, 1, 2, 3, 4, 6, 5])
```

[8] ✓ 0.0s Python

```
··· array(['BARBUNYA', 'BOMBAY', 'CALI', 'DERMASON', 'HOROZ', 'SIRA', 'SEKER'],
      dtype=object)
```

Normalizing dataset

+ Code + Markdown

```
X_train, X_test, y_train, y_test = train_test_split(
    df.iloc[:, :-2],
    df["Class_numerical"],
    test_size=0.2,
    random_state=42,
    shuffle=True,
    stratify=df["Class_numerical"],
)
```

[9] ✓ 0.0s Python

Normalizing dataset

```
X_train, X_test, y_train, y_test = train_test_split(
    df.iloc[:, :-2],
    df["Class_numerical"],
    test_size=0.2,
    random_state=42,
    shuffle=True,
    stratify=df["Class_numerical"],
)
```

[9] ✓ 0.0s Python

```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X_train)
X_train = pd.DataFrame(scaled_features, columns=df.columns[:-2])
X_test = pd.DataFrame(scaler.transform(X_test), columns=df.columns[:-2])
X_train.head()
```

[10] ✓ 0.0s Python

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFa
0	-0.371362	-0.531131	-0.695370	0.086657	-1.293800	-1.515964	-0.380039	-0.361098	0.857421	1.256100	1.516864	1.446832	-0.339818	1.118082	1.4
1	0.028952	0.498818	0.783518	-0.540352	2.401812	1.493455	0.035323	0.150320	-2.941719	-0.843829	-2.378161	-1.999948	0.551406	-1.321497	-1.8
2	0.727078	0.795005	0.824382	0.924849	0.076902	0.330836	0.717120	0.923243	1.203117	0.618792	0.254498	-0.201104	-1.160932	-0.787674	-0.2
3	0.732131	0.883592	1.020897	0.725643	0.614895	0.721194	0.735383	0.928419	0.870459	-0.479395	-0.253981	-0.715959	-0.969786	-1.032437	-0.7
4	-0.133509	-0.257497	-0.481242	0.494139	-1.381023	-1.713495	-0.144039	-0.049766	0.713863	1.247235	1.531060	1.569816	-0.786632	0.919368	1.6

+ Code + Markdown

As you can see in below histograms, the range of features are normalized

```
_ = X_train.iloc[:, :-1].hist(figsize=(12, 12))
```

[11] ✓ 0.5s Python

Outputs are collapsed ···

Model Training

Random Forest

Random Forest

```
# params = {
#     "n_estimators": [100, 200, 300, 400],
#     "max_depth": [8, 10, 12],
#     "ccp_alpha": [5e-4, 1e-3],
# }
stratified_split = StratifiedShuffleSplit(n_splits=10, test_size=0.1, random_state=42)
# gridsearch_rf = GridSearchCV(
#     RandomForestClassifier(n_jobs=-1),
#     params,
#     cv=stratified_split,
#     n_jobs=-1,
#     verbose=3,
#     scoring=custom_calculate_f1_score
# )
# gridsearch_rf.fit(X_train, y_train)
# gridsearch_rf.best_params_

✨If you uncomment the commented code above, you will get the following output
# Output - {'ccp_alpha': 0.0005, 'max_depth': 12, 'n_estimators': 300}
```

[14] ✓ 0.0s Python

```python
random_forest_cv = {}
random_forest_cv_models = []

for i, (train_index, test_index) in enumerate(stratified_split.split(X_train, y_train)):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    random_forest = RandomForestClassifier(n_estimators=300, max_depth=12, ccp_alpha=5e-4, n_jobs=-1)
    random_forest.fit(X_train_fold, y_train_fold)

    y_pred = random_forest.predict(X_val_fold)
    y_pred_proba = random_forest.predict_proba(X_val_fold)[:, 1]
    random_forest_cv[i+1] = get_all_metrics(y_val_fold, y_pred)
    random_forest_cv[i+1]['Brier Score'] = np.mean((y_pred_proba - y_val_fold)**2)
    random_forest_cv[i+1]['Brier Skill Score'] = random_forest_cv[i+1]['Brier Score'] / (np.mean((y_val_fold - np.mean(y_pred_proba))**2))
    random_forest_cv_models.append(random_forest)

random_forest_cv['mean'] = pd.DataFrame(random_forest_cv).mean(axis=1)
pd.DataFrame(random_forest_cv)
```

[15] ✓ 5.2s Python

Outputs are collapsed ···

