🌳 Decision Trees | 🔍 K-Nearest Neighbors

🎓 CSCI 467 – Machine Learning

🎓 Insructor: Prof. Victor Adamchik

🎓 TA: Sampad Mohanty

🌳 Trees 🧮 Impurity 🧮 Entropy 🧱 Stumps→Depth 🔎 KNN 📈 Boundaries

In [1]:
```
from IPython.display import HTML, display

display(HTML("""
<style>
/* ======= Softer theme ======= */
:root{
  --bg-grad-1: linear-gradient(90deg,#fdfbfb,#ebedee); /* subtle grey gradie
  --bg-soft: #fafafa;
  --card: #ffffff;
  --ink: #222;
  --muted: #666;
  --accent: #2563eb;   /* blue */
  --accent-2: #7c3aed; /* purple */
  --accent-3: #d97706; /* amber */
  --radius: 10px;
  --shadow: 0 4px 10px rgba(0,0,0,.05);
  --shadow-soft: 0 3px 8px rgba(0,0,0,.04);
}

/* General text */
.jp-RenderedHTMLCommon, .rendered_html{
  color: var(--ink);
  line-height: 1.55;
  font-size: 15px;
}

/* Headings */
.jp-RenderedHTMLCommon h1, .rendered_html h1{
  font-size: 30px !important;
  padding: 10px;
  border-radius: var(--radius);
  background: var(--bg-grad-1);
  color: var(--ink);
  text-align: center;
```

```
  margin: 20px 0 14px 0;
  box-shadow: var(--shadow);
}
.jp-RenderedHTMLCommon h2, .rendered_html h2{
  font-size: 22px !important;
  padding: 6px 10px;
  border-left: 4px solid var(--accent);
  margin: 18px 0 10px 0;
  color: var(--ink);
  background: #f9fafb;
}
.jp-RenderedHTMLCommon h3, .rendered_html h3{
  font-size: 18px !important;
  padding: 4px 8px;
  border-left: 3px solid var(--accent-3);
  margin: 14px 0 8px 0;
  background: #fcfcfc;
}

/* Paragraphs */
.jp-RenderedHTMLCommon p, .rendered_html p{
  margin: 8px 0;
}

/* Lists */
.jp-RenderedHTMLCommon ul, .jp-RenderedHTMLCommon ol,
.rendered_html ul, .rendered_html ol{
  padding-left: 22px;
  margin: 8px 0;
}

/* Links */
.jp-RenderedHTMLCommon a, .rendered_html a{
  color: var(--accent-2);
  text-decoration: none;
}
.jp-RenderedHTMLCommon a:hover, .rendered_html a:hover{
  text-decoration: underline;
}

/* Code */
.jp-RenderedHTMLCommon code, .rendered_html code{
  background: #f3f4f6;
  color: #111;
  padding: 2px 4px;
  border-radius: 4px;
  font-size: 90%;
}
.jp-RenderedHTMLCommon pre code, .rendered_html pre code{
  display: block;
  padding: 12px;
  border-radius: 8px;
  background: #f9fafb;
  box-shadow: var(--shadow-soft);
}
```

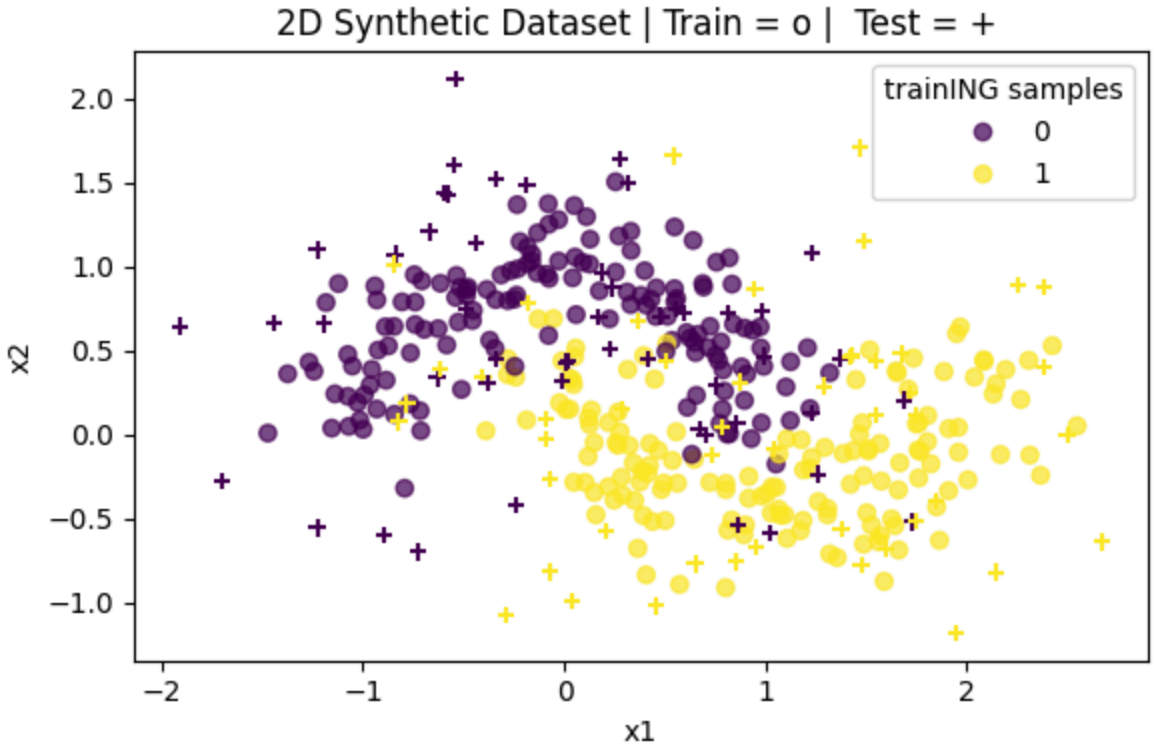
```
/* Tables */
.jp-RenderedHTMLCommon table, .rendered_html table{
  border-collapse: collapse;
  width: 100%;
  background: var(--card);
  border-radius: var(--radius);
  overflow: hidden;
  box-shadow: var(--shadow-soft);
}
.jp-RenderedHTMLCommon th, .rendered_html th{
  padding: 8px;
  background: #f3f4f6;
  text-align: left;
}
.jp-RenderedHTMLCommon td, .rendered_html td{
  padding: 8px;
  border-top: 1px solid #e5e7eb;
}

/* Blockquotes */
.jp-RenderedHTMLCommon blockquote, .rendered_html blockquote{
  background: #f9fafb;
  border-left: 4px solid var(--accent);
  color: #374151;
  padding: 10px 14px;
  margin: 10px 0;
  border-radius: 6px;
}

/* HR divider */
.jp-RenderedHTMLCommon hr, .rendered_html hr{
  border: none;
  height: 2px;
  background: #e5e7eb;
  margin: 16px 0;
}

/* Images */
.jp-RenderedHTMLCommon img, .rendered_html img{
  border-radius: 8px;
  box-shadow: var(--shadow-soft);
}

/* Badges (optional) */
.badge{
  display:inline-block; padding:2px 8px; border-radius:999px;
  background:#eef2ff; color:#3730a3; font-size:12px; margin:2px 4px;
}
</style>
"""))
```

SYNTHETIC DATASET

Below is a synthetic dataset with two classes.

