VASILIS ODYSSEOS

DECISION TREES

ID3 Algorithm and Post-Pruning

Datasets Used

- Hepatitis dataset
  - 155 instances
  - 19 attributes
  - Classification of patient’s survival
- Horse Colic Dataset
  - 368 instances
  - 27 attributes
  - Classification of whether lesion was surgical or not
- Mushroom Dataset
  - 8124 instances
  - 22 attributes
  - Classification of whether mushroom is poisonous or edible

+others

```
def open_file(filename):
 df = pd.read_csv(filename)
 cont_attr = get_continuous(df)
 continuous(df, cont_attr, 4)
 df = df.sample(frac=1).reset_index(drop=True)
    training = df.sample(frac=0.8)
    rest = df.drop(training.index).reset_index(drop=True)
    training = training.reset_index(drop=True)
    testing = rest.sample(frac=0.5)
    validation = rest.drop(testing.index)
    testing = testing.reset_index(drop=True)
    validation = validation.reset_index(drop=True)
```

Importing data

- Import using Pandas dataframe
- Split continuous data into 4 bins after checking if numerical attributes have more than 4 unique values
- Randomly divide the data into training, testing, validation samples

```python
def decision_tree(df, tree=None, used=None):
    # Recursively builds a decision tree from a given dataframe until entropy(df)
    # is 0 or all attributes used

    truth_col = df.keys()[-1]

    if used is None:
        used = []
    # get best attribute based on information gain
    attr = find_best(df[df.columns.difference(used, sort=False)])
    used.append(attr)
    attributes = list(df)

    values = df[attr].unique()

    if tree is None:
        tree = {}
        tree[attr] = {}

    # loop through unique values of attr and create a branch for each value.
    # If entropy of sorting on value is 0, the branch is a lead
    for value in values:
        sub = df[df[attr] == value].reset_index(drop=True)

        if entropy(sub) == 0 or (len(attributes) - 1) == len(np.unique(used)):
            unique, counts = np.unique(sub[truth_col], return_counts=True)
            tree[attr][value] = unique[np.argmax(counts)]
        else:
            tree[attr][value] = decision_tree(sub, used=used)
    return tree
```

Decision Tree algorithm

- Use attribute with best information gain as the root of the tree
- For each possible value of the attribute, create a new branch by recursively calling the decision_tree function on the subset of the data with that value
- If the entropy of the subset is 0 or all attributes have been used, then the branch is a leaf and the tree stops growing

```python
def pruning(df, tree):
    # prunes a decision tree by finding all nodes whose children are all leaves,
    # then checking to see if simplifying the node to a leaf gives a better accuracy,
    # and removes the node that gives the greatest gain in accuracy
    # essentially tries to find the shortest tree without loss of accuracy

    nodes = []
    # test on current tree for benchmark
    default_accuracy = test_all(df, tree)
    max_accuracy = 0
    # save current tree as best possible tree (that we know of)
    best_tree = tree
    # get all nodes with all leaf children
    get_nodes(tree, nodes)

    # test if simplifying each node produces a better or equal accuracy than the benchmark
    for i in range(len(nodes)):
        tcopy = copy.deepcopy(tree)
        tcopy = simplify_node(tcopy, nodes[i])
        accuracy = test_all(df, tcopy)
        if accuracy >= max_accuracy:
            best_tree = copy.deepcopy(tcopy)
            max_accuracy = accuracy
    # recursively prune the tree if the accuracy of the pruned tree is greater or equal
    # to the benchmark, otherwise return the current tree
    if (max_accuracy >= default_accuracy):
        return pruning(df, best_tree)
    else:
        return tree
```