```python
# Mode function returns the values that appear most frequently in the array
y_pred = stats.mode([model.predict(X_test) for model in random_forest_cv_models]).mode

ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

RocCurveDisplay.from_predictions(y_test, y_pred)

results_comparison['RandomForest'] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "RandomForest"] = np.mean((y_pred - y_test) ** 2)
results_comparison.loc["Brier Skill Score", "RandomForest"] = results_comparison["RandomForest"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```

SVM

Additional Algorithm - SVM

```python
# params = {
#     "C": [10, 100, 1000],
#     "kernel": ["linear", "rbf", "sigmoid", "poly"],
#     'degree': [2, 4, 6]
# }
stratified_split = StratifiedShuffleSplit(n_splits=10, test_size=0.1, random_state=42)
# gridsearch_svc = GridSearchCV(
#     SVC(random_state=42),
#     params,
#     cv=stratified_split,
#     n_jobs=-1,
#     verbose=3,
#     scoring=custom_calculate_f1_score
# )
# gridsearch_svc.fit(X_train, y_train)
# gridsearch_svc.best_params_

# If you uncomment the commented code above, you will get the following output
# Output - {'C': 100, 'degree': 2, 'kernel': 'rbf'}
```

[17] ✓ 0.0s Python

```python
svc_cv = {}
svc_cv_models = []

for i, (train_index, test_index) in enumerate(stratified_split.split(X_train, y_train)):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    svc = SVC(C= 100, degree= 2, kernel= 'rbf', probability=True, random_state=42)
    svc.fit(X_train_fold, y_train_fold)

    y_pred = svc.predict(X_val_fold)
    y_pred_proba = svc.predict_proba(X_val_fold)[:, 1]
    svc_cv[i+1] = get_all_metrics(y_val_fold, y_pred)
    svc_cv[i+1]['Brier Score'] = np.mean((y_pred_proba - y_val_fold)**2)
    svc_cv[i+1]['Brier Skill Score'] = svc_cv[i+1]['Brier Score'] / (np.mean((y_val_fold - np.mean(y_pred_proba))**2))
    svc_cv_models.append(svc)

svc_cv['mean'] = pd.DataFrame(svc_cv).mean(axis=1)
pd.DataFrame(svc_cv)
```

[18] ✓ 13.6s Python

Outputs are collapsed ···

```python
# Mode function returns the values that appear most frequently in the array
y_pred = stats.mode([model.predict(X_test) for model in svc_cv_models]).mode
matrix = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

results_comparison['SVM'] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "SVM"] = np.mean((y_pred - y_test) ** 2)
results_comparison.loc["Brier Skill Score", "SVM"] = results_comparison["SVM"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```

[19] ✓ 0.9s Python

KNN

Additional Algorithm - KNN

```python
# params = {
#     "n_neighbors": [3, 5, 7],
#     "weights": ["uniform", "distance"],
#     "algorithm": ["ball_tree", "kd_tree", "brute"],
#     "leaf_size": [10, 30, 50],
#     "p": [1, 2, 3],
# }
stratified_split = StratifiedShuffleSplit(n_splits=6, test_size=0.1, random_state=42)
# gridsearch_knn = GridSearchCV(
#     KNeighborsClassifier(n_jobs=-1),
#     params,
#     cv=stratified_split,
#     n_jobs=-1,
#     verbose=3,
#     scoring=custom_calculate_f1_score
# )
# gridsearch_knn.fit(X_train, y_train)
# gridsearch_knn.best_params_

# If you uncomment the commented code above, you will get the following output
# Output - {'algorithm': 'ball_tree',
#  'leaf_size': 10,
#  'n_neighbors': 5,
#  'p': 2,
#  'weights': 'distance'}
```

[20] ✓ 0.0s Python

```python
knn_cv = {}
knn_cv_models = []

for i, (train_index, test_index) in enumerate(stratified_split.split(X_train, y_train)):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    knn = KNeighborsClassifier(
        algorithm="ball_tree",
        leaf_size=10,
        n_neighbors=5,
        p=2,
        weights="distance",
        n_jobs=-1,
    )
    knn.fit(X_train_fold, y_train_fold)

    y_pred = knn.predict(X_val_fold)
    y_pred_proba = knn.predict_proba(X_val_fold)[:, 1]
    knn_cv[i + 1] = get_all_metrics(y_val_fold, y_pred)
    knn_cv[i+1]['Brier Score'] = np.mean((y_pred_proba - y_val_fold)**2)
    knn_cv[i+1]['Brier Skill Score'] = knn_cv[i+1]['Brier Score'] / (np.mean((y_val_fold - np.mean(y_pred_proba))**2))
    knn_cv_models.append(knn)

knn_cv["mean"] = pd.DataFrame(knn_cv).mean(axis=1)
pd.DataFrame(knn_cv)
```

[21] ✓ 0.3s Python