We will use this dataset to get insights into the Decision Tree and KNN classifiers.

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn.datasets import make_moons

from sklearn.tree import plot_tree


NOISE = 0.2
NSAMPLES = 300
# Reproducible 2D dataset suitable for stumps vs deeper trees
X, y = make_moons(n_samples=NSAMPLES, noise=NOISE, random_state=42)

# TEST DATASET
Xtest, ytest = make_moons(n_samples=NSAMPLES//3, noise=NOISE*3, random_state


# Plot (single chart, default matplotlib styling)
plt.figure(figsize=(6, 4))
plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.7, marker='o')
plt.legend(*plt.gca().collections[0].legend_elements(), title="trainING samp

plt.scatter(Xtest[:, 0], Xtest[:, 1], c=ytest, alpha=1, marker='+')
# plt.legend(*plt.gca().collections[1].legend_elements(), title="classes tes



# plt.title(f"2D Synthetic Dataset: make_moons (n={NSAMPLES}, noise={NOISE})
plt.title(f"2D Synthetic Dataset | Train = o |  Test = + ")
plt.xlabel("x1")
plt.ylabel("x2")
plt.tight_layout()
plt.show()
```



In [3]:
```
print(X.shape, " | ", y.shape)
print(Xtest.shape, " | ", ytest.shape)
```

```
(300, 2)  |  (300,)
(100, 2)  |  (100,)
```

WARMUP

> The variables X and y contain the training datapoints (in 2D) and the corresponding labels (class0 or class1)

1. Based on the figure above and the printed shapes of the variables, what is the number of training sample points we have? How many test samples? (2 POINTS)

ANS:

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

def plot_decision_boundary(
    clf,
    X,
    y=None,
    ax=None,
    title=None,
    feature_names=None,
    resolution=400,
    padding=0.5,
```

```
    proba=False,
    alpha_fill=0.3,
    scatter_kwargs=None,
):
    """
    Plot the decision boundary of a 2D classifier.

    Parameters
    ----------
    clf : fitted classifier
        Any estimator with a .predict method. If `proba=True` and
        estimator has .predict_proba, class probability contours are drawn.
        If .decision_function exists, it is used for continuous contours.
    X : array-like, shape (n_samples, 2)
        Training features (exactly two columns).
    y : array-like, optional, shape (n_samples,)
        Class labels. If provided, points are colored by class.
    ax : matplotlib.axes.Axes, optional
        Axes to draw on; if None, a new figure and axes are created.
    title : str, optional
        Title for the plot.
    feature_names : list[str] of length 2, optional
        Names for the x/y axes.
    resolution : int, optional
        Grid resolution in each dimension (default 400).
    padding : float, optional
        Extra margin around the min/max of X (default 0.5).
    proba : bool, optional
        If True and classifier supports predict_proba, fills by max class pr
        If False, fills by predicted class index.
    alpha_fill : float, optional
        Alpha for the contourf background.
    scatter_kwargs : dict, optional
        Extra kwargs passed to plt.scatter for the data points.
    """
    X = np.asarray(X)
    assert X.shape[1] == 2, "X must be 2D: shape (n_samples, 2)"

    if ax is None:
        fig, ax = plt.subplots(figsize=(6, 5))
    else:
        fig = ax.figure

    # Grid bounds
    x_min, x_max = X[:, 0].min() - padding, X[:, 0].max() + padding
    y_min, y_max = X[:, 1].min() - padding, X[:, 1].max() + padding

    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, resolution),
        np.linspace(y_min, y_max, resolution),
    )
    grid = np.c_[xx.ravel(), yy.ravel()]

    # Evaluate model on grid
    Z = None
    Z_cont = None
```

```
    if proba and hasattr(clf, "predict_proba"):
        # Use max class probability for smooth shading
        P = clf.predict_proba(grid)
        if P.ndim == 1:
            # Some estimators can return shape (n_samples,) for binary
            P = np.c_[1 - P, P]
        Z_cont = P.max(axis=1).reshape(xx.shape)
        Z = P.argmax(axis=1).reshape(xx.shape)
    elif hasattr(clf, "decision_function"):
        df = clf.decision_function(grid)
        # Multiclass decision_function: take argmax for class regions
        if df.ndim == 1:
            Z_cont = df.reshape(xx.shape)
            Z = (df > 0).astype(int).reshape(xx.shape)
        else:
            Z = df.argmax(axis=1).reshape(xx.shape)
            # A crude confidence proxy for fill (max margin)
            Z_cont = df.max(axis=1).reshape(xx.shape)
    else:
        Z = clf.predict(grid).reshape(xx.shape)

    # Background fill
    if Z is not None:
        cs = ax.contourf(xx, yy, Z if Z_cont is None else Z, alpha=alpha_fil
    else:
        # Fallback if only continuous scores available
        ax.contourf(xx, yy, Z_cont, alpha=alpha_fill)

    # Decision boundary lines (only if we have discrete classes)
    if Z is not None:
        ax.contour(xx, yy, Z, levels=np.unique(Z), linewidths=1, linestyles=

    # Scatter training data
    skw = dict(s=25, edgecolor="k", alpha=0.9)
    if scatter_kwargs:
        skw.update(scatter_kwargs)

    if y is not None:
        scatter = ax.scatter(X[:, 0], X[:, 1], c=y, **skw)
        classes = np.unique(y)
        handles = [
            mpatches.Patch(color=scatter.cmap(scatter.norm(c)), label=f"clas
            for c in classes
        ]
        ax.legend(handles=handles, title="classes")
    else:
        ax.scatter(X[:, 0], X[:, 1], **skw)

    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    if feature_names is None:
        ax.set_xlabel("x1")
        ax.set_ylabel("x2")
    else:
        ax.set_xlabel(feature_names[0])
```

```
        ax.set_ylabel(feature_names[1])
    if title is not None:
        ax.set_title(title)
    fig.tight_layout()
    return ax
```

INTRO TO SCIKIT-LEARN

> We will be using the popular and standard scikit-learn machine learning library in this notebook

> To familiarize you with this library, you uneed to read the relevant documentation on your own and implement the missing pieces of the code

> The missing pieces of the code are preceded by a comment that is prefixed with "COMPLETE"

> You are encouraged to understand the code that precedes or follows the missing lines to be fluent with the sklearn library. Every machine learning job/internship interview out there will test you on familiarity with sklearn !!!