Post-pruning algorithm

- Get all nodes that have all-leaf-children
- For each node, test if removing it yields a greater or equal accuracy on the validation data
- Remove the node with the greatest increase in accuracy
- Recursively prune the tree until accuracy drops

```python
def test(hypothesis, tree):
    # test a hypothesis by traversing the tree and comparing the leaf
    # of the tree to the label of the hypothesis

    if type(tree) is dict:
        try:
            return test(hypothesis, tree[list(tree)[0]][hypothesis[list(tree)[0]]])
        except:
            print("Label found in test set that does not exist in the decision tree.")
            return 0
    else:
        if(hypothesis[-1] == tree):
            return 1
        else:
            return 0


def test_all(df, tree):
    # tests all hypotheses in the set agains the tree

    count = 0
    for i in range(len(df[df.keys()[-1]])):
        count += test(df.loc[i, :], tree)
    return count / len(df[df.keys()[-1]])
```

Testing the trees

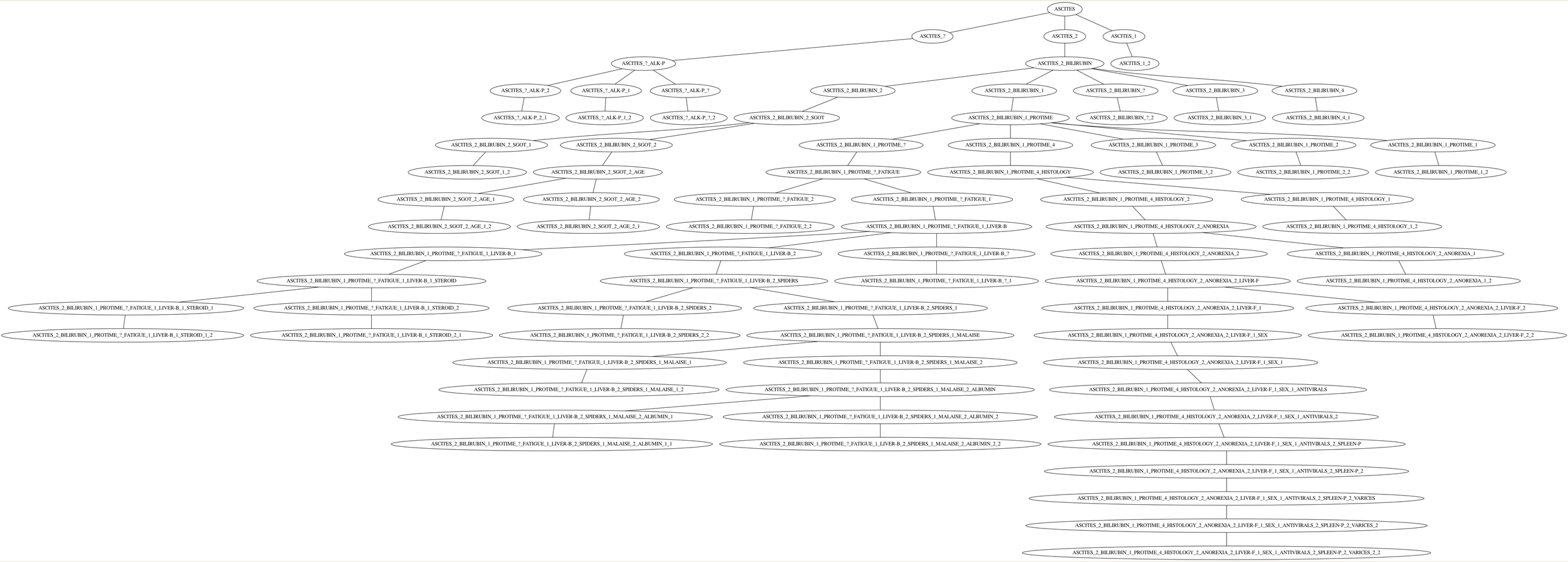
- For each instance in the given dataframe
  - Traverse the tree using the values in the instance
  - Return 1 if the classification given by the tree matches that of the instance
  - Return 0 if the classification given by the tree doesn't match, or if a certain attribute value is missing from the tree

Results

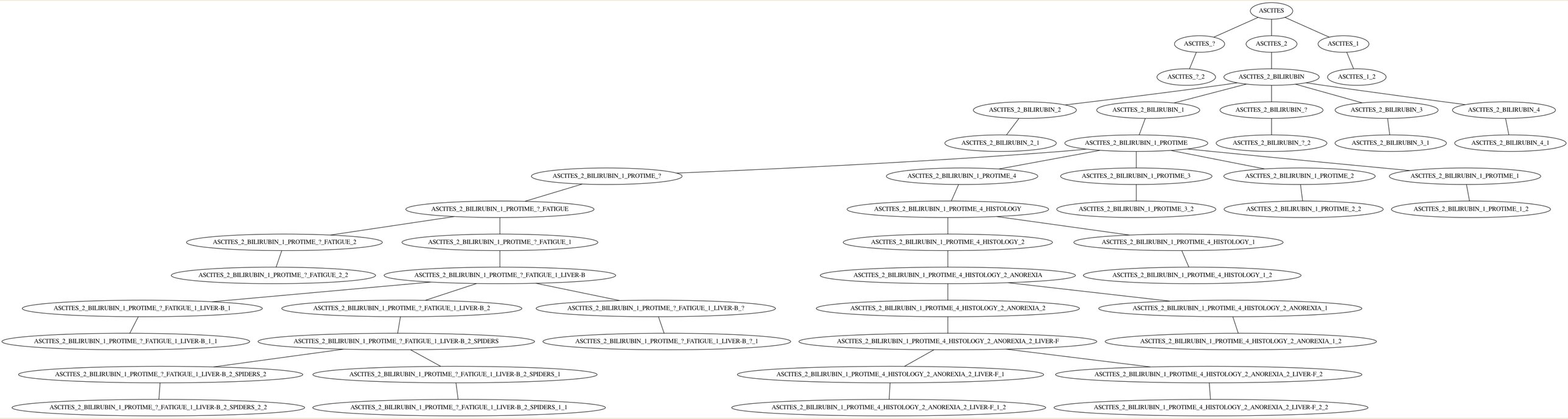
Hepatitis Dataset

- Testing on training data:
  - Original Accuracy: 0.8467741935483871
  - Pruned Accuracy: 0.75
- Testing on test data:
  - Original Accuracy: 0.8125
  - Pruned Accuracy: 0.8125
- Testing on validation data:
  - Original Accuracy: 0.6666666666666
  - Pruned Accuracy: 0.8

