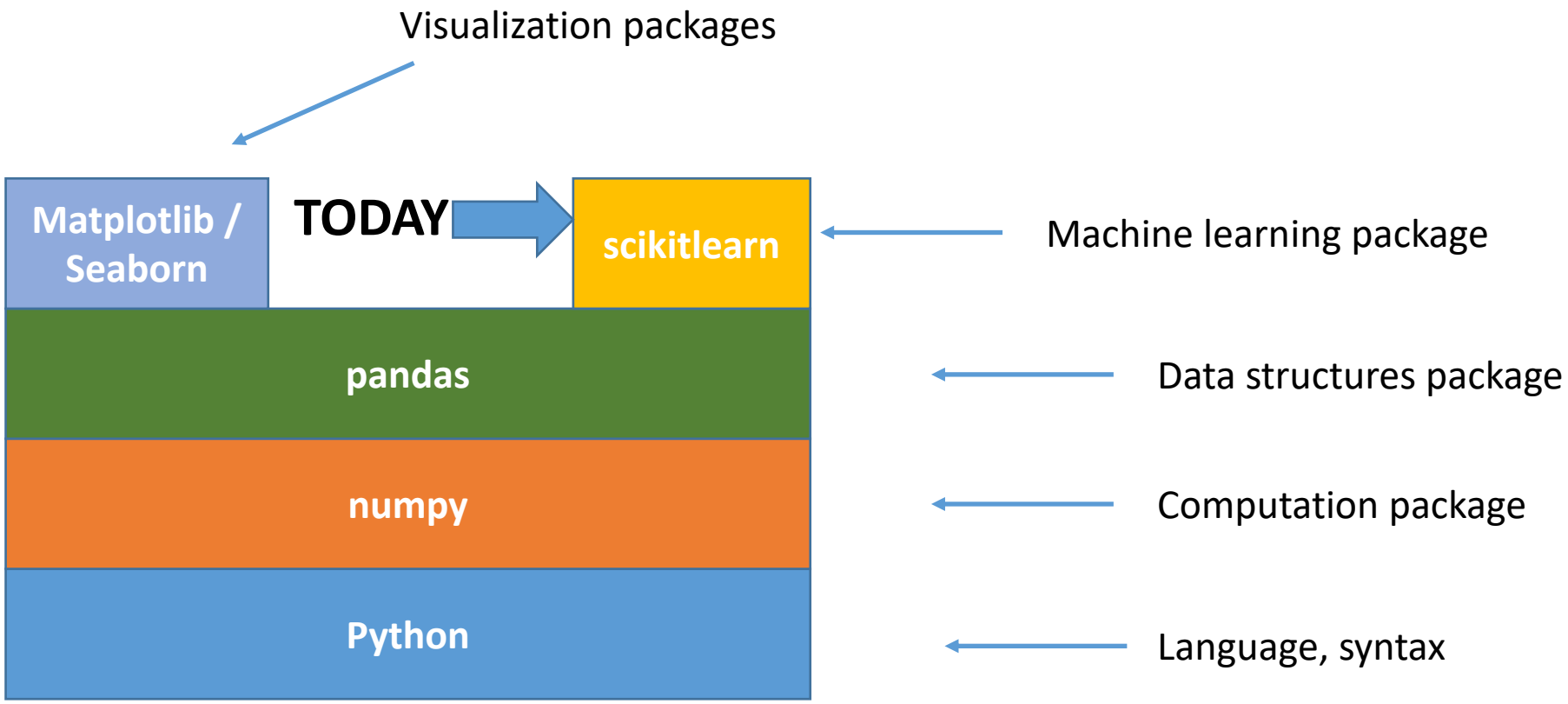
Introduction to Machine Learning Classification for Data Exploration

module 10

This course


Visualization packages
Matplotlib / Seaborn
TODAY
scikitlearn
Machine learning package
pandas
Data structures package
numpy
Computation package
Python
Language, syntax

Machine Learning

Set of techniques to find nontrivial patterns in one table. The main tasks are:

- **Classification**: explain a categorical (usually binary) column given all other columns
- **Clustering**: partition the rows into homogeneous groups
- **Regression**: explain a numeric column given all other columns

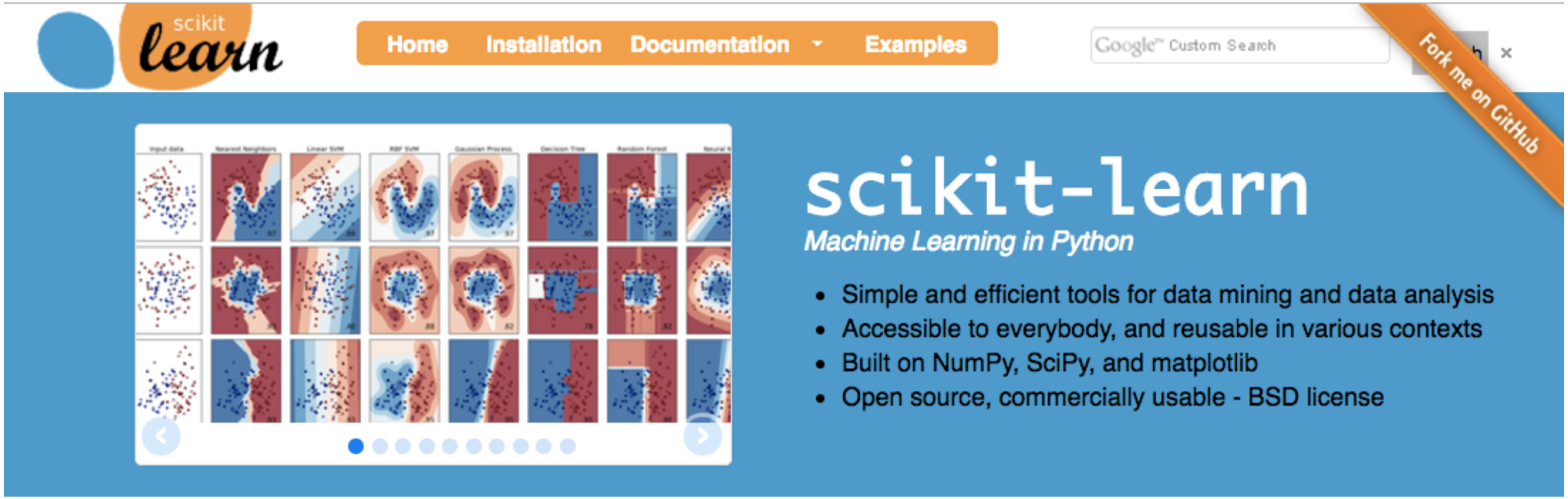
Scikit-learn : ML in Python

scikit-learn is a Python module integrating classic machine learning algorithms in the tightly-knit world of scientific Python packages (numpy, scipy, matplotlib, statsmodel)

Tools for:

- Classification
- Clustering
- Dimensionality Reduction
- Model Selection
- Regression (but if you are a statistician, you'd prefer statsmodel)

Scikit-learn : http://scikit-learn.org/



Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

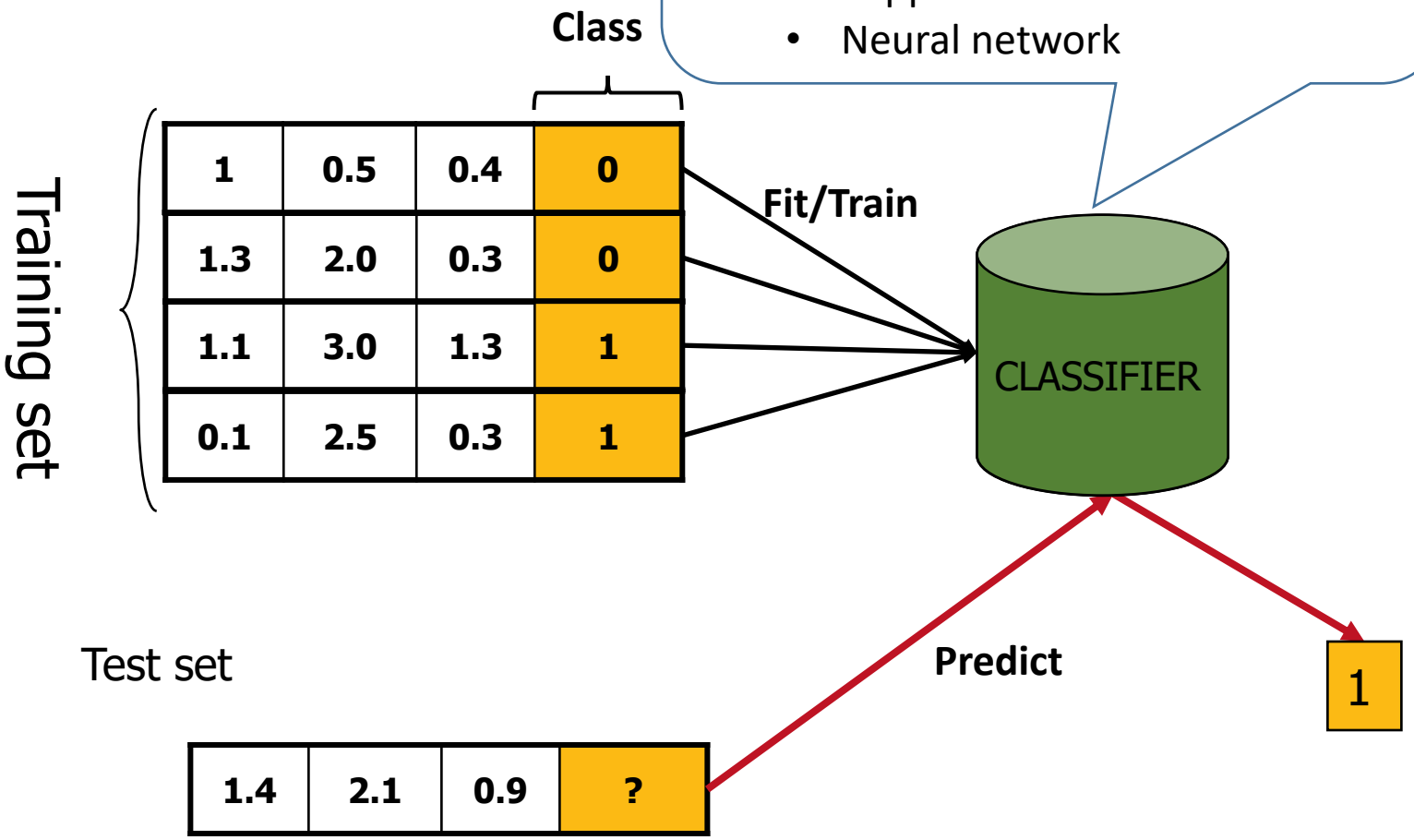
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Classification