DECISION TREE CLASSIFIER IN SKLEARN

> 2. COMPLETE: Import the decision tree classifier from sklearn (2 POINTS)

In [5]:
```
### COMPLETE: Import the correct classifier
from sklearn.tree import DecisionTreeClassifier

# Fit a decision stump ( a tree of depth 1 )
MAXDEPTH = 1

stump = DecisionTreeClassifier(criterion='gini', max_depth=MAXDEPTH, random_

# COMPLETE: call the fit function with the appropriate arguments
stump.fit(X, y)
```

Out[5]:

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=1, random_state=42)
```

STRUCTURE OF THE DECISION TREE OBJECT IN SKLEARN

In the above code cell, click on the DecisionTreeClassifier object printed to expand it.

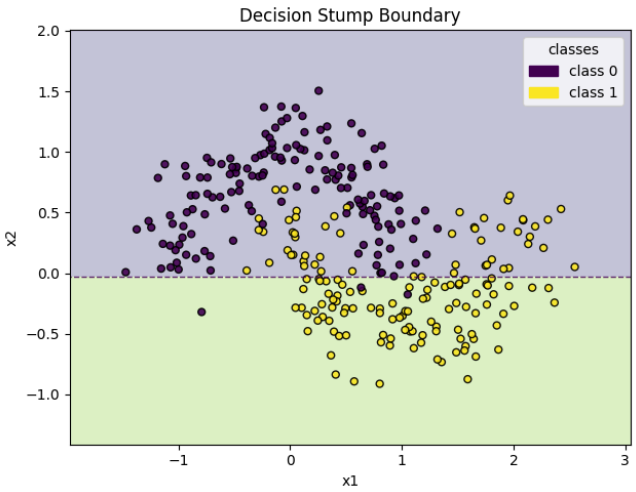
Try to understand the various information present in each row of the object as much as possible.

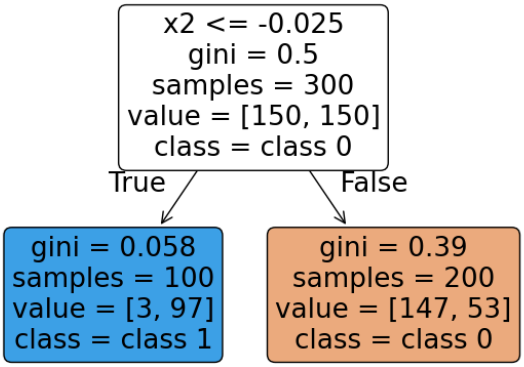
3. Are there any rows that are of different colors than the rest? Why? (2 POINTS)

YOUR ANSWER:

In [6]:

```
# Make (1, 2) subplot: left = boundary, right = tree
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Use your plot_decision_boundary function AS-IS
plot_decision_boundary(
    stump, X, y,
    ax=axes[0],
    title="Decision Stump Boundary",
    feature_names=["x1", "x2"]
)

# Tree diagram
plot_tree(
    stump,
    feature_names=["x1", "x2"],
    class_names=["class 0", "class 1"],
    filled=True,
    rounded=True,
    ax=axes[1]
)

plt.tight_layout()
plt.show()
```





UNDERSTANDING DECISION STUMPS

Decision stumps are decision trees with depth = 1.

In the cell above, the decision boundary (left figure) and the decision stump itself (right boundary) are plotted

4. Answer the following questions based on the figure below. 5 POINTS)

Is the decision boundary linear? (1 POINT)

ANS: YES/NO

Is the decision boundary parallel to one of the axes? (1 POINT)

ANS: YES/NO

Can the decision boundaries in a decision stump be not parallel to any of the axes? (1 POINT)

ANS: YES/NO

The right figure has the nodes of the tree with different colors (top is white, bottom left is blue and bottom right is orange). What do you think blue and orange/red represent here? (1 POINT)

ANS:

Which leaf node is associated to the class label 0 and which to class label 1? (1 POINT)

ANS:

GINI IMPURITY

6. Complete the code below to calculate the gini index (4 POINTS)

If p_0 and p_1 are the fraction of samples labeled class0 and class1 in a region/area, the gini index/score is given by

$$gini(p_0, p_1) = 1 - p_0^2 - p_1^2$$

Note that the gini impurity is symmetric in its two arguments, that is,

$$gini(p_0, p_1) = gini(p_1, p_0)$$

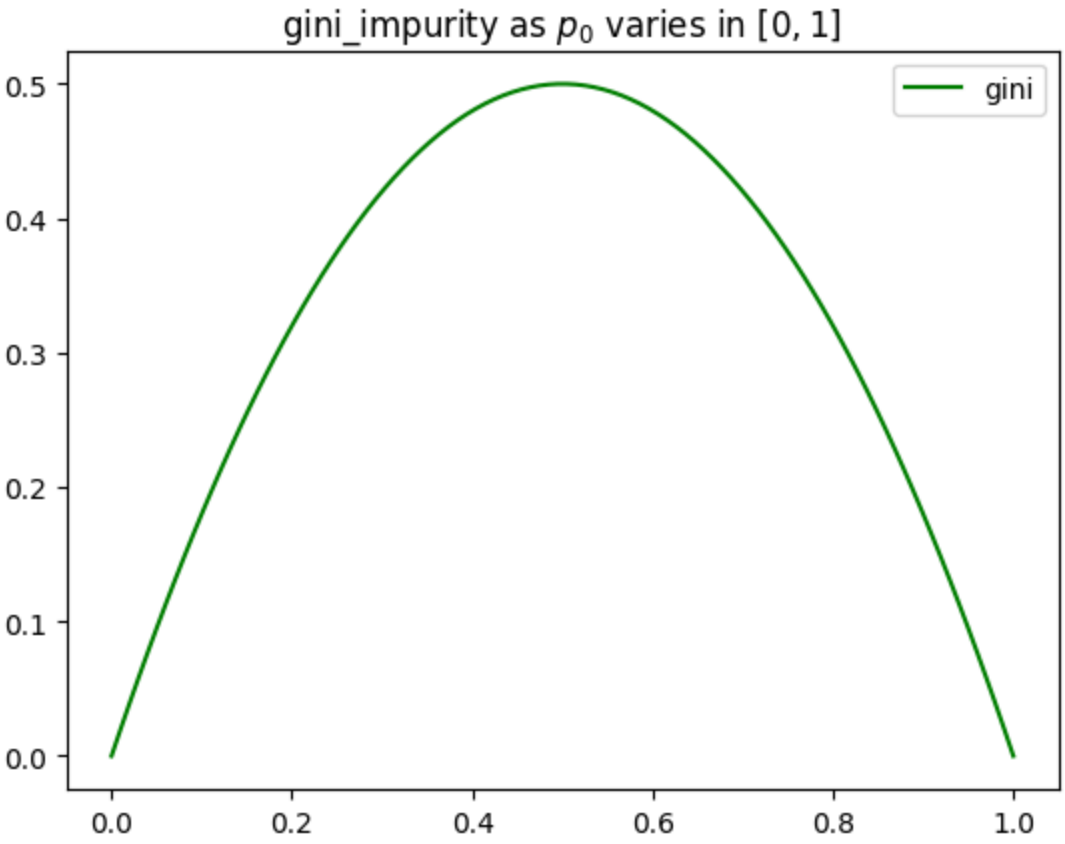
NOTE: If there are only two classes, $p_1 = 1 - p_0$ and hence gini impurity can be plotted as a function with respect to just $p_0 \in [0, 1]$

Based on this definition of gini impurity, implement the incomplete function to calculate the gini score where there are n0 and n1 samples labeled class0 and class1 respectively

In [7]:
```
###  COMPLETE: Implement the gini index/score function given number of class

