Report for HW2

H24111057 姚博瀚 11/27/2023

Section 1: 了解資料

Part 1: 特徵（Feature）的資料屬性

首先我想先確認有無missing value（code 1），再來分出哪些特徵為二元、連續還是離散的型態，code 2依照特徵的值是否只有兩種、整數、小數分為二元、離散、連續。接著我想看每個特徵的數值的分位數以及最大最小值，以describe()產生後再匯出成圖片（code 3）。

Listing 1: Identify missing values

```
missing_values = df_train.isnull().sum()
df_missing_values = pd.DataFrame(missing_values)
df_missing_values.columns = ['Count']
print(df_missing_values)
-----------------------------------------------------------------
                      Count
profile pic               0
nums/length username      0
fullname words            0
nums/length fullname      0
name==username            0
description length        0
external URL              0
private                   0
posts                     0
followers                 0
follows                   0
fake                      0
```

Listing 2: Identify data type

```
import numpy as np
import pandas as pd

# Identify numeric features
numeric_features = X_train.select_dtypes(include=[np.number]).columns.tolist()

# Initialize lists for different types of features
binary_features = []
```

```
continuous_features = []
discrete_features = []

# Define a threshold to differentiate between binary and continuous/discrete
binary_threshold = 2

# Loop through each numeric feature
for feature in numeric_features:
    unique_values = X_train[feature].nunique()
    # Check if the feature is binary
    if unique_values <= binary_threshold:
        binary_features.append(feature)
    else:
        # Check if the feature has only integer values to classify as discrete
        if X_train[feature].dtype == np.int64:
            discrete_features.append(feature)
        else:
            continuous_features.append(feature)

# Print the results
print("Numeric Features:")
if len(numeric_features) == len(X_train.columns):
    print("All features are numeric")
else:
    print(numeric_features)
print("\nBinary Features:")
print(binary_features)
print("\nContinuous Features:")
print(continuous_features)
print("\nDiscrete Features:")
print(discrete_features)
----------------------------------------------------
Numeric Features:
All features are numeric

Binary Features:
['profile pic', 'name==username', 'external URL', 'private']

Continuous Features:
['nums/length username', 'nums/length fullname']

Discrete Features:
['fullname words', 'description length', '#posts', '#followers', '#follows']
```

Listing 3: Summary of features

```
import matplotlib.pyplot as plt

# X_train is a DataFrame
summary = X_train.describe()

# Format the values with 4 decimal places
formatted_values = summary.values.round(4)

# Create a figure and axis with specified DPI
fig, ax = plt.subplots(figsize=(16, 2), dpi=300)

# Plot the table with adjusted fontsize and expanded space for column labels
ax.axis('off')  # Turn off axis
table = ax.table(cellText=formatted_values,
rowLabels=summary.index,
colLabels=summary.columns,
loc='center',
cellLoc='center',
rowLoc='center',
colColours=['#f0f0f0'] * len(summary.columns))  # Set background color for column labels

table.auto_set_font_size(False)
table.set_fontsize(10)  # Adjust the fontsize

# Adjust the space for column labels
table.auto_set_column_width([0] + list(range(len(summary.columns))))

# Save the figure as an image with adjusted DPI
fig.savefig('summary_features_table.png', dpi=300)

# Display the summary table
plt.show()
```

由Figure 1可得知訓練集（train.csv）共有576個樣本，並且post、followers、follows這三個特徵的數值皆偏大，後續可能進行標準化或是什麼轉換以降低此特徵的影響力過大。

Part 2: 目標（**Target**）的資料屬性

由以下code可近一步得知訓練集的576個樣本之中各有288個假帳號與288個真帳號。

Listing 4: Check Target's value

```
# y_train only contains fake columns extracted from train.csv
```

```
y_train.value_counts()
------------------------------------------
0    288
1    288
Name: fake, dtype: int64
```

	profile pic	nums/length username	fullname words	nums/length fullname	name==username
count	576.0	576.0	576.0	576.0	576.0
mean	0.7014	0.1638	1.4601	0.0361	0.0347
std	0.458	0.2141	1.0526	0.1251	0.1832
min	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	1.0	0.0	0.0
50%	1.0	0.0	1.0	0.0	0.0
75%	1.0	0.31	2.0	0.0	0.0
max	1.0	0.92	12.0	1.0	1.0

	description length	external URL	private	#posts	#followers	#follows
count	576.0	576.0	576.0	576.0	576.0	576.0
mean	22.6233	0.1163	0.3819	107.4896	85307.2361	508.3819
std	37.703	0.3209	0.4863	402.0344	910148.4577	917.9812
min	0.0	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	0.0	0.0	39.0	57.5
50%	0.0	0.0	0.0	9.0	150.5	229.5
75%	34.0	0.0	1.0	81.5	716.0	589.5
max	150.0	1.0	1.0	7389.0	15338538.0	7500.0

