DPA PROJECT - CSP 571

NOTEBOOK AND REPORT

Sachin shivanna - A20552795

Nandini Devaraj - A20593534

Vikas Ravikumar Karjigi - A20581687

ABSTRACT

Using Scikit-Learn for model research and fit construction and ONNX for model deployment and edge inference, this project investigates building a strong machine learning pipeline. In order to obtain optimal performance, the research entails creating different classifiers, employing dimensionality reduction techniques such as PCA, and converting data from CSV to Parquet format for efficient processing. According on our research, the Random Forest Classifier has the best overall accuracy; however, there are still issues with resolving class imbalance. To address this problem and enhance model performance even more, future research will investigate cutting-edge methods like SMOTE and Balanced Random Forest.

Overview

Problem Statement

The primary goal of this project is to develop an efficient and effective machine learning pipeline for classification tasks that can handle large datasets and maintain robust performance during deployment.

Relevant Literature

Addressing imbalanced datasets in Random Forest (SFU CSPMP, 2024).

Evaluating Random Forest Models in classification tasks (Analytics Vidhya, 2024).

Proposed Methodology

step 1. **Data preprocessing:** Conversion of CSV files to Parquet format for faster data access, and handling missing values.

step 2. **Feature extraction:** Applying PCA to reduce data dimensionality.

step 3. **Model training:** Implementing and comparing multiple classifiers including Decision Tree and Random Forest.

step 4. **Model deployment:** Using ONNX for model conversion and runtime inference.

In [1]:
```
import numpy as np
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Data Processing

In order to improve data loading efficiency and drastically cut down on access time by about 20 times, the data pretreatment pipeline converts CSV files to Parquet format. Scaling the features to guarantee equitable contribution during model training comes next, after null values have been checked for and suitably imputed.

csv to parquet:

Convert the csv file to parquet file for further use to reduce access time.

Ref : https://robertochiosa.medium.com/a-python-guide-to-compress-csv-data-into-parquet-8c797136aa38

In [2]:
```
%%time
df_csv = pd.read_csv('data_public.csv')
table = pa.Table.from_pandas(df_csv)
```

```
CPU times: user 5.88 s, sys: 402 ms, total: 6.28 s
Wall time: 6.46 s
```

In [3]:
```
%%time
pq.write_table(table, 'data_parquet')

df = pd.read_parquet('data_parquet')

df.head()
```

```
CPU times: user 1.25 s, sys: 1.15 s, total: 2.41 s
Wall time: 2.59 s
```

Out[3]:

	A	B	C	D	E	F	G	H	I
0	231.420023	-12.210984	217.624839	-15.611916	140.047185	76.904999	131.591871	198.160805	82.873279
1	-38.019270	-14.195695	9.583547	22.293822	-25.578283	-18.373955	-0.094457	-33.711852	-8.356041
2	-39.197085	-20.418850	21.023083	19.790280	-25.902587	-19.189004	-2.953836	-25.299219	-6.612401
3	221.630408	-5.785352	216.725322	-9.900781	126.795177	85.122288	108.857593	197.640135	82.560019
4	228.558412	-12.447710	204.637218	-13.277704	138.930529	91.101870	115.598954	209.300011	89.961688

DATA ANALYSIS

In [4]:
```
#Check if there are any missing values
df.isna().sum()
#As there are no null values, there is not imputation required.
```

Out[4]:

	0
A	0
B	0
C	0
D	0
E	0
F	0
G	0
H	0
I	0
J	0
K	0
L	0
M	0
N	0
O	0
Class	0

dtype: int64

Visualisation & Feature Extraction

In [5]:
```
missing_val = df.isnull().sum()
missing_val = missing_val[missing_val != 0]
missing_val.index
```

Out[5]: `Index([], dtype='object')`

In [6]:
```
sns.heatmap(df.corr(), cmap="Oranges", xticklabels=True, yticklabels=True)
```

Out[6]: `<Axes: >`

The correlation map above shows that while the features are highly connected with the labels, they are also highly correlated with other features, including A, E, M, and many others.

To lessen the dimensionality of the data, a PCA is a smart idea.

Train / Test Split

In [7]:
```
from sklearn.model_selection import train_test_split
X = pd.DataFrame(df.iloc[:,0:len(df.columns)-2])
y = pd.DataFrame(data=df['Class'])