def gini_impurity(n0, n1):
    total = n0 + n1
    p0 = n0/total
    p1 = n1/total

    return 1  - p0**2 - p1**2
```

In [8]:
```
### Plot with respect to p0
_p0 = np.linspace(0,1,100)
_p1 = 1 - _p0
_g = gini_impurity(_p0, _p1)
plt.plot(_p0,_g, color='green', label='gini')
plt.title("gini_impurity as $p_0$ varies in $[0,1]$")
plt.legend()
```

Out[8]:
```
<matplotlib.legend.Legend at 0x7f6914da5d10>
```



MORE INTUITIONS ON GINI IMPURITY

7. Answer the following based on the plot above (3 POINTS)

As you can see, the gini impurity is symmetric about the point where it peaks.

Where does the gini impurity peak as we vary p_0 from 0 to 1 (and hence p_1 varies from 1 to 0) ? What is the gini impurity value at that peak (2 POINTS)

ANS:

What is the intuition behind the gini impurity being zero at the extremes, i.e when p_0 and p_1. The answer needs to qualitative and not quantitave (i.e, answer should not be based on the formula/expression of gini impurity)

ANS:

In [9]:
```
def entropy(n0,n1):
    total = n0 + n1
    p0 = n0/total
```

```
    p1 = n1/total
    return -p0 * np.log2(p0) - p1 * np.log2(p1)
```

GINI IMPURITY VS ENTROPY

Entropy of a given set (with samples from two different classes) lying in that set is given by

$$entropy(p_0, p_1) = -p_0 log_2(p_0) - p_1 log_2(p_1)$$

where p_0 and p_1 are the fraction of samples in the set belonging to class0 and class1

In the cell above, we have implement the code for entropy based on the number of samples in each class n0 and n1

In the cell below, we have plotted the entropy along with gini impurity.

8. Answer the following based on the plot below (4 POINTS)

At what value of p_0 does the entropy peak and what is that maximum value? (2 POINT)

ANS:

Are the two function convex or concave? (1 POINT)

ANS: CONVEX/CONCAVE

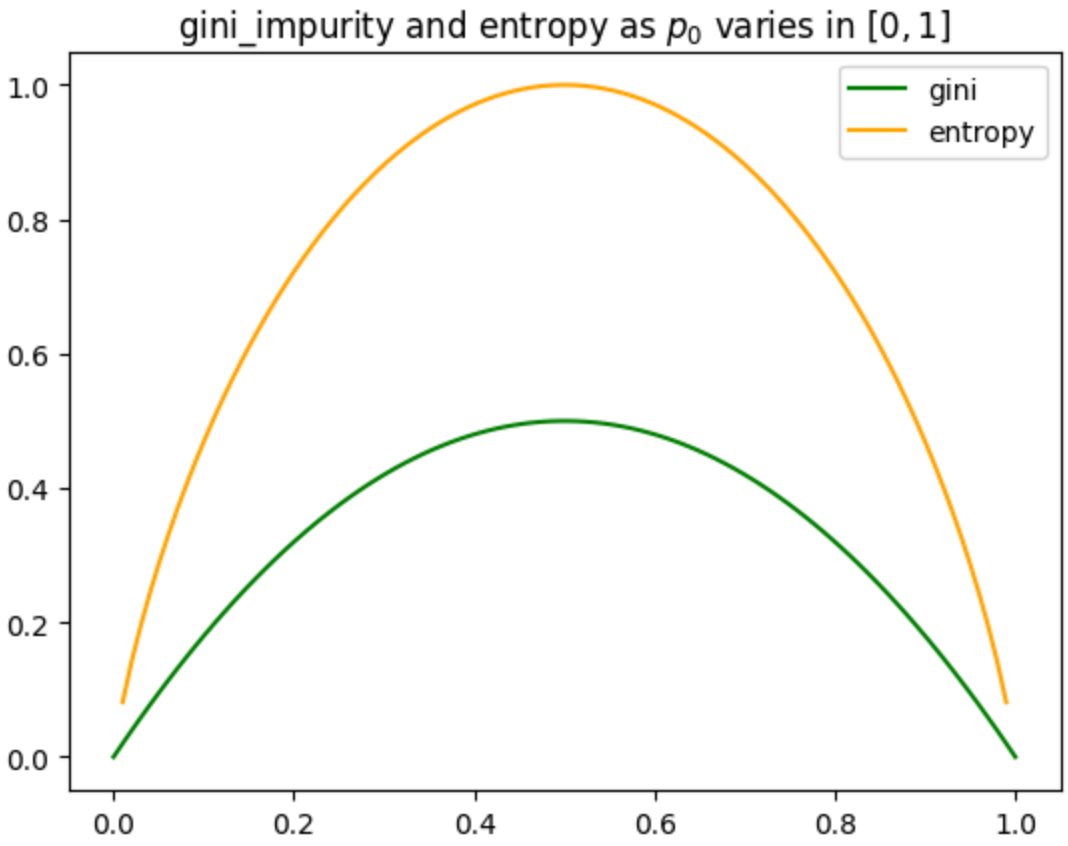
In what aspects (other than convexity/concavity) are the two functions (gini impurity and entropy) similar? (1 POINT)

ANS:

In [10]:
```
_h = entropy(_p0,_p1)
plt.plot(_p0, _g, color='green', label='gini')
plt.plot(_p0, _h, color='orange' , label='entropy')
plt.legend()
plt.title("gini_impurity and entropy as $p_0$ varies in $[0,1]$")
```

