Assignment 8: Decision Trees

Machine Learning – University of Wrocław

Winter Semester 2025/2026

Note: Solutions should include Python code and brief comments explaining key implementation decisions. The goal of this assignment is to **discover** methods through experimentation – try to understand the intuition first before checking official sklearn implementations.

Task 1: WroBank – Stabilizing Unstable Classifiers (3 pts)

Scenario: The company "WroBank" is building a credit risk assessment system. A single decision tree gives different results with each training (instability problem). The client complains: "Yesterday the model said YES, today it says NO for the same application!" Your boss asks: "How can we get stable predictions without giving up trees?" Your task is to discover the stabilization method.

Data: Use synthetic credit data generated by `make_classification` with 1000 samples and 10 features.

Instructions:

(a) **Instability of a single tree:**

- Train 10 decision trees on the same data (different `random_state`: 0, 1, ..., 9)
- For each test point, calculate the variance of probability predictions across trees
- Visualize the variance distribution – how many points have high instability?
- Draw decision boundaries of several trees in 2D (use the first two features)

(b) **Prediction aggregation (Bootstrap Aggregating):**

- Implement bootstrap sampling: n samples with replacement from n original
- Train 50 trees, each on a different bootstrap sample
- For classification: apply majority voting
- Calculate and compare the prediction variance with point (a) – how much did it decrease?

(c) **Random feature subsets:**

- Modify the procedure: at each split, randomly select $\sqrt{p}$ features (where p = number of features)
- Implement the function `build_randomized_tree()` using the parameter `max_features='sqrt'`
- Compare trees with and without feature randomization – which are more "diverse"?

- Examine the correlation between tree predictions in both variants (Pearson on probabilities)

(d) **Final analysis:**

- Examine the effect of the number of trees: 10, 50, 100, 200, 500 – where does "saturation" occur?
- Implement OOB (Out-of-Bag) error: evaluate the model without a validation set
- Compare results with `sklearn.ensemble.RandomForestClassifier` – are they consistent?
- Explain mathematically why this method reduces variance (formula for variance of the mean)

Starter code:

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from collections import Counter
import matplotlib.pyplot as plt

def generate_credit_data(n_samples=1000, n_features=10, random_state=42)
    :
    """Generates synthetic credit data."""
    X, y = make_classification(
        n_samples=n_samples, n_features=n_features,
        n_informative=6, n_redundant=2, n_clusters_per_class=2,
        random_state=random_state
    )
    return X, y

def bootstrap_sample(X, y, random_state=None):
    """Draws a bootstrap sample (n samples with replacement from n)."""
    rng = np.random.RandomState(random_state)
    n = len(X)
    indices = rng.choice(n, size=n, replace=True)
    oob_indices = list(set(range(n)) - set(indices))
    return X[indices], y[indices], oob_indices

def train_single_trees(X_train, y_train, n_trees=10):
    """Trains n single trees with different seeds."""
    trees = []
    for seed in range(n_trees):
        tree = DecisionTreeClassifier(random_state=seed)
        tree.fit(X_train, y_train)
        trees.append(tree)
    return trees

def calculate_prediction_variance(trees, X):
    """Calculates prediction variance across trees."""
    # TODO: Collect predict_proba from all trees
    # TODO: Calculate variance for each point
    pass

def ensemble_predict(trees, X):
```

```
    """Aggregates predictions from multiple trees via majority voting.
       """
    # TODO: Collect predictions from all trees
    # TODO: For each point, return the most frequent class
    pass

def calculate_oob_error(X, y, trees, oob_indices_list):
    """Calculates OOB error without a validation set."""
    # TODO: For each point, use only trees that didn't see it
    # TODO: Vote and compare with the true label
    pass
```

Task 2: WroSecure – Intrusion Detection via Isolation (4 pts)

Scenario: The company "WroSecure" monitors network traffic looking for cyberattacks. Problem: anomalies are rare (1-5% of traffic), diverse, and hard to define. Classical methods (labeling, rules) don't work. Your colleague made a provocative observation: "Anomalies are *weird*, so it should be easy to ISOLATE them from the rest." Your task is to investigate this intuition.

Data: Synthetic network traffic data with normal patterns and anomalies (DDoS, port scan).

Instructions:

(a) **Random isolation trees:**

- Build a tree with completely random splits: random feature, random threshold in range [min, max]
- **NOTE:** Don't use labels! This is an unsupervised method
- Assign each point an "isolation depth" (path length to leaf)
- Visualize the depth histogram: normal vs anomalies – what do you see?

(b) **Geometric intuition:**

- Generate 2D data: normal cluster + a few anomalies on the edges
- Draw successive tree splits on the plot step by step
- Show that anomalies are isolated after 2-3 splits, normal points after 8-10
- Explain: "rare points = few neighbors = quick isolation"

(c) **Ensemble and anomaly score:**

- Build 100 random isolation trees (each on a subsample of 256 points)
- Average the path length $E[h(x)]$ for each point
- Normalize to anomaly score: $s(x, n) = 2^{-E[h(x)]/c(n)}$
- where $c(n) = 2H(n-1) - 2(n-1)/n$ (expected depth of BST for n elements)

(d) **Evaluation and comparison:**

- Compare with LOF (Local Outlier Factor) and One-Class SVM
- Calculate precision, recall, F1 on labeled data (test set)
- Examine the effect of: number of trees (50, 100, 200), max_samples (64, 128, 256)

- Compare with `sklearn.ensemble.IsolationForest` – is the implementation consistent?

Starter code:

```
import numpy as np
from scipy.special import digamma  # for H(n) = digamma(n+1) - digamma
    (1)

def generate_network_data(n_normal=950, n_anomaly=50, random_state=42):
    """Generates network traffic data with anomalies."""
    np.random.seed(random_state)
    # Normal traffic - 3 operational clusters
    normal = np.vstack([
        np.random.randn(n_normal//3, 4) * 0.5 + [50, 100, 0.8, 30],
        np.random.randn(n_normal//3, 4) * 0.5 + [55, 110, 0.75, 35],
        np.random.randn(n_normal//3, 4) * 0.5 + [48, 95, 0.85, 28],
    ])
    # Anomalies - various attack patterns
    anomalies = np.vstack([
        np.random.randn(n_anomaly//2, 4) * 0.3 + [200, 500, 0.1, 5],   #
            DDoS
        np.random.randn(n_anomaly//2, 4) * 0.3 + [10, 10, 0.99, 100],  #
            Port scan
    ])
    X = np.vstack([normal, anomalies])
    y = np.array([0]*len(normal) + [1]*len(anomalies))
    return X, y

class IsolationTree:
    """Single isolation tree."""
    def __init__(self, max_depth=None):
        self.max_depth = max_depth
        self.root = None

    def fit(self, X, current_depth=0):
        """Builds a tree with random splits (without labels!)."""
        n_samples, n_features = X.shape

        # Stopping condition
        if n_samples <= 1 or (self.max_depth and current_depth >= self.
            max_depth):
            return {'type': 'leaf', 'size': n_samples, 'depth':
                current_depth}

        # TODO: Randomly select feature and threshold
        # feature = np.random.randint(0, n_features)
        # min_val, max_val = X[:, feature].min(), X[:, feature].max()
        # threshold = np.random.uniform(min_val, max_val)

        # TODO: Split data and recursively build subtrees
        pass

    def path_length(self, x, node=None, current_depth=0):
        """Calculates the isolation depth of point x."""
        # TODO: Recursively traverse the tree until reaching a leaf
        pass
```

```python
def c_factor(n):
    """Calculates the expected BST depth for n elements."""
    if n <= 1:
        return 0
    return 2 * (np.log(n - 1) + 0.5772156649) - 2 * (n - 1) / n

def anomaly_score(path_lengths, n_samples):
    """Calculates the normalized anomaly score."""
    c = c_factor(n_samples)
    return 2 ** (-np.array(path_lengths) / c)
```

Task 3: WroGeo – Geometry of Decision Boundaries (3 pts)

Scenario: The company "WroGeo" classifies terrain types based on satellite data. They noticed that decision trees create "angular" terrain maps instead of smooth boundaries. The client asks: "Why does my forest look like pixel graphics from the 90s?" Your task is to understand the geometric limitations of trees and their consequences.

Data: Synthetic 2D data: half-moons (`make_moons`), circles (`make_circles`), spirals.

Instructions:

(a) **Visualization of decision boundaries:**

- Generate 2D data: `make_moons`, `make_circles`, spirals
- Train trees with depths: 1, 3, 5, 10, None (no limit)
- Draw decision boundaries for each depth (meshgrid + contourf)
- Describe observations: how does depth affect "smoothness"?

(b) **Why perpendicular to axes?:**

- Implement step-by-step visualization of tree construction
- Show that each split $x_i < t$ divides the space with a line perpendicular to axis i
- Draw a "split diagram" for a simple tree (depth 3)
- Compare with SVM (curved boundaries with RBF kernel) on the same data

(c) **How many splits are needed?:**

- For a circle of radius r: how many splits are needed to approximate with accuracy ϵ?
- Implement the function `count_splits_for_circle(radius, tolerance)`
- Plot: tolerance vs number of leaves – what kind of growth? (linear, quadratic, exponential?)
- Conclusions for real data with "circular" clusters

(d) **Oblique Decision Trees:**

- Trees with oblique splits: $a_1 x_1 + a_2 x_2 < t$
- Implement a simple variant (search through several coefficient combinations)
- Compare decision boundaries with axis-parallel tree
- Why doesn't sklearn implement this by default? (Hint: computational complexity)

Starter code:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons, make_circles
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

def make_spirals(n_samples=500, noise=0.1, random_state=42):
    """Generates spiral-shaped data."""
    np.random.seed(random_state)
    n = n_samples // 2
    theta = np.linspace(0, 4*np.pi, n)
    r = theta / (4*np.pi)
    x1 = r * np.cos(theta) + noise * np.random.randn(n)
    y1 = r * np.sin(theta) + noise * np.random.randn(n)
    x2 = -r * np.cos(theta) + noise * np.random.randn(n)
    y2 = -r * np.sin(theta) + noise * np.random.randn(n)
    X = np.vstack([np.column_stack([x1, y1]), np.column_stack([x2, y2])
        ])
    y = np.array([0]*n + [1]*n)
    return X, y

def plot_decision_boundary(model, X, y, ax, title=""):
    """Draws the decision boundary of a model."""
    h = 0.02  # grid step
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, alpha=0.4, cmap='RdYlBu')
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap='RdYlBu', edgecolors='black')
    ax.set_title(title)

def visualize_tree_splits(tree, X, y, ax):
    """Visualizes successive tree splits."""
    # TODO: Recursively draw split lines
    # Use tree.tree_.feature, tree.tree_.threshold
    pass

def count_splits_for_circle(radius, tolerance):
    """Counts how many splits are needed to approximate a circle."""
    # TODO: Simulate a tree approximating a circle
    pass
```

Good luck! Remember that the goal is **understanding**, not just implementation. Try to answer the question "why?" for each task.