class_counts = y.value_counts()
class_percentages = (class_counts / len(y)) * 100
class_percentages
```

Out[7]:

Class	count
3	47.460083
2	37.490417
1	15.049500

dtype: float64

In [8]:
```
y.isnull().values
```

Out[8]:
```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    stratify=y,
                                                    random_state=42)
X.head()
```

Out[9]:

	A	B	C	D	E	F	G	H	I
0	231.420023	-12.210984	217.624839	-15.611916	140.047185	76.904999	131.591871	198.160805	82.873279
1	-38.019270	-14.195695	9.583547	22.293822	-25.578283	-18.373955	-0.094457	-33.711852	-8.356041
2	-39.197085	-20.418850	21.023083	19.790280	-25.902587	-19.189004	-2.953836	-25.299219	-6.612401
3	221.630408	-5.785352	216.725322	-9.900781	126.795177	85.122288	108.857593	197.640135	82.560019
4	228.558412	-12.447710	204.637218	-13.277704	138.930529	91.101870	115.598954	209.300011	89.961688

In [10]:
```
y.head()
```

Out[10]:

	Class
0	3
1	2
2	2
3	3
4	3

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

from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import SelectKBest

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_scaled)
df_train_scaled = pd.concat([X_train_scaled.reset_index(drop=True), y_train.reset_index(drop=True)]
df_train_scaled.tail()
```

Out[11]:

	0	1	2	3	4	5	6	7	8
839995	-0.699520	0.559578	-0.595247	0.712416	-0.814404	-0.260933	-0.503614	-0.717387	-0.968314
839996	-0.638317	0.041224	-0.555640	0.590153	-0.752334	-0.323515	-0.591538	-0.708540	-0.971112
839997	1.395852	0.414448	1.287152	0.055095	1.372297	1.306311	1.424369	1.351994	1.362479
839998	-0.616734	0.330633	-0.635060	0.598741	-0.722039	-0.254298	-0.420213	-0.739310	-0.944551
839999	-0.825564	-1.663080	-1.054183	-1.939207	-0.612724	-1.749766	-1.210277	-0.521088	0.510337

In [12]:
```
pca = PCA()
df_pca = pca.fit_transform(df_train_scaled.drop('Class',axis=1))
exp_percetange = pca.explained_variance_ratio_*100
for ratio in exp_percetange:
    print(f'{ratio:.2f}%')
```

```
72.73%
25.47%
0.64%
0.43%
0.37%
0.07%
0.06%
0.05%
0.05%
0.04%
0.03%
0.02%
0.02%
0.01%
```

Since the first two features, as we can see, account for about 95% of the variance, the model will be simpler if the dimensionality is reduced to 2.