```
/tmp/ipykernel_227122/1882047823.py:5: RuntimeWarning: divide by zero encoun
tered in log2
  return -p0 * np.log2(p0) - p1 * np.log2(p1)
/tmp/ipykernel_227122/1882047823.py:5: RuntimeWarning: invalid value encount
ered in multiply
  return -p0 * np.log2(p0) - p1 * np.log2(p1)
```

Out[10]: Text(0.5, 1.0, 'gini_impurity and entropy as p_0 varies in $[0,1]$')



GINI IMPURITY FOR NODES IN THE ABOVE DECISION STUMP

> 9. For each of the three nodes in the decision stump, make sure the gini score matches with the ones printed in the tree by calling the 'gini_score' function and passing the appropriate number of samples as shown in the tree nodes. (3 POINTS)

In [11]:
```
### COMPLETE: Pass the appropriate arguments for n0 and n1 for the gini_scor
### NOTE: Although the gini impurity is symmetric to its two arguments n0 ar
###  you will lose points if you swap the values of n0 and n1 even if the gi

print("Top Node (depth 0):",gini_impurity(n0=150,n1=150))
print("Bottom Left Node (depth 1):",gini_impurity(n0=97,n1=3))
print("Bottom Right Node (depth 1):",gini_impurity(147,53))
```

```
Top Node (depth 0): 0.5
Bottom Left Node (depth 1): 0.05820000000000004
Bottom Right Node (depth 1): 0.38955000000000006
```

GINI IMPURITY REDUCTION AT A SPLIT

The top node in the decision tree contained all the training points (300 of them), 150 from each class. It is no wonder that the gini impurity was 0.5 (which is the maximum value)

However, when the node is split, the two children nodes have lesser gini impurities than the parent node.

The weighted gini impurity of the two children are calculated using the number of class0 and class1 samples in the left and rigt children l_0, l_1, r_0, r_1 where $n_0 = l_0 + r_0$ and $n_1 = l_1 + r_1$ are the number of class0 samples in parent node.

$$\text{weighted gini impurity of children} = \frac{\text{samples in left child}}{\text{total samples in parent}} \times \text{gini right child}$$

$$||$$

$$\frac{l_0 + l_1}{n_1 + n_2} \times \text{gini left child} + \frac{r_0 + r_1}{n_1 + n_2} \times \text{gini right child}$$

The in the gini impurity reduced by splitting the parent node is given by

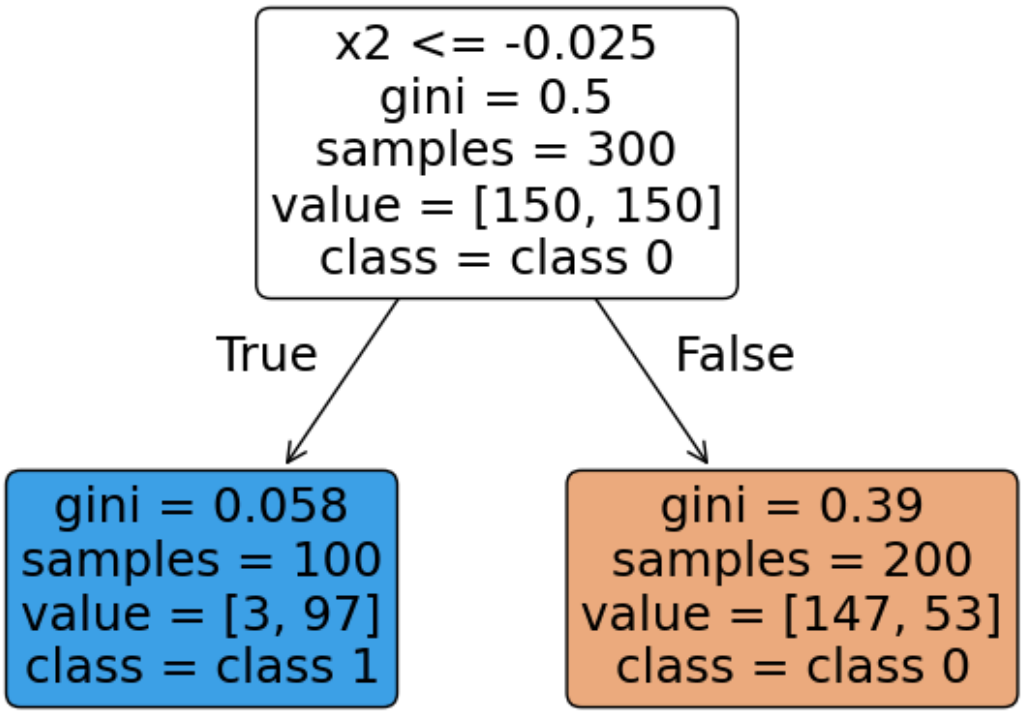
$$\Delta\text{gini} = \text{gini of parent} - \text{weighted gini impurity of children}$$

10) Find the gini impurity reduction (= purity improvement) in our decision stump [stump shown below]. Show all steps and not just the final answer (4 POINTS)

YOUR ANS:

In [12]:
```
# Tree diagram
plot_tree(
```

```
    stump,
    feature_names=["x1", "x2"],
    class_names=["class 0", "class 1"],
    filled=True,
    rounded=True,
    ax=None
)

plt.tight_layout()
plt.show()
```



GINI VS ENTROPY REDUCTION

11) What is the use of the $\Delta gini$ in the decision tree algorithm? Give your answer qualitatively. (2 POINTS)

ANS:

12) How does the decision tree algorithm decide which of the two features (x_1 or x_2) and the threshold to split a parent node on? What would be the problem if we had a lot of features, say 10,000 of them? What could be a possible solution when we have a large number of features?(4 POINTS)

ANS:

> 13) Would the splits (the feature to split on and the corresponding threshold) be the same if we used entropy instead of gini impurity? (1 POINT)

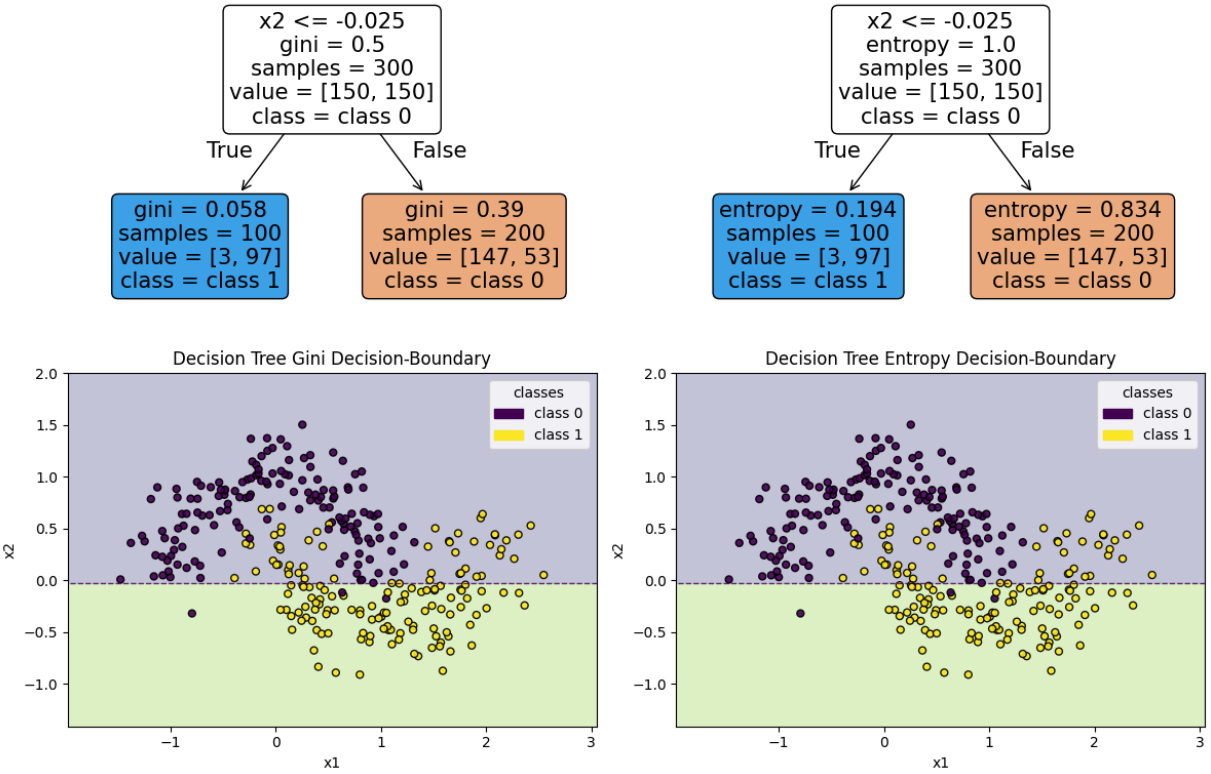
ANS:

> 14) In the code below, play with the MAXDEPTH parameter and plot the decision trees obtained using reduction in gini impurity and entropy? What is the minimum value of MAXDEPTH that leads to different decision trees for gini impurity and entropy? (2 POINTS)

ANS:

In [13]:
```
# COMPLETE: Change the depth of the decision tree to find out the minimum de
MAXDEPTH = 1

tree_gini = DecisionTreeClassifier(criterion='gini', max_depth=MAXDEPTH, rar
tree_gini = tree_gini.fit(X,y)

tree_entropy = DecisionTreeClassifier(criterion='entropy', max_depth=MAXDEPT
tree_entropy = tree_entropy.fit(X,y)

# Make (1, 2) subplot: left = boundary, right = tree
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Tree diagram
plot_tree(
    tree_gini,
    feature_names=["x1", "x2"],
    class_names=["class 0", "class 1"],
    filled=True,
    rounded=True,
    ax=axes[0,0]
)

# Tree diagram
plot_tree(
    tree_entropy,
    feature_names=["x1", "x2"],
    class_names=["class 0", "class 1"],
    filled=True,
    rounded=True,
    ax=axes[0,1]
)

# Use your plot_decision_boundary function AS-IS
plot_decision_boundary(
    tree_gini, X, y,
    ax=axes[1,0],
    title="Decision Tree Gini Decision-Boundary",
    feature_names=["x1", "x2"]
)
```

```
# Use your plot_decision_boundary function AS-IS
plot_decision_boundary(
    tree_entropy, X, y,
    ax=axes[1,1],
    title="Decision Tree Entropy Decision-Boundary",
    feature_names=["x1", "x2"]
)


plt.tight_layout()
plt.show()
```



DEEP DECISION TREES

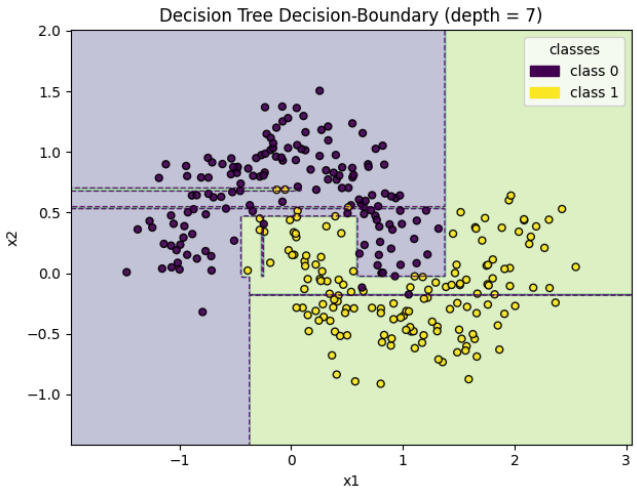
> We will now explore what happens when we increase the depth in decision trees.

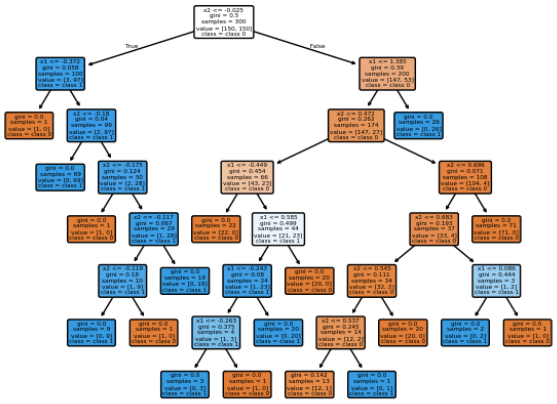
> 15) Below is a decision tree with depth = 7 that was fitted on the same dataset. What do you think is the problem with this model? Will it predict well on unseen test dataset? (2 POINTS)

ANS:

In [14]:
```
# Fit a depth 2 tree
MAXDEPTH = 7
tree2 = DecisionTreeClassifier(criterion='gini',max_depth=MAXDEPTH, random_s
tree2.fit(X, y)
```

```python
# Make (1, 2) subplot: left = boundary, right = tree
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Use your plot_decision_boundary function AS-IS
plot_decision_boundary(
    tree2, X, y,
    ax=axes[0],
    title=f"Decision Tree Decision-Boundary (depth = {MAXDEPTH})",
    feature_names=["x1", "x2"]
)

# Tree diagram
plot_tree(
    tree2,
    feature_names=["x1", "x2"],
    class_names=["class 0", "class 1"],
    filled=True,
    rounded=True,
    ax=axes[1]
)

plt.tight_layout()
plt.show()
```





TRAINING VS TEST ACCURACY

16) In the code cell below, import the accuracy_score function from the correct sklearn module (1 POINT)

The graph below shows the training accuracy increases as we increase depth when trained/fitted on the training data X,y. We also plot the testing accuracy as the depth is increased using Xtest, ytest.

17) Why did the trainging accuracy increase but the test accuracy plateaue and then drop?(2 POINTS)

ANS:

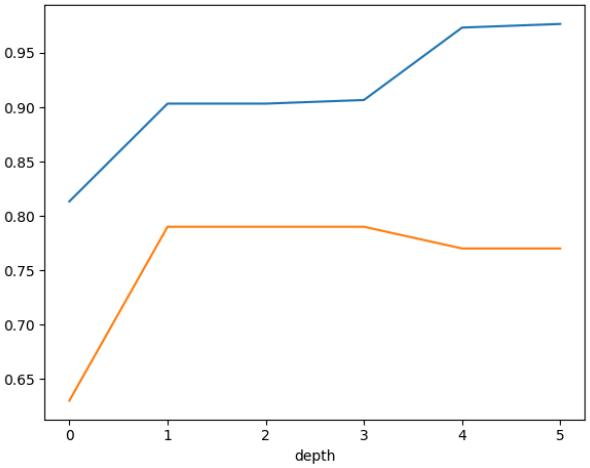
> **18) Why exactly is the problem with a decision tree model when we increasing the depth arbitrarily? The answer cannot be the same as the prior question (2 POINTS)**

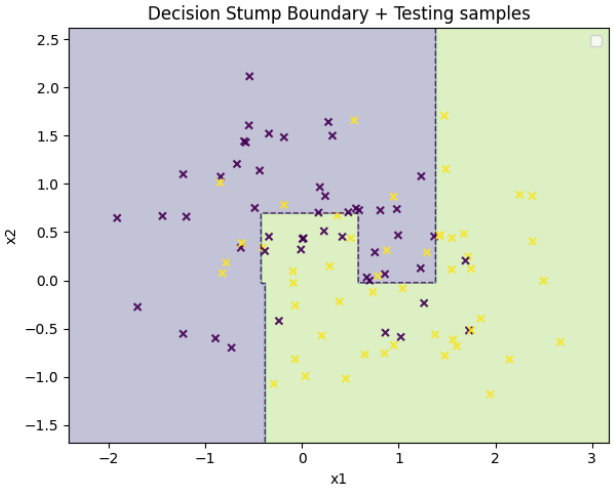
ANS:

In [15]:
```
#COMPLETE: Import the accuracy_score from the correct sklearn module
from sklearn.metrics import accuracy_score
```

In [16]:
```
train_accuracy, test_accuracy = [],[]

DEPTHS = list(range(1,7))
for MAXDEPTH in DEPTHS:
    tree = DecisionTreeClassifier(criterion='entropy', max_depth=MAXDEPTH, r
    tree = tree.fit(X,y)

    train_accuracy.append( accuracy_score(y, tree.predict(X))  )
    test_accuracy.append(  accuracy_score(ytest, tree.predict(Xtest)) )

# Make (1, 2) subplot: left = boundary, right = tree
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Use your plot_decision_boundary function AS-IS
plot_decision_boundary(
    tree, Xtest, ytest,
    ax=axes[1],
    title="Decision Stump Boundary + Testing samples",
    feature_names=["x1", "x2"],
    scatter_kwargs = {"marker":"x"}
)



axes[0].plot(train_accuracy, label='training accuracy')
axes[0].plot(test_accuracy, label='testing accuracy')
axes[0].set_xlabel("depth")

plt.legend()
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_227122/15381826.py:108: UserWarning: You passed a edgecolor/e
dgecolors ('k') for an unfilled marker ('x').  Matplotlib is ignoring the ed
gecolor in favor of the facecolor.  This behavior may change in the future.
  scatter = ax.scatter(X[:, 0], X[:, 1], c=y, **skw)
/tmp/ipykernel_227122/3722750224.py:29: UserWarning: No artists with labels
found to put in legend.  Note that artists whose label start with an undersc
ore are ignored when legend() is called with no argument.
  plt.legend()
```





K NEAREST NEIGHBOR (KNN)

Just like the decision tree classifier we say above, we want to implement our own sklearn style KNN classifier on which we can call fit and predict. Below is a code implementation for KNN. Take some time to understand the code

In [17]:
```
from scipy.stats import mode


class KNN:
    # Initialize with the k value
    def __init__(self,k=1):
        self.k = k

        self.X = None
        self.y = None

    # Fit the given training data.
    # Since in KNN, there is not much to do other than comparing the distanc
    # we store all the training samples in the class's static variables X ar
    def fit(self,X,y):
        self.X = X
        self.y = y

    def predict(self,Xtest):
        # for each item in Xtest, predict its label
        votes = []
        for x in Xtest:
            distances = self.distances(x)
            neighbor_indices = np.argsort(distances)[:self.k]
            neighbor_labels = [self.y[i] for i in neighbor_indices]
            vote = mode(neighbor_labels, keepdims=True).mode[0]
            votes.append(int(vote))
```

```
        return np.array(votes)

    def distances(self,x1):
        return [np.linalg.norm(x1 - x) for x in self.X]
```

TRAINING ACCURACY IN KNN

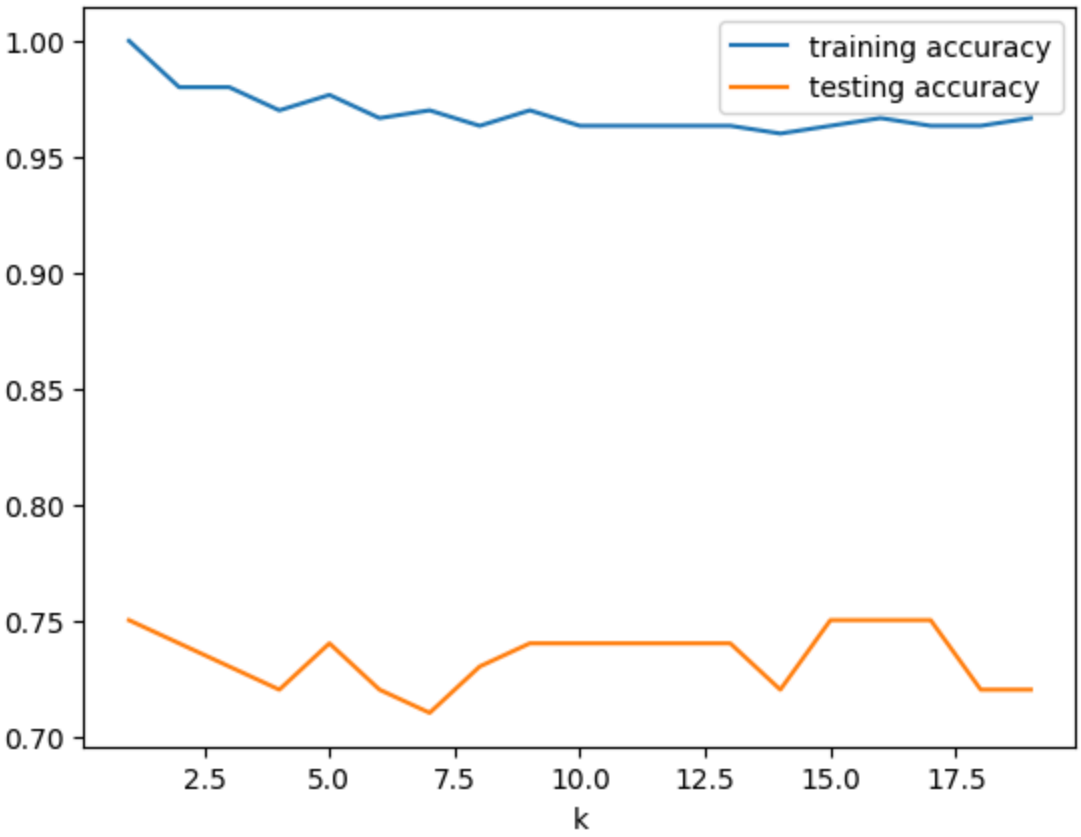
Below is a plot of KNN training accuracy as we increase k

19) Why is the training accuracy 1 (or 100%) for k=1? (2 POINTS)

ANS:

In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_training_accuracy, knn_test_accuracy = [],[]

ks = list(range(1,20))
for k in ks:
    knn = KNN(k)
    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X,y)

    knn_training_accuracy.append(accuracy_score(y,knn.predict(X)))
    knn_test_accuracy.append(accuracy_score(ytest,knn.predict(Xtest)))

plt.plot(ks,knn_training_accuracy, label="training accuracy")
plt.plot(ks,knn_test_accuracy, label="testing accuracy")
plt.xlabel("k")
plt.legend()
```

Out[18]:
```
<matplotlib.legend.Legend at 0x7f691111ccd0>
```



KNN Decision Boundary

20) What is the fundamental difference in the decision boundaries produced in KNN and Decision Trees? (2 POINTS)

ANS:

21) Describe qualitatively how the decision boundary changes as we increase k in KNN from 1 to 3 to 5 in teh following plot (2 POINTS)

ANS:

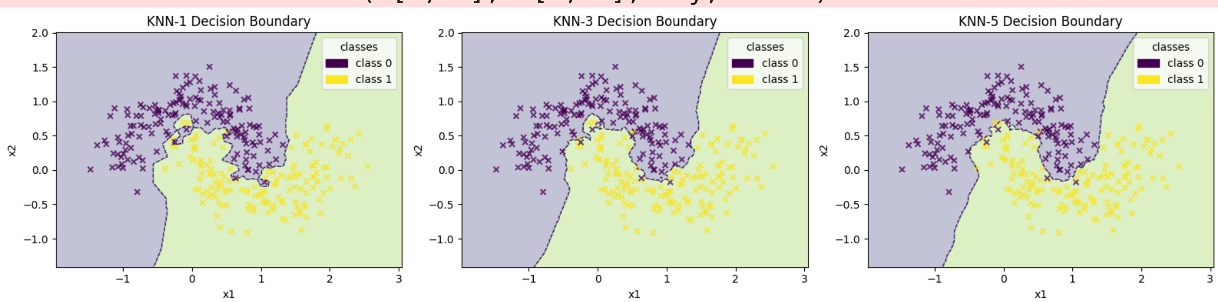
22) As you can see, we try to use odd numbers for the k values in KNN. What is a good reason for that? (2 POINTS)

ANS:

In [19]:
```
from sklearn.neighbors import KNeighborsClassifier

fig,axes = plt.subplots(1,3, figsize=(16,4))

for i,k in enumerate([1,3,5]):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X,y)
    # Use your plot_decision_boundary function AS-IS
```

```
    plot_decision_boundary(
        knn, X, y,
        ax=axes[i],
        title=f"KNN-{k} Decision Boundary",
        feature_names=["x1", "x2"],
        scatter_kwargs = {"marker":"x", 'alpha':0.75}
    )
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_227122/15381826.py:108: UserWarning: You passed a edgecolor/e
dgecolors ('k') for an unfilled marker ('x').  Matplotlib is ignoring the ed
gecolor in favor of the facecolor.  This behavior may change in the future.
  scatter = ax.scatter(X[:, 0], X[:, 1], c=y, **skw)
/tmp/ipykernel_227122/15381826.py:108: UserWarning: You passed a edgecolor/e
dgecolors ('k') for an unfilled marker ('x').  Matplotlib is ignoring the ed
gecolor in favor of the facecolor.  This behavior may change in the future.
  scatter = ax.scatter(X[:, 0], X[:, 1], c=y, **skw)
/tmp/ipykernel_227122/15381826.py:108: UserWarning: You passed a edgecolor/e
dgecolors ('k') for an unfilled marker ('x').  Matplotlib is ignoring the ed
gecolor in favor of the facecolor.  This behavior may change in the future.
  scatter = ax.scatter(X[:, 0], X[:, 1], c=y, **skw)
```



In [ ]: