MUHAMMAD IQBAL DHARMAWAN | 1103194125

TECHNICAL REPORT FOR MACHINE LEARNING MID-TEARM EXAM

Introduction

In machine learning, Decision Tree, Random Forest, and Self-Training are three commonly used algorithms for classification tasks. Decision Trees are tree-like structures that use a set of rules to classify data. Random Forest is an ensemble learning algorithm that combines multiple Decision Trees to increase accuracy. Self-Training is a semi-supervised learning approach that uses labeled and unlabeled data to improve the classification performance. This report will provide a detailed overview of these three algorithms, their advantages, disadvantages, and use cases.

Correlation heatmap

A correlation heatmap is a graphical representation of the correlation between variables in a dataset. It is a useful tool for exploring the relationships between variables and identifying patterns in the data. The heatmap is a color-coded matrix that displays the correlation coefficient between pairs of variables. The values range from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.

The heatmap can be generated using various libraries in Python, such as Seaborn and Matplotlib. It is often used in exploratory data analysis to identify potential multicollinearity between features, which can affect the accuracy of the model. Multicollinearity occurs when two or more features in a dataset are highly correlated with each other, making it difficult for the model to distinguish the effect of each feature on the outcome variable.

By examining the heatmap, we can identify pairs of variables that are highly correlated and remove one of them from the model to improve its performance. It can also be used to identify clusters of variables that are highly correlated with each other, which can be useful in feature selection and dimensionality reduction.

Correlation heatmap implementation with sklearn and google collab:

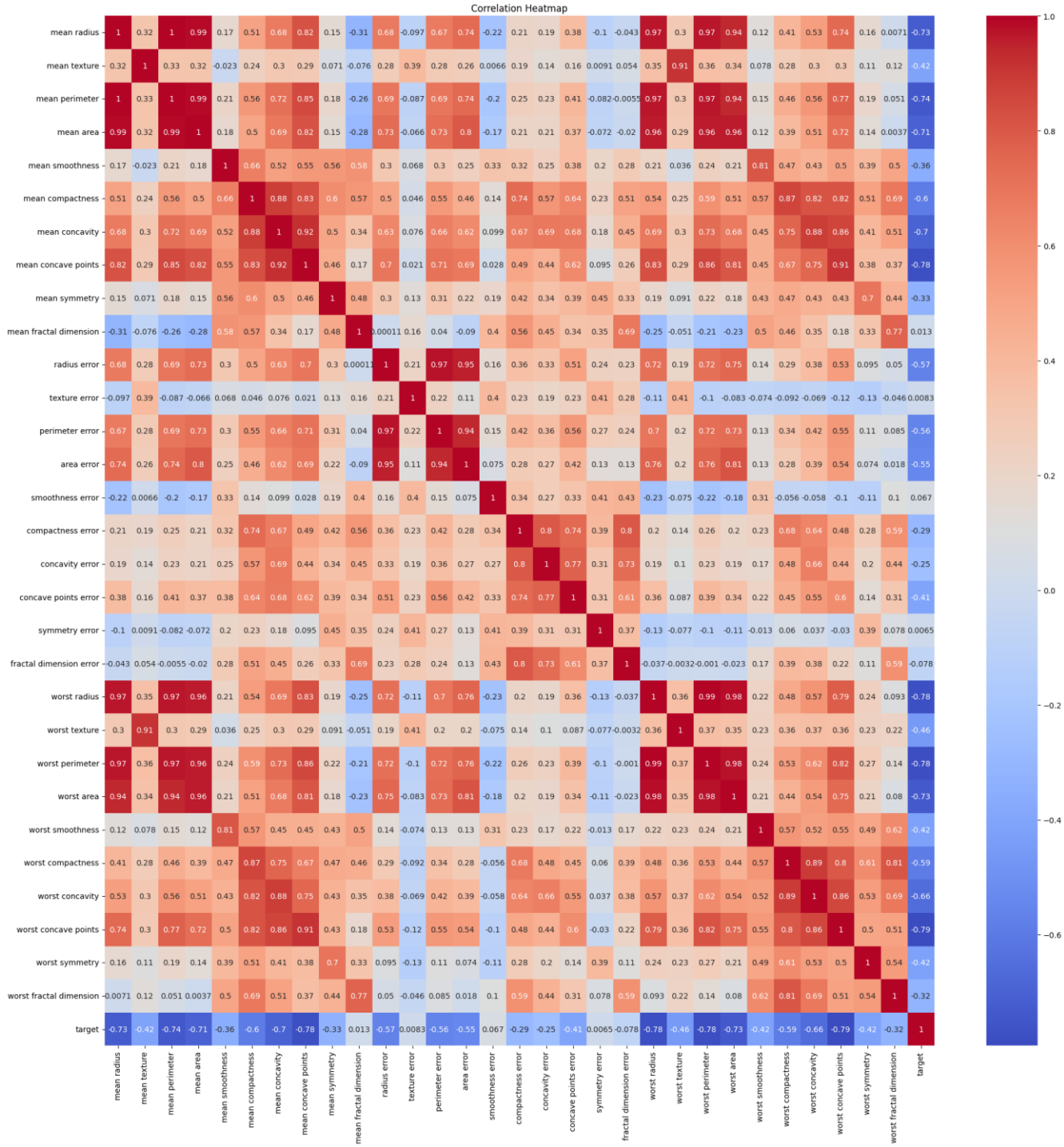
Source code:

```
import seaborn as sns
import pandas as pd
from sklearn.datasets import load_breast_cancer

cancer_dataset = load_breast_cancer()
cancer_df = pd.DataFrame(cancer_dataset.data, columns=cancer_dataset.feature_names)
cancer_df['target'] = cancer_dataset.target
corr_matrix = cancer_df.corr(method='pearson')

# Create a heatmap using the correlation matrix
plt.figure(figsize=(24,24))
plt.title('Correlation Heatmap')
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

plt.show()
```

MUHAMMAD IQBAL DHARMAWAN | 1103194125

TECHNICAL REPORT FOR MACHINE LEARNING MID-TEARM EXAM

Output:



MUHAMMAD IQBAL DHARMAWAN | 1103194125

TECHNICAL REPORT FOR MACHINE LEARNING MID-TEARM EXAM

Decision Tree

A decision tree is a supervised learning algorithm that can be used for both classification and regression tasks. It works by recursively splitting the dataset based on the values of input features to create a tree-like structure. Each node of the tree represents a test on a particular feature, and each branch represents the outcome of that test. The goal of a decision tree is to create a model that predicts the class label of a new input based on the features of that input.

Advantages:

- Easy to understand and interpret
- Can handle both categorical and numerical data
- Requires less data preprocessing and feature engineering

Disadvantages:

- Can be prone to overfitting
- Not suitable for complex relationships between features
- Can be biased towards features with many levels

Use Cases:

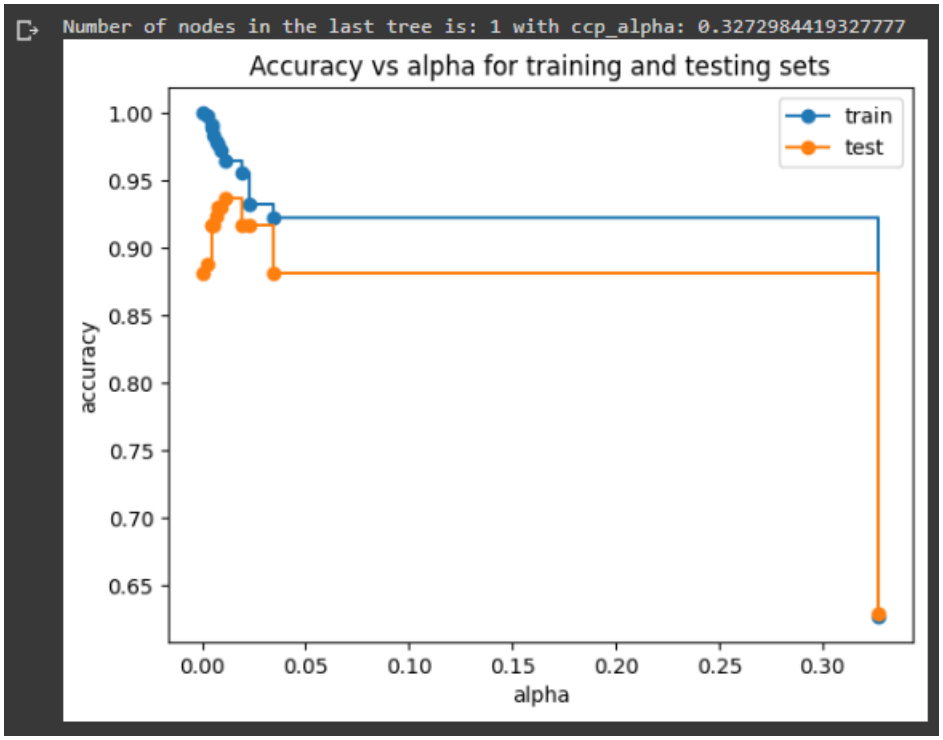
- Customer segmentation
- Fraud detection
- Medical diagnosis

Decision tree training with sklearn and google collab:

Source code:

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier(random_state=0)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)

train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker="o", label="train", drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker="o", label="test", drawstyle="steps-post")
ax.legend()
plt.show()
```

Output:



Random Forest

Random Forest is an ensemble learning algorithm that combines multiple Decision Trees to improve accuracy and reduce overfitting. It works by creating a set of Decision Trees on random subsets of the data and features. The final prediction is made by aggregating the predictions of all the trees in the forest. Random Forest is a powerful algorithm that can handle high-dimensional datasets with complex relationships between features.

Advantages:

- Reduced overfitting compared to a single Decision Tree
- Can handle both categorical and numerical data
- Can handle missing data
- Provides feature importance measures

Disadvantages:

- Can be slow for large datasets
- Difficult to interpret the results
- Can have a high memory footprint

Use Cases:

- Predicting stock prices
- Credit risk assessment
- Image classification

MUHAMMAD IQBAL DHARMAWAN | 1103194125

TECHNICAL REPORT FOR MACHINE LEARNING MID-TEARM EXAM

Random forest training with sklearn and google collab:

Source code:

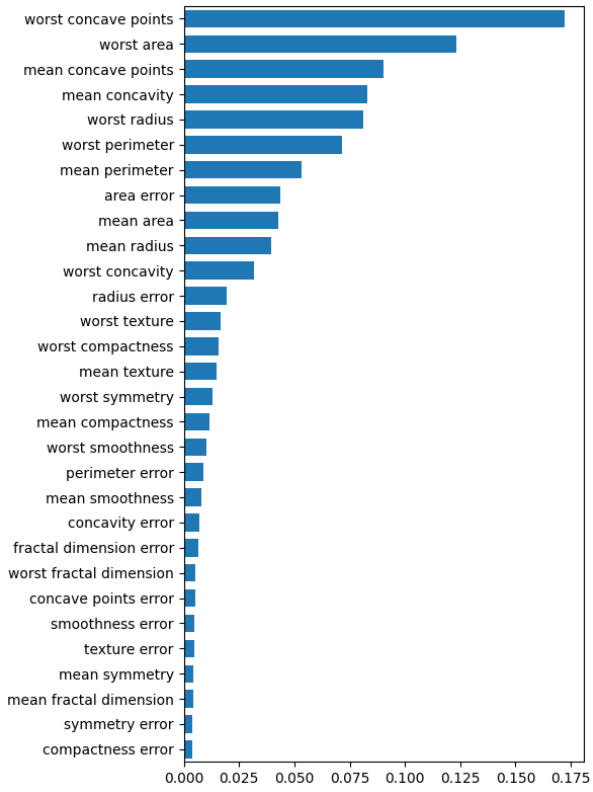
```
from collections import defaultdict

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import spearmanr
from scipy.cluster import hierarchy
from scipy.spatial.distance import squareform

from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split

data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
print("Accuracy on test data: {:.2f}".format(clf.score(X_test, y_test)))

result = permutation_importance(clf, X_train, y_train, n_repeats=10, random_state=42)
perm_sorted_idx = result.importances_mean.argsort()

tree_importance_sorted_idx = np.argsort(clf.feature_importances_)
tree_indices = np.arange(0, len(clf.feature_importances_)) + 0.5
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 8))
ax1.barh(tree_indices, clf.feature_importances_[tree_importance_sorted_idx], height=0.7)
ax1.set_yticks(tree_indices)
ax1.set_yticklabels(data.feature_names[tree_importance_sorted_idx])
ax1.set_ylim((0, len(clf.feature_importances_)))
ax2.boxplot(
    result.importances[perm_sorted_idx].T,
    vert=False,
    labels=data.feature_names[perm_sorted_idx],
)
fig.tight_layout()
plt.show()
```

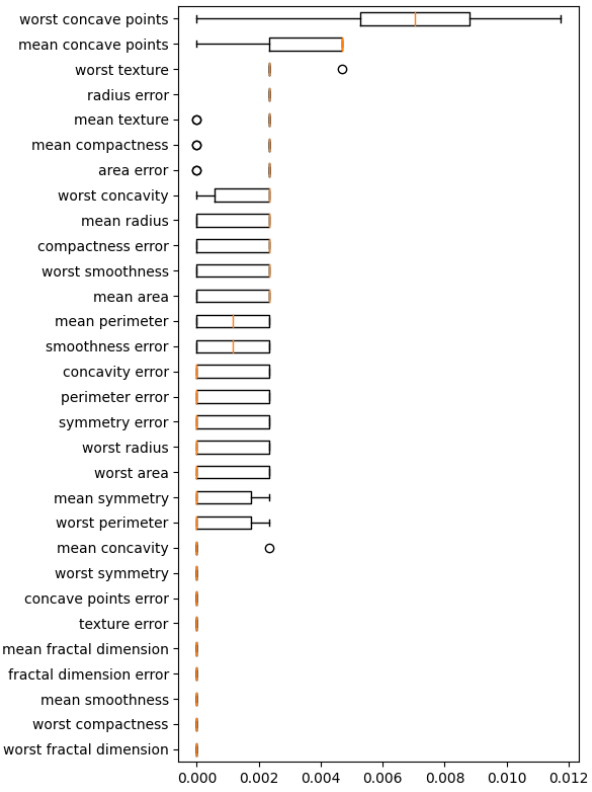
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 8))
corr = spearmanr(X).correlation

# Ensure the correlation matrix is symmetric
corr = (corr + corr.T) / 2
np.fill_diagonal(corr, 1)

# We convert the correlation matrix to a distance matrix before performing
# hierarchical clustering using Ward's linkage.
distance_matrix = 1 - np.abs(corr)
dist_linkage = hierarchy.ward(squareform(distance_matrix))
dendro = hierarchy.dendrogram(
    dist_linkage, labels=data.feature_names.tolist(), ax=ax1, leaf_rotation=90
)
dendro_idx = np.arange(0, len(dendro["ivl"]))

ax2.imshow(corr[dendro["leaves"], :][:, dendro["leaves"]])
ax2.set_xticks(dendro_idx)
ax2.set_yticks(dendro_idx)
ax2.set_xticklabels(dendro["ivl"], rotation="vertical")
ax2.set_yticklabels(dendro["ivl"])
fig.tight_layout()
plt.show()

cluster_ids = hierarchy.fcluster(dist_linkage, 1, criterion="distance")
cluster_id_to_feature_ids = defaultdict(list)
for idx, cluster_id in enumerate(cluster_ids):
    cluster_id_to_feature_ids[cluster_id].append(idx)
selected_features = [v[0] for v in cluster_id_to_feature_ids.values()]

X_train_sel = X_train[:, selected_features]
X_test_sel = X_test[:, selected_features]

clf_sel = RandomForestClassifier(n_estimators=100, random_state=42)
clf_sel.fit(X_train_sel, y_train)
print(
    "Accuracy on test data with features removed: {:.2f}".format(
        clf_sel.score(X_test_sel, y_test)
    )
)
```

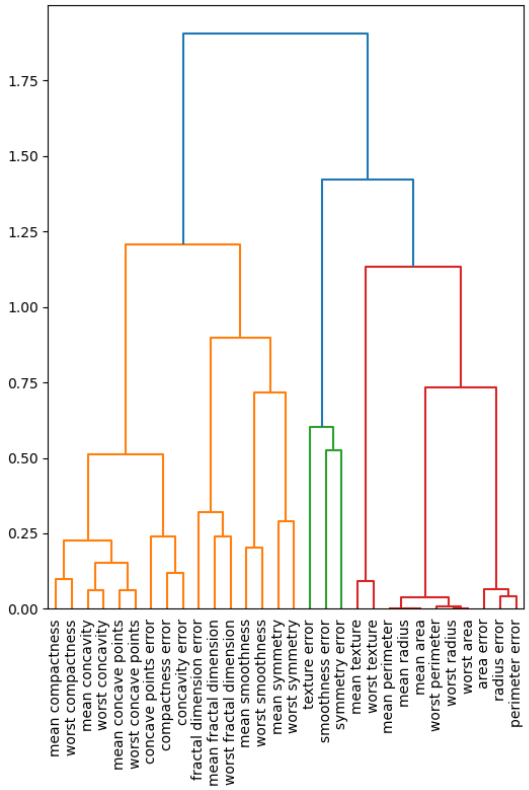
MUHAMMAD IQBAL DHARMAWAN | 1103194125

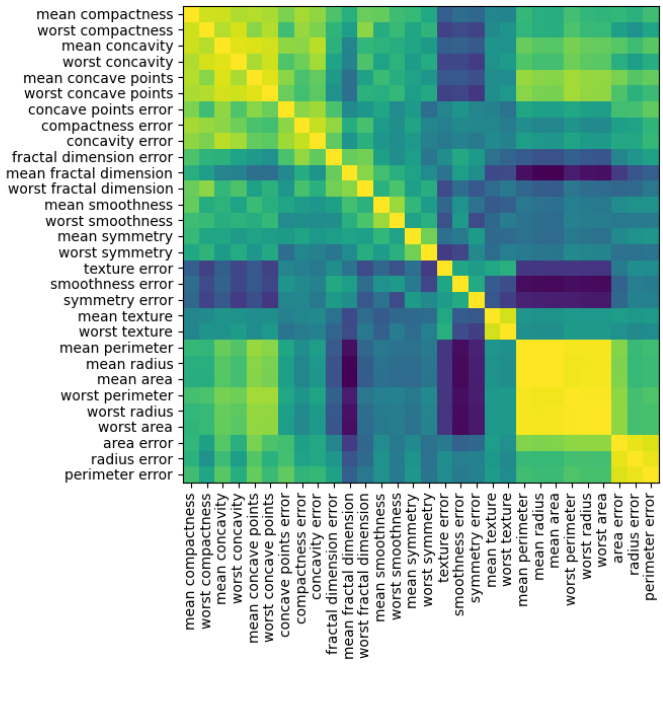
TECHNICAL REPORT FOR MACHINE LEARNING MID-TEARM EXAM

Output:









TECHNICAL REPORT FOR MACHINE LEARNING MID-TEARM EXAM

Self-Training

Self-Training is a semi-supervised learning approach that uses labeled and unlabeled data to improve the classification performance. It works by first training a classifier on the labeled data and then using that classifier to predict the labels of the unlabeled data. The predicted labels are then added to the labeled data, and the process is repeated until convergence. Self-Training can be useful when there is a limited amount of labeled data and a large amount of unlabeled data.

Advantages:

- Can improve the classification performance with limited labeled data
- Can handle unbalanced datasets
- can be used with any classifier

Disadvantages:

- Can be prone to error propagation
- Can lead to overconfidence in the classifier
- Requires careful selection of the threshold for adding predicted labels to the labeled data

Use Cases:

- Text classification
- Sentiment analysis
- Image recognition

Self training training with sklearn and google collab:

Source code:

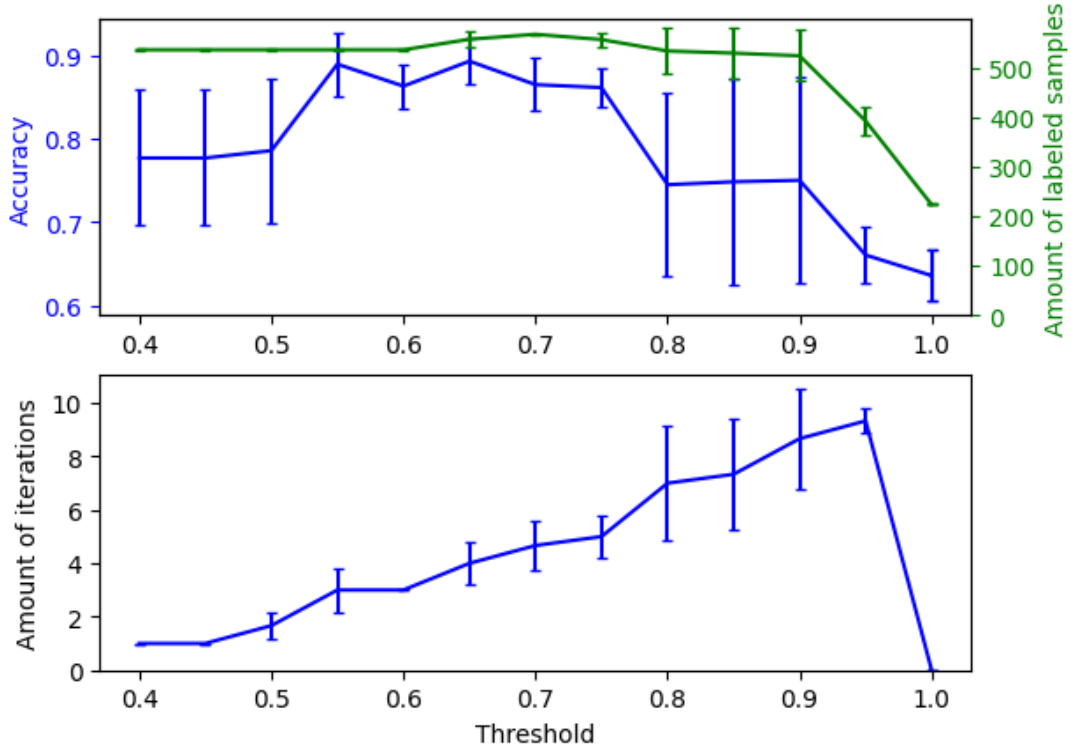
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.semi_supervised import SelfTrainingClassifier
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle

n_splits = 3

X, y = datasets.load_breast_cancer(return_X_y=True)
X, y = shuffle(X, y, random_state=42)
y_true = y.copy()
y[50:] = -1
total_samples = y.shape[0]

base_classifier = SVC(probability=True, gamma=0.001, random_state=42)

x_values = np.arange(0.4, 1.05, 0.05)
x_values = np.append(x_values, 0.99999)
scores = np.empty((x_values.shape[0], n_splits))
amount_labeled = np.empty((x_values.shape[0], n_splits))
amount_iterations = np.empty((x_values.shape[0], n_splits))

for i, threshold in enumerate(x_values):
    self_training_clf = SelfTrainingClassifier(base_classifier, threshold=threshold)

    # We need manual cross validation so that we don't treat -1 as a separate
    # class when computing accuracy
    skfolds = StratifiedKFold(n_splits=n_splits)
    for fold, (train_index, test_index) in enumerate(skfolds.split(X, y)):
        X_train = X[train_index]
        y_train = y[train_index]
        X_test = X[test_index]
        y_test = y[test_index]
        y_test_true = y_true[test_index]

        self_training_clf.fit(X_train, y_train)
```

MUHAMMAD IQBAL DHARMAWAN | 1103194125

TECHNICAL REPORT FOR MACHINE LEARNING MID-TEARM EXAM

```
        # The amount of labeled samples that at the end of fitting
        amount_labeled[i, fold] = (
            total_samples
            - np.unique(self_training_clf.labeled_iter_, return_counts=True)[1][0]
        )
        # The last iteration the classifier labeled a sample in
        amount_iterations[i, fold] = np.max(self_training_clf.labeled_iter_)

        y_pred = self_training_clf.predict(X_test)
        scores[i, fold] = accuracy_score(y_test_true, y_pred)


ax1 = plt.subplot(211)
ax1.errorbar(
    x_values, scores.mean(axis=1), yerr=scores.std(axis=1), capsize=2, color="b"
)
ax1.set_ylabel("Accuracy", color="b")
ax1.tick_params("y", colors="b")

ax2 = ax1.twinx()
ax2.errorbar(
    x_values,
    amount_labeled.mean(axis=1),
    yerr=amount_labeled.std(axis=1),
    capsize=2,
    color="g",
)
ax2.set_ylim(bottom=0)
ax2.set_ylabel("Amount of labeled samples", color="g")
ax2.tick_params("y", colors="g")

ax3 = plt.subplot(212, sharex=ax1)
ax3.errorbar(
    x_values,
    amount_iterations.mean(axis=1),
    yerr=amount_iterations.std(axis=1),
    capsize=2,
    color="b",
)
ax3.set_ylim(bottom=0)
ax3.set_ylabel("Amount of iterations")
ax3.set_xlabel("Threshold")

plt.show()
```

Output:



Conclusion

Decision Tree, Random Forest, and Self-Training are three commonly used algorithms in machine learning for classification tasks. Decision Tree is a simple and easy-to-understand algorithm that can handle both categorical and numerical data. Random Forest is a powerful algorithm that combines multiple Decision Trees to improve accuracy and reduce overfitting. Self-Training is a semi-supervised learning approach that can improve the classification performance with limited labeled data. Each algorithm has its own advantages and disadvantages, and the choice of algorithm depends on the specific problem and data at hand.