In [13]:
```
df_pca = pd.DataFrame(df_pca)
df_pca.head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8
0	-2.002208	1.429344	0.006437	-0.008226	0.006927	-0.064878	-0.006448	-0.088163	-0.142396
1	-1.985210	1.363794	0.237184	-0.045310	0.058191	-0.054961	0.005127	0.029273	-0.012220
2	-2.039002	1.415636	-0.144471	-0.206608	-0.069038	-0.037181	-0.039752	-0.038070	-0.009791
3	-3.310290	-3.791050	0.345070	-0.515810	-0.133839	0.051192	0.039618	-0.029132	-0.081056
4	4.582659	-0.169875	-0.007113	0.146830	0.194771	0.092028	-0.190730	0.051793	-0.061020

In [14]:
```
selector = SelectKBest(k=2)
X_new = selector.fit_transform(df_pca, y_train)
X_new.shape
```

Out[14]: `(840000, 2)`

In [15]:
```
X_new = pd.DataFrame(X_new)
X_new.head()
```

Out[15]:

	0	1
0	-2.002208	1.429344
1	-1.985210	1.363794
2	-2.039002	1.415636
3	-3.310290	-3.791050
4	4.582659	-0.169875

In [16]:
```
X_new[0]
```

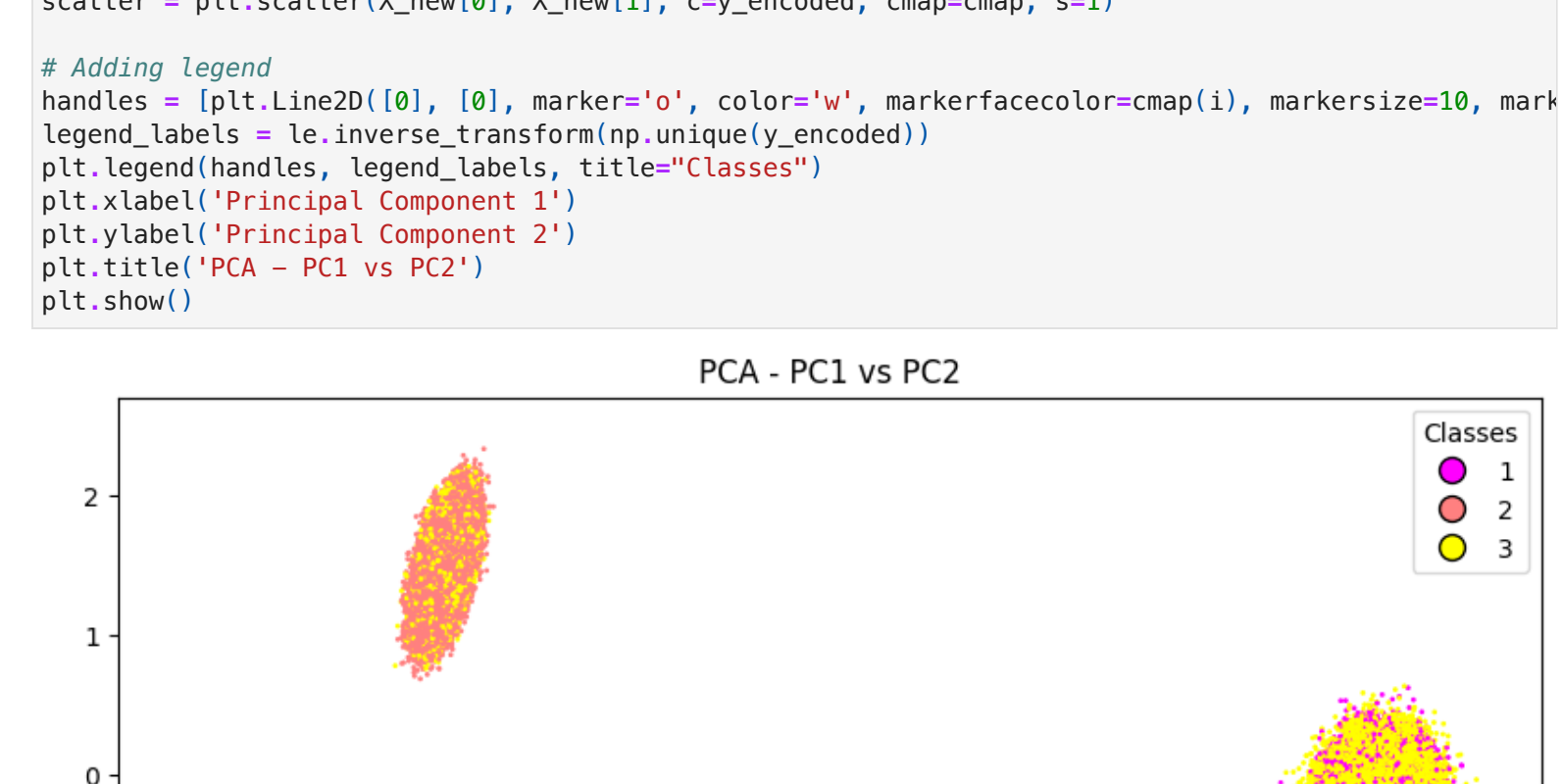
Out[16]:

	0
0	-2.002208
1	-1.985210
2	-2.039002
3	-3.310290
4	4.582659
...	...
839995	-1.930350
839996	-2.076061
839997	4.474101
839998	-1.877825
839999	-2.872944

840000 rows × 1 columns

dtype: float64

In [17]:
```
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

plt.figure(figsize=(10, 8))

le = LabelEncoder()
y_encoded = le.fit_transform(y_train)

# Create a color map from the encoded labels
cmap = plt.get_cmap('spring', np.unique(y_encoded).size)

scatter = plt.scatter(X_new[0], X_new[1], c=y_encoded, cmap=cmap, s=1)

