Group 6:

Using ML-generated synthetic data to boost performance of security classifiers

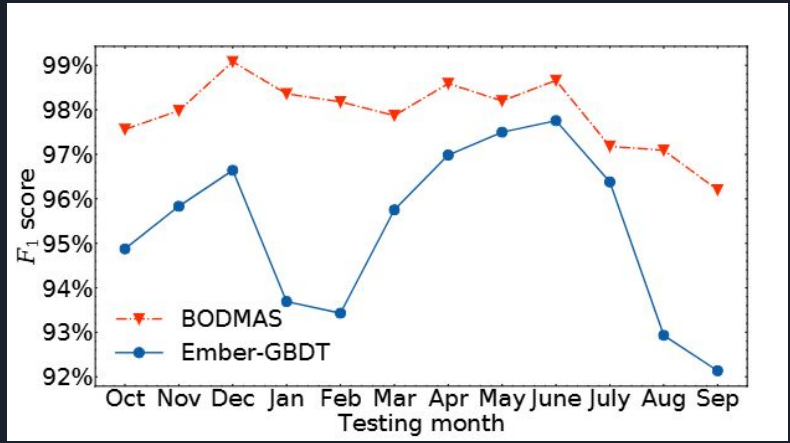
Samuel Fisher, Karan Reddy Kandala, Harrison Bickerstaff

Introduction

- Using BODMAS and TabDDPM to generate synthetic data for malware classifiers
- Focusing on n = 40 families to address drop in performance from May-July
- Why this is a big deal
  - Privacy concerns
  - Lack of updated malware
  - Constantly evolving programs

BODMAS Review

- Open malware data set that poses many advantages to other currently available data sets
- Still shows signs of “concept drift”
- GBDT classifier experiences large drop in performance when family size = 40
- Most likely due to an increase in testing an underrepresented families



Dataset	Malware Time	Family	# Families	# Samples	# Benign	# Malware	Malware Binaries	Feature Vectors
Microsoft	N/A (Before 2015)	●	9	10,868	0	10,868	◑	○
Ember	01/2017–12/2018	◑	N/A	2,050,000	750,000	800,000	○	●
UCSB-Packed	01/2017*–03/2018	○	N/A	341,445	109,030	232,415	●	○
SOREL-20M	01/2017–04/2019	○	N/A	19,724,997	9,762,177	9,962,820	●	●
BODMAS (Our)	**08/2019–09/2020**	●	**581**	134,435	77,142	57,293	●	●

TabDDPM In Depth

Forward Diffusion: Start with real tabular data and gradually add small amounts of noise at each step until the data becomes almost random.

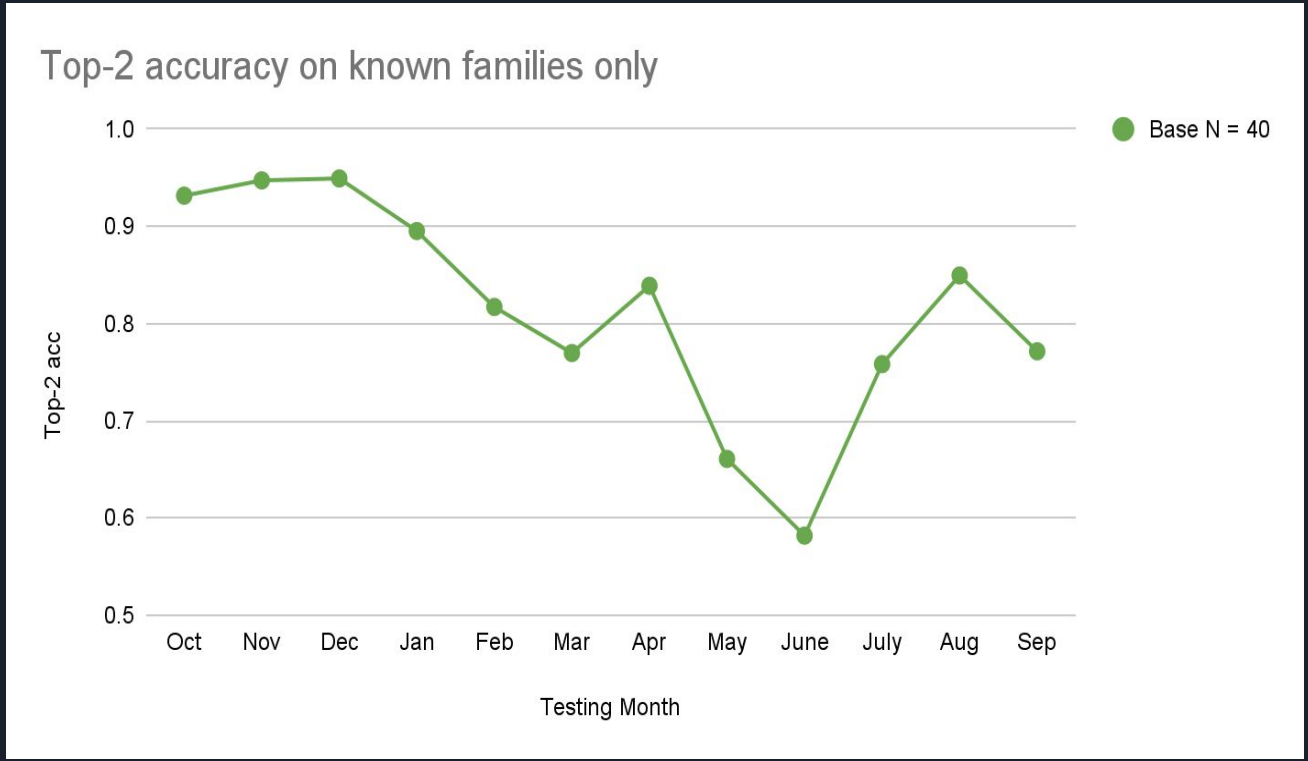
Reverse Denoising: Train a neural network to learn how to remove the noise step by step, reconstructing clean data from partially corrupted inputs.

Sampling: Once trained, generate new synthetic rows by starting from random noise and iteratively applying the learned denoising process in reverse.

Reconstruction: Convert the denoised outputs back into numerical and categorical columns, yielding realistic synthetic tabular data.

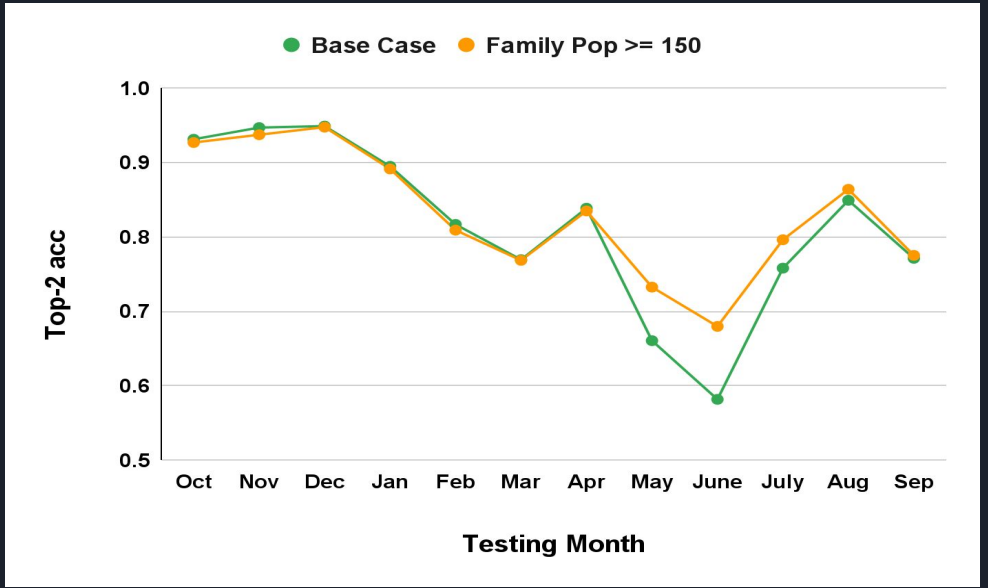
Evaluation: TabDDPM evaluates the generated synthetic data by F1 score.

Previous Work



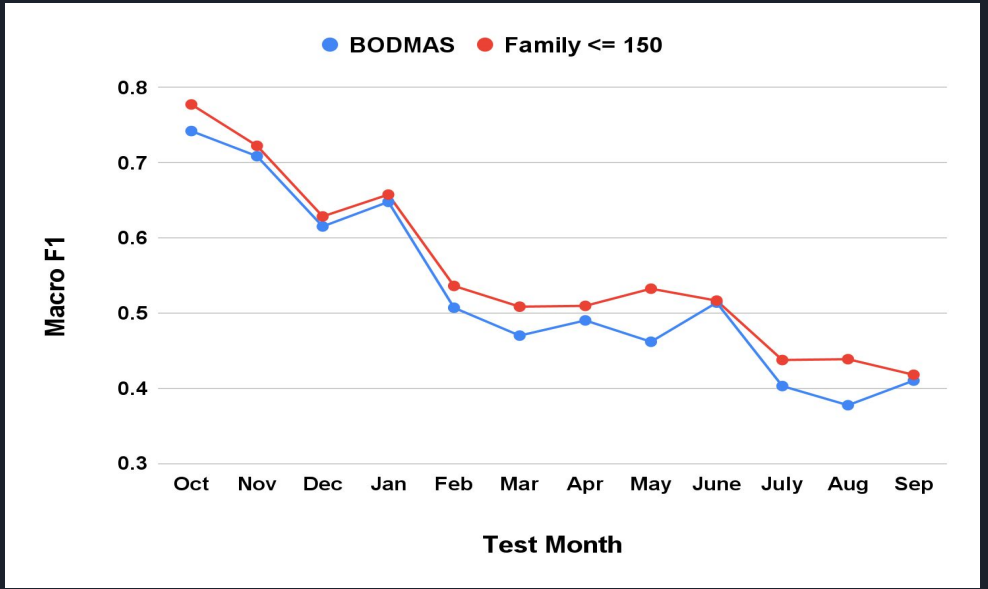
Previous Work

- Replicated Bodmas findings
- Tuned TabDDPM for most accurate synthetic data generation
- Found measurable improvements in top 2 - accuracy with the addition of synthetic samples for May-July



Previous Work

- Replicated Bodmas findings
- Tuned TabDDPM for most accurate synthetic data generation
- Found measurable improvements in top 2 - accuracy with the addition of synthetic samples for May-July
- Found measurable improvements in Macro F1 score compared to Bodmas.



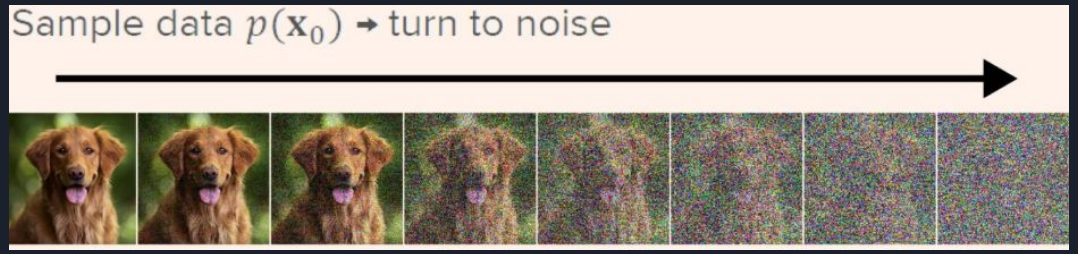
Addressing Feedback

- More balanced data augmentation techniques
- Demonstration of how TabDDPM and Data Generation works
- Replacing Top-2 accuracy with Macro F1 Score

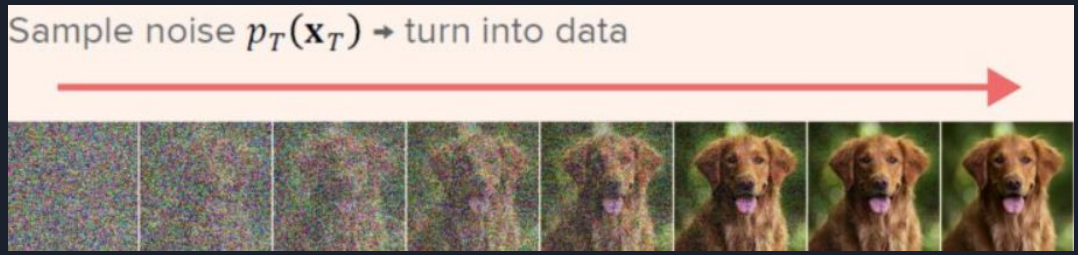
Tab DDPM Visualized

- Forward Diffusion adds noise to a sample
- Reverse Diffusion removes noise from a sample
- TabDDPM tends to compress values to the mean when not confident

Forward Diffusion

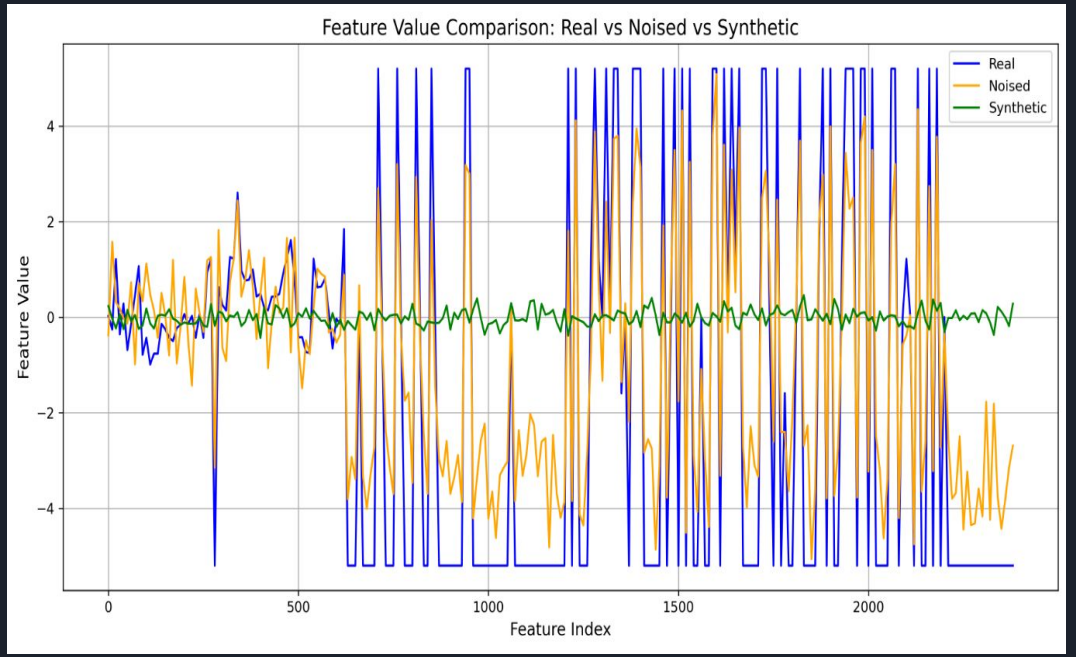


Reverse Diffusion

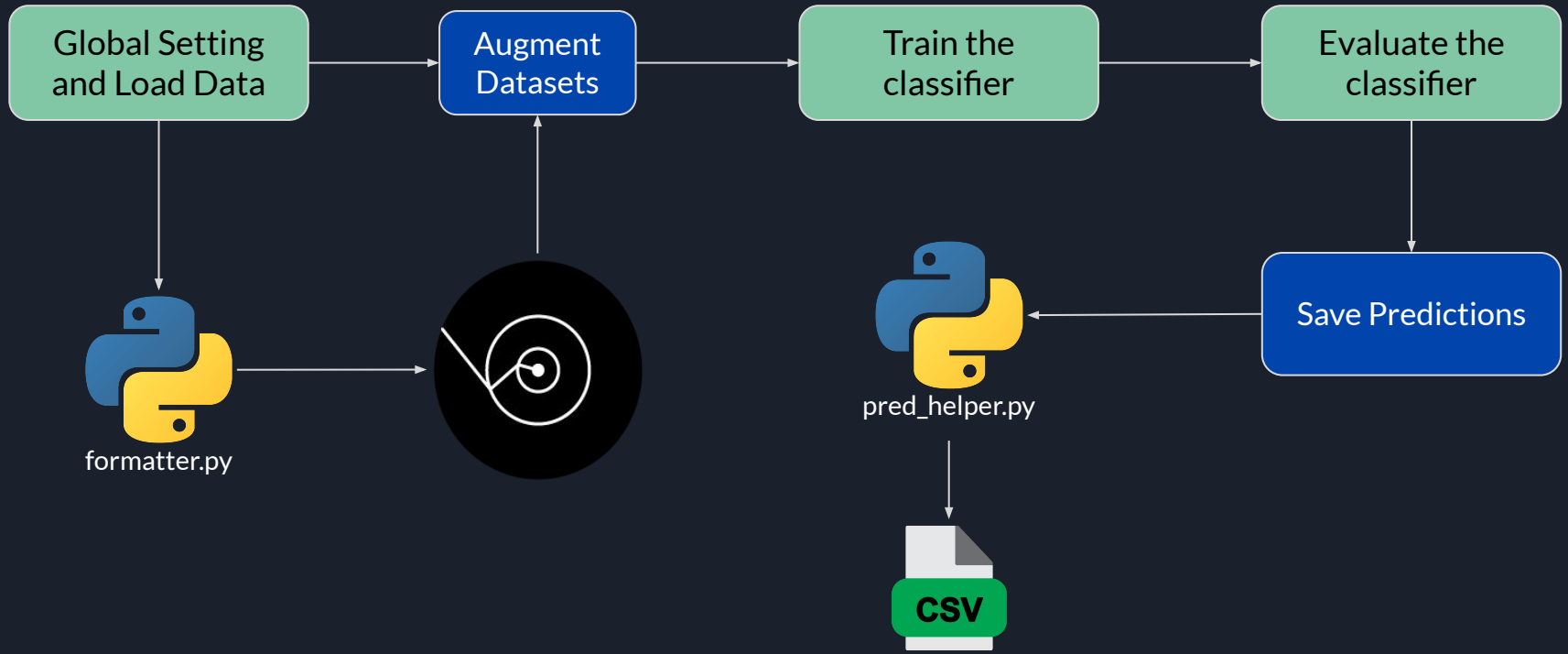


Tab DDPM Visualized

- Forward Diffusion adds noise to a sample
- Reverse Diffusion removes noise from a sample
- TabDDPM tends to compress values to the mean when not confident



Core Methodology

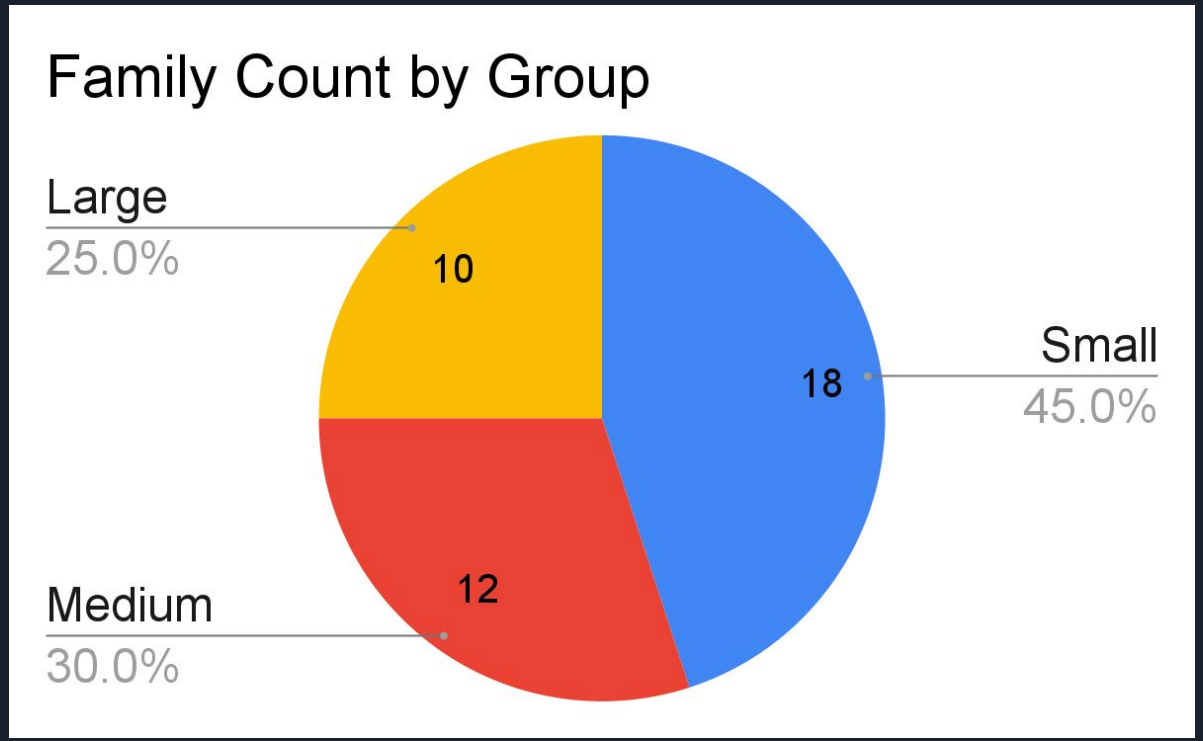


Data Generation Technique

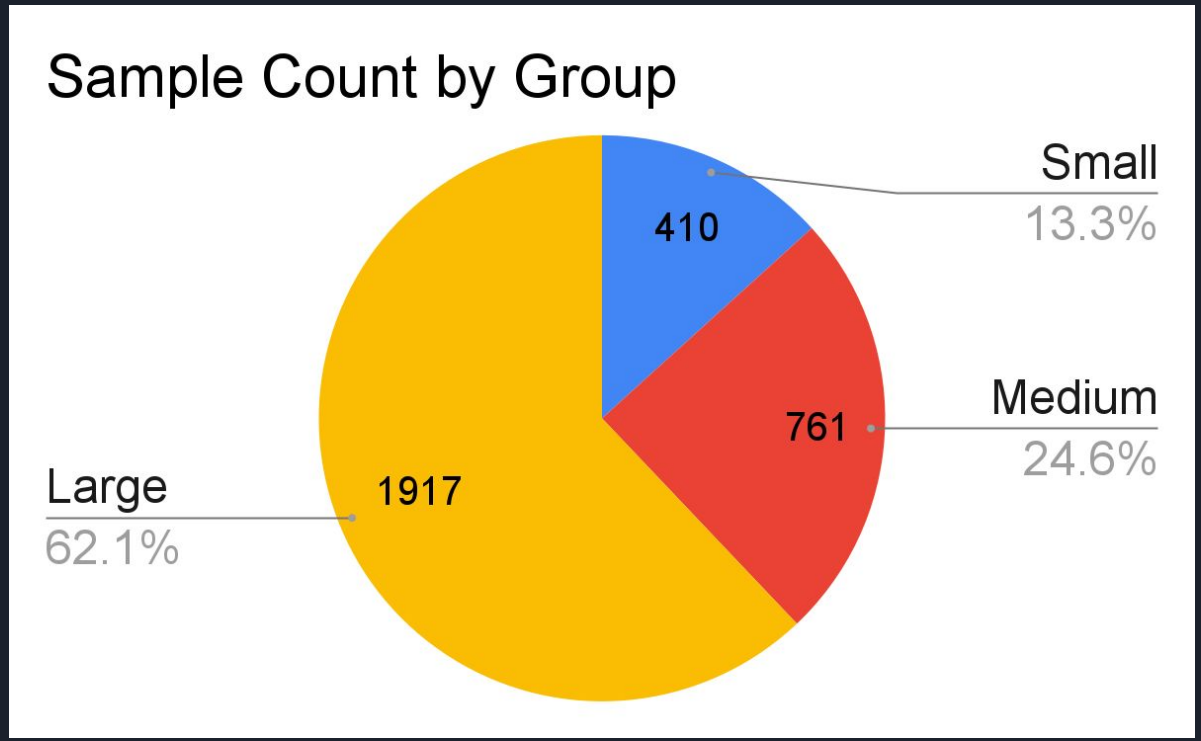
```
# Settings
breaks = [40, 100] # Small < [0] <= Medium < [1] <= Large
sample_methods = {"s": True, "m": True, "l": False} # Determines which sampling method to use
```

```
# Get list of labels to keep
family_counts = df['y_label'].value_counts()

s_valid = family_counts[family_counts < breaks[0]].index
sdf = df[df['y_label'].isin(s_valid)]

m_valid = family_counts[(family_counts >= breaks[0]) & (family_counts < breaks[1])].index
mdf = df[df['y_label'].isin(m_valid)]

l_valid = family_counts[family_counts >= breaks[1]].index
ldf = df[df['y_label'].isin(l_valid)]
```

Data Generation Technique



Data Generation Technique



Data Generation Technique

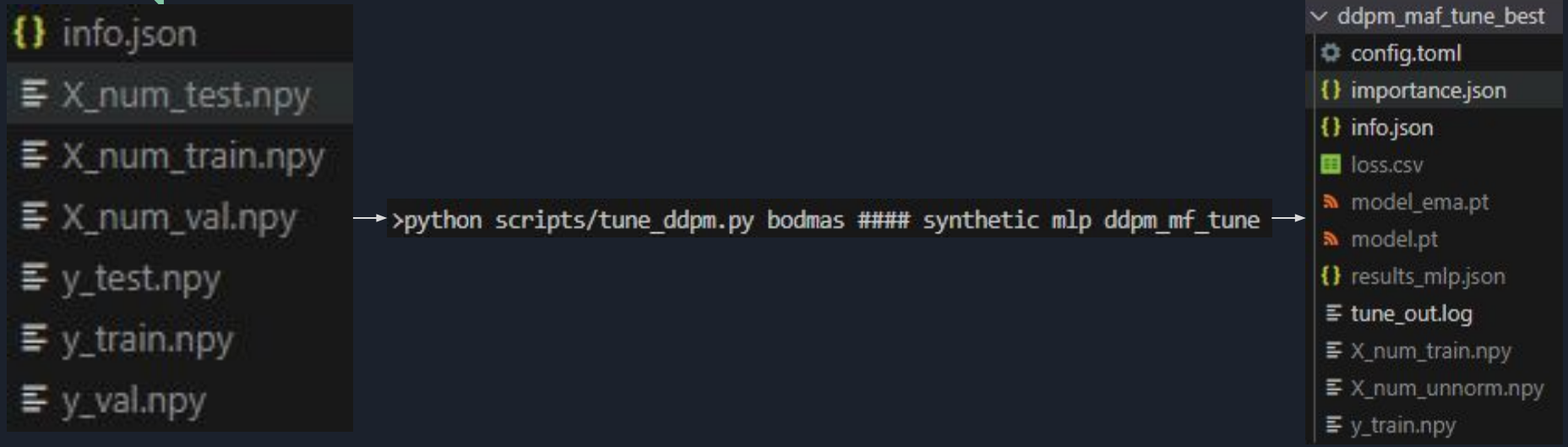
```python
def make_maps(df, size_set):
    # Remapping y_label to be subsequent
    uniques = df["y_label"].unique()

    # Create mappings in both directions
    orig_to_temp = pd.Series(data=range(len(uniques)), index=uniques)
    pd.Series(index=range(len(uniques)), data=uniques).to_json(f'data/bodmas/map_{size_set}.json')

    return orig_to_temp
```

```python
# Apply the maps to the dataframes
sdf["y_label"] = sdf["y_label"].map(s_map)
mdf["y_label"] = mdf["y_label"].map(m_map)
ldf["y_label"] = ldf["y_label"].map(l_map)
```

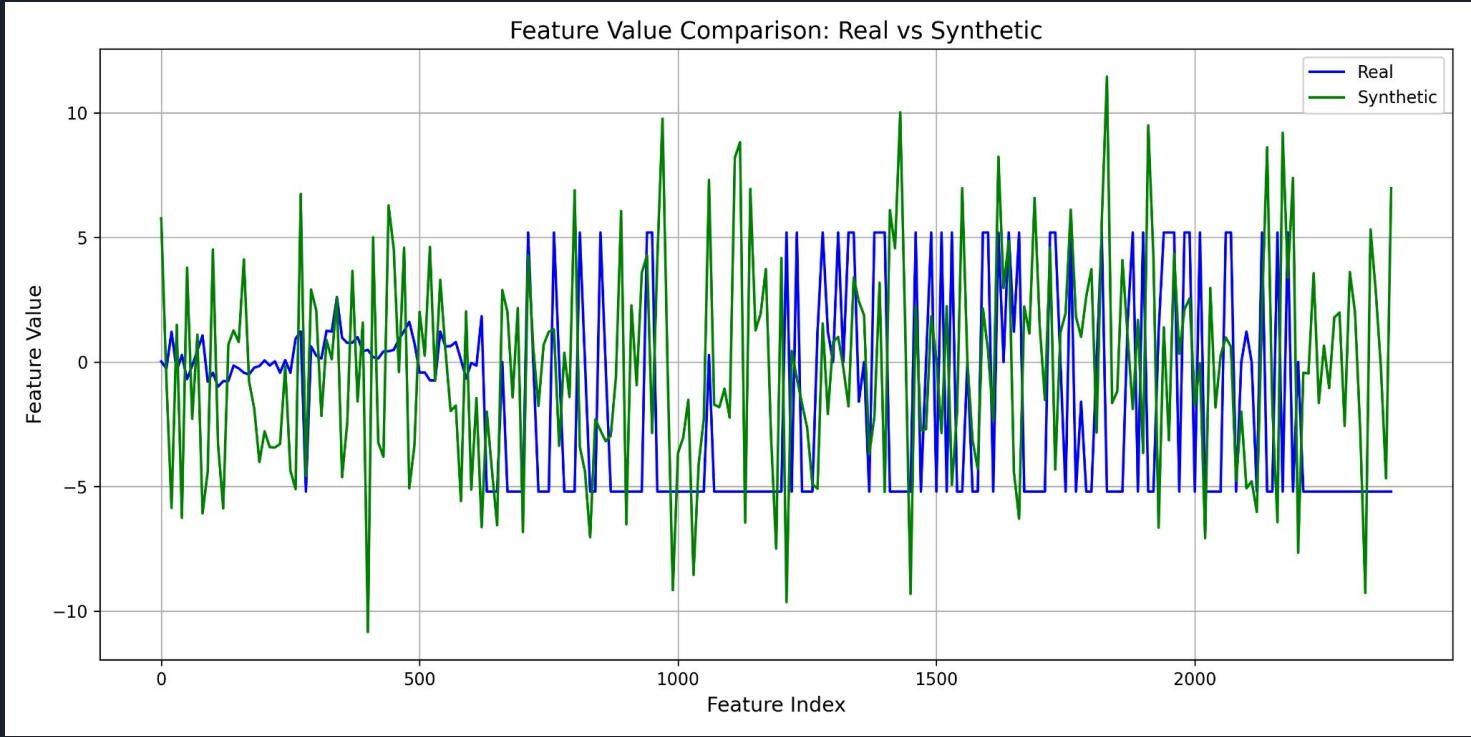
Data Generation Technique

```python
# Double - Intended for smaller families
def double_sample(df):
    # Split into train/val/test splits, random_state allows reproducibility
    rs = 42

    # Create a training split
    train = df.groupby("y_label").sample(frac=.5, random_state=rs).index
    trainA = df.loc[train]
    trainB = df.drop(train)

    # Create a validation split
    valA = trainB.groupby("y_label").sample(frac=.5, random_state=rs)
    valB = trainA.groupby("y_label").sample(frac=.5, random_state=rs)

    # Create a testing split
    testA = trainB.drop(valA.index)
    testB = trainA.drop(valB.index)

    return trainA, valA, testA, trainB, valB, testB
```

Data Generation Technique



Data Generation Technique



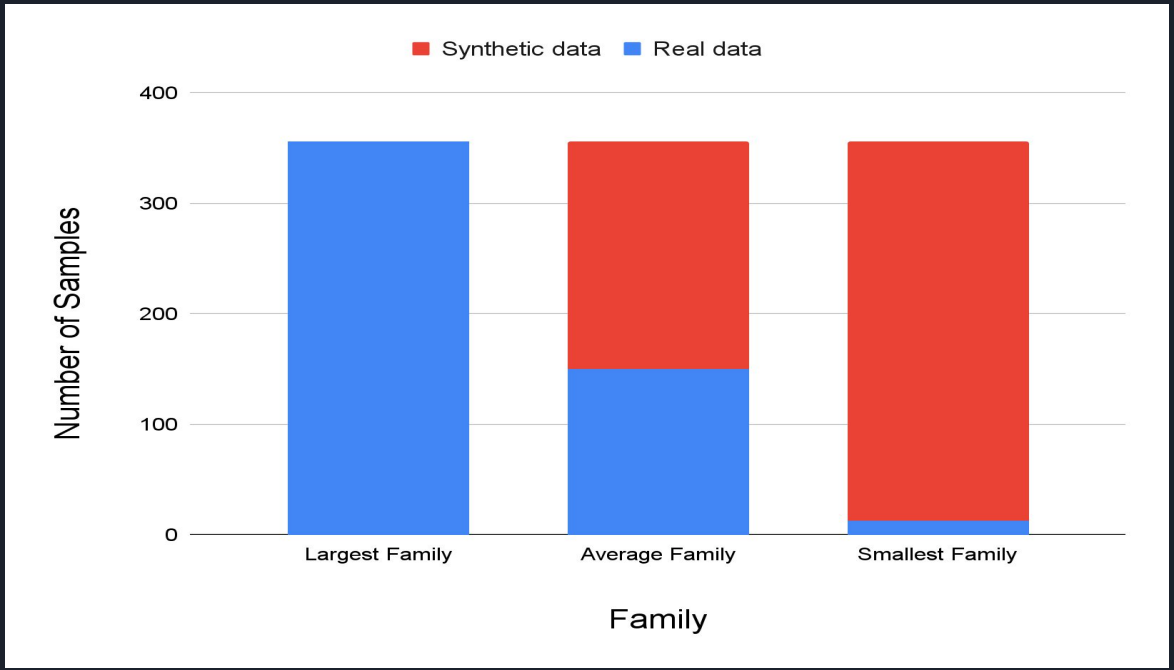
Data Augmentation Technique

- Our data augmentation technique uses two variables
  - Max Family Size (MFS):
    - The largest a family can be after augmentation
  - Synth Needed (SN)
    - The amount of synthetic data to add for a given family
    - If there is not enough synthetic data, then it uses all available data

```
max_family_size = real_counts.max()
synth_needed = max_family_size - count
df_arr.append(synth_sub_df.sample(n=min(synth_needed,len(synth_sub_df)), random_state=rs))
```

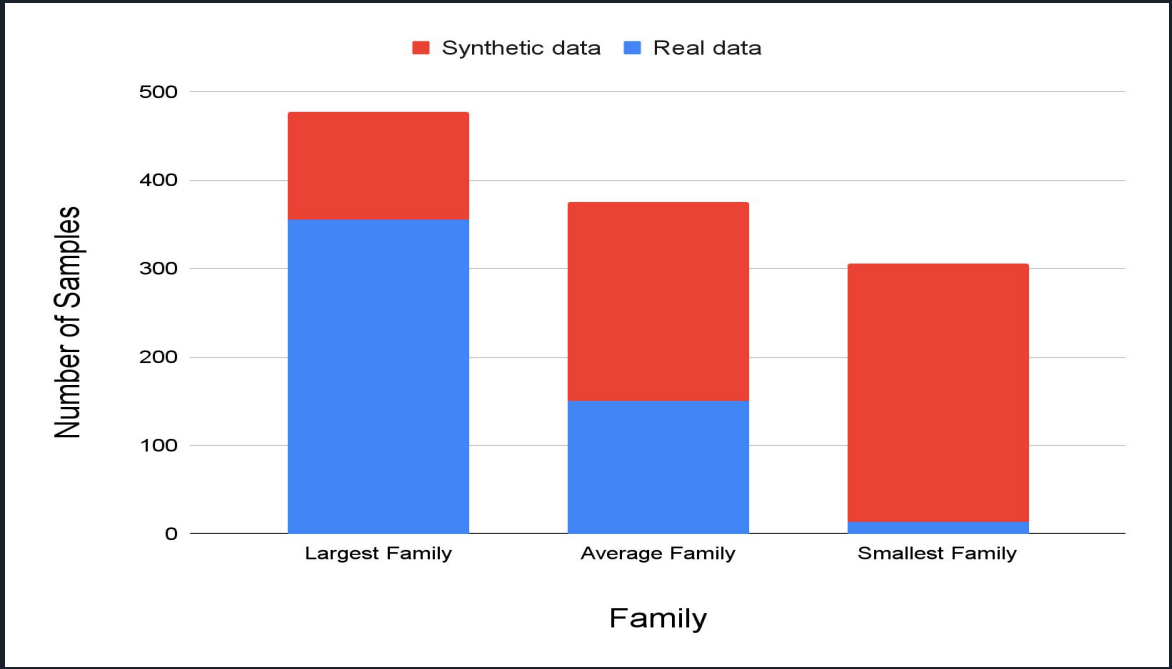
Data Augmentation Technique

Version 1: MFS = Largest Family Size; SN = MFS - Current Family Size



Data Augmentation Technique

Version 2: MFS = 600; SN = (MFS - Current Family Size) / 2



Data Augmentation Technique

Version	Max Family Size	Synth Needed
1	real_counts.max()	Max Family Size - Current Family Size
2	600	(Max Family Size - Current Family Size) / 2
3	real_counts.max()	(Max Family Size - Current Family Size) / 2
4	real_counts.max() * 2	Max Family Size - Current Family Size
5	real_counts.max() * 1.5	Max Family Size - Current Family Size

Additionally ran V1 with a upscale of 12, 20, 25, 50, and 100

Evaluation metric

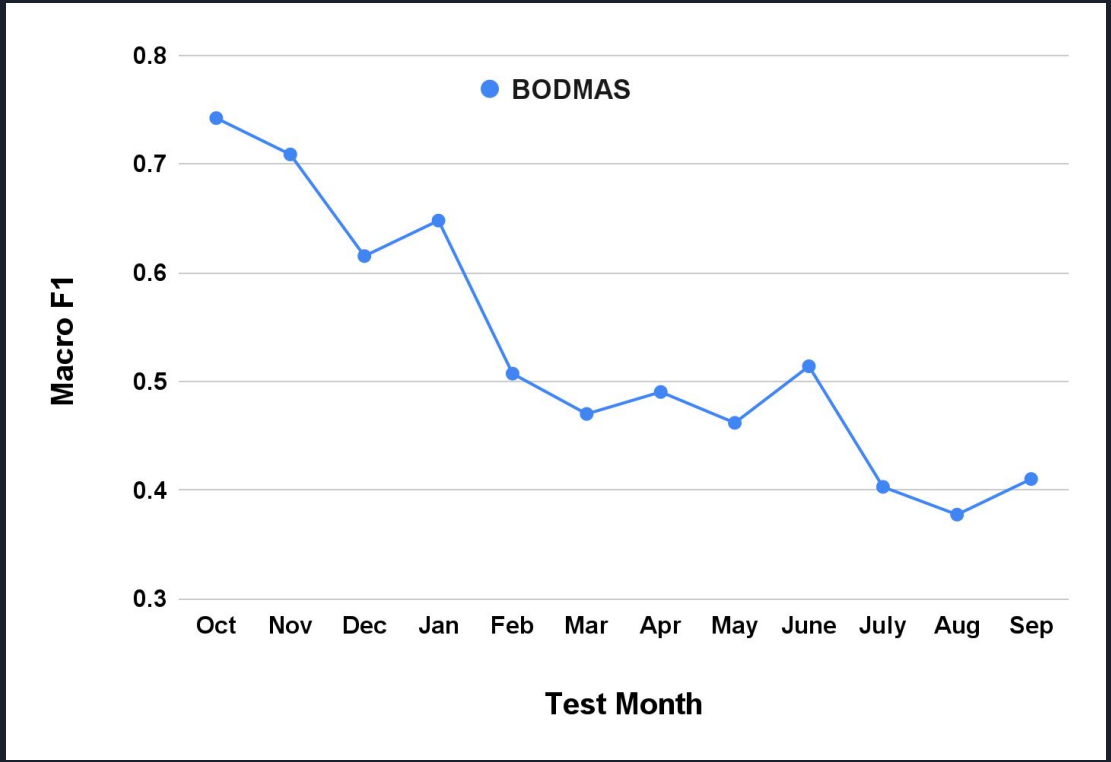
What is Macro F1- Score?

- Computes the F1 Score for each class independently, then calculates the unweighted mean.
  - Balance of precision and Recall
  - Fair performance evaluation for multiclass classification

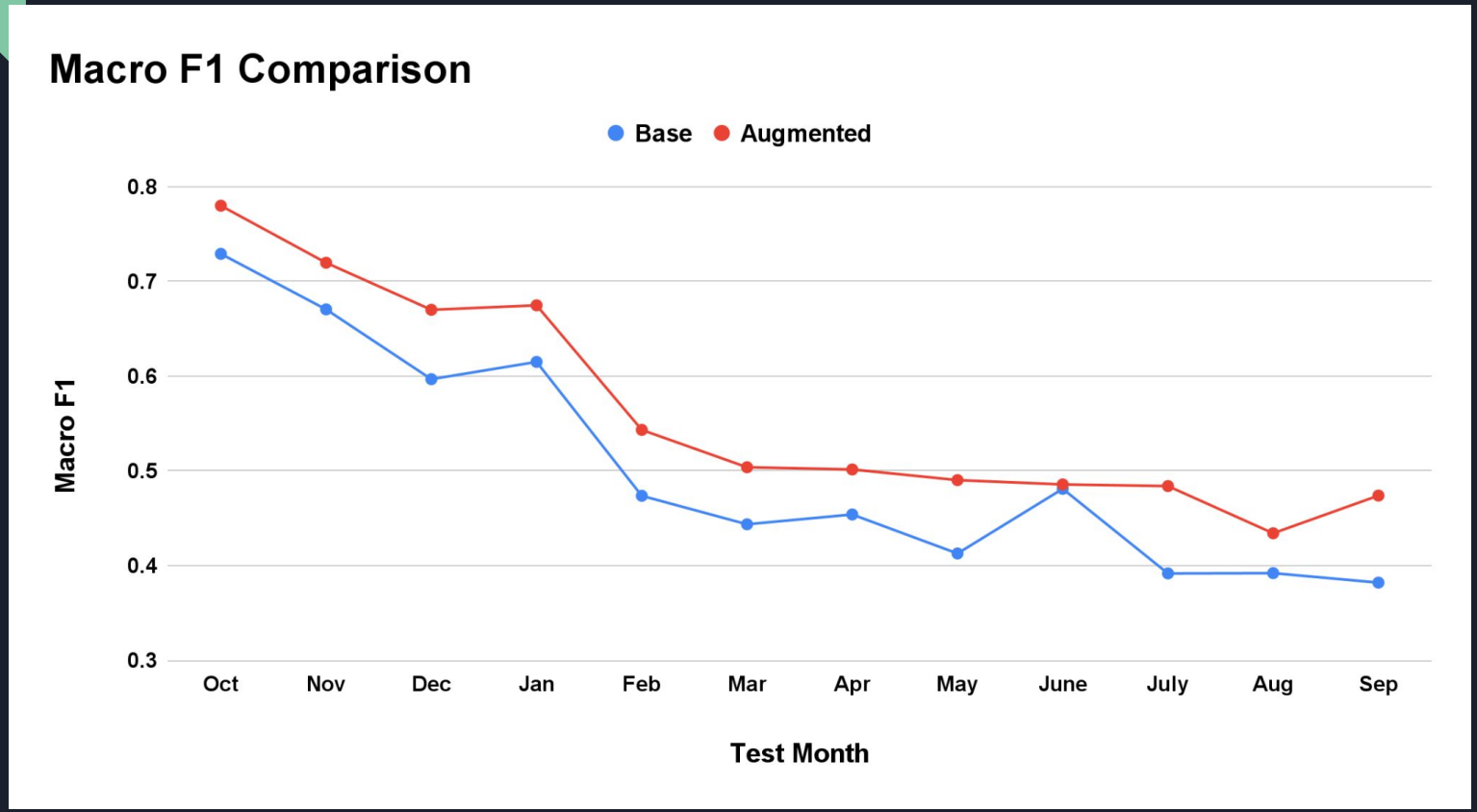
$$F_1 = \frac{2 \cdot \text{Precision} \cdot Recall}{(Precision + Recall)}$$

$$\text{Macro } F_1 = \frac{1}{k} \sum_{i=1}^{k} F_{1i}$$

Macro-F1 Baseline

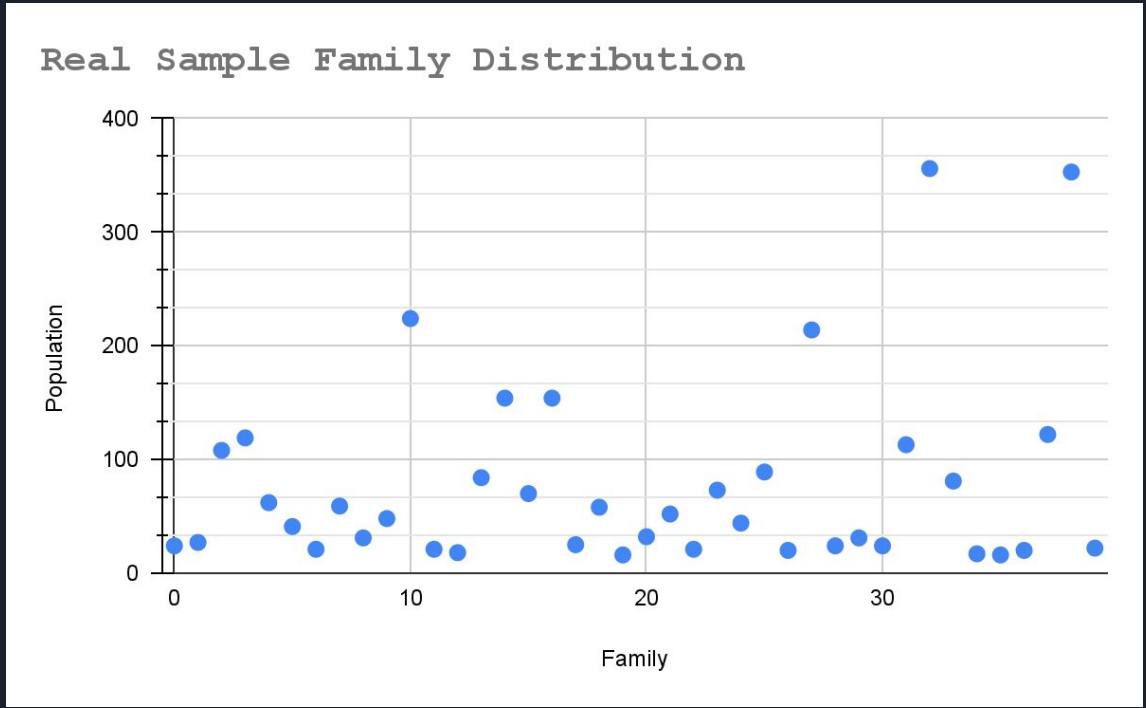


Augmented and Baseline BODMAS



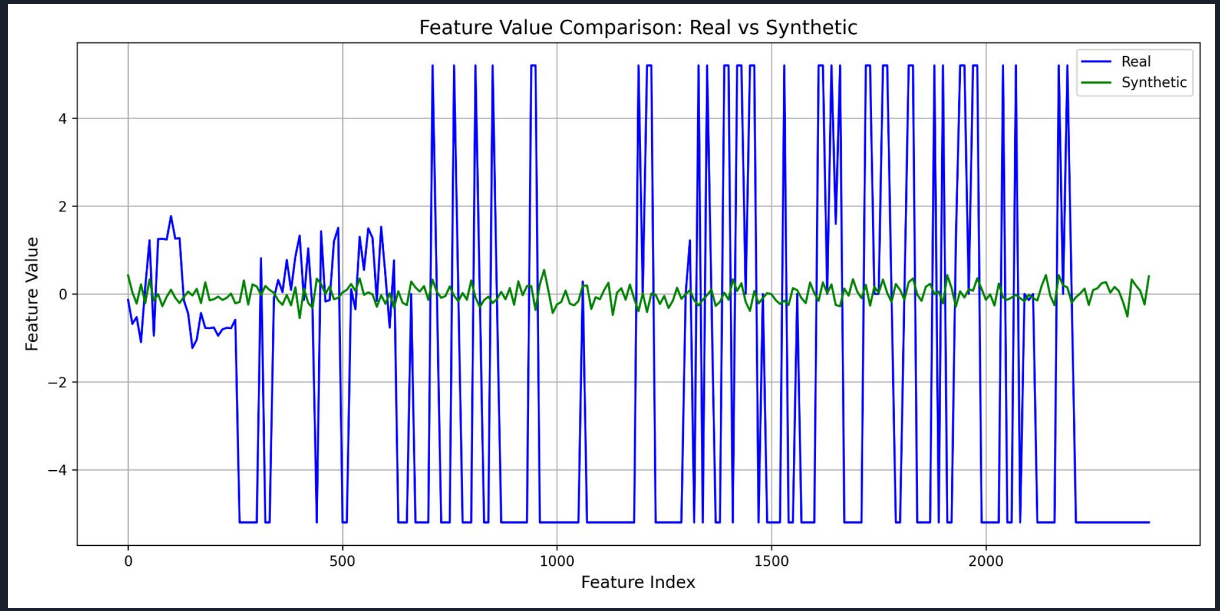
Limitations

- Limited family populations for training



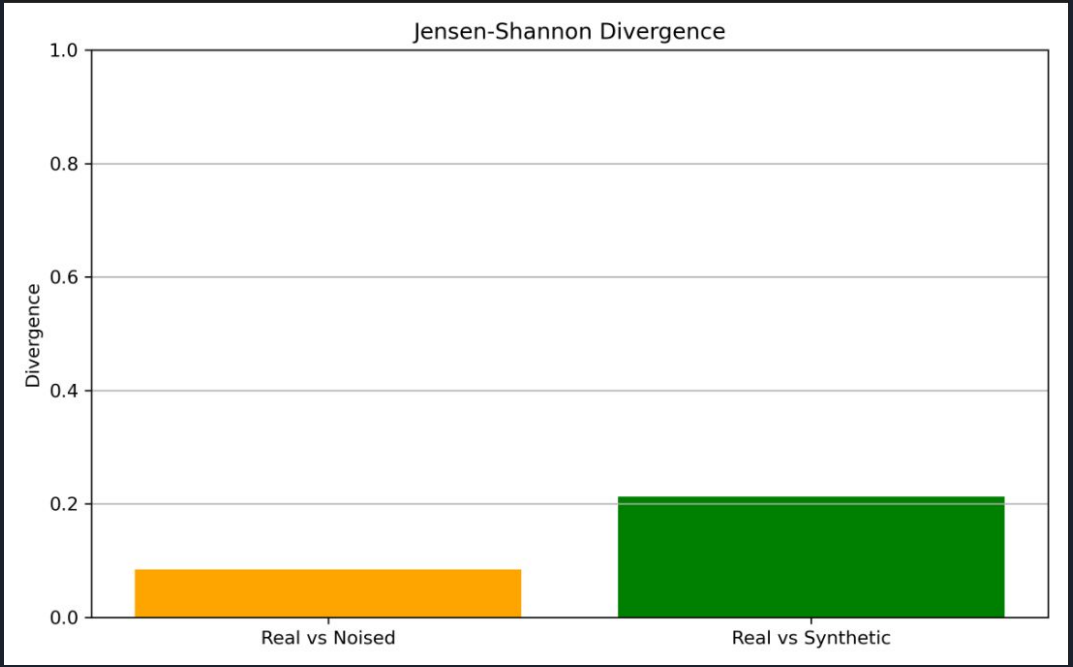
Limitations

- Feature values being compressed to 0 generally means TabDDPM is not confident in its generated feature values



Limitations

- TabDDPM producing low quality samples measured by JS-Divergence
- JS divergence of 0.2 or higher is generally considered bad



Future work

- Creating higher quality synthetic data
- Using js divergence or another metric to filter out bad synthetic samples
- Possibly using another classifier to try to detect synthetic samples
- Try training TabDDPM using Unnormalized data
- Further tuning of data augmentation method

Q & A