Original Tree



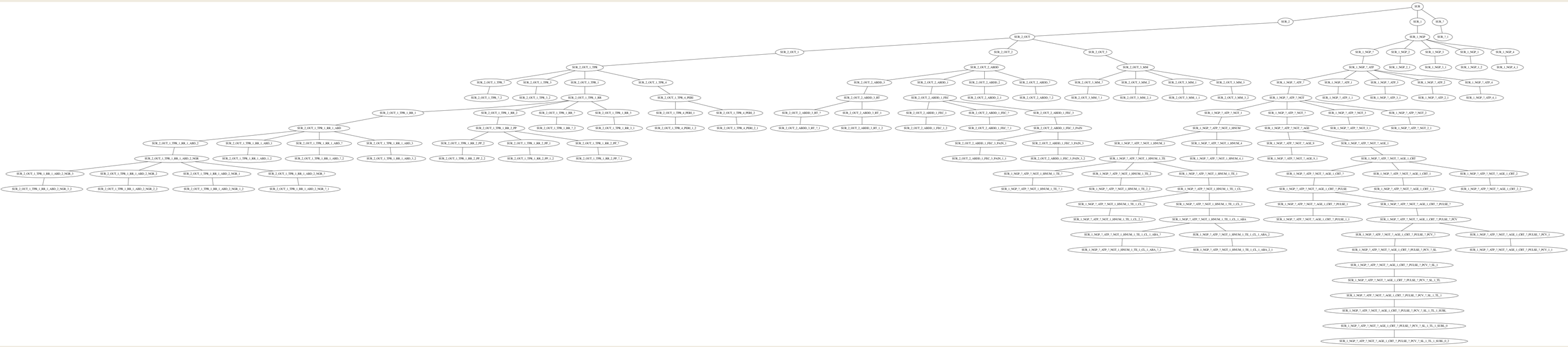
Pruned Tree


ASCITES
ASCITES_?
ASCITES_2
ASCITES_1
ASCITES_?_2
ASCITES_2_BILIRUBIN
ASCITES_1_2
ASCITES_2_BILIRUBIN_2
ASCITES_2_BILIRUBIN_1
ASCITES_2_BILIRUBIN_?
ASCITES_2_BILIRUBIN_3
ASCITES_2_BILIRUBIN_4
ASCITES_2_BILIRUBIN_2_1
ASCITES_2_BILIRUBIN_1_PROTIME
ASCITES_2_BILIRUBIN_?_2
ASCITES_2_BILIRUBIN_3_1
ASCITES_2_BILIRUBIN_4_1
ASCITES_2_BILIRUBIN_1_PROTIME_?
ASCITES_2_BILIRUBIN_1_PROTIME_4
ASCITES_2_BILIRUBIN_1_PROTIME_3
ASCITES_2_BILIRUBIN_1_PROTIME_2
ASCITES_2_BILIRUBIN_1_PROTIME_1
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY
ASCITES_2_BILIRUBIN_1_PROTIME_3_2
ASCITES_2_BILIRUBIN_1_PROTIME_2_2
ASCITES_2_BILIRUBIN_1_PROTIME_1_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_1
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_2_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_1_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_1
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_?
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_2
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_1
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_1_1
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_2_SPIDERS
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_?_1
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_2_LIVER-F
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_1_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_2_SPIDERS_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_2_SPIDERS_1
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_2_LIVER-F_1
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_2_LIVER-F_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_2_SPIDERS_2_2
ASCITES_2_BILIRUBIN_1_PROTIME_?_FATIGUE_1_LIVER-B_2_SPIDERS_1_1
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_2_LIVER-F_1_2
ASCITES_2_BILIRUBIN_1_PROTIME_4_HISTOLOGY_2_ANOREXIA_2_LIVER-F_2_2

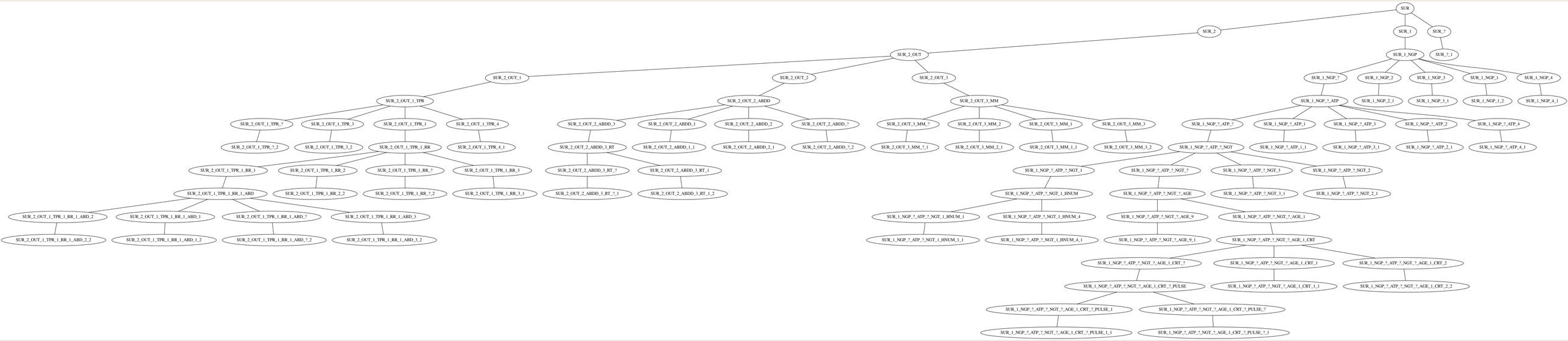
Horse Colic Dataset

- Testing on training data:
  - Original Accuracy: 0.8916666666666667
  - Pruned Accuracy: 0.8583333333333333
- Testing on test data:
  - Original Accuracy: 0.8
  - Pruned Accuracy: 0.8333333333333334
- Testing on validation data:
  - Original Accuracy: 0.7666666666666667
  - Pruned Accuracy: 0.8333333333333334

Original Tree



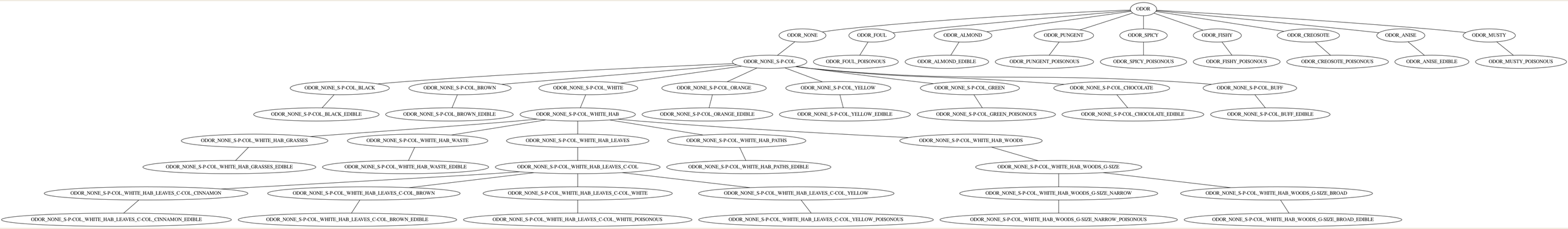
Pruned Tree


SUR
SUR_2
SUR_1
SUR_?
SUR_2_OUT
SUR_1_NGP
SUR_?_1
SUR_2_OUT_1
SUR_2_OUT_2
SUR_2_OUT_3
SUR_1_NGP_?
SUR_1_NGP_2
SUR_1_NGP_3
SUR_1_NGP_1
SUR_1_NGP_4
SUR_2_OUT_1_TPR
SUR_2_OUT_2_ABDD
SUR_2_OUT_3_MM
SUR_1_NGP_?_ATP
SUR_1_NGP_2_1
SUR_1_NGP_3_1
SUR_1_NGP_1_2
SUR_1_NGP_4_1
SUR_2_OUT_1_TPR_?
SUR_2_OUT_1_TPR_3
SUR_2_OUT_1_TPR_1
SUR_2_OUT_1_TPR_4
SUR_2_OUT_2_ABDD_3
SUR_2_OUT_2_ABDD_1
SUR_2_OUT_2_ABDD_2
SUR_2_OUT_2_ABDD_?
SUR_2_OUT_3_MM_?
SUR_2_OUT_3_MM_2
SUR_2_OUT_3_MM_1
SUR_2_OUT_3_MM_3
SUR_1_NGP_?_ATP_?
SUR_1_NGP_?_ATP_1
SUR_1_NGP_?_ATP_3
SUR_1_NGP_?_ATP_2
SUR_1_NGP_?_ATP_4
SUR_2_OUT_1_TPR_?_2
SUR_2_OUT_1_TPR_3_2
SUR_2_OUT_1_TPR_1_RR
SUR_2_OUT_1_TPR_4_1
SUR_2_OUT_2_ABDD_3_RT
SUR_2_OUT_2_ABDD_1_1
SUR_2_OUT_2_ABDD_2_1
SUR_2_OUT_2_ABDD_?_2
SUR_2_OUT_3_MM_?_1
SUR_2_OUT_3_MM_2_1
SUR_2_OUT_3_MM_1_1
SUR_2_OUT_3_MM_3_2
SUR_1_NGP_?_ATP_?_NGT
SUR_1_NGP_?_ATP_1_1
SUR_1_NGP_?_ATP_3_1
SUR_1_NGP_?_ATP_2_1
SUR_1_NGP_?_ATP_4_1
SUR_2_OUT_1_TPR_1_RR_1
SUR_2_OUT_1_TPR_1_RR_2
SUR_2_OUT_1_TPR_1_RR_?
SUR_2_OUT_1_TPR_1_RR_3
SUR_2_OUT_2_ABDD_3_RT_?
SUR_2_OUT_2_ABDD_3_RT_1
SUR_1_NGP_?_ATP_?_NGT_1
SUR_1_NGP_?_ATP_?_NGT_?
SUR_1_NGP_?_ATP_?_NGT_3
SUR_1_NGP_?_ATP_?_NGT_2
SUR_2_OUT_1_TPR_1_RR_1_ABD
SUR_2_OUT_1_TPR_1_RR_2_2
SUR_2_OUT_1_TPR_1_RR_?_2
SUR_2_OUT_1_TPR_1_RR_3_1
SUR_2_OUT_2_ABDD_3_RT_?_1
SUR_2_OUT_2_ABDD_3_RT_1_2
SUR_1_NGP_?_ATP_?_NGT_1_HNUM
SUR_1_NGP_?_ATP_?_NGT_?_AGE
SUR_1_NGP_?_ATP_?_NGT_3_1
SUR_1_NGP_?_ATP_?_NGT_2_1
SUR_2_OUT_1_TPR_1_RR_1_ABD_2
SUR_2_OUT_1_TPR_1_RR_1_ABD_1
SUR_2_OUT_1_TPR_1_RR_1_ABD_?
SUR_2_OUT_1_TPR_1_RR_1_ABD_3
SUR_1_NGP_?_ATP_?_NGT_1_HNUM_1
SUR_1_NGP_?_ATP_?_NGT_1_HNUM_4
SUR_1_NGP_?_ATP_?_NGT_?_AGE_9
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1
SUR_2_OUT_1_TPR_1_RR_1_ABD_2_2
SUR_2_OUT_1_TPR_1_RR_1_ABD_1_2
SUR_2_OUT_1_TPR_1_RR_1_ABD_?_2
SUR_2_OUT_1_TPR_1_RR_1_ABD_3_2
SUR_1_NGP_?_ATP_?_NGT_1_HNUM_1_1
SUR_1_NGP_?_ATP_?_NGT_1_HNUM_4_1
SUR_1_NGP_?_ATP_?_NGT_?_AGE_9_1
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_?
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_1
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_2
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_?_PULSE
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_1_1
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_2_2
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_?_PULSE_1
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_?_PULSE_?
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_?_PULSE_1_1
SUR_1_NGP_?_ATP_?_NGT_?_AGE_1_CRT_?_PULSE_?_1

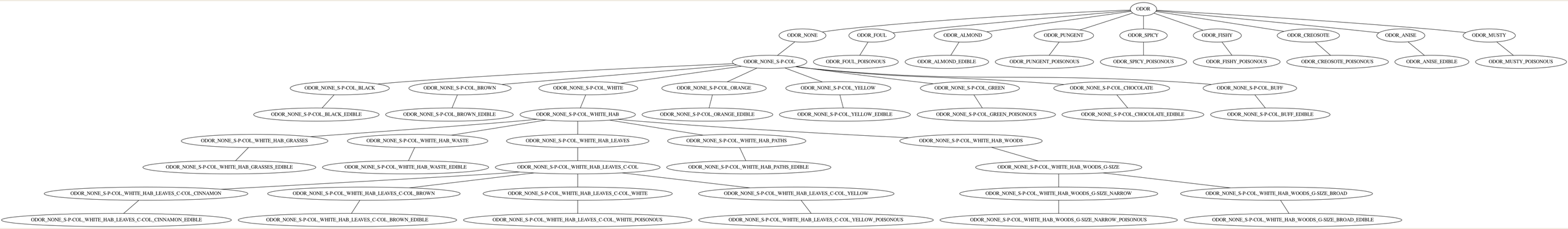
Mushroom Dataset

- Testing on training data:
  - Original Accuracy: 1.0
  - Pruned Accuracy: 1.0
- Testing on test data:
  - Original Accuracy: 1.0
  - Pruned Accuracy: 1.0
- Testing on validation data:
  - Original Accuracy: 1.0
  - Pruned Accuracy: 1.0

Original Tree



Pruned Tree


ODOR
ODOR_NONE
ODOR_FOUL
ODOR_ALMOND
ODOR_PUNGENT
ODOR_SPICY
ODOR_FISHY
ODOR_CREOSOTE
ODOR_ANISE
ODOR_MUSTY
ODOR_NONE_S-P-COL
ODOR_FOUL_POISONOUS
ODOR_ALMOND_EDIBLE
ODOR_PUNGENT_POISONOUS
ODOR_SPICY_POISONOUS
ODOR_FISHY_POISONOUS
ODOR_CREOSOTE_POISONOUS
ODOR_ANISE_EDIBLE
ODOR_MUSTY_POISONOUS
ODOR_NONE_S-P-COL_BLACK
ODOR_NONE_S-P-COL_BROWN
ODOR_NONE_S-P-COL_WHITE
ODOR_NONE_S-P-COL_ORANGE
ODOR_NONE_S-P-COL_YELLOW
ODOR_NONE_S-P-COL_GREEN
ODOR_NONE_S-P-COL_CHOCOLATE
ODOR_NONE_S-P-COL_BUFF
ODOR_NONE_S-P-COL_BLACK_EDIBLE
ODOR_NONE_S-P-COL_BROWN_EDIBLE
ODOR_NONE_S-P-COL_WHITE_HAB
ODOR_NONE_S-P-COL_ORANGE_EDIBLE
ODOR_NONE_S-P-COL_YELLOW_EDIBLE
ODOR_NONE_S-P-COL_GREEN_POISONOUS
ODOR_NONE_S-P-COL_CHOCOLATE_EDIBLE
ODOR_NONE_S-P-COL_BUFF_EDIBLE
ODOR_NONE_S-P-COL_WHITE_HAB_GRASSES
ODOR_NONE_S-P-COL_WHITE_HAB_WASTE
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES
ODOR_NONE_S-P-COL_WHITE_HAB_PATHS
ODOR_NONE_S-P-COL_WHITE_HAB_WOODS
ODOR_NONE_S-P-COL_WHITE_HAB_GRASSES_EDIBLE
ODOR_NONE_S-P-COL_WHITE_HAB_WASTE_EDIBLE
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL
ODOR_NONE_S-P-COL_WHITE_HAB_PATHS_EDIBLE
ODOR_NONE_S-P-COL_WHITE_HAB_WOODS_G-SIZE
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_CINNAMON
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_BROWN
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_WHITE
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_YELLOW
ODOR_NONE_S-P-COL_WHITE_HAB_WOODS_G-SIZE_NARROW
ODOR_NONE_S-P-COL_WHITE_HAB_WOODS_G-SIZE_BROAD
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_CINNAMON_EDIBLE
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_BROWN_EDIBLE
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_WHITE_POISONOUS
ODOR_NONE_S-P-COL_WHITE_HAB_LEAVES_C-COL_YELLOW_POISONOUS
ODOR_NONE_S-P-COL_WHITE_HAB_WOODS_G-SIZE_NARROW_POISONOUS
ODOR_NONE_S-P-COL_WHITE_HAB_WOODS_G-SIZE_BROAD_EDIBLE