# Adding legend
handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=cmap(i), markersize=10, mark
legend_labels = le.inverse_transform(np.unique(y_encoded))
plt.legend(handles, legend_labels, title="Classes")
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA - PC1 vs PC2')
plt.show()
```



Model Training

Feature Engineering

To minimize dimensionality, the features were scaled and processed using PCA. SelectPercentile was used to select the most important features, hence improving the model's performance.

Evaluation Metrics

To ensure a comprehensive assessment of the models' performance, accuracy, precision, recall, and F1-score measures were used, with a preference for overall accuracy.

Model / Pipeline Selection

The pipelines evolved as follows: As previously indicated, we begin by implementing scaling and PCA.

Then we compare various models to find which performs best.

Pipeline 1 - Decision Tree Classifier

Pipeline 2 – Random Forest Classifier

Pipeline 3: Random Forest Classifier with n_estimators=20

After testing the pipeline with SAMPLE data, using the Random Forest Classifier resulted in a 10% boost in accuracy, as well as a considerable increase in recall for each class. However, training time has grown by ten times.

The accuracy and f1 scores were then maintained by fine-tuning the Random Forest Classifier settings in Pipeline3, with only a threefold increase in training time over the Decision Tree Classifier.

The min_number in the leaf nodes was not increased, and as shown in the graph above, the classes are quite densely intermixed.

In [18]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

from sklearn.pipeline import Pipeline
```

In [19]:
```
#PIPE LINE 1
pipeline1 = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('selector', SelectPercentile(percentile=90)),
    ('classifier', DecisionTreeClassifier())
])
```

In [20]:
```
%%time
pipeline1.fit(X_train, y_train)
```

```
CPU times: user 2min 22s, sys: 487 ms, total: 2min 23s
Wall time: 2min 24s
```

Out[20]:
```
Pipeline
  StandardScaler
  PCA
  SelectPercentile
  DecisionTreeClassifier
```

In [21]:
```
print(classification_report(pipeline1.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

           1       0.38      0.36      0.37     56708
           2       0.73      0.75      0.74    131583
           3       0.60      0.59      0.59    171709

    accuracy                           0.61    360000
   macro avg       0.57      0.57      0.57    360000
weighted avg       0.61      0.61      0.61    360000
```

In [22]:
```
#PIPELINE 2

pipeline2 = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('selector', SelectPercentile(percentile=90)),
    ('classifier', RandomForestClassifier())
])
```

In [23]:
```
%%time
pipeline2.fit(X_train, y_train)
print(classification_report(pipeline2.predict(X_test), y_test))
```

```
              precision    recall  f1-score   support

           1       0.30      0.50      0.38     32865
           2       1.00      0.75      0.86    179793
           3       0.64      0.74      0.69    147342

    accuracy                           0.73    360000
   macro avg       0.65      0.67      0.64    360000
weighted avg       0.79      0.73      0.74    360000

CPU times: user 33min 4s, sys: 3.56 s, total: 33min 7s
Wall time: 33min 32s
```

In [24]:
```
#PIPELINE 3
pipeline3 = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('selector', SelectPercentile(percentile=90)),
    ('classifier', RandomForestClassifier(n_estimators=20))
])
```