Figure 1: Summary table of features

Section 2: 前處理

由code 1已知所有的特徵以及目標皆無遺失值，並且已知特徵皆是數字型態（code 2）。由於我後續建立的Decision Tree在decision code要分割樣本的時候可以直接處理二元、離散、連續性的特徵（皆以兩個樣本的數值中點當分割依據），因此特徵不做任何的轉換皆已適合模型的輸入。

但為了避免post、followers、follows這三個特徵的影響力過大，我對這三個特徵的數值進行0.3次方的轉換，目的是把數值的尺度縮小。原本是想進行0.5次方的轉換（也就是把特徵的數值開根號），但做0.3次方的轉換讓followers 的最大值比較接近其餘沒做轉換的特徵的最大值（約150）。如Figure 2所示，左邊為原來的資料的分布圖，右邊則為做完0.3次方轉換後的分布圖。Figure 3則為做完轉換後的summary table。後續模型的輸入皆為直接用轉換過後的資料（X_train_root）。

Section 3: 建立模型

依照講義的作法，使用Entropy來計算node的impurity，再依照Information gain 來決定要用哪個特徵以及那個特徵的哪個threshold來分割目前的node。簡而言之，當使用fit fuction時，會呼叫_grow_tree(呼叫_best_split來找出最好的分割feature與threshhold)來遞迴地生成下面的child node。註1：左child node是小於threshold；右child node是大於threshold。註2：二元

特徵的threshold只有一個，為1；離散與連續性特徵的thresholds是每兩個unique value的平均。

Listing 5: Decision Tree Class

```
import numpy as np
import pandas as pd

# Decision Tree Node class
class Node:
    def __init__(self, entropy, num_samples, num_samples_per_class, predicted_class):
        self.entropy = entropy
        self.num_samples = num_samples
        self.num_samples_per_class = num_samples_per_class
        self.predicted_class = predicted_class
        self.feature_index = 0
        self.threshold = 0
        self.children = {}

# Decision Tree class
class DecisionTree:
    def __init__(self, max_depth = 30):
        self.max_depth = max_depth

    def fit(self, X, y):
        self.n_classes_ = len(set(y))
        self.n_features_ = X.shape[1]
        self.tree_ = self._grow_tree(X, y)

    def _entropy(self, y):
        m = len(y)
        class_probs = [np.sum(y == c) / m for c in range(self.n_classes_)]
        return -np.sum(p * np.log2(p) if p > 0 else 0 for p in class_probs)

    def _information_gain(self, y, y_left, y_right):
        ent_parent = self._entropy(y)
        ent_left = self._entropy(y_left)
        ent_right = self._entropy(y_right)
        weight_left = len(y_left) / len(y)
        weight_right = len(y_right) / len(y)
        return ent_parent - (weight_left * ent_left + weight_right * ent_right)

    def _best_split(self, X, y):
```

```python
        m, n = X.shape
        if m <= 1:
            return None, None
        best_info_gain = 0
        best_idx, best_thr = None, None

        for idx in range(self.n_features_):
            feature_values = X.iloc[:, idx]
            unique_values = feature_values.unique()

            # If the feature is binary, consider only one threshold
            if len(unique_values) == 2:
                thresholds = [unique_values[0]]
            else:
            # For continuous or discrete features, consider midpoints between unique values
                thresholds = [(unique_values[i] + unique_values[i + 1]) / 2 for i in range(len(unique_values) - 1)]

            for thr in thresholds:
                # Split the dataset based on the current threshold
                y_left = y[feature_values < thr]
                y_right = y[feature_values >= thr]

                # Calculate information gain
                info_gain = self._information_gain(y, y_left, y_right)

                # Update best split if current information gain is higher
                if info_gain > best_info_gain:
                    best_info_gain = info_gain
                    best_idx = idx
                    best_thr = thr

        return best_idx, best_thr

    def _grow_tree(self, X, y, depth=0):
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
        predicted_class = np.argmax(num_samples_per_class)
        node = Node(
                    entropy=self._entropy(y),
                    num_samples=len(y),
                    num_samples_per_class=num_samples_per_class,
                    predicted_class=predicted_class
                    )

        if depth < self.max_depth:
```

```
            idx, thr = self._best_split(X, y)
            if idx is not None:
                feature_values = X.iloc[:, idx]
                X_left, y_left = X[feature_values < thr], y[feature_values < thr]
                X_right, y_right = X[feature_values >= thr], y[feature_values >= thr]
                node.feature_index = idx
                node.threshold = thr
                node.children['left'] = self._grow_tree(X_left, y_left, depth + 1)
                node.children['right'] = self._grow_tree(X_right, y_right, depth + 1)
        return node

    def predict(self, X):
        return [self._predict(inputs) for inputs in X.values]

    def _predict(self, inputs):
        node = self.tree_
        while 'left' in node.children:
            feature_value = inputs[node.feature_index]

            # Check if the feature value is numeric
            if isinstance(feature_value, (int, float)):
                if feature_value < node.threshold:
                    node = node.children['left']
                else:
                    node = node.children['right']
            else:
                # Handle non-numeric feature values
                raise ValueError("Unsupported feature type in decision tree prediction.")

        return node.predicted_class
```