- Goal: Explain or predict categorical target (outcome) variable
- Examples: Purchase/no purchase, fraud/no fraud, creditworthy/not creditworthy…
- Each row is a case (customer, student, applicant)
- Each column is a variable
- Target variable is often binary (0/1)

Classification for prediction

- Some classifiers are interpretable
  - Decision trees
  - Logistic regression
- Some aren't
  - Support vector machines
  - Neural network



Classification Jargon:

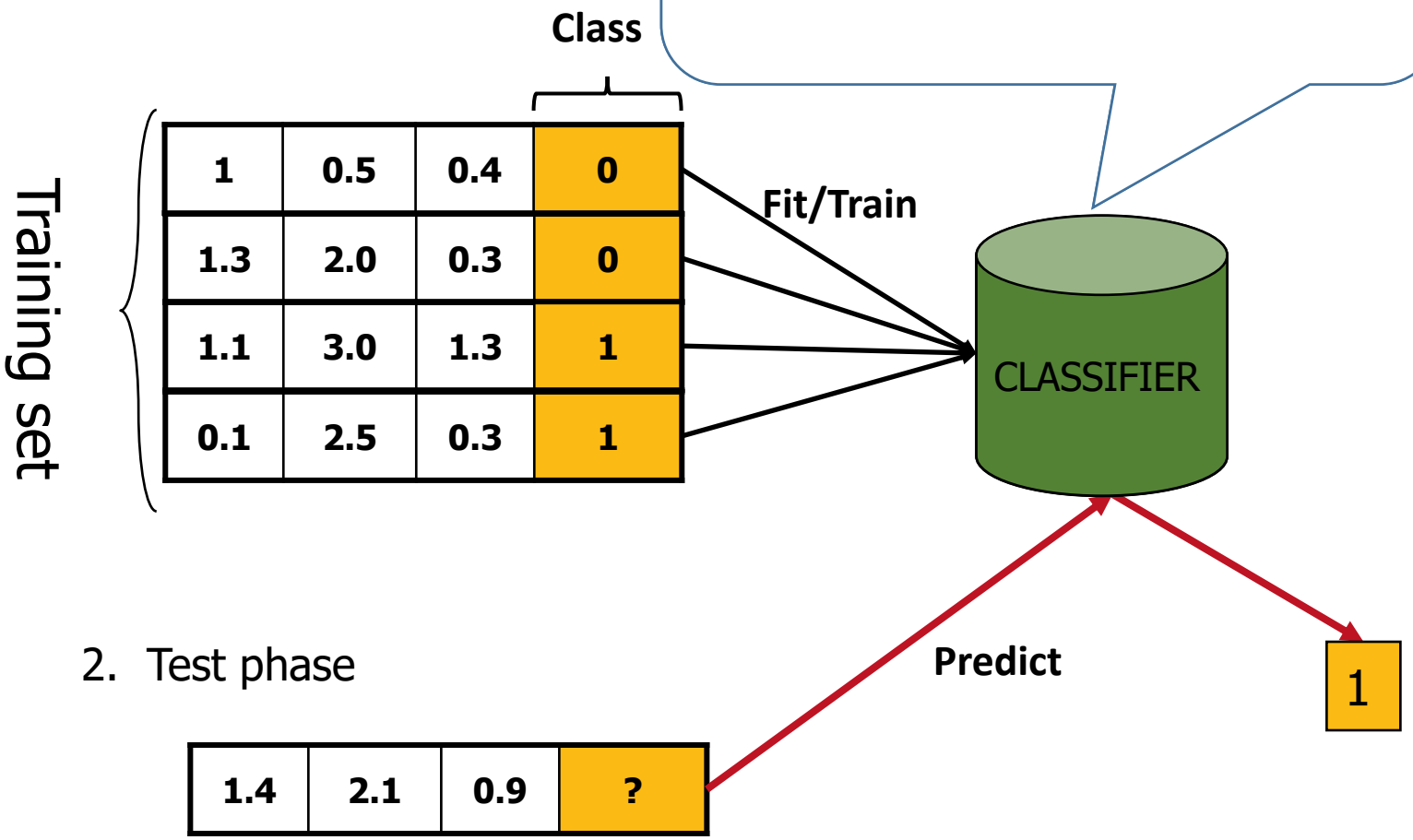
- **Class**: the target attribute that we want to predict
- **Training set**: the table (DataFrame) used to learn
- **Classifier**: the entity that learns the differences between classes
- **Fit/train**: the task of learning
- **Predict**: after training, the task of predicting the class of new objects

Fit and predict in classification

- All models (Classification and regression) implement at least two methods:
  - fit(x, y) - Fit the model to the given dataset
  - predict(x) - Predict the y values associated with the x values

Classification for data exploration

- The classifier must be interpretable
  - Decision trees
  - Logistic regression



Classification Jargon:

- **Class**: the target attribute that we want to predict
- **Training set**: the table (DataFrame) used to learn
- **Classifier**: the entity that learns the differences between classes
- **Fit/train**: the task of learning
- **Predict**: after training, the task of predicting the class of new objects

Classification for Data Exploration

Decision Trees

To visualize decision trees – Windows

- Before we start, we need to install a couple of things which will make it possible to visualize decision trees.

Decision Tree in a nutshell

GOAL: build a tree of decisions to predict the class of an object

1. Recursively partition the training set with the goal of minimizing classification errors, using the “most” helpful attribute
2. Many methods to choose the attribute for partitioning
   - Maximize information gain
   - Minimize gini impurity

Let’s see an example…

Example: Fountain owners

- Goal: Classify 24 households as owning or not owning fountains

- Attributes = Income, Lot Size

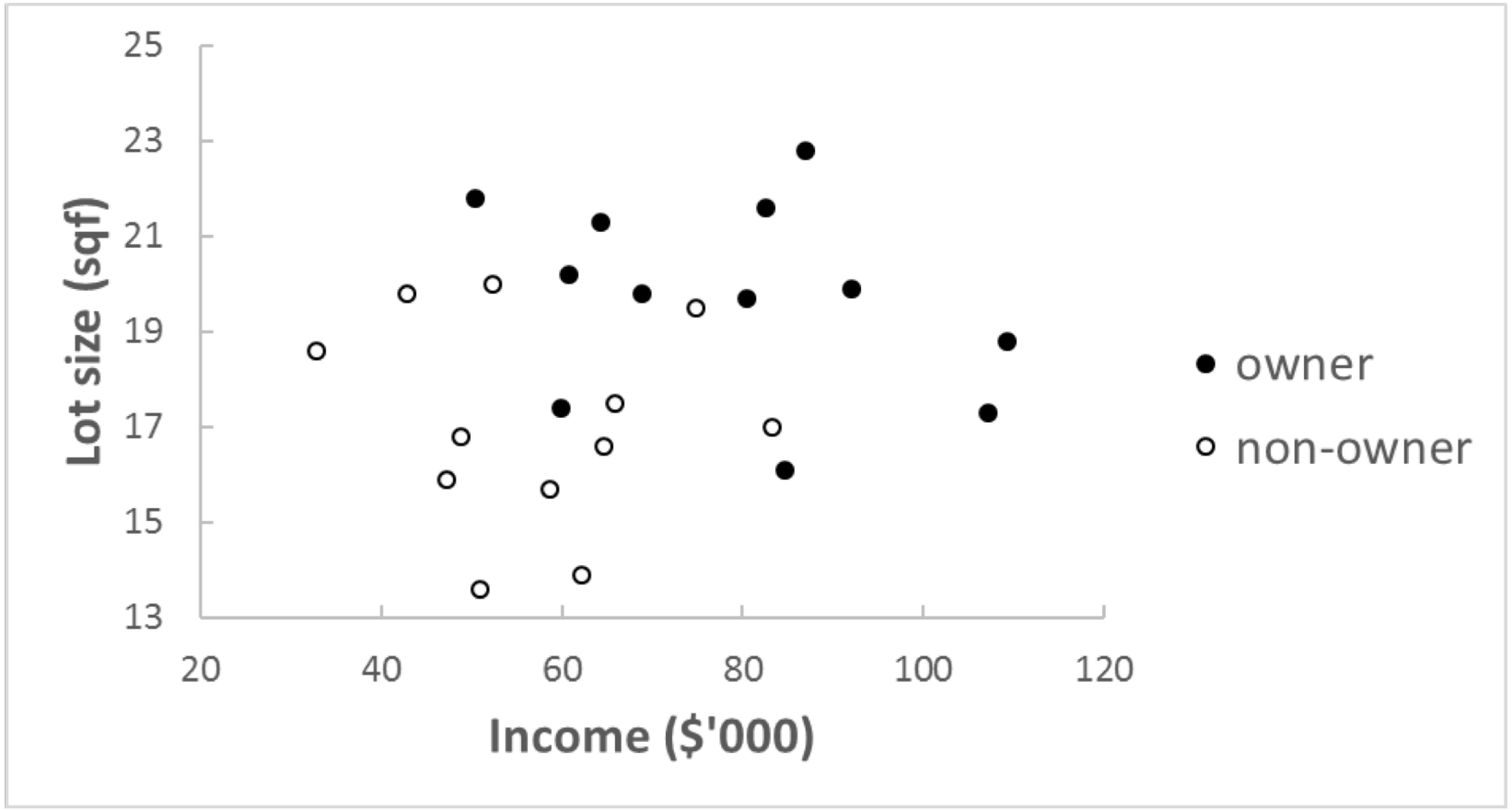
Training set

Income	Lot size	Ownership
59.9	17.4	owner
84.6	16.1	owner
64.3	21.3	owner
60.8	20.2	owner
86.9	22.8	owner
109.3	18.8	owner
107.1	17.3	owner
82.5	21.6	owner
68.9	19.8	owner
92	19.9	owner
50.4	21.8	owner
80.4	19.7	owner
74.9	19.5	non-owner
52.3	20	non-owner
64.6	16.6	non-owner
42.8	19.8	non-owner
83.3	17	non-owner
48.9	16.8	non-owner
58.6	15.7	non-owner
65.9	17.5	non-owner
47.2	15.9	non-owner
32.8	18.6	non-owner
50.9	13.6	non-owner
62.2	13.9	non-owner

We want to build a tree that tells us the difference between:

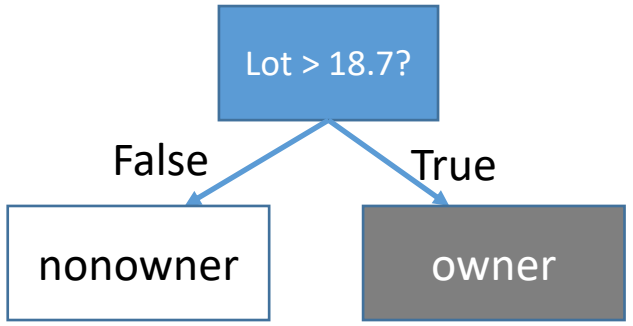
- **Owners: those who own a fountain**
- **Non-owners: those who do not**

Here is the data set



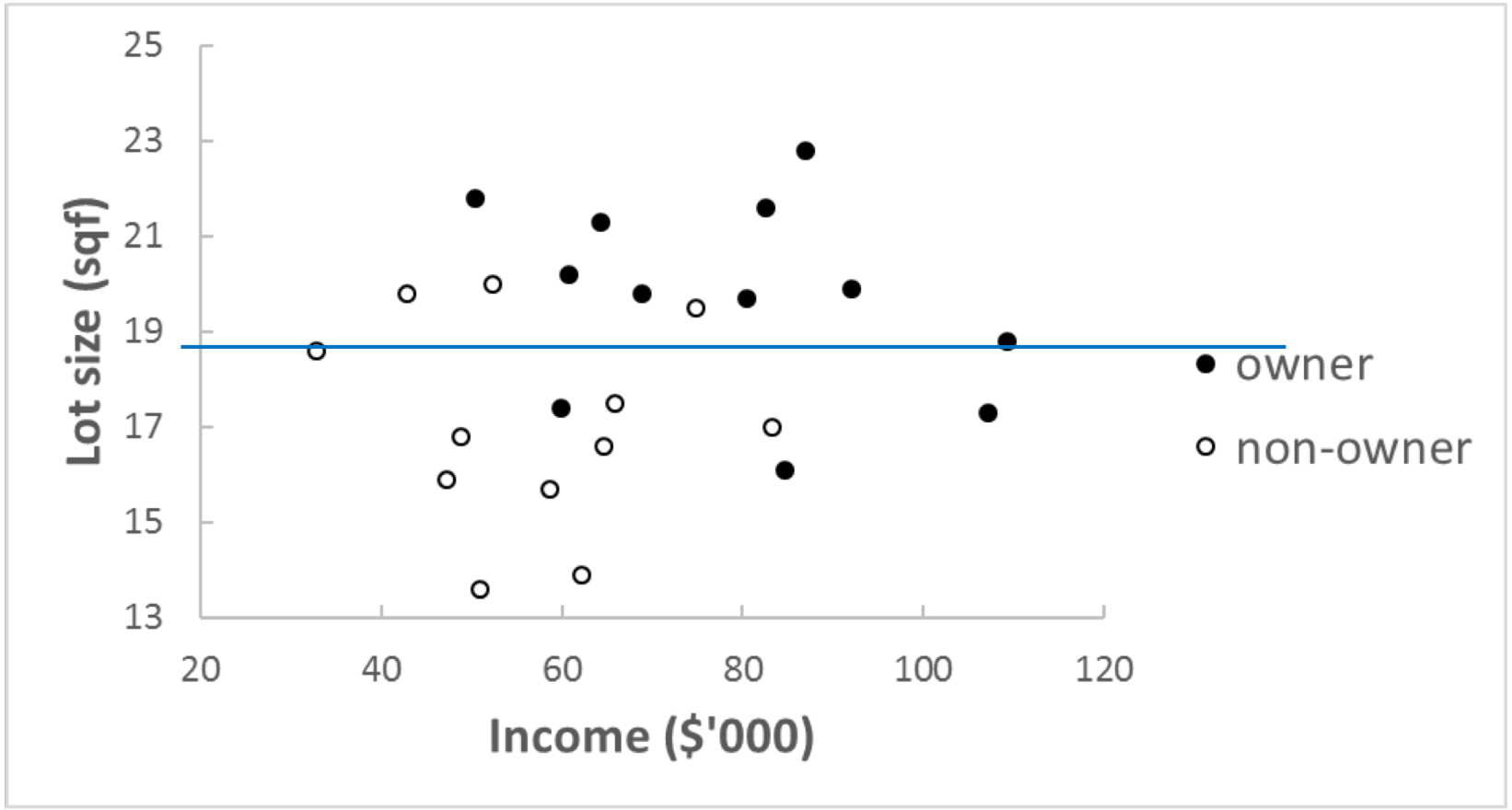
First split

Decision tree of depth 1



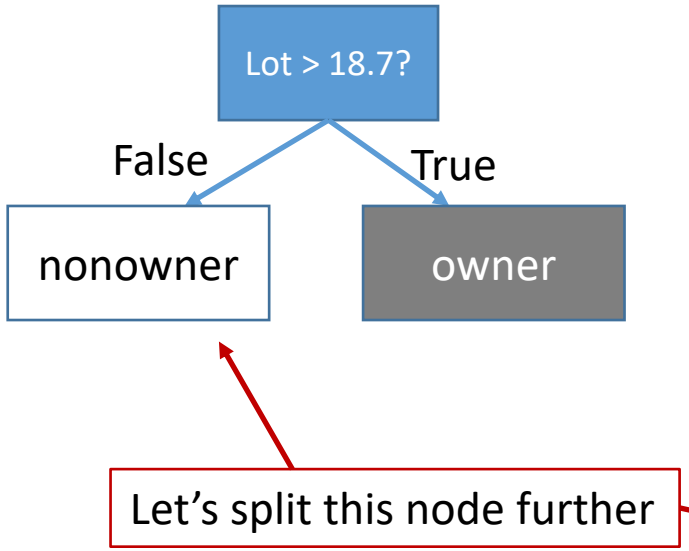
Suppose that the one above is the final tree.

How many classification errors does this tree make on the training set?

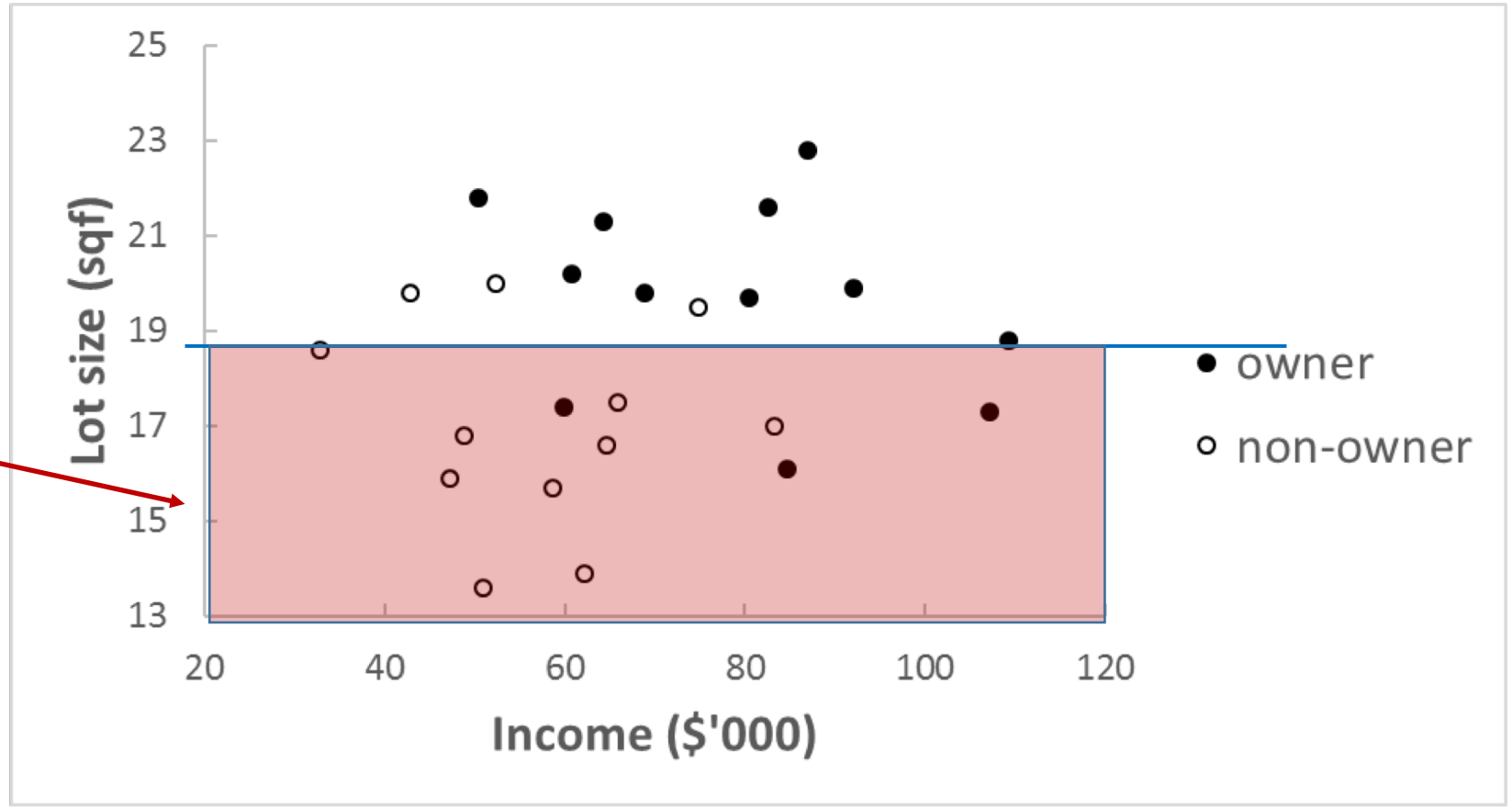


First split

Decision tree of depth 1

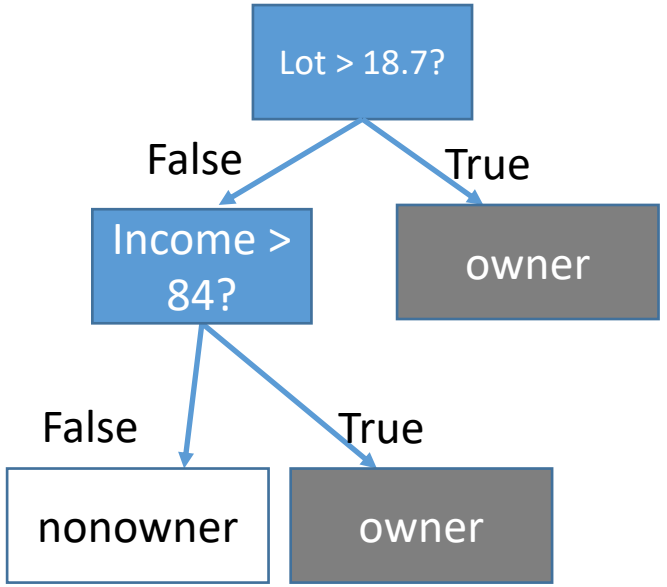


Let's split this node further



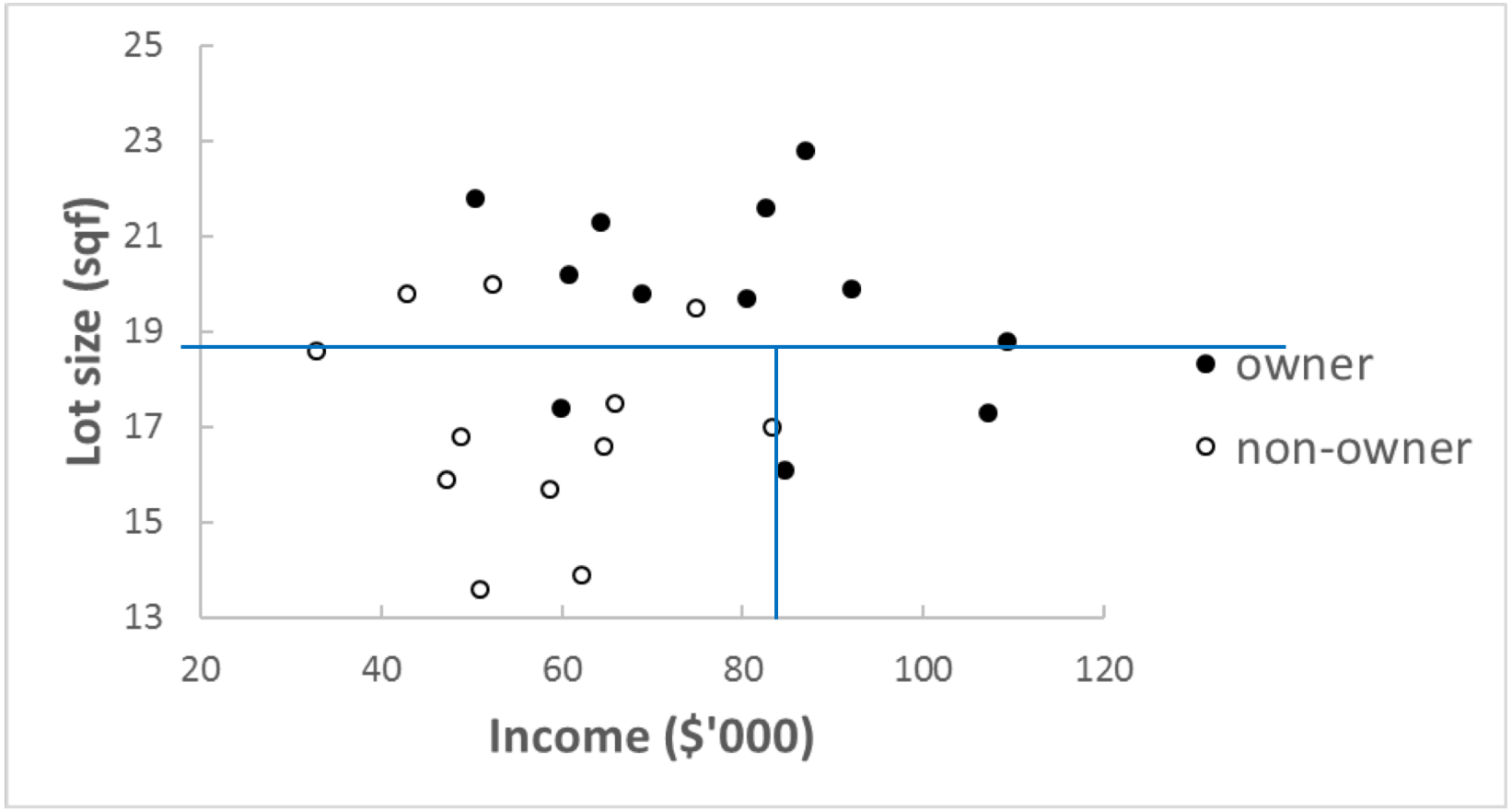
Second split

Decision tree of depth 1



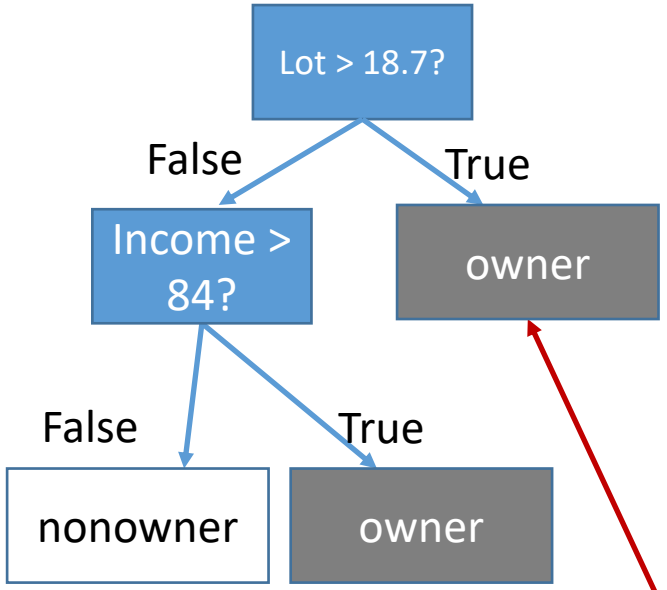
Suppose that the one above is the final tree.

How many classification errors does this tree make on the training set?

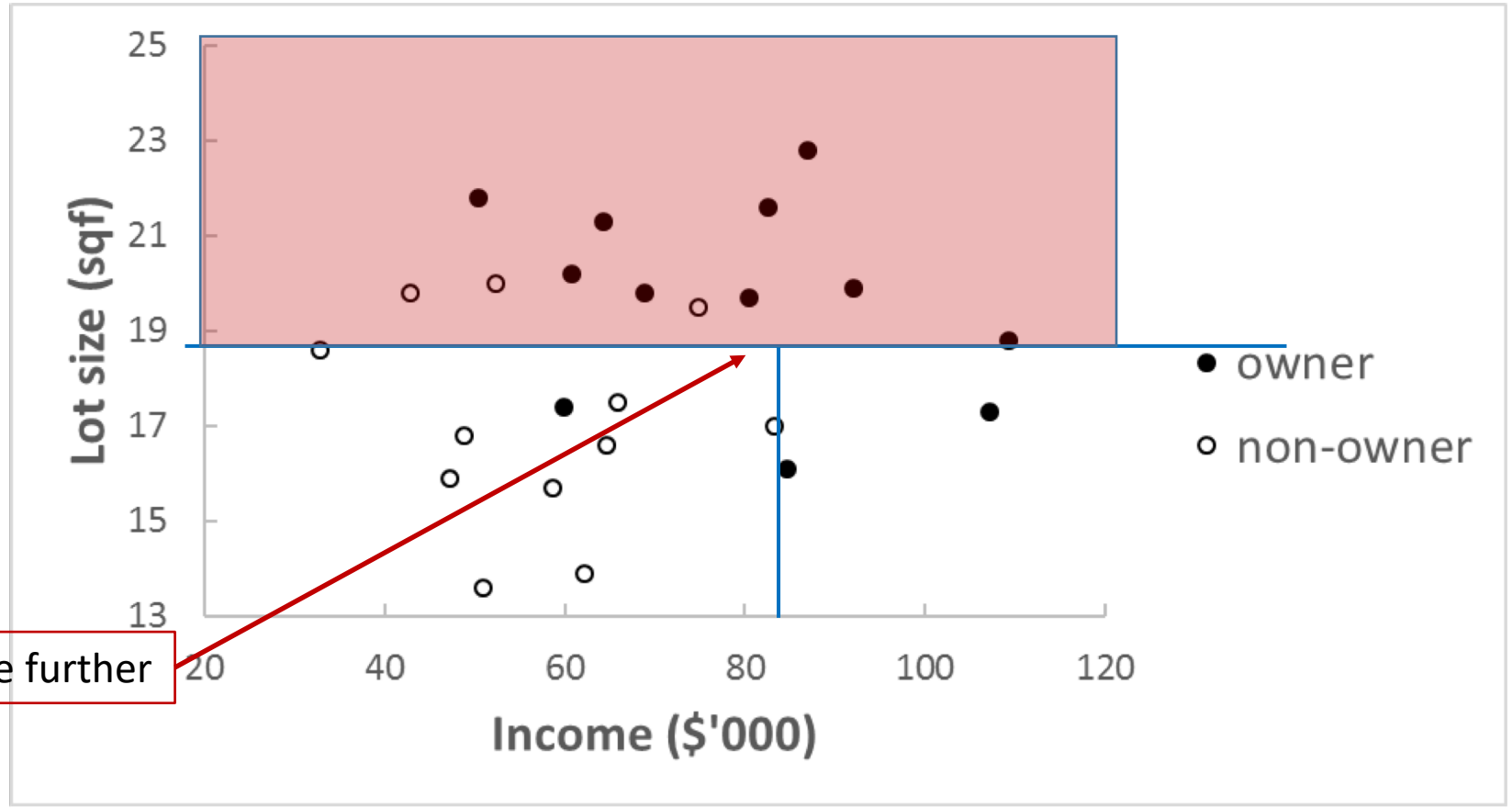


Second split

Decision tree of depth 2

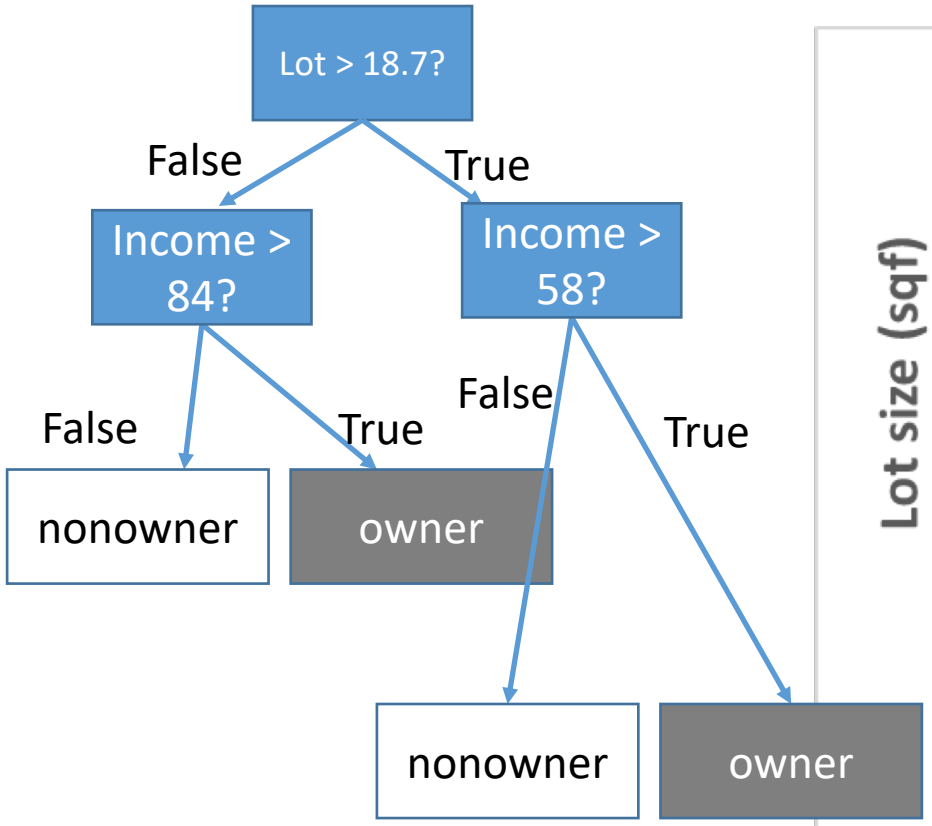


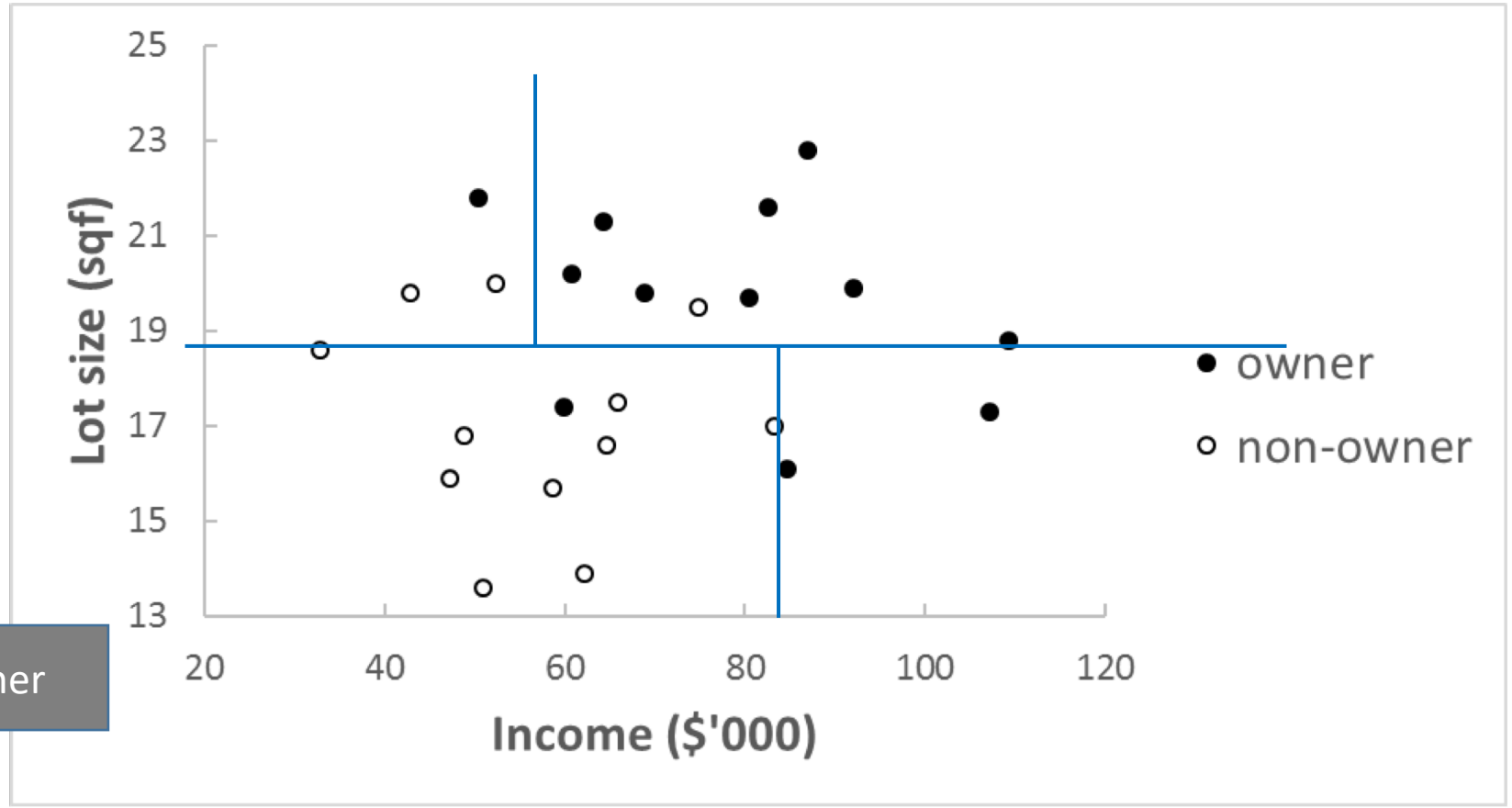
Let's split this node further



Second split

Decision tree of depth 2





How many classification errors does this tree make on the training set?

How to train a decision tree in scikit learn

```
import sklearn as sk
import sklearn.tree as tree
from IPython.display import Image
import pydotplus
```

```
dt = tree.DecisionTreeClassifier(max_depth=2)
```

For better interpretability, make your tree shallow

```
dt.fit(X,Y)
```

```
dt_feature_names = list(X.columns)
dt_target_names = np.array(Y.unique(),dtype=np.string_)
tree.export_graphviz(dt, out_file='tree.dot',
    feature_names=dt_feature_names, class_names=dt_target_names,
    filled=True)
graph = pydotplus.graph_from_dot_file('tree.dot')
Image(graph.create_png())
```

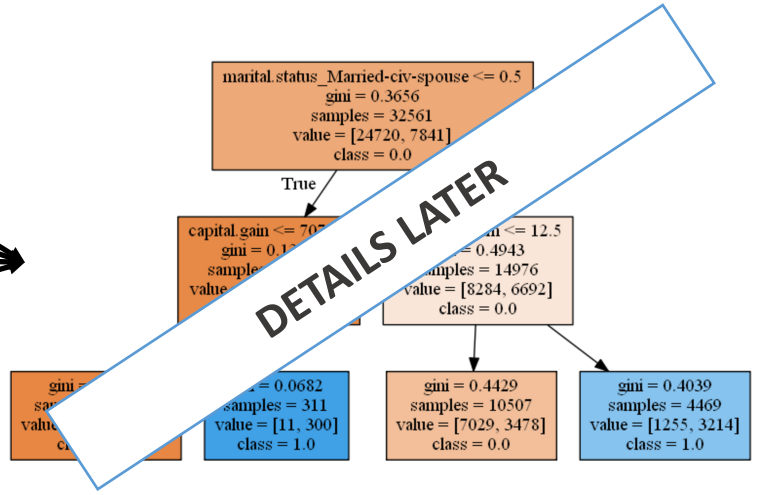
To view tree

X

Y

1	0.5	0.4	0
1.3	2.0	0.3	0
1.1	3.0	1.3	1
0.1	2.5	0.3	1

Fit/Train



Predicting High Income

Case study similar to project

Today's data set

- https://www.kaggle.com/uciml/adult-census-income
- This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). *The prediction task is to determine whether a person makes over $50K a year*.

```
df.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	cap
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0

Let's try to train a decision tree to predict income

```
dt = tree.DecisionTreeClassifier(max_depth=2)
```

```
# can't deal with strings
dt.fit(X=df.drop('income',axis=1),y=df.income)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-c05d087851c1> in <module>()
      1 # can't deal with strings
----> 2 dt.fit(X=df.drop('income',axis=1),y=df.income)

C:\Users\msamorani\AppData\Local\Enthought\Canopy\User\lib\site-packages\sklearn\tree\tree.pyc in f
 check_input, X_idx_sorted)
```

We need all numeric values
So, let's clean up the data set

CLEANING

Transform the "?" into NaN

```
df.replace('?', np.NaN, inplace=True)
```

CLEANING

```
for c in df.columns:
    print (c + ': '+ str(df[c].nunique()))
```

```
age: 73
workclass: 8
fnlwgt: 21648
education: 16
education.num: 16
marital.status: 7
occupation: 14
relationship: 6
race: 5
sex: 2
capital.gain: 119
capital.loss: 92
hours.per.week: 94
native.country: 41
income: 2
```

Let's make dummies for 'workclass','marital.status','occupation','relationship','race','sex', and let's exclude education (as education.num is its numeric code)

```
df2 = pd.get_dummies(data=df,columns=['workclass','marital.status','occupation','relationship','race','sex', 'native.country'])
```

```
df2.drop(['education'],axis=1,inplace=True)
```

CLEANING

Income must become binary too

```
df2.income = df2.apply(lambda row : 1.0 if row['income'].startswith('>') else 0.0, axis=1)
```

```
df2.head()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	income	workclass_Federal-gov	v g
0	90	77053	9	0	4356	40	0.0	0	0
1	82	132870	9	0	4356	18	0.0	0	0
2	66	186061	10	0	4356	40	0.0	0	0
3	54	140359	4	0	3900	40	0.0	0	0
4	41	264663	10	0	3900	40	0.0	0	0

5 rows × 90 columns

```
df2.income.mean()
```

```
0.24080955744602439
```

TRAIN THE DECISION TREE

Train the decision tree

Make X and Y

```
X = df2.drop('income',axis=1)
Y = df2.income
```

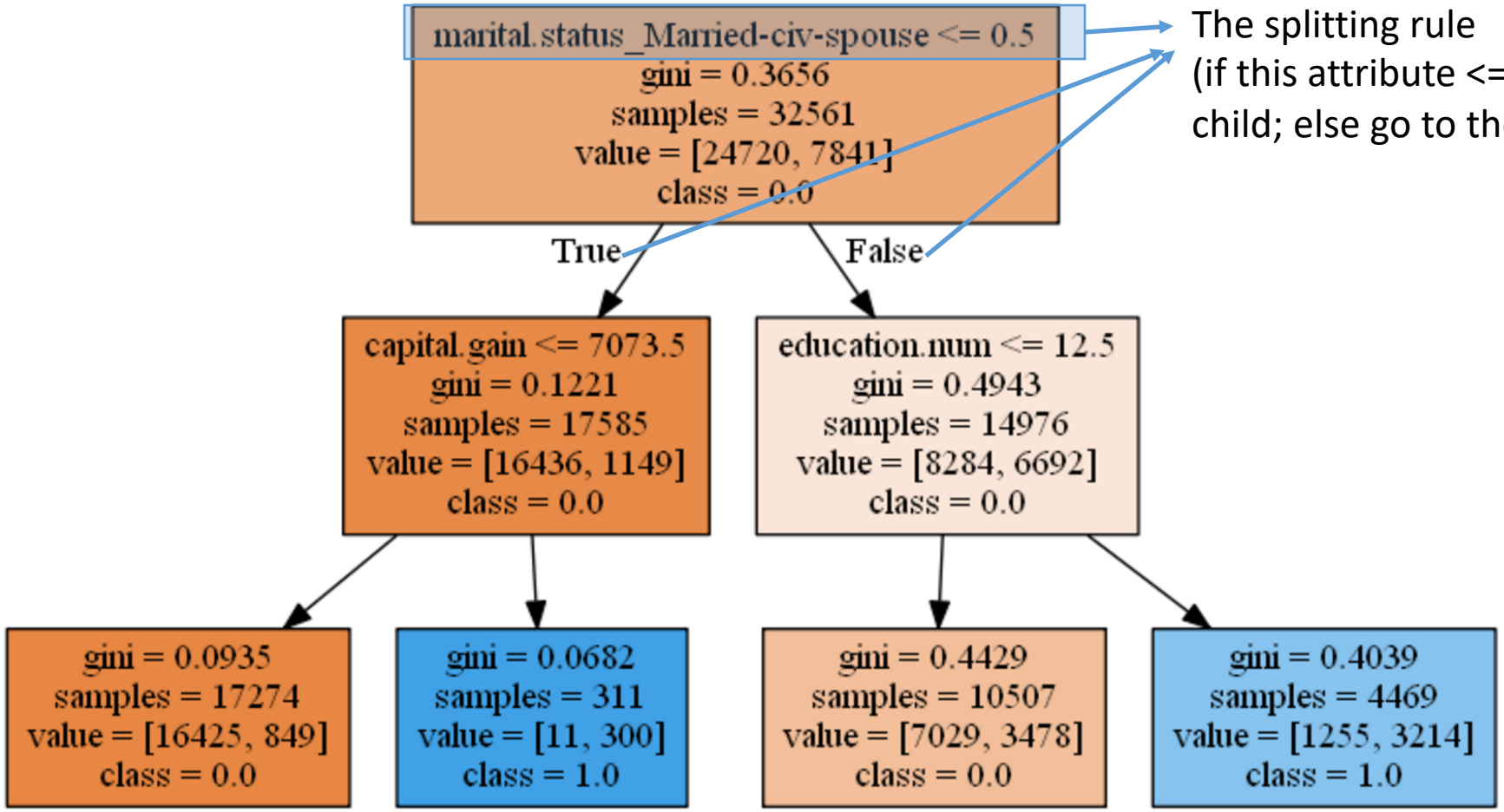
Build the tree

```
dt = tree.DecisionTreeClassifier(max_depth=2)
dt.fit(X,Y)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

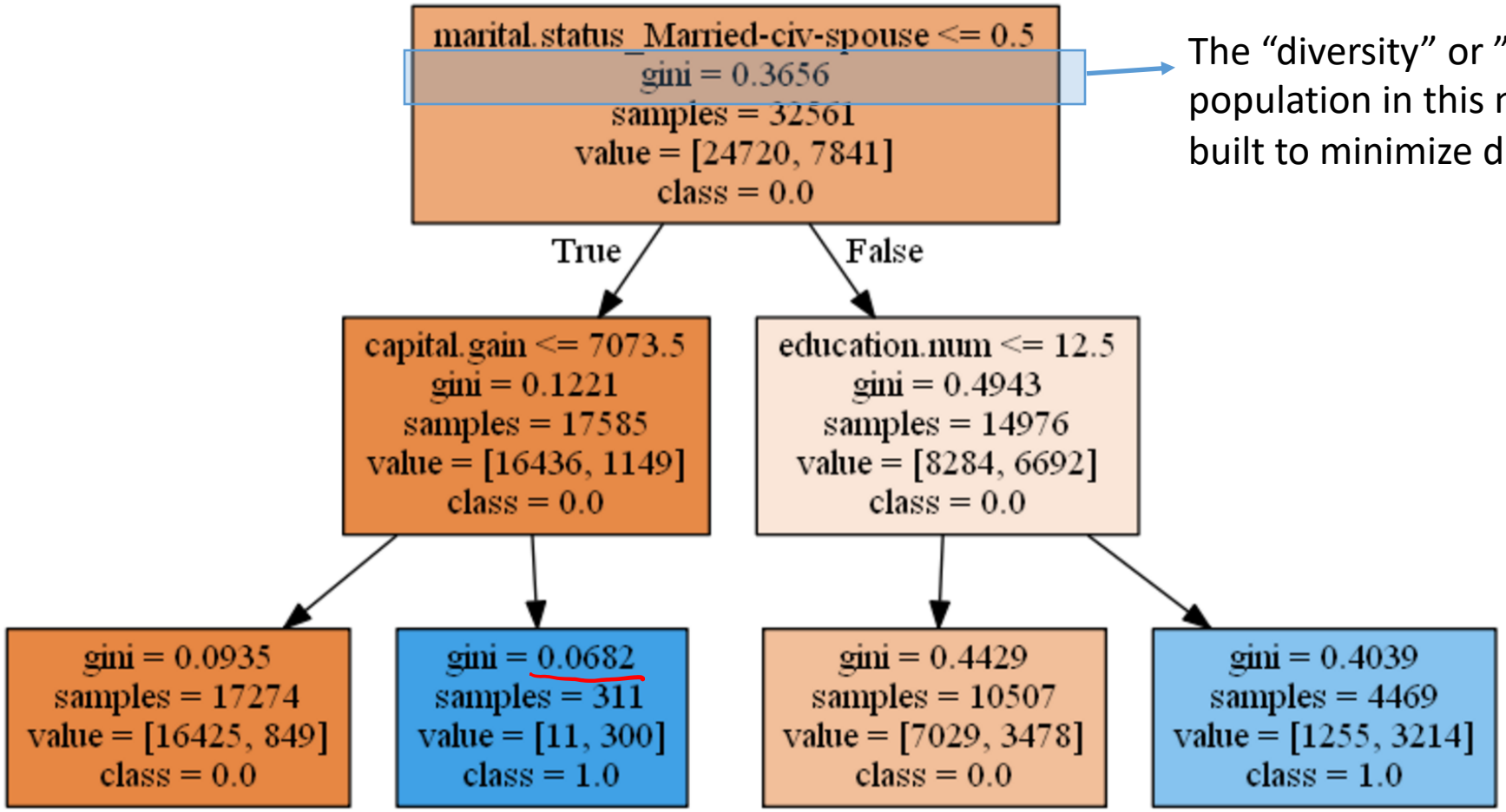
```
dt_feature_names = list(X.columns)
dt_target_names = np.array(Y.unique(),dtype=np.string_)
tree.export_graphviz(dt, out_file='tree.dot',
    feature_names=dt_feature_names, class_names=dt_target_names,
    filled=True)
graph = pydotplus.graph_from_dot_file('tree.dot')
Image(graph.create_png())
```

Interpreting the Tree



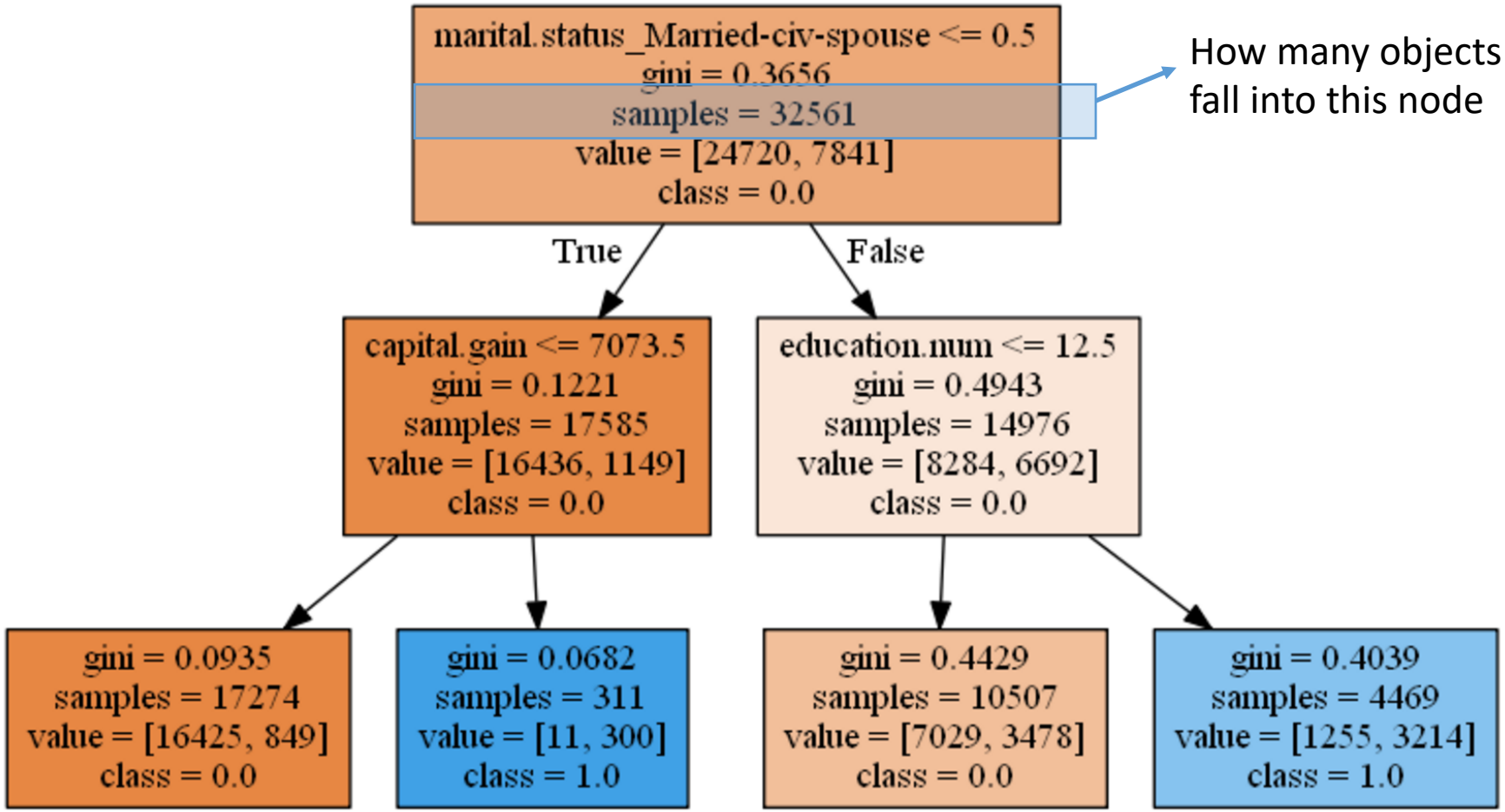
The splitting rule
(if this attribute <= 0.5, go to the left child; else go to the right child)

Interpreting the Tree

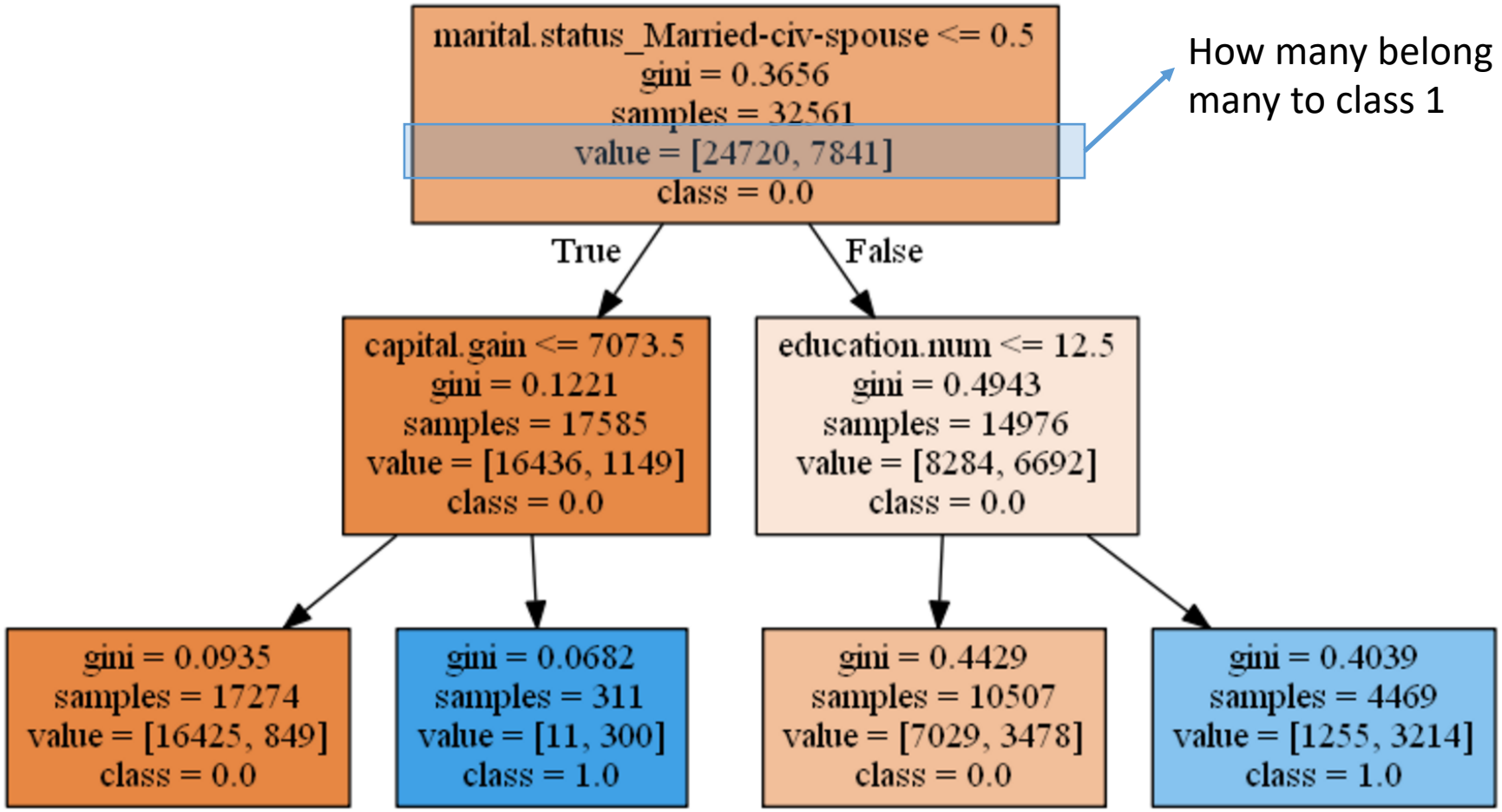


The “diversity” or ”impurity” of the population in this node. The tree is built to minimize diversity in its leaves

Interpreting the Tree

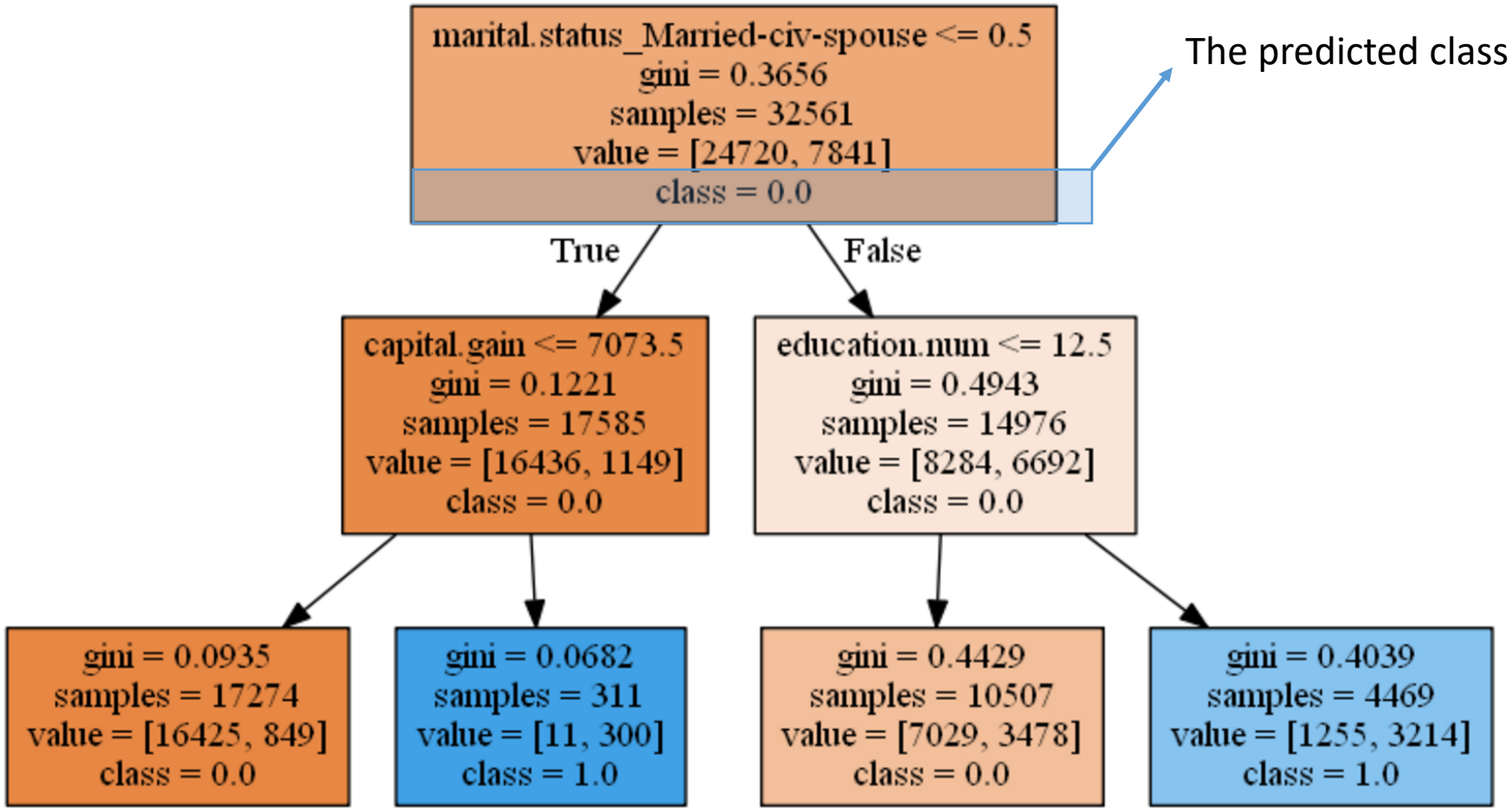


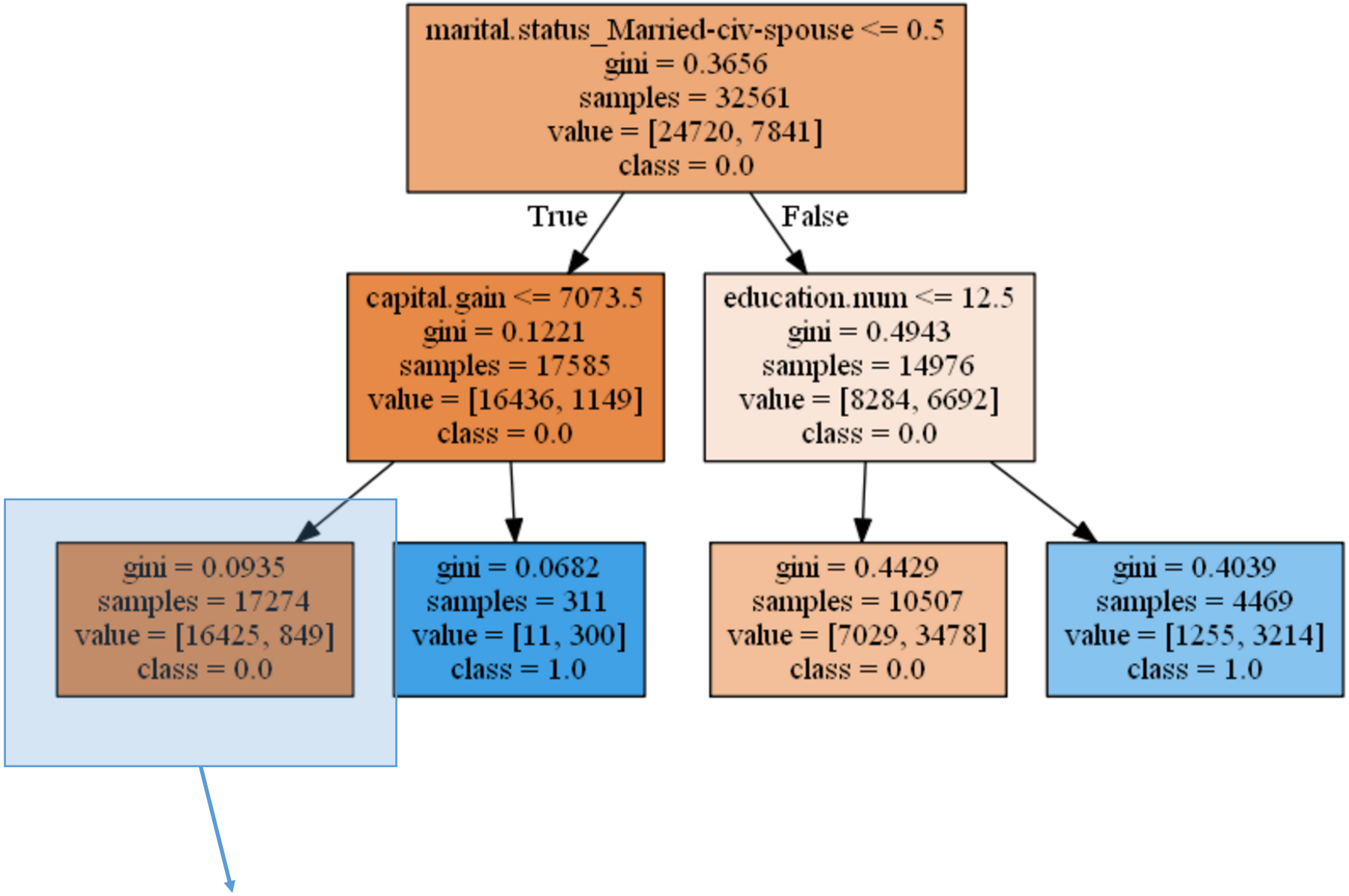
Interpreting the Tree



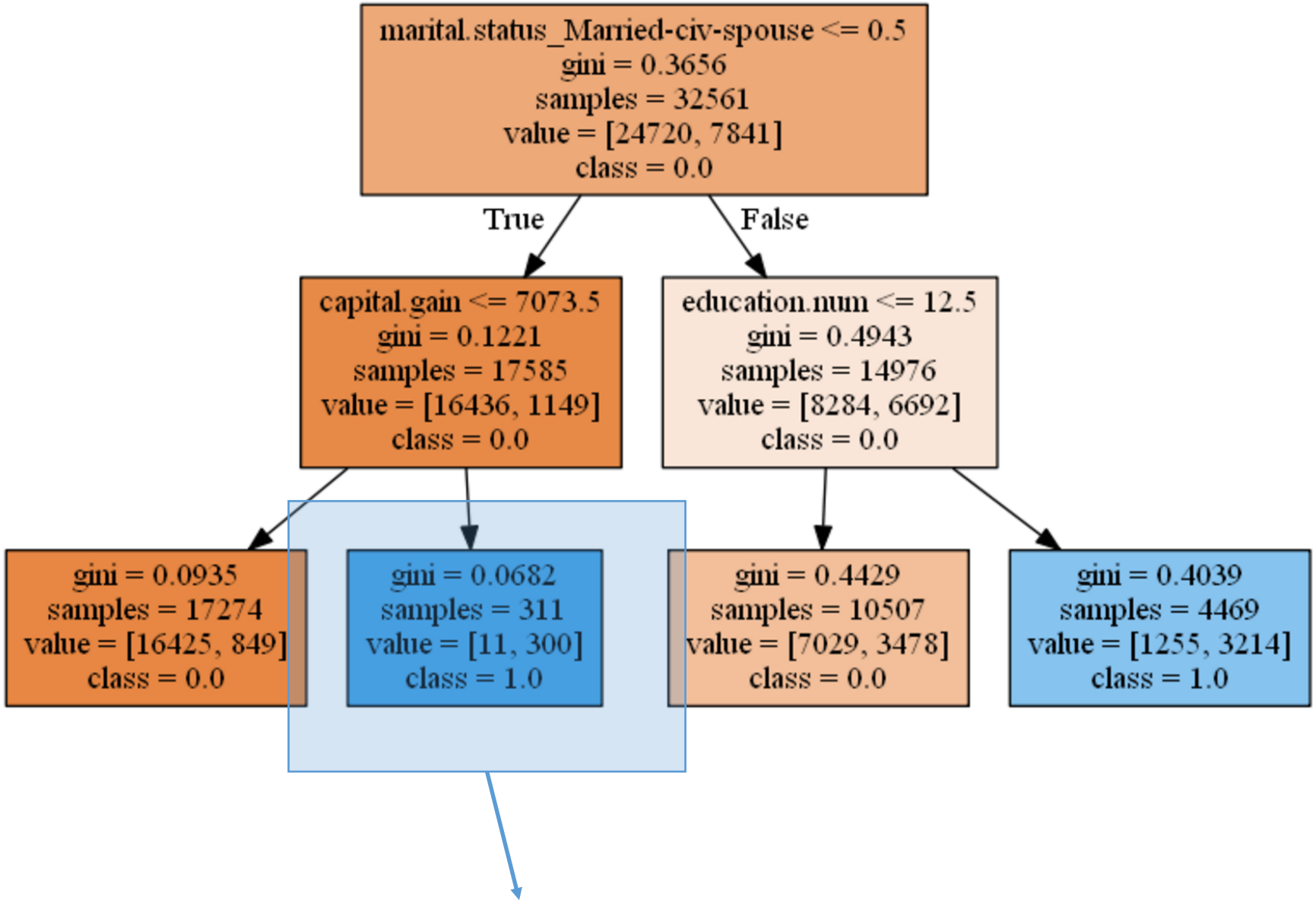
How many belong to class 0 and how many to class 1

Interpreting the Tree

