Homework 5 - CS348 Spring 2019

Description - In this assignment, you will run and analyze binary classification using decision trees.

Getting Started - You should complete the assignment using your own installation of Python 3 and the packages numpy, pandas, matplotlib, and seaborn. Download the assignment from Moodle and unzip the file. This will create a directory with this file, 'HW05.ipynb'.

You will also need to install the pydotplus library by running `pip install pydotplus` or `conda install pydotplus` in the terminal.

Deliverables - The assignment has a single deliverable: this jupyter notebook file saved as a pdf. Please answer all coding and writing questions in the body of this file. Once all of the answers are complete, download the file by navigating the following menus: File -> Download as -> PDF via LaTeX. Submit the downloaded pdf file on gradescope. Alternatively, you can save the file as a pdf via the following: File -> Print Preview -> Print as pdf.

Data Sets - In this assignment, you will a single dataset from the sci-kit learn repository on breast cancer.

Academic Honesty Statement - Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is considered cheating. Posting your code to public repositories such as GitHub is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

This academic honesty statement does not restrict you from reading official documentation or using other web resources for understanding the syntax of python, related data science libraries, or properties of distributions.

In [38]:
```
# Do not import any other libraries other than those listed here.
import pydotplus
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as skl
import seaborn as sns

from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.tree._tree import TREE_LEAF
from sklearn.externals.six import StringIO
from sklearn.datasets import load_breast_cancer
from IPython.display import Image
```

Problem 1 - Decision Tree Classifiers

In this problem you'll use a Decision Tree model to classify whether a tumor is benign or malignant in a breast cancer dataset.

In [39]:
```
# Loading data.
data = load_breast_cancer()
```

In [40]:
```
x = data['data']
y = data['target']

x_train = x[:500]
y_train = y[:500]
x_val = x[500:]
y_val = y[500:]
```

In [41]:
```
def get_num_leaves(model):
    return sum(model.tree_.children_left < 0)

def print_decision_tree(model):
    # Taken from https://medium.com/@rnbrown/creating-and-visualizing-de
cision-trees-with-python-f8e8fa394176
    dot_data = StringIO()

    export_graphviz(model, out_file=dot_data,
                    filled=True, rounded=True,
                    special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return Image(graph.create_png())

# Modifed from David Dale's solution at https://stackoverflow.com/questi
ons/49428469/pruning-decision-trees.
def prune_tree(model, threshold):
    prune_index(model.tree_, 0, threshold)
    return model

def prune_index(inner_tree, index, threshold):
    # Recursively call prune_index until you hit the leaf nodes.
    if inner_tree.children_left[index] != TREE_LEAF:
        prune_index(inner_tree, inner_tree.children_left[index], thresho
ld)
        prune_index(inner_tree, inner_tree.children_right[index], thresh
old)
    if inner_tree.value[index].sum() < threshold:
        # turn node into a leaf by "unlinking" its children
        inner_tree.children_left[index] = TREE_LEAF
        inner_tree.children_right[index] = TREE_LEAF
```

Part 1 (10 points)

Using sci-kit learn's `DecisionTreeClassifier` class with `random_state=0` , fit a model between features `x_train` and targets `y_train` . Use the function `print_decision_tree(model)` to visually inspect the trained decision tree model.

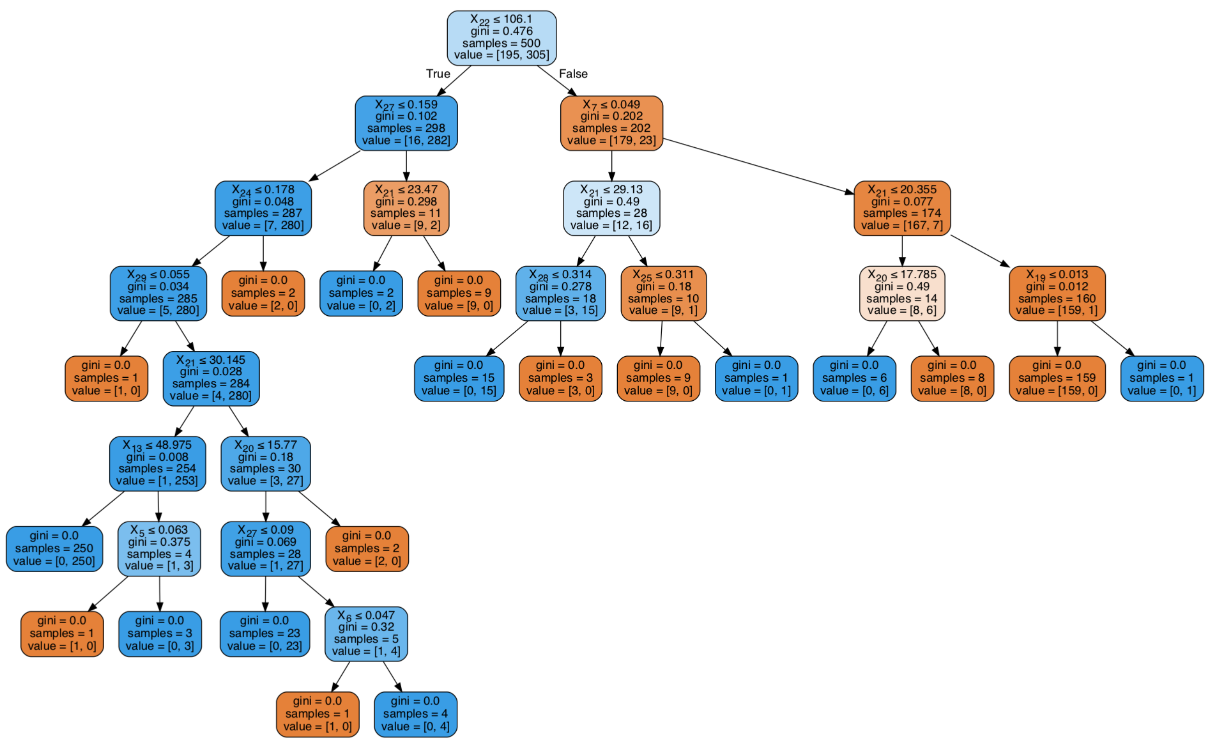
Using only the printed decision tree, evaluate the following sample probabilities.

```
P(y_train=1)
P(y_train=1|X22>106.1)
P(y_train=0|X22>106.1, X7>0.049)
```

In [42]:

```
# Part 1 Solution

# --- write code here ---
clf = DecisionTreeClassifier(random_state=0)
clf = clf.fit(x_train, y_train)

print_decision_tree(clf)
```

Out[42]:



Part 1 Written Response

P(y_train=1)=305/500=61/100

P(y_train=1|X22>106.1)=23/202

P(y_train=0|X22>106.1, X7>0.049)=167/174

Part 2 (20 points)

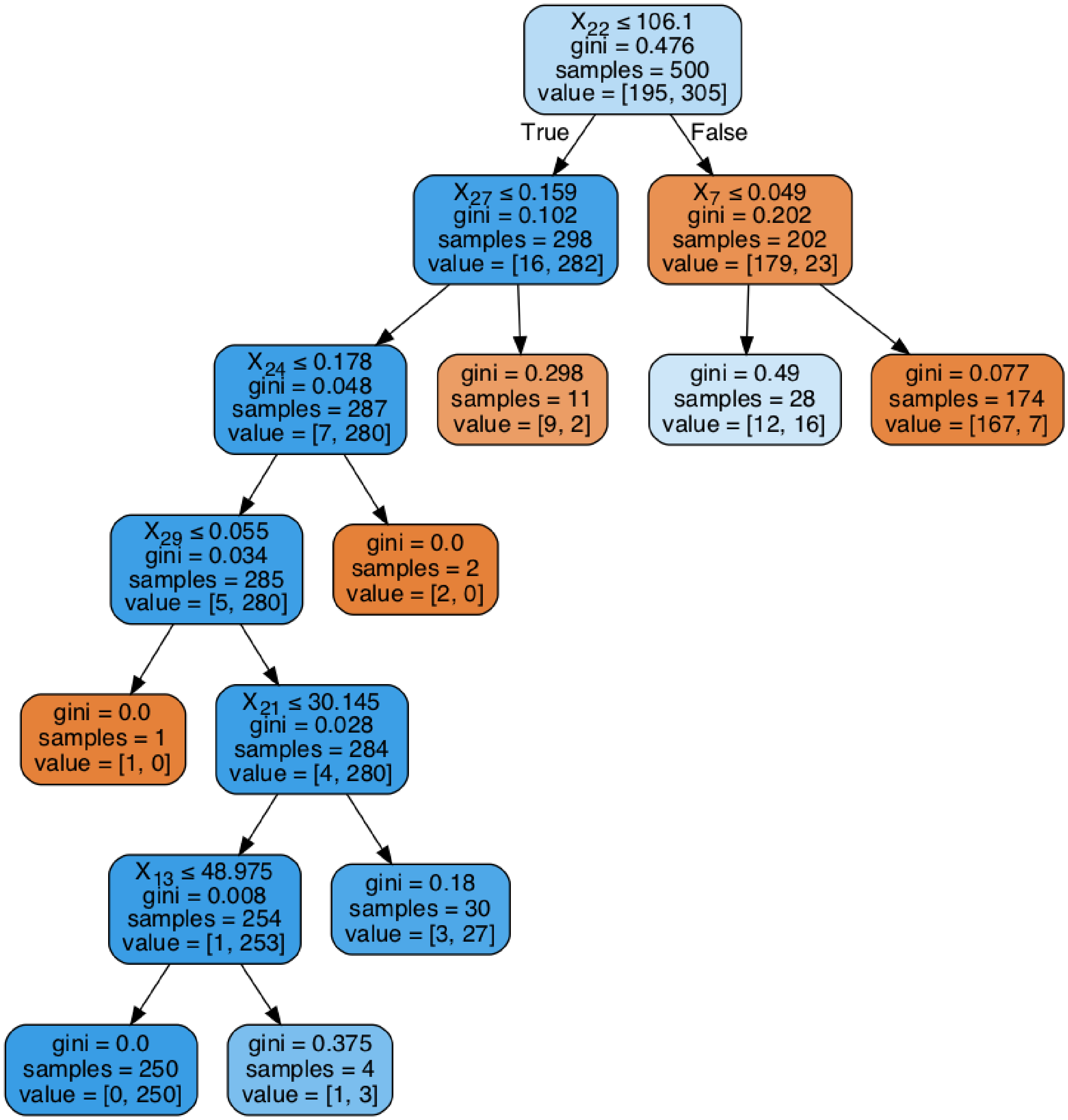
Again using sci-kit learn's `DecisionTreeClassifier` class with `random_state=0` , fit a model between features `x_train` and targets `y_train` . Use the function `prune_tree` with a threshold of `200` to prune the trained decision tree model. Use the function `print_decision_tree(model)` to visually inspect the pruned decision tree model.

Using only the two printed decision trees, describe an instance of `x` such that the model you trained in part 1 would predict `y=0` and the pruned model would predict class `y=1` . Be specific about variable values.

Note: The function `prune_tree(model, threshold)` takes as input a fully trained decision tree model and returns a modified decision tree model where every decision node with `samples < threshold` is converted into a leaf node.

In [43]:
```
# Part 2 Solution

# --- write code here ---
clf = DecisionTreeClassifier(random_state=0)
clf = clf.fit(x_train, y_train)
prune_tree(clf, 200)
print_decision_tree(clf)
```

Out[43]:



Part 2 Written Response

If X22 <= 106.1 & X27 <= 0.159 & X24 <= 0.178 & X29 > 0.055 & X21 > 30.145 & X20 > 15.77, model in part1 would predict y=0, model in part2 would predict y=1.

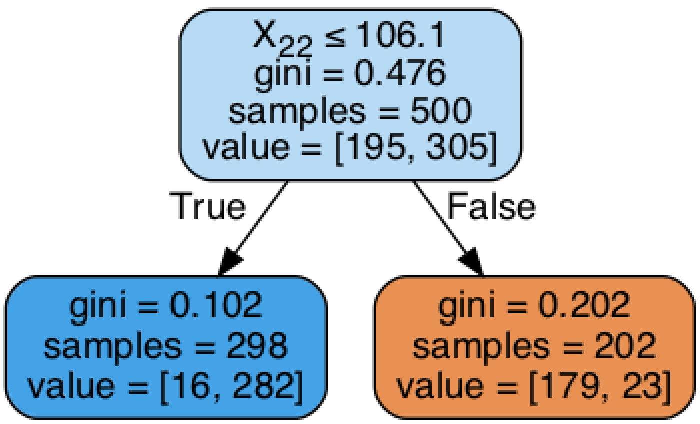
Part 3 (30 points)

Again using sci-kit learn's `DecisionTreeClassifier` class with `random_state=0` and `max_depth=1` , fit a model between features `x_train` and targets `y_train` . Use the function `print_decision_tree(model)` to visually inspect the pruned decision tree model.

Using only the printed decision trees, describe an instance of `x` such that the models you trained in part 1 and part 2 would both predict `y=0` , but the model with `max_depth=1` would predict `y=1` .

In [44]:
```
# Part 3 Solution

# --- write code here ---
clf = DecisionTreeClassifier(random_state=0, max_depth=1)
clf = clf.fit(x_train, y_train)
print_decision_tree(clf)
```

Out[44]:



Part 3 Written Response

If X22 <= 106.1 & X27 > 0.159, models in part1 and part2 would predict y=0, but the model in part3 would predict y=1.

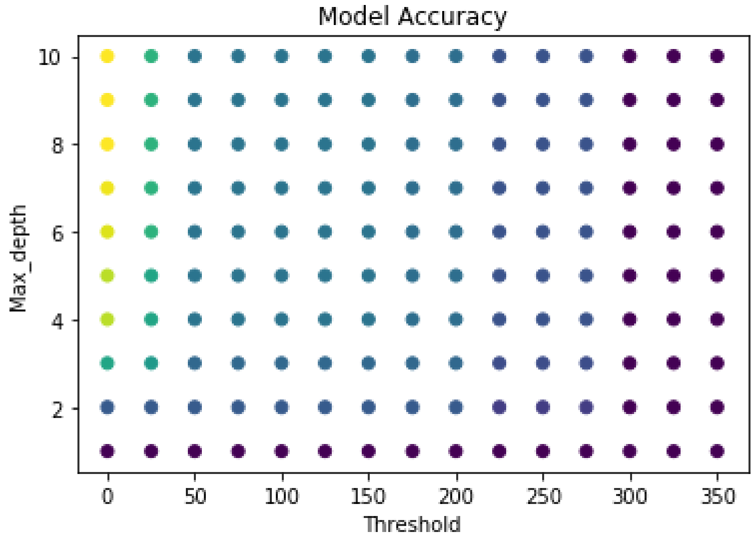
Part 4 (20 points)

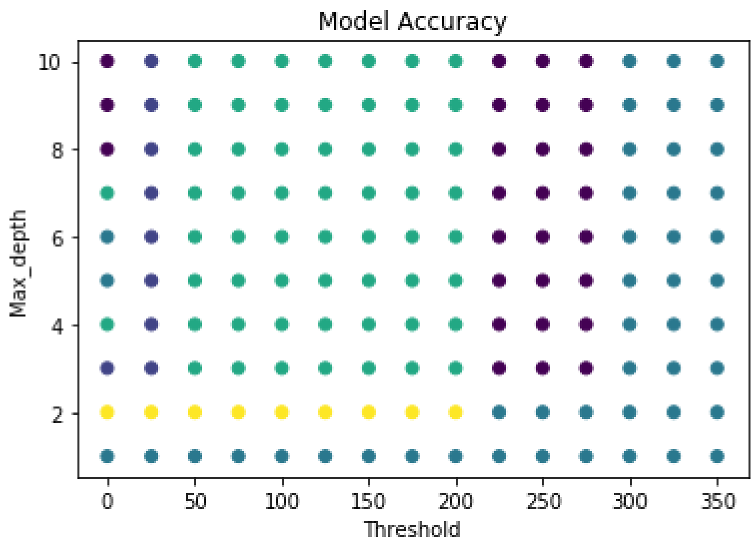
For every combination of `max_depth` between 1 and 10 and `threshold` between 0 and 350 in increments of 25, train a decision tree classifer on `x_train` and `y_train` and then prune the trained using the `prune_tree` function. Create a scatterplot with `threshold` on the horizontal axis and `max_depth` on the vertical axis, using color to show the classification accuracy of each point. Make 2 scatterplots using this same format, one with training accuracies and the other with validation accuracies.

Based on these scatterplots, which values should we select for `max_depth` and `threshold` ? If there are multiple models with comparable accuracies, describe other criteria we may want to consider when selecting a decision tree model.

Hint: You may find it easier to use `plt.scatter` or `ax.scatter` than `sns.scatterplot` for this problem.

In [55]:
```
# Part 4 Solution

# --- write code here ---

def scatterFunc(x_train, y_train, x_test, y_test):
    depth = []
    threshold = []
    train_acc = []
    val_acc = []
    for x in range(1, 11, 1):
        for y in range(0, 375, 25):
            depth.append(x)
            threshold.append(y)
            clf = DecisionTreeClassifier(random_state=0, max_depth=x)
            clf = clf.fit(x_train, y_train)
            prune_tree(clf, y)
            sc1 = clf.score(x_train, y_train)
            sc2 = clf.score(x_test, y_test)
            train_acc.append(sc1)
            val_acc.append(sc2)
    return depth, threshold, train_acc, val_acc

plt.figure()
d1, t1, a1, a2 = scatterFunc(x_train, y_train, x_val, y_val)
plt.scatter(t1, d1, c=a1)
plt.xlabel("Threshold")
plt.ylabel("Max_depth")
plt.title("Model Accuracy")
plt.show()

plt.figure()
plt.scatter(t1, d1, c=a2)
plt.xlabel("Threshold")
plt.ylabel("Max_depth")
plt.title("Model Accuracy")
plt.show()
```





Part 4 Written Response

Based on the scatterplots, as yellow dots indicate higher classification accuracy, we should select 2 for max_depth and 0-200 for threshold.

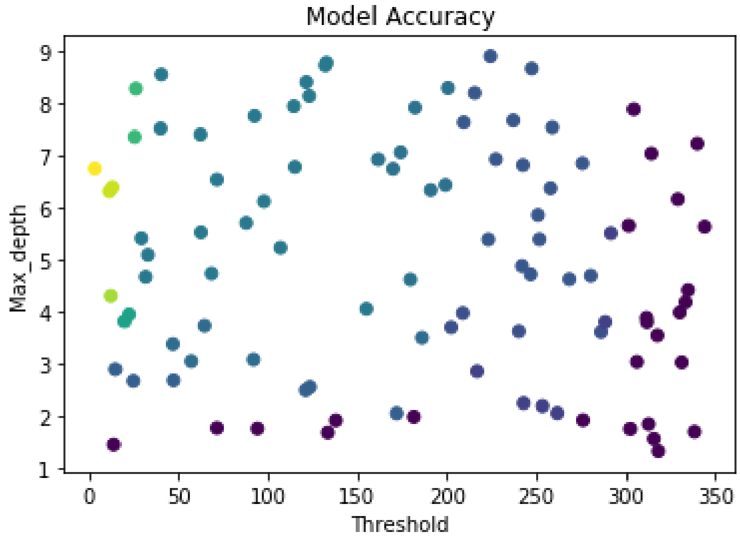
When there are multiple models with comparable accuracies, we should consider having the tree with low variance, which means keeping the depth low. At the meantime, we should present more detail about the tree, thus the threshold should be low. So the criteria will be the tradeoff between the low variance and more detail.

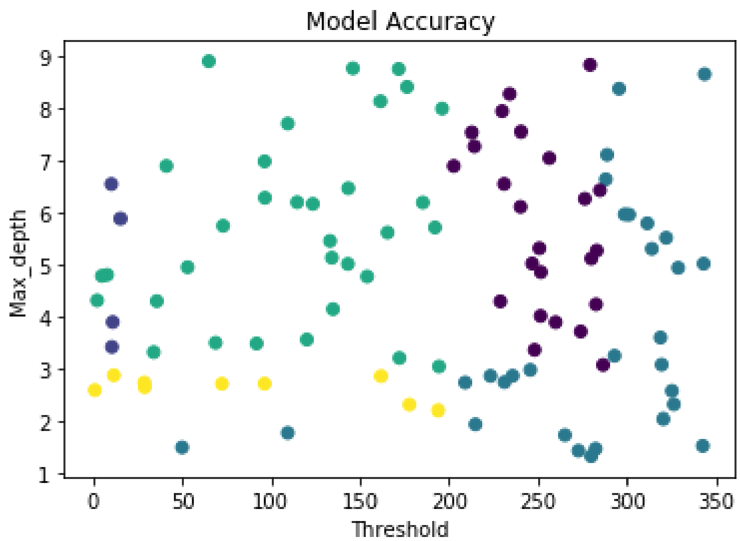
Part 5 (20 Points)

In this problem you'll explore an alternative hyperparameter search strategy, random search. For each of 100 iterations train a decision tree classifier on `x_train` and `y_train` using a `max_depth` sampled from a discrete uniform distribution between 1 and 9. Then prune the trained model using a `threshold` sampled from a uniform distribution between 0 and 350. Construct 2 plots identical to the plots you produced in part 4 using these new randomly sampled hyperparameters and models.

Do the results of this random search change your selection from part 4? If not, describe a setting in which random search would be preferable to grid search.

In [66]:
```
# Part 5 Solution

# --- write code here ---
def scatterFunc(x_train, y_train, x_test, y_test):
    depth = []
    threshold = []
    train_acc = []
    val_acc = []
    for iteration in range(0, 100):
        dep=np.random.uniform(1,9)
        depth.append(dep)
        thres=np.random.uniform(0,350)
        threshold.append(thres)
        clf = DecisionTreeClassifier(random_state=0, max_depth=dep)
        clf = clf.fit(x_train, y_train)
        prune_tree(clf, thres)
        sc1 = clf.score(x_train, y_train)
        sc2 = clf.score(x_test, y_test)
        train_acc.append(sc1)
        val_acc.append(sc2)
    return depth, threshold, train_acc, val_acc

plt.figure()
d1, t1, a1, a2 = scatterFunc(x_train, y_train, x_val, y_val)
plt.scatter(t1, d1, c=a1)
plt.xlabel("Threshold")
plt.ylabel("Max_depth")
plt.title("Model Accuracy")
plt.show()

plt.figure()
d1, t1, a1, a2 = scatterFunc(x_train, y_train, x_val, y_val)
plt.scatter(t1, d1, c=a2)
plt.xlabel("Threshold")
plt.ylabel("Max_depth")
plt.title("Model Accuracy")
plt.show()
```





Part 5 Written Response

As mentioned in part 4, yellow dot in the scatterplot indicates the higher classification accuracy. In this part, as the scatterplots indicate, that the yellows commonly appear around threshold=0-200, max_depth=2-3. Thus I would select (threshold, max_depth)=(0-200, 2-3) for the decision tree model. This doesn't change from part 4.

A setting that random search would be preferable to grid search is when the dataset has lower dimension, then random search would take less time to find the right (threshold, max_depth) set with less number of iterations. However, in grid search, we have to spend time to find the near best hyperparameters till the last set sample.

```
In [ ]:
```