Section 4: 優化

為了使整個模型不過於複雜以及為了避免overfitting，繼承了原本的Decision Tree class來修改_best_split、_grow_tree以增加以下四個conditions於PrePrunedDecisionTree class：

1. min_samples_split(default=4)：當要分割當前node所需的最小樣本數。
2. min_samples_leaf(default=2)：要成為leaf node(terminal node)所需的最小樣本數。
3. min_info_gain(default=0.05)：每次分隔node的最小所需imformation gain。

4. max_features(default=None)：每次分隔時考慮的特徵數量，會隨機抽出所設的特徵數量。None為考慮所有的特徵。

註：max_depth為原本就有的超參數。

Listing 6: Pre-Pruned Decision Tree Class

```python
import numpy as np
import pandas as pd
import random

# Pre-Pruned Decision Tree class
class PrePrunedDecisionTree(DecisionTree):
    def __init__(self, max_depth=30, min_samples_split=4, min_samples_leaf=2, min_info_gain=0.05, max_features=None):
        super().__init__(max_depth=max_depth)
        self.min_samples_split = min_samples_split
        self.min_samples_leaf = min_samples_leaf
        self.min_info_gain = min_info_gain
        self.max_features = max_features

    def _best_split(self, X, y):
        m, n = X.shape
        if m <= 1:
            return None, None

        best_info_gain = 0
        best_idx, best_thr = None, None

        # Select a random subset of features if max_features is specified
        if self.max_features is not None:
            all_features = range(self.n_features_)
            selected_features = random.sample(all_features, min(self.max_features, self.n_features_))
        else:
            selected_features = range(self.n_features_)

        for idx in selected_features:
            feature_values = X.iloc[:, idx]
            unique_values = feature_values.unique()

            # If the feature is binary, consider only one threshold
            if len(unique_values) == 2:
                thresholds = [unique_values[1]]
            else:
                # For continuous or discrete features, consider midpoints between unique values
                thresholds = [(unique_values[i] + unique_values[i + 1]) / 
```

```
                2 for i in range(len(unique_values) - 1)]

            for thr in thresholds:
                # Split the dataset based on the current threshold
                y_left = y[feature_values < thr]
                y_right = y[feature_values >= thr]

                # Apply pre-pruning conditions
                if (len(y_left) < self.min_samples_leaf or len(y_right) < self.min_samples_leaf):
                    continue

                # Calculate information gain
                info_gain = self._information_gain(y, y_left, y_right)

                # Update best split if current information gain is higher
                if info_gain > best_info_gain:
                    best_info_gain = info_gain
                    best_idx = idx
                    best_thr = thr

        # Apply pre-pruning conditions
        if best_info_gain < self.min_info_gain:
            return None, None

        return best_idx, best_thr


    def _grow_tree(self, X, y, depth=0):
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
        predicted_class = np.argmax(num_samples_per_class)
        node = Node(
                    entropy=self._entropy(y),
                    num_samples=len(y),
                    num_samples_per_class=num_samples_per_class,
                    predicted_class=predicted_class
                    )

        if depth < self.max_depth:
            idx, thr = self._best_split(X, y)
            if idx is not None:
                feature_values = X.iloc[:, idx]
                X_left, y_left = X[feature_values < thr], y[feature_values < thr]
                X_right, y_right = X[feature_values >= thr], y[feature_values >= thr]
```

```

                # Apply pre-pruning conditions
                if (
                    len(y_left) >= self.min_samples_split
                    and len(y_right) >= self.min_samples_split
                    ):
                    node.feature_index = idx
                    node.threshold = thr
                    node.children['left'] = self._grow_tree(X_left, y_left, depth + 1)
                    node.children['right'] = self._grow_tree(X_right, y_right, depth + 1)
        return node
```

Listing 7: Accuracy before pruning

```
import numpy as np
import pandas as pd

df_train = pd.read_csv('train.csv')
X_train = df_train.drop('fake', axis=1)
y_train = df_train['fake']

df_test = pd.read_csv('test.csv')
X_test = df_test.drop('fake', axis=1)
y_test = df_test['fake']

X_train_root = X_train.copy()
X_train_root['#followers'] = X_train_root['#followers']**0.3
X_train_root['#follows'] = X_train_root['#follows']**0.3
X_train_root['#posts'] = X_train_root['#posts']**0.3

X_test_root = X_test.copy()
X_test_root['#followers'] = X_test_root['#followers']**0.3
X_test_root['#follows'] = X_test_root['#follows']**0.3
X_test_root['#posts'] = X_test_root['#posts']**0.3

model_root = DecisionTree(max_depth=7)
model_root.fit(X_train_root, y_train)

y_pred_root = model_root.predict(X_test_root)
accuracy_root = np.mean(y_pred_root == y_test.values.flatten())
print(f"Accuracy before prepruning: {accuracy_root:.5f}")
-------------------------------------------------------------------------
Accuracy before prepruning: 0.88333
```

Listing 8: Accuracy after pruning

```python

import itertools

# Define the parameter grid to search
param_grid = {
    'max_depth': [5, 7, 9],
    'min_samples_split': [3, 4, 5],
    'min_samples_leaf': [2, 3, 4],
    'min_info_gain': [0.01, 0.04, 0.08],
    'max_features': [4, 6, 8]
}

# Create the decision tree model
base_decision_tree = PrePrunedDecisionTree()

best_accuracy = 0
best_params = None

# Perform grid search
for params in itertools.product(*param_grid.values()):
    param_dict = dict(zip(param_grid.keys(), params))

    # Create the decision tree model with current parameters
    decision_tree = PrePrunedDecisionTree(**param_dict)

    # Fit the model on the training data
    decision_tree.fit(X_train_root, y_train)

    # Make predictions on the test set
    y_pred = decision_tree.predict(X_test_root)

    # Evaluate accuracy
    #accuracy = accuracy_score(y_test, y_pred)
    accuracy = np.mean(y_pred == y_test)


    # Update best parameters if the current model is better
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_params = param_dict

# Print the best hyperparameters and accuracy
print("Best Hyperparameters:", best_params)
print("Test Accuracy after pruning:", best_accuracy)
------------------------------------------------------------------
Best Hyperparameters: {'max_depth': 7, 'min_samples_split': 3, '↩
```

```
    min_samples_leaf': 2, 'min_info_gain': 0.01, 'max_features': 8}
Test Accuracy after pruning: 0.95
```

Section 5: 解釋

Figure 4 為最終的決策樹結構，依此圖來看，follows、followers、posts為用來做決策的主要特徵，如果要再做優化可能仍須再把這三個的數值尺度降低，或者是max_features要設小一點。

Listing 9: output structure

```python
#from sklearn.tree import export_graphviz
import graphviz

# Helper function to convert ID3 tree to a format compatible with export_graphviz
def tree_to_graphviz(tree, feature_names, class_names, dpi=300, size=None):
    def recurse(node, graph):
        # Combine nodes for decision split
        if 'left' in node.children:
            #decision_label = f"If feature {node.feature_index} < {node.threshold}"
            #decision_label = f"If feature {node.feature_index} < {node.threshold}\nClass: {node.predicted_class}\nEntropy = {node.entropy}\nSamples = {node.num_samples}\nClass Distribution: {node.num_samples_per_class}"
            decision_label = f"If {feature_names[node.feature_index]} < {node.threshold}\nClass: {node.predicted_class}\nEntropy = {node.entropy}\nSamples = {node.num_samples}\nClass Distribution: {node.num_samples_per_class}"

            graph.node(str(id(node)), label=decision_label, shape='box', fillcolor='#78bceb')
            left_child = node.children['left']
            right_child = node.children['right']

            recurse(left_child, graph)
            recurse(right_child, graph)

            graph.edge(str(id(node)), str(id(left_child)), label='True')
            graph.edge(str(id(node)), str(id(right_child)), label='False')
        else:
            # Leaf node
```

```
            leaf_label = f"Class: {node.predicted_class}\nEntropy = {node.entropy}\nSamples = {node.num_samples}\nClass Distribution: {node.num_samples_per_class}"
            graph.node(str(id(node)), label=leaf_label, shape='box', fillcolor='#78bceb')

    graph = graphviz.Digraph(format='png')
    root_label = f"Root\nEntropy = {tree.entropy}\nSamples = {tree.num_samples}\nClass Distribution: {tree.num_samples_per_class}"
    graph.node(str(id(tree)), label=root_label, shape='box', fillcolor='#78bceb')
    recurse(tree, graph)

    # Set DPI and size attributes
    if dpi is not None:
        graph.attr(dpi=str(dpi))
    if size is not None:
        graph.attr(size=size)

    return graph

------------------------------------------------
feature_names = list(X_train.columns)
class_names = list(map(str, set(y_train)))
# Set the desired DPI
dpi = 300
# Set the desired size in inches (width, height)
size = None
graph = tree_to_graphviz(prepruned_model_root.tree_, feature_names, class_names, dpi=dpi, size=size)
graph.render("prepruned_model_root")
```

