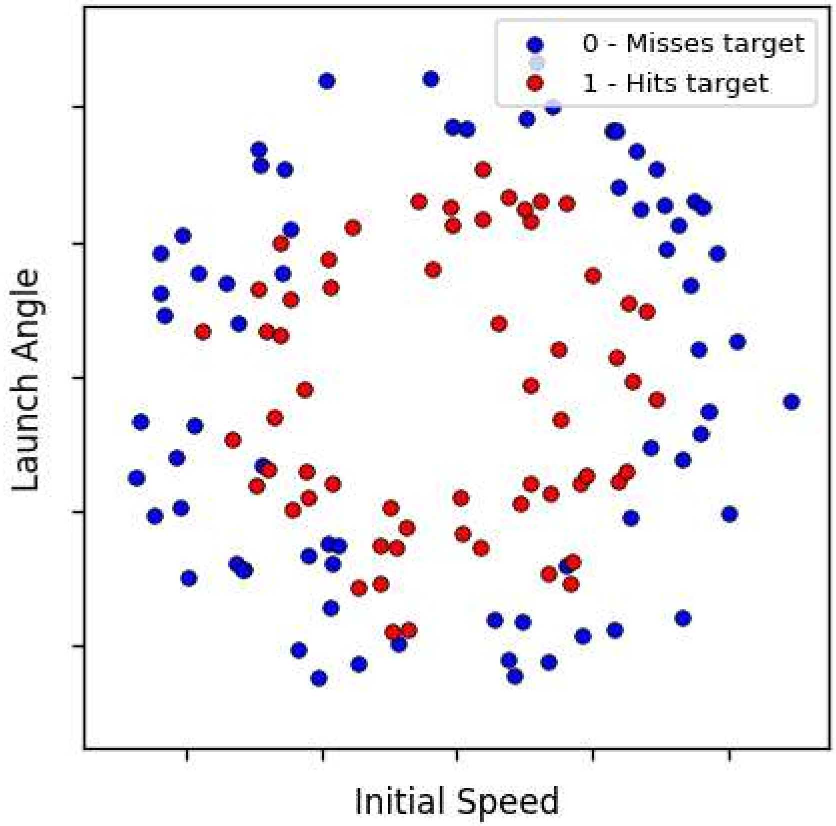
M5-L1 Problem 3

Let's revisit the initial speed vs. launch angle data from the logistic regression module. This time, you will train a decision tree classifier to predict whether a projectile launched with a given speed and angle will hit a target.

Run this cell to load the data and decision tree tools:

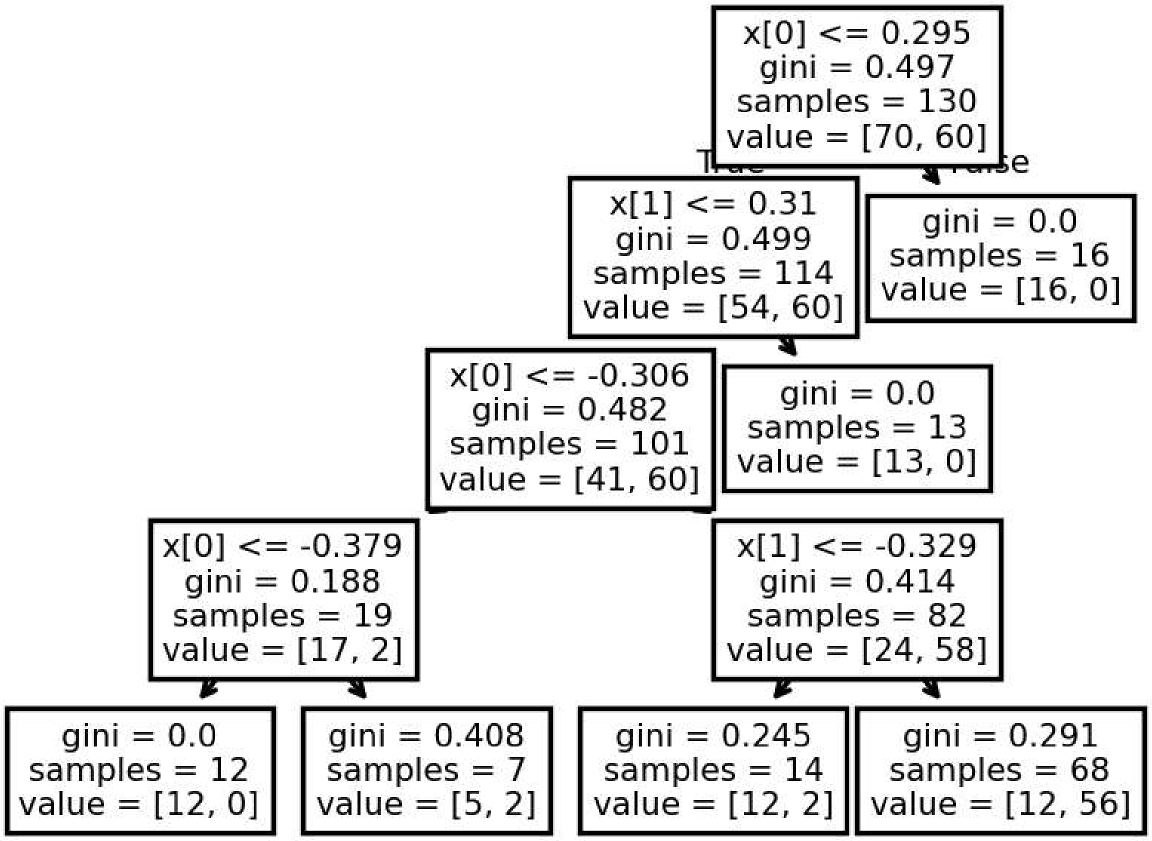
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from matplotlib.colors import ListedColormap

y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
x1 = np.array([0.02693745, 0.41186575, 0.10363585, 0.08489663, 0.09512868, 0.311211
x2 = np.array([0.3501823 , 0.10349458, 0.20137442, 0.37973165, 0.71062143, 0.253770
X = np.vstack([x1, x2]).T

def plot_data(X,y):
    colors=["blue","red"]
    labels = ["0 - Misses target", "1 - Hits target"]
    for i in range(2):
        plt.scatter(X[y==i,0],X[y==i,1],s=20,c=colors[i],edgecolors="black",linewid
        plt.xlabel("Initial Speed")
        plt.ylabel("Launch Angle")
        plt.legend(loc="upper right",prop={'size':8})
        ax = plt.gca()
        ax.set_xticklabels([])
        ax.set_yticklabels([])
        plt.xlim([-0.55,.55])
        plt.ylim([-0.55,.55])

plt.figure(figsize=(4,4),dpi=120)
plot_data(X,y)
plt.show()
```



Training a decision tree classifier.

Below, a decision tree of max depth 4 is trained, and the tree is visualized with `plot_tree()`.

In [2]:
```
dt = DecisionTreeClassifier(max_depth=4)
dt.fit(X,y)

plt.figure(figsize=(4,3),dpi=250)
plot_tree(dt)
plt.show()
```



Accuracy on training data

Compute the accuracy on the training data with the provided function `get_dt_accuracy(dt, X, y)`. Print the result.

In [8]:
```
def get_dt_accuracy(dt, X, y):
    pred = dt.predict(X)
    return 100*np.sum(pred == y)/len(y)

# YOUR CODE GOES HERE
acc = get_dt_accuracy(dt,X,y)
print(acc)
```

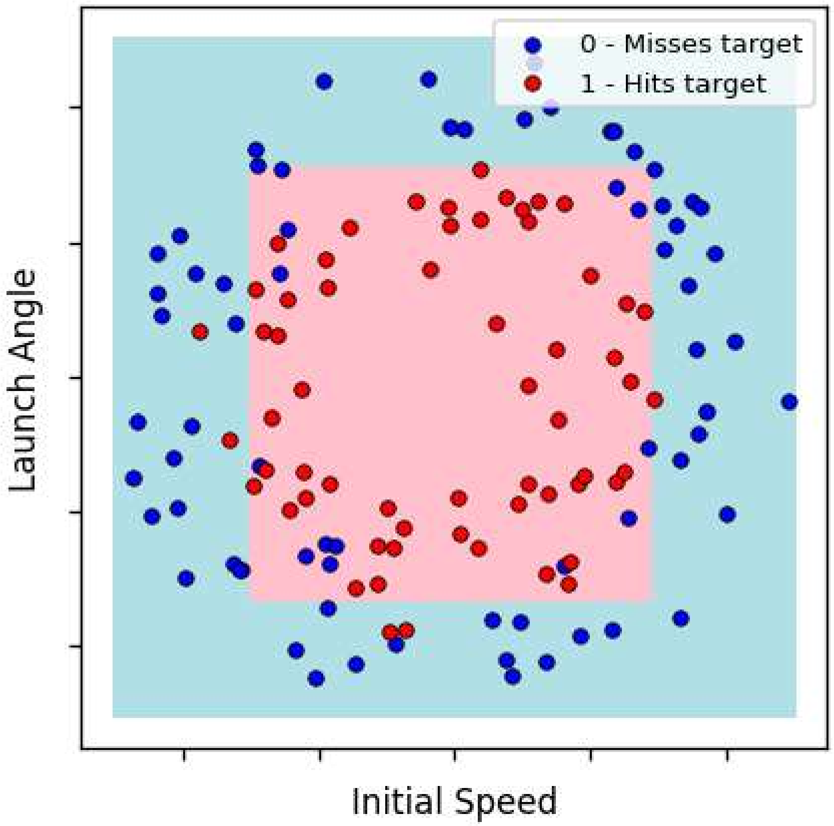
```
87.6923076923077
```

Visualizing tree predictions

By evaluating the model on a meshgrid of results, we can look at how our model performs on the input space:

In [4]:
```
vals = np.linspace(-.5,.5,100)
x1grid, x2grid = np.meshgrid(vals, vals)
X_test = np.vstack([x1grid.flatten(), x2grid.flatten()]).T
```

```
pred = dt.predict(X_test)

plt.figure(figsize=(4,4),dpi=120)
bgcolors = ListedColormap(["powderblue","pink"])
plt.pcolormesh(x1grid, x2grid, pred.reshape(x1grid.shape), shading="nearest",cmap=b
plot_data(X,y)
plt.show()
```



Expanded feature set

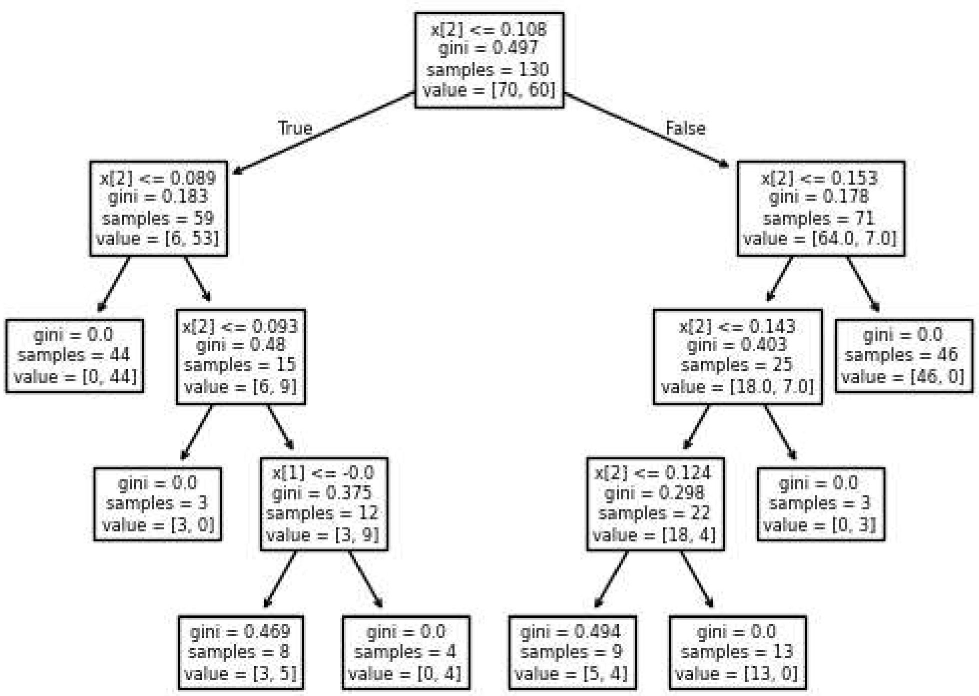
Now, we will add a third feature that (for this problem) happens to be very useful. That feature is $x_1^2 + x_2^2$. A new training input `X_ex` is generated below containing this additional feature.

Train a new decision tree, max depth 4, on this data. Then visualize the tree with `plot_tree()`.

In [9]:
```
def feature_expand(X):
    x1 = X[:,0].reshape(-1, 1)
    x2 = X[:,1].reshape(-1, 1)
    columns = [x1, x2, x1*x1 + x2*x2]
    return np.concatenate(columns, axis=1)

X_ex = feature_expand(X)

# YOUR CODE GOES HERE
# Train a new decision tree on X_ex, y
```

```python
# Plot the tree

dt = DecisionTreeClassifier(max_depth=4)
dt.fit(X_ex,y)

plot_tree(dt)
```

Out[9]:
```
[Text(0.5, 0.9, 'x[2] <= 0.108\ngini = 0.497\nsamples = 130\nvalue = [70, 60]'),
 Text(0.16666666666666666, 0.7, 'x[2] <= 0.089\ngini = 0.183\nsamples = 59\nvalue
= [6, 53]'),
 Text(0.3333333333333333, 0.8, 'True  '),
 Text(0.08333333333333333, 0.5, 'gini = 0.0\nsamples = 44\nvalue = [0, 44]'),
 Text(0.25, 0.5, 'x[2] <= 0.093\ngini = 0.48\nsamples = 15\nvalue = [6, 9]'),
 Text(0.16666666666666666, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.3333333333333333, 0.3, 'x[1] <= -0.0\ngini = 0.375\nsamples = 12\nvalue =
[3, 9]'),
 Text(0.25, 0.1, 'gini = 0.469\nsamples = 8\nvalue = [3, 5]'),
 Text(0.4166666666666667, 0.1, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.8333333333333334, 0.7, 'x[2] <= 0.153\ngini = 0.178\nsamples = 71\nvalue =
[64.0, 7.0]'),
 Text(0.6666666666666667, 0.8, '  False'),
 Text(0.75, 0.5, 'x[2] <= 0.143\ngini = 0.403\nsamples = 25\nvalue = [18.0, 7.
0]'),
 Text(0.6666666666666666, 0.3, 'x[2] <= 0.124\ngini = 0.298\nsamples = 22\nvalue =
[18, 4]'),
 Text(0.5833333333333334, 0.1, 'gini = 0.494\nsamples = 9\nvalue = [5, 4]'),
 Text(0.75, 0.1, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
 Text(0.8333333333333334, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9166666666666666, 0.5, 'gini = 0.0\nsamples = 46\nvalue = [46, 0]')]
```



Accuracy on training data: expanded features

Compute the accuracy of this new model its training data. It should have increased. Note that the useful features to expand will vary significantly from problem to problem.

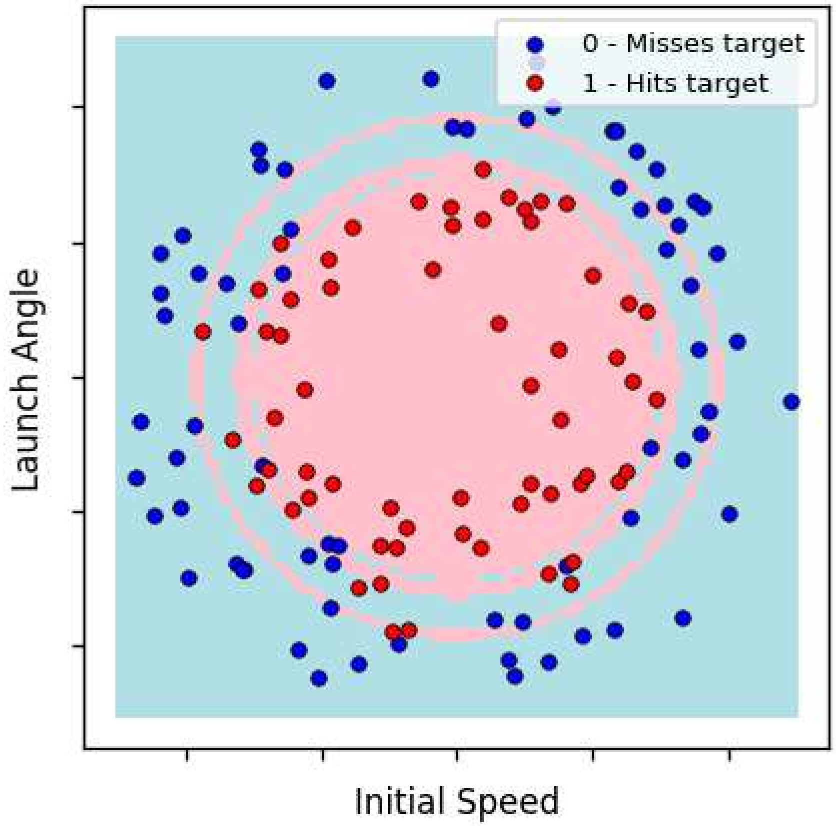
In [10]:
```
# YOUR CODE GOES HERE
acc_new = get_dt_accuracy(dt,X_ex,y)
print(acc_new)
```

```
94.61538461538461
```

Visualizing expanded feature results

Use your model to make a prediction called `pred` on the data `X_test_ex`, an expanded meshgrid of points, as indicated. This code will plot the class decisions. Note the difference between this and the previous model, which only had speed and angle as features.

In [11]:
```
X_test_ex = feature_expand(X_test)
# YOUR CODE GOES HERE
# Have your model make a prediction, `pred` on X_test_ex

pred = dt.predict(X_test_ex)


plt.figure(figsize=(4,4),dpi=120)
bgcolors = ListedColormap(["powderblue","pink"])
plt.pcolormesh(x1grid, x2grid, pred.reshape(x1grid.shape), shading="nearest",cmap=b
plot_data(X,y)
plt.show()
```



In [ ]: