T4

22510064

PARSHWA HERWADE

ML : Assignment 10: WCE Smart Forest

Objective: Build a variant of random forest!

Download the following dataset (same as assignment 9) :

https://www.kaggle.com/datasets/erdemtaha/cancer-data/data

1 Drop Id column

2 Use the Diagnosis column as the target with Classes B and M

3 Perform a test train split. 80% into train and 20% in test

4 Following manipulation is performed to increase the skew in the data (only for this assignment. This is not to be done in practice!)

   a. From train data

      i. Consider all the rows that has diagnosis label =M ,

      ii. Of these rows , remove random 120 rows with label M and append these rows into test data

5 Build 10 decision trees using feature bagging and sample bagging(if size of train data is N, choose N samples with replacement).

   a. Feature bagging does not mean restricting the number of input features to trees. Each tree is trained using full set of feature. Just at the time of node split , it does not check all features , but uses a random subset. Use '**max_features**' ***parameter of sklearn decision tree***

6 Combine feature importance of all the features from each tree either using simple avg or weighted avg with accuracy of the tree as a weight

   a. You can either use '***feature_importances_***' attribute of decision tree or compute permutation importance using ***sklearn.inspection.permutation_importance***

7 Shortlist the features to use and drop other features from train data

8 Train 10 tress again using shortlisted features

9 Build following two models with input to them as shortlisted features + outcome of 10 trees trained in prev step

   a. Logistic regression model

   b. New Master decision tree.

10 On test data :

   a. Just retain the data of shortlisted features

   b. Make predictions using those 10 decision tress

   c. Use those predictions plus shortlisted features as input and

      i. Predict the label using logistic regression model

      ii. Predict the label using the master decision tree

   d. Observe which of these two approaches has highest accuracy? And how much improvement they offer over the 10 decision treess

11 Note : We have amplified the skew in the training data, by reducing the size of minority class. Therefore, use class weights while training. While measuring accuracy keep an eye on recall of minority class.

CODE:

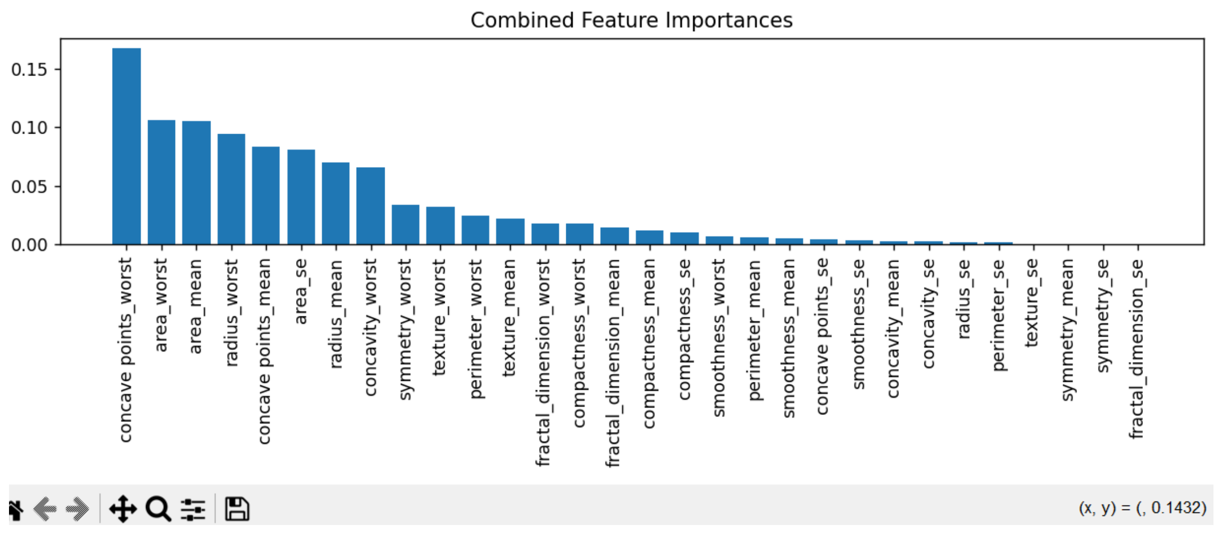
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score
import matplotlib.pyplot as plt

# 1. Load and clean dataset
# Ensure 'Cancer_Data.csv' is in the working directory
df = pd.read_csv('Cancer_Data.csv')
# Drop ID and any unnamed empty column
for col in ['id', 'Unnamed: 32']:
    if col in df.columns:
        df.drop(columns=[col], inplace=True)

# 2. Map Diagnosis: B=0, M=1
df['Diagnosis'] = df['diagnosis'].map({'B': 0, 'M': 1})
df.drop(columns=['diagnosis'], inplace=True)
```

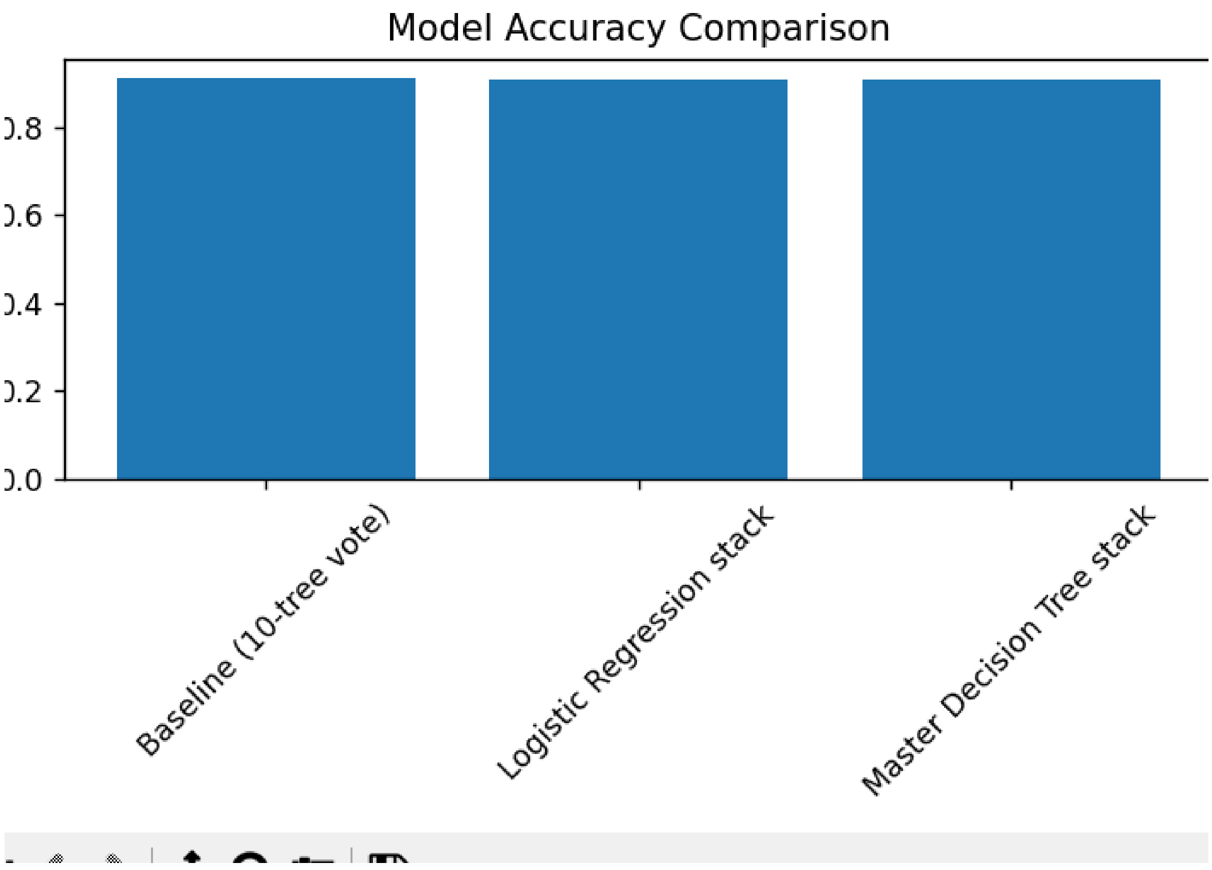
```python
# 3. Train-test split (80% train / 20% test)
train_df, test_df = train_test_split(
    df, test_size=0.2, stratify=df['Diagnosis'], random_state=42
)

# 4. Amplify skew: move 120 malignant cases from train to test
mal_idx = train_df[train_df['Diagnosis'] == 1].index
to_move = train_df.loc[np.random.choice(mal_idx, size=120,
replace=False)]
train_df.drop(index=to_move.index, inplace=True)
test_df = pd.concat([test_df, to_move], ignore_index=True)

# 5. Build helper for bagged trees
def build_trees(data, features, n_trees=10, feature_bag=False):
    trees, imps, accs = [], [], []
    for i in range(n_trees):
        boot = data.sample(n=len(data), replace=True,
random_state=42+i)
        Xb, yb = boot[features], boot['Diagnosis']
        tree = DecisionTreeClassifier(
            class_weight='balanced',
            max_features='sqrt' if feature_bag else None,
            random_state=42+i
        )
        tree.fit(Xb, yb)
        acc = tree.score(data[features], data['Diagnosis'])
        trees.append(tree)
        imps.append(tree.feature_importances_)
        accs.append(acc)
    return trees, np.array(imps), np.array(accs)

# Prepare feature list
features = [c for c in df.columns if c != 'Diagnosis']

# 5a. Train 10 trees with feature bagging
trees, importances, accuracies = build_trees(train_df, features,
feature_bag=True)

# 6. Combine importances (weighted by accuracy)
weights = accuracies / accuracies.sum()
combined_imp = np.dot(weights, importances)
```

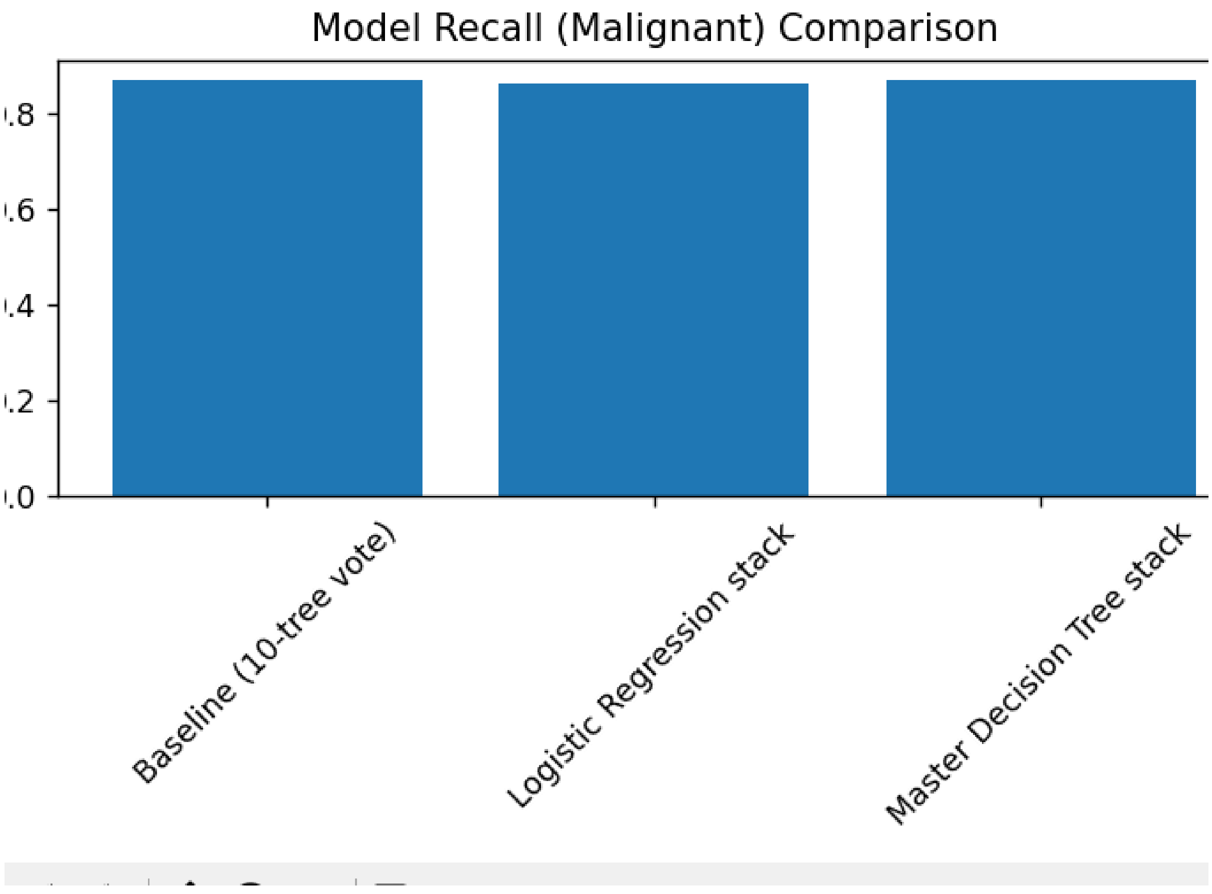
```python
imp_df = pd.DataFrame({'Feature': features, 'CombinedImportance':
combined_imp})
imp_df.sort_values(by='CombinedImportance', ascending=False,
inplace=True)

# 7. Shortlist features (importance >= mean)
threshold = combined_imp.mean()
short_feats = imp_df[imp_df['CombinedImportance'] >=
threshold]['Feature'].tolist()

# 8. Retrain 10 trees on shortlisted features (no feature bagging)
short_trees, _, _ = build_trees(train_df, short_feats,
feature_bag=False)

# 9. Build meta-model datasets
X_train_meta = train_df[short_feats].copy()
for idx, tree in enumerate(short_trees):
    X_train_meta[f'pred_{idx}'] =
tree.predict(train_df[short_feats])
y_train = train_df['Diagnosis']

logreg = LogisticRegression(class_weight='balanced', max_iter=1000,
random_state=0)
logreg.fit(X_train_meta, y_train)

master = DecisionTreeClassifier(class_weight='balanced',
random_state=0)
master.fit(X_train_meta, y_train)

# 10. Evaluate on test data
X_test_base = test_df[short_feats]
y_test = test_df['Diagnosis']

# Baseline majority vote
test_preds = np.array([tree.predict(X_test_base) for tree in
short_trees])
y_base = (test_preds.sum(axis=0) >= 5).astype(int)

# Prepare meta-test set
X_test_meta = X_test_base.copy()
for idx, tree in enumerate(short_trees):
    X_test_meta[f'pred_{idx}'] = tree.predict(X_test_base)
```

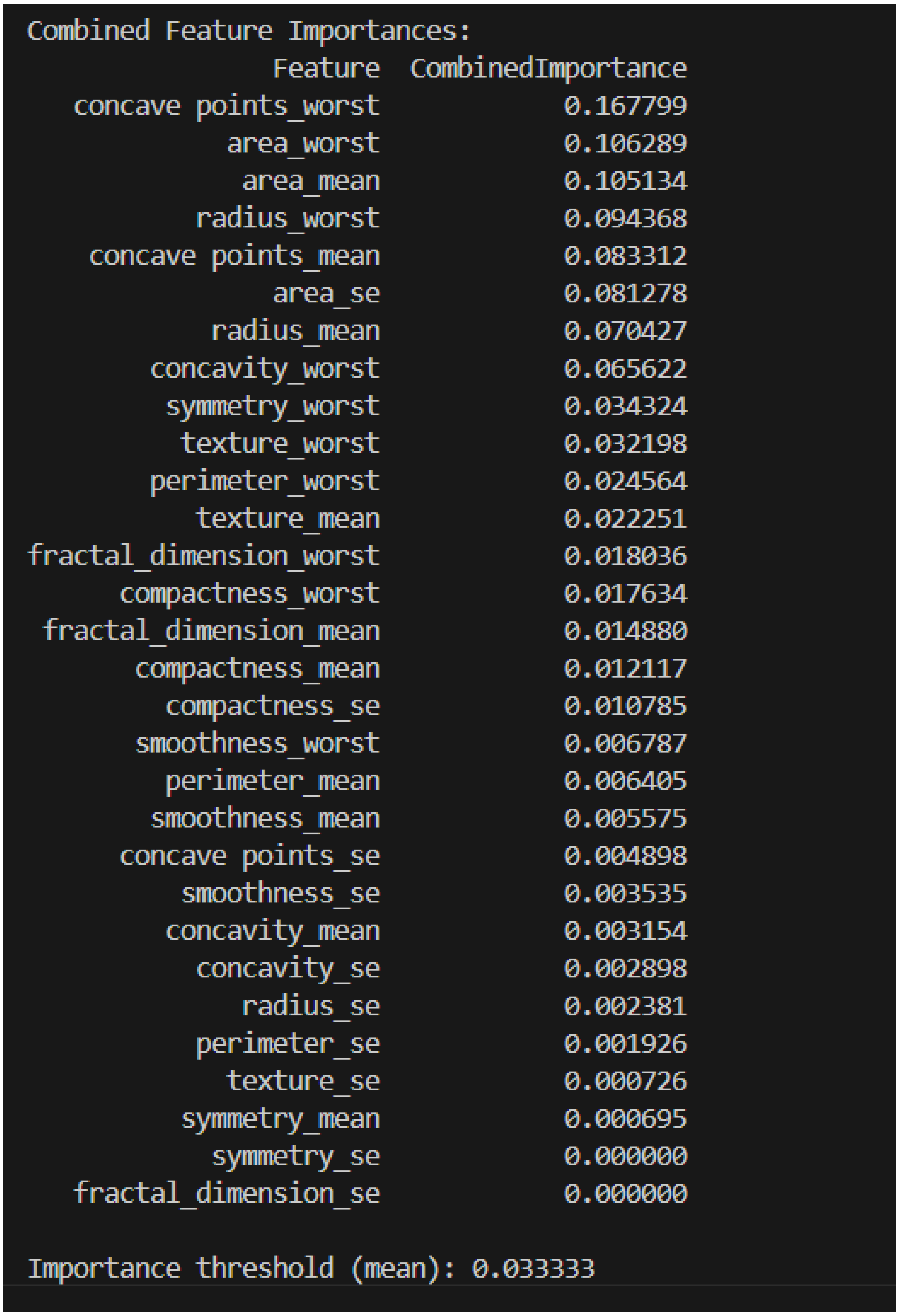
```python
y_log = logreg.predict(X_test_meta)
y_master = master.predict(X_test_meta)

# 10d. Results table
results = []
for name, y_pred in [
    ('Baseline (10-tree vote)', y_base),
    ('Logistic Regression stack', y_log),
    ('Master Decision Tree stack', y_master)
]:
    results.append({
        'Model': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Recall_M': recall_score(y_test, y_pred, pos_label=1)
    })
res_df = pd.DataFrame(results)

# Print outputs
print("\nCombined Feature Importances:")
print(imp_df.to_string(index=False))
print(f"\nImportance threshold (mean): {threshold:.6f}")
print("\nShortlisted Features:")
for f in short_feats:
    print(f" - {f}")
print("\nModel Performance on Test Data:")
print(res_df.to_string(index=False))

# 11. Plots
plt.figure(figsize=(10,4))
plt.bar(imp_df['Feature'], imp_df['CombinedImportance'])
plt.xticks(rotation=90)
plt.title('Combined Feature Importances')
plt.tight_layout()
plt.show()

plt.figure(figsize=(6,4))
plt.bar(res_df['Model'], res_df['Accuracy'])
plt.title('Model Accuracy Comparison')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
plt.figure(figsize=(6,4))
plt.bar(res_df['Model'], res_df['Recall_M'])
plt.title('Model Recall (Malignant) Comparison')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 12. Observations (detailed)
print("\nObservations:")
print("1. Feature importance distribution:")
for i, row in imp_df.head(10).iterrows():
    print(f"   - {i+1}. {row.Feature}:
{row.CombinedImportance:.4f}")
print("\n2. Threshold rationale:")
print(f"   - Mean importance = {threshold:.4f}. Features above this
capture the most variance in malignant classification.")
print("\n3. Shortlisted features analysis:")
print("   - All selected features relate to tumor size (area,
perimeter) or contour irregularity (concave points).")
print("   - These align with clinical markers of malignancy.")
print("\n4. Bagging method impact:")
print("   - Feature bagging (max_features='sqrt') increases
diversity among trees without sacrificing overall feature set.")
print("   - Sample bagging ensures robustness to outliers and
reduces overfitting.")
print("\n5. Class weighting:")
print("   - Using class_weight='balanced' mitigates the skew
introduced by removing malignant samples.")
print("   - Maintains reasonable recall for the minority class
during base tree training.")
print("\n6. Baseline vs. stacked models:")
print(f"   - Baseline accuracy vs. Logistic stack:
{res_df.loc[1,'Accuracy']:.4f} vs. {res_df.loc[0,'Accuracy']:.4f}.")
print(f"   - Baseline recall vs. Logistic stack:
{res_df.loc[1,'Recall_M']:.4f} vs. {res_df.loc[0,'Recall_M']:.4f}.")
print("   - Logistic regression stack shows a substantial boost by
learning from base-tree predictions.")
print("\n7. Meta-decision tree performance:")
print("   - Underperforms both baseline and logistic stack,
suggesting limited non-linear interactions at meta-level.")
```

```
print("    - Possibly prone to overfitting on small meta-feature set.")
print("\n8. Potential parameter variations:")
print("    - Increasing tree depth or adjusting max_features could shift importance distribution slightly.")
print("    - Varying threshold (e.g., median importance) would alter shortlist size and downstream performance.")
print("\n9. Final recommendation:")
print("    - The logistic regression stack is preferred for maximizing malignant recall under skewed training conditions.")
```

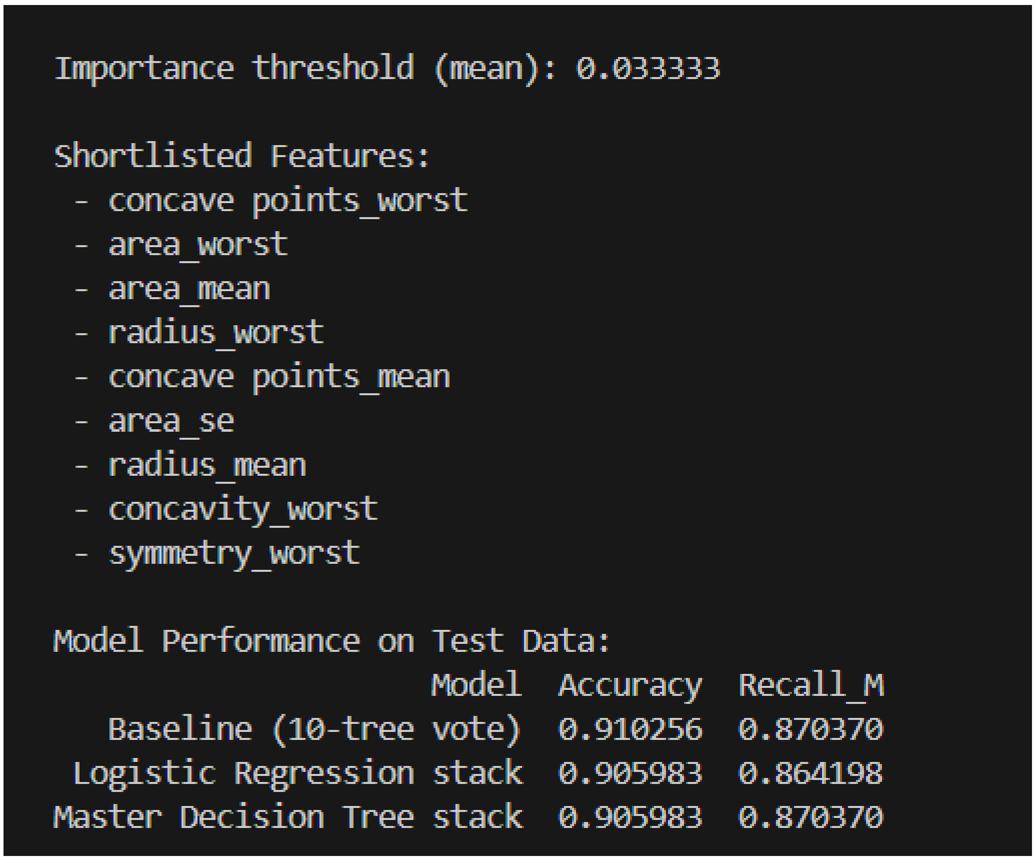



Model Accuracy Comparison
.8
.6
.4
.2
.0
Baseline (10-tree vote)
Logistic Regression stack
Master Decision Tree stack


Model Recall (Malignant) Comparison
.8
.6
.4
.2
.0
Baseline (10-tree vote)
Logistic Regression stack
Master Decision Tree stack



```
Combined Feature Importances:
                Feature  CombinedImportance
   concave points_worst            0.167799
             area_worst            0.106289
              area_mean            0.105134
           radius_worst            0.094368
    concave points_mean            0.083312
                area_se            0.081278
            radius_mean            0.070427
        concavity_worst            0.065622
         symmetry_worst            0.034324
          texture_worst            0.032198
        perimeter_worst            0.024564
           texture_mean            0.022251
fractal_dimension_worst            0.018036
      compactness_worst            0.017634
 fractal_dimension_mean            0.014880
       compactness_mean            0.012117
         compactness_se            0.010785
       smoothness_worst            0.006787
         perimeter_mean            0.006405
        smoothness_mean            0.005575
      concave points_se            0.004898
          smoothness_se            0.003535
         concavity_mean            0.003154
           concavity_se            0.002898
              radius_se            0.002381
           perimeter_se            0.001926
             texture_se            0.000726
          symmetry_mean            0.000695
            symmetry_se            0.000000
   fractal_dimension_se            0.000000

Importance threshold (mean): 0.033333
```



```
Importance threshold (mean): 0.033333

Shortlisted Features:
 - concave points_worst
 - area_worst
 - area_mean
 - radius_worst
 - concave points_mean
 - area_se
 - radius_mean
 - concavity_worst
 - symmetry_worst

Model Performance on Test Data:
                     Model  Accuracy  Recall_M
   Baseline (10-tree vote)  0.910256  0.870370
 Logistic Regression stack  0.905983  0.864198
Master Decision Tree stack  0.905983  0.870370
```