In [25]:
```
%%time
pipeline3.fit(X_train, y_train)
```

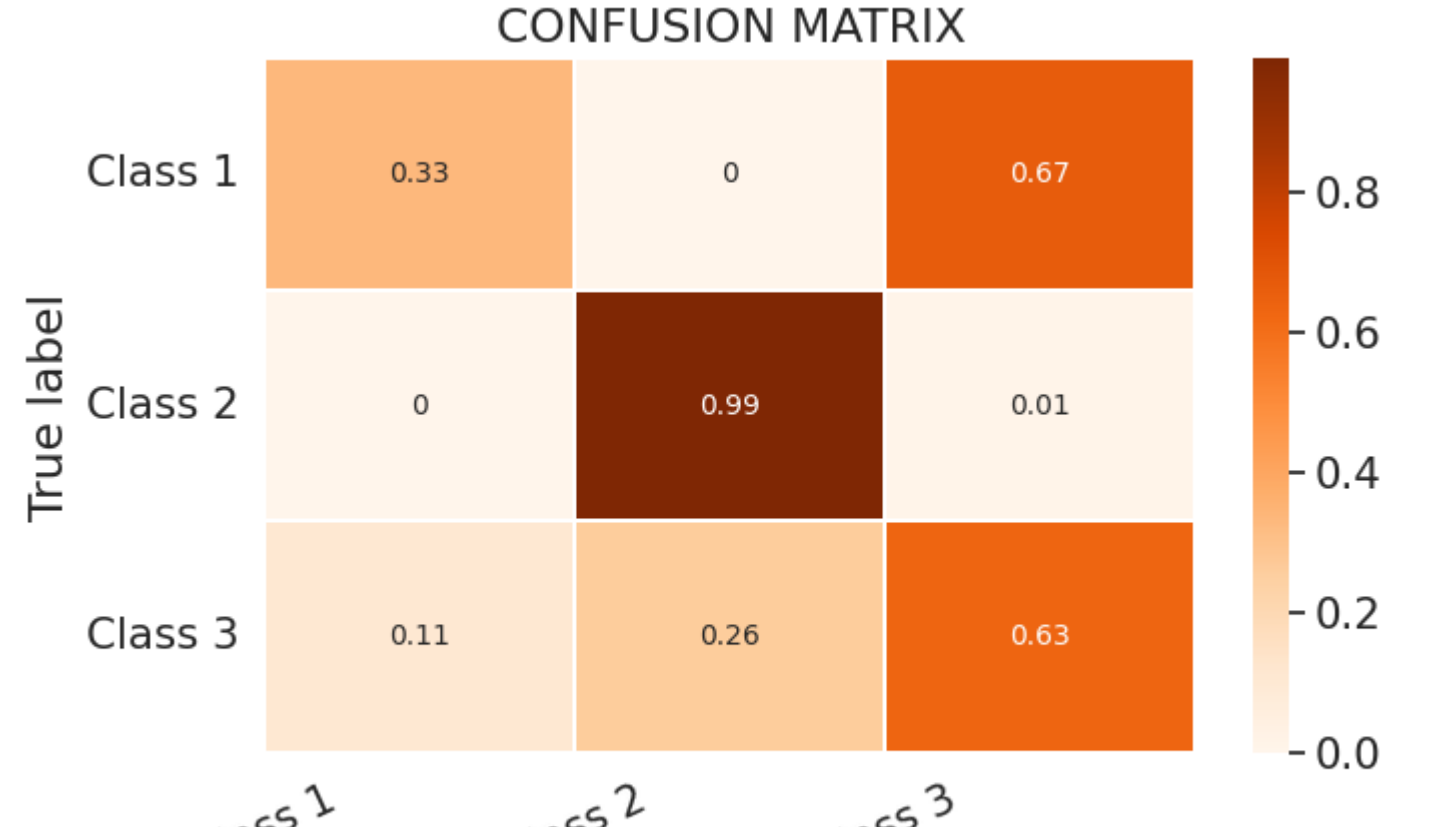
```
CPU times: user 6min 29s, sys: 892 ms, total: 6min 30s
Wall time: 6min 33s
```

Out[25]:
```
Pipeline
  StandardScaler
  PCA
  SelectPercentile
  RandomForestClassifier
```

MODEL VALIDATION

In [26]:
```
y_pred = pipeline3.predict(X_test)
```

In [27]:
```
#CONFUSION MATRIX

from sklearn.metrics import confusion_matrix
import seaborn as sns

# Get and reshape confusion matrix data
matrix = confusion_matrix(y_test, y_pred)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(7.5,4.5))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Oranges, linewidths=0.2)

# Add labels to the plot
class_names = ['Class 1', 'Class 2', 'Class 3']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('CONFUSION MATRIX')
plt.show()
```



CLASSIFICATION REPORT

In [28]:
```
print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

           1       0.33      0.49      0.40     36669
           2       0.99      0.75      0.85    178001
           3       0.63      0.74      0.68    145330

    accuracy                           0.72    360000
   macro avg       0.65      0.66      0.64    360000
weighted avg       0.78      0.72      0.74    360000
```

Testing Results:

The chosen Random Forest model was validated using a stratified train/test split to ensure balanced class representation. The final model had an overall accuracy of 72%, with Class 2 having the best accuracy and Class 1 having the lowest, indicating class imbalance problems.

Performance Criteria:

The model's performance was evaluated using a confusion matrix and a classification report, which highlighted precision, recall, and F1-score for each class. The confusion matrix demonstrated that Class 1 is frequently misclassified as Class 3, highlighting the need for improved handling of imbalanced data.

Biases/Risks:

Class imbalance was recognized as a serious danger, especially for the minority class's (Class 1) accuracy and F1-score. This mismatch can result in biased models that underperform for minority groups.

In [29]:
```
# Creating the ONNX File
!pip install skl2onnx

from skl2onnx.common.data_types import FloatTensorType
from skl2onnx import convert_sklearn
```

```
Collecting skl2onnx
  Downloading skl2onnx-1.17.0-py2.py3-none-any.whl.metadata (3.2 kB)
Collecting onnx>=1.2.1 (from skl2onnx)
  Downloading onnx-1.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (16 k
B)
Requirement already satisfied: scikit-learn>=1.1 in /usr/local/lib/python3.10/dist-packages (from s
kl2onnx) (1.5.2)
Collecting onnxconverter-common>=1.7.0 (from skl2onnx)
  Downloading onnxconverter_common-1.14.0-py2.py3-none-any.whl.metadata (4.2 kB)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from onnx>=
1.2.1->skl2onnx) (1.26.4)
Requirement already satisfied: protobuf>=3.20.2 in /usr/local/lib/python3.10/dist-packages (from on
nx>=1.2.1->skl2onnx) (4.25.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from onnxconve
rter-common>=1.7.0->skl2onnx) (24.2)
Collecting protobuf>=3.20.2 (from onnx>=1.2.1->skl2onnx)
  Downloading protobuf-3.20.2-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl.metadata
(679 bytes)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit
-learn>=1.1->skl2onnx) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from sciki
t-learn>=1.1->skl2onnx) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (fro
m scikit-learn>=1.1->skl2onnx) (3.5.0)
Downloading skl2onnx-1.17.0-py2.py3-none-any.whl (298 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 298.4/298.4 kB 11.2 MB/s eta 0:00:00
Downloading onnx-1.17.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.0/16.0 MB 35.5 MB/s eta 0:00:00
Downloading onnxconverter_common-1.14.0-py2.py3-none-any.whl (84 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.5/84.5 kB 5.3 MB/s eta 0:00:00
Downloading protobuf-3.20.2-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 32.7 MB/s eta 0:00:00
Installing collected packages: protobuf, onnx, onnxconverter-common, skl2onnx
  Attempting uninstall: protobuf
    Found existing installation: protobuf 4.25.5
    Uninstalling protobuf-4.25.5:
      Successfully uninstalled protobuf-4.25.5
ERROR: pip's dependency resolver does not currently take into account all the packages that are ins
talled. This behaviour is the source of the following dependency conflicts.
grpcio-status 1.62.3 requires protobuf>=4.21.6, but you have protobuf 3.20.2 which is incompatible.
tensorflow 2.17.1 requires protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,
>=3.20.3, but you have protobuf 3.20.2 which is incompatible.
tensorflow-metadata 1.13.1 requires protobuf<5,>=3.20.3, but you have protobuf 3.20.2 which is inco
mpatible.
Successfully installed onnx-1.17.0 onnxconverter-common-1.14.0 protobuf-3.20.2 skl2onnx-1.17.0
```

In [30]:
```
%%time
input_types = dict([(x, FloatTensorType([None, 1])) for x in X_train.columns.values])

try:
    model_onnx = convert_sklearn(pipeline3,
                                 'pipeline_onnx',
                                initial_types=list(input_types.items()))
except Exception as e:
    print(e)

with open("pipeline.onnx", "wb") as f:
    f.write(model_onnx.SerializeToString())
```

```
CPU times: user 2min, sys: 13.5 s, total: 2min 14s
Wall time: 2min 39s
```

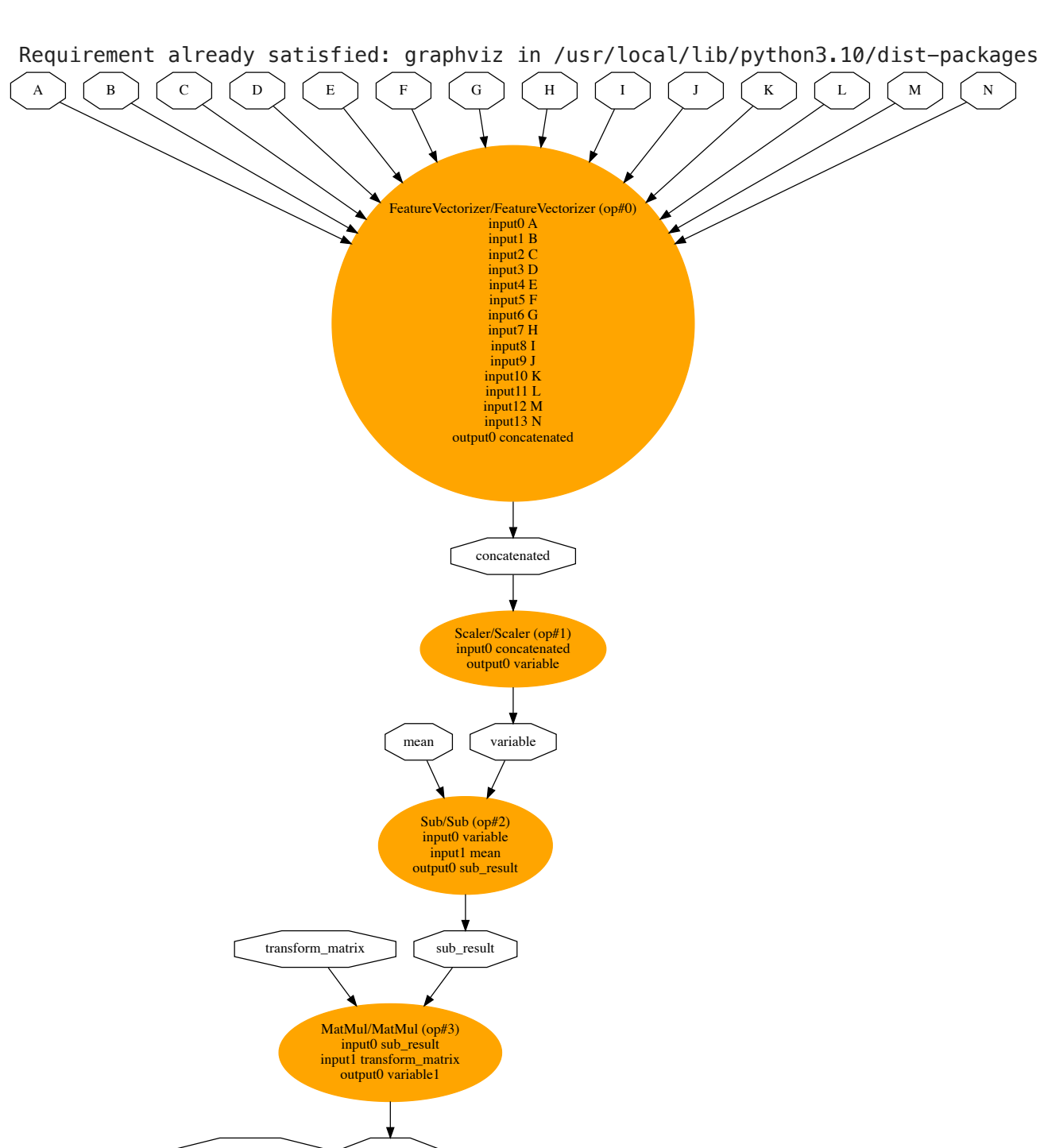
In [31]:
```
!pip install onnxruntime
import onnxruntime as rt

inputs_onnx = {k: np.array(v).astype(np.float32)[:, np.newaxis] for k, v in X_test.to_dict(orient='

session_onnx = rt.InferenceSession("pipeline.onnx")
predict_onnx = session_onnx.run(None, inputs_onnx)

print(classification_report(predict_onnx[0], y_test))
```

```
Collecting onnxruntime
  Downloading onnxruntime-1.20.1-cp310-cp310-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metada
ta (4.5 kB)
Collecting coloredlogs (from onnxruntime)
  Downloading coloredlogs-15.0.1-py2.py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: flatbuffers in /usr/local/lib/python3.10/dist-packages (from onnxrun
time) (24.3.25)
Requirement already satisfied: numpy>=1.21.6 in /usr/local/lib/python3.10/dist-packages (from onnxr
untime) (1.26.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from onnxrunti
me) (24.2)
Requirement already satisfied: protobuf in /usr/local/lib/python3.10/dist-packages (from onnxruntim
e) (3.20.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from onnxruntime)
(1.13.1)
Collecting humanfriendly>=9.1 (from coloredlogs->onnxruntime)
  Downloading humanfriendly-10.0-py2.py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from
sympy->onnxruntime) (1.3.0)
Downloading onnxruntime-1.20.1-cp310-cp310-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (13.3 M
B)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 26.7 MB/s eta 0:00:00
Downloading coloredlogs-15.0.1-py2.py3-none-any.whl (46 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.0/46.0 kB 3.0 MB/s eta 0:00:00
Downloading humanfriendly-10.0-py2.py3-none-any.whl (86 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.8/86.8 kB 3.1 MB/s eta 0:00:00
Installing collected packages: humanfriendly, coloredlogs, onnxruntime
Successfully installed coloredlogs-15.0.1 humanfriendly-10.0 onnxruntime-1.20.1
              precision    recall  f1-score   support

           1       0.33      0.49      0.40     36669
           2       0.99      0.75      0.85    178001
           3       0.63      0.74      0.68    145330

    accuracy                           0.72    360000
   macro avg       0.65      0.66      0.64    360000
weighted avg       0.78      0.72      0.74    360000
```