```python
# Mode function returns the values that appear most frequently in the array
y_pred = stats.mode([model.predict(X_test) for model in knn_cv_models]).mode
matrix = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

results_comparison['KNN'] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "KNN"] = np.mean((y_pred - y_test) ** 2)
results_comparison.loc["Brier Skill Score", "KNN"] = results_comparison["KNN"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```

[22] ✓ 0.4s Python

Outputs are collapsed ···

Conv1D

Additional Deep Learning Algorithm - Conv1D

```python
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps" if torch.backends.mps.is_available() else "cpu"
)
print(f"Using {device} device for torch models")
```

[23] ✓ 0.0s Python

```
Using mps device for torch models
```

```python
class Conv1DNNModel(nn.Module):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        self.conv1d_relu_stack = nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=128, kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm1d(128),
            nn.Conv1d(in_channels=128, out_channels=64, kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Conv1d(in_channels=64, out_channels=32, kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm1d(32),
            nn.Flatten(),
            nn.Linear(32 * 10, 64),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Linear(64, 2),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.conv1d_relu_stack(x)
```

[24] ✓ 0.0s Python

```
learning_rate = 1e-2
batch_size = 64
epochs = 20

conv1d_model = Conv1DNNModel()
loss_fn = nn.BCEWithLogitsLoss()
summary(conv1d_model, input_size=(batch_size, 1, 16))
```

[25] ✓ 0.0s Python

Outputs are collapsed ···

```
y_train_one_hot = pd.get_dummies(y_train.values, dtype=np.float32)
y_test_one_hot = pd.get_dummies(y_test.values, dtype=np.float32)

test_dataset = TensorDataset(
    torch.tensor(X_test.values.reshape((-1, 1, 16)), dtype=torch.float32),
    torch.tensor(y_test.values, dtype=torch.float32),
)
```

[26] ✓ 0.0s Python

```
stratified_cv_split = StratifiedShuffleSplit(n_splits=10, test_size=0.1, random_state=42)

conv1d_cv = TorchKFoldCrossValidation(
    model_class=Conv1DNNModel,
    loss_fn=loss_fn,
    learning_rate=learning_rate,
    batch_size=batch_size,
    epochs=epochs,
    cv=stratified_cv_split,
    device=device
)
# conv1d_cv.fit(X_train.values.reshape(-1,1,16), y_train_one_hot.values)

## Load models, it is faster than to train the model again
## If you want to train the model then comment out the below line and uncomment the above code line
conv1d_cv.load_models()
```

[27] ✓ 0.0s Python

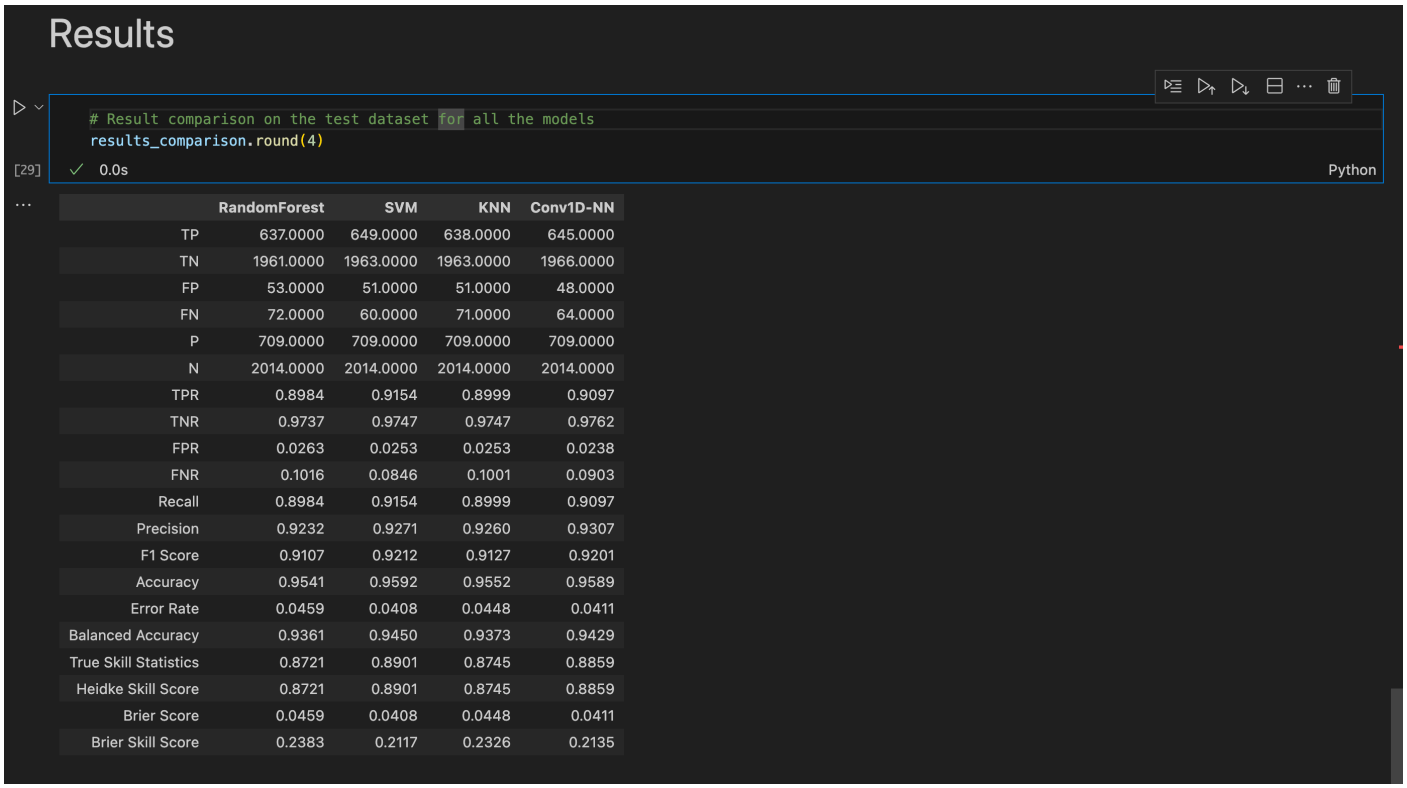
```
y_pred = conv1d_cv.predict(test_dataset.tensors[0]).argmax(axis=1)
matrix = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

results_comparison["Conv1D-NN"] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "Conv1D-NN"] = np.mean((y_pred - y_test) ** 2)
results_comparison.loc["Brier Skill Score", "Conv1D-NN"] = results_comparison["Conv1D-NN"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```

[28] ✓ 1.9s Python

Outputs are collapsed ···

Results



	RandomForest	SVM	KNN	Conv1D-NN
TP	637.0000	649.0000	638.0000	645.0000
TN	1961.0000	1963.0000	1963.0000	1966.0000
FP	53.0000	51.0000	51.0000	48.0000
FN	72.0000	60.0000	71.0000	64.0000
P	709.0000	709.0000	709.0000	709.0000
N	2014.0000	2014.0000	2014.0000	2014.0000
TPR	0.8984	0.9154	0.8999	0.9097
TNR	0.9737	0.9747	0.9747	0.9762
FPR	0.0263	0.0253	0.0253	0.0238
FNR	0.1016	0.0846	0.1001	0.0903
Recall	0.8984	0.9154	0.8999	0.9097
Precision	0.9232	0.9271	0.9260	0.9307
F1 Score	0.9107	0.9212	0.9127	0.9201
Accuracy	0.9541	0.9592	0.9552	0.9589
Error Rate	0.0459	0.0408	0.0448	0.0411
Balanced Accuracy	0.9361	0.9450	0.9373	0.9429
True Skill Statistics	0.8721	0.8901	0.8745	0.8859
Heidke Skill Score	0.8721	0.8901	0.8745	0.8859
Brier Score	0.0459	0.0408	0.0448	0.0411
Brier Skill Score	0.2383	0.2117	0.2326	0.2135

Conv-1D and SVM performed better overall than other algorithms. This is because neural networks has ability to determine and fit complex relationships among different attributes and label. SVM with RBF kernel function separates data into higher dimensions, that's why it was able to predict accurately.
KNN and Random Forest also did good job and were very close to the Conv-1D neural network and SVM, this is because the correlation between attributes and labels were proportional.

Link to Git Repository: https://github.com/amishfaldu-njit/cs-634104-final-term-project
Dataset can be found at: https://archive.ics.uci.edu/dataset/602/dry+bean+dataset