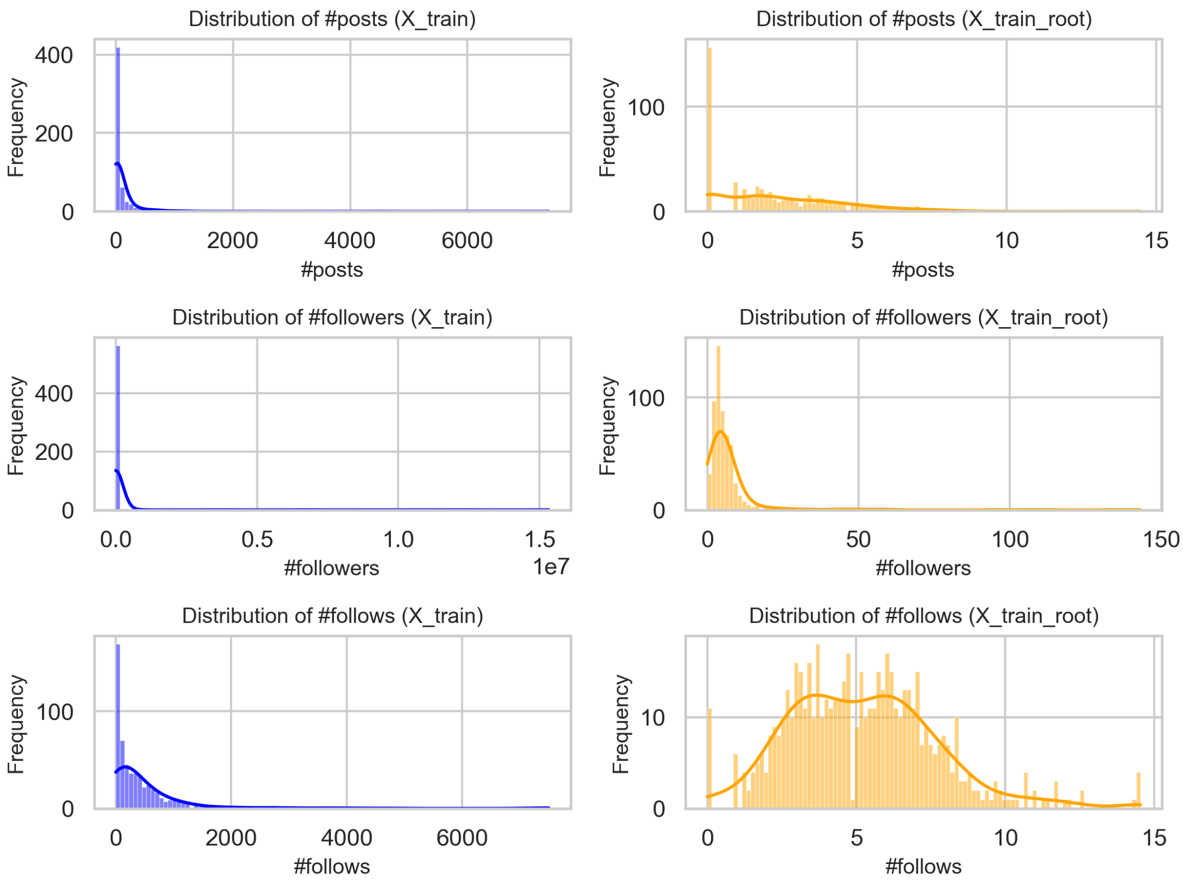

Figure 2: Distribution comparison(post, followers, follows are transformed to the power of 0.3)

	profile pic	nums/length username	fullname words	nums/length fullname	name==username
count	576.0	576.0	576.0	576.0	576.0
mean	0.7014	0.1638	1.4601	0.0361	0.0347
std	0.458	0.2141	1.0526	0.1251	0.1832
min	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	1.0	0.0	0.0
50%	1.0	0.0	1.0	0.0	0.0
75%	1.0	0.31	2.0	0.0	0.0
max	1.0	0.92	12.0	1.0	1.0

	description length	external URL	private	#posts	#followers	#follows
count	576.0	576.0	576.0	576.0	576.0	576.0
mean	22.6233	0.1163	0.3819	2.3739	7.1458	5.224
std	37.703	0.3209	0.4863	2.1771	12.6298	2.5185
min	0.0	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	0.0	0.0	3.0014	3.372
50%	0.0	0.0	0.0	1.9332	4.5005	5.1078
75%	34.0	0.0	1.0	3.7441	7.1858	6.7787
max	150.0	1.0	1.0	14.4736	143.1315	14.5385

Figure 3: Summary table of features (post, followers, follows are transformed to the power of 0.3)

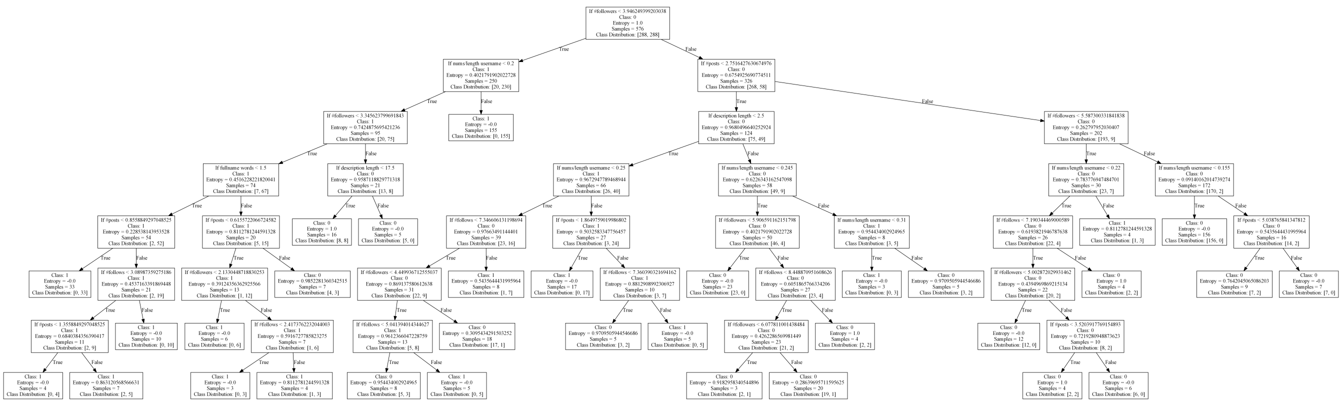

Figure 4: Tree Structure