In [32]:
```
from onnx.tools.net_drawer import GetPydotGraph, GetOpNodeProducer
from onnx.tools.net_drawer import GetPydotGraph, GetOpNodeProducer
from IPython.display import display
```

In [34]:
```
!pip install graphviz
import graphviz
pydot_graph = GetPydotGraph(model_onnx.graph,
                            name=model_onnx.graph.name,
                            rankdir="TB",
                            node_producer=GetOpNodeProducer("docstring",
                                                            color="orange",
                                                            fillcolor="orange",
                                                            style="filled"))

# Convert the Pydot graph to a string and modify the size attribute
dot_string = pydot_graph.to_string()
dot_string = dot_string.replace('rankdir=TB;', 'rankdir=TB; size="30,60"; dpi="50";')

# Create a Graphviz Source object from the modified string
source = graphviz.Source(dot_string)

# Display the graph directly in the Jupyter notebook cell
display(source)
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (0.20.3)
```



CONCLUSION

Results:

The adjusted Standard Random Forest was selected as the final model based to its superior overall accuracy.

According to the confusion matrix, **Class 2** has the best accuracy, while **Class 1** has the lowest and is incorrectly labeled as Class 3.

It reflects the scatterplot seen above. Class 1 is **intermixed** with Class 3.

As previously stated, **class imbalance** contributed to bias, and Class1, with the smallest size, has the lowest accuracy and F1 score.

Recommendations

To tackled the class imblance issue, techniques such as Balanced Random Forest(BRF), SMOTE (Synthetic Minority Oversampling Technique) using Standard Random Forest, can be explored. These techniques can enhance the performance of the minority class without compromising the model's overall accuracy, as discussed in the article *"Surviving in a Random Forest with Imbalanced Datasets" (SFU CSPMP, 2024)*.

Caution

While the methods mentioned above may boost the F1 score of the minority class, it is important to also consider maintaining overall accuracy and minimizing misclassification of the majority class.

DATA SOURCES

This data is provdied by the professor for the imPlementation of the project CSP571.

SOURCE CODE

Listings

1. Jupyter Notebook *(.ipynb & .pdf)*
2. Dataset *(.csv(as a zip) & parquet)*
3. ONNX File *(pipeline.onnx)*

To use the trained model in runtime 3. ONNX Model Diagram *(.pdf)*
The graph for the ONNX Model is attached as seperate pdf file, due to issues in the diagram appropriately in the notebook pdf. The model's architecture and conversion process are detailed, ensuring reproducibility and ease of deployment. 4. Requirements *(.txt)*

Dependencies:

The project relies on several open-source libraries, primarily:

- Python 3.10+
- Pandas for data manipulation.
- ONNXRuntime 1.18+ for inference, to run the IR 10 model.
- Seaborn and Matplotlib for data visualization.
- Scikit-Learn 1.2+ for building and evaluating models.
- Graphviz for visualizing model architecture.
- ONNX 1.16+ for model conversion, which creates a model in the ONNX IR version of 10.
- NumPy for numerical computations.
- PyArrow for efficient data storage in Parquet format.

Installation Instructions

1. Install the required dependencies.
2. Install the graphviz also into the system using the link provided in the references.
3. Open the Jupyter Notebook and run the cells sequentially.

Reproducibility

The project is developed with reproducibility in mind, offering detailed instructions and all essential files to allow others to replicate and expand upon the work.

10. Bibliography

(Chicago style - AMS/AIP)

- ChatGPT. “Assistance with Project Report on the Abstract, Overview & Bibliography sections.” OpenAI, accessed November 15, 2024.

https://chatgpt.com/?model=gpt-4o.

- Stack Overflow. “Pandas Concat Increases Number of Rows.” Accessed November 19, 2024.

https://stackoverflow.com/questions/50368145/pandas-concat-increases-number-of-rows.

- Analytics Vidhya. “Evaluating a Random Forest Model.” Medium, accessed November 19, 2024.

https://medium.com/analytics-vidhya/evaluating-a-random-forest-model-9d165595ad56.

- ONNX Runtime. “Compatibility.” Accessed November 19, 2024.

https://onnxruntime.ai/docs/reference/compatibility.html.

- Graphviz. “Graphviz.” Accessed November 19, 2024.

https://pypi.org/project/graphviz/.

- SFU CSPMP. “Surviving in a Random Forest with Imbalanced Datasets.” Medium, accessed November 19, 2024.

https://medium.com/sfu-cspmp/surviving-in-a-random-forest-with-imbalanced-datasets-b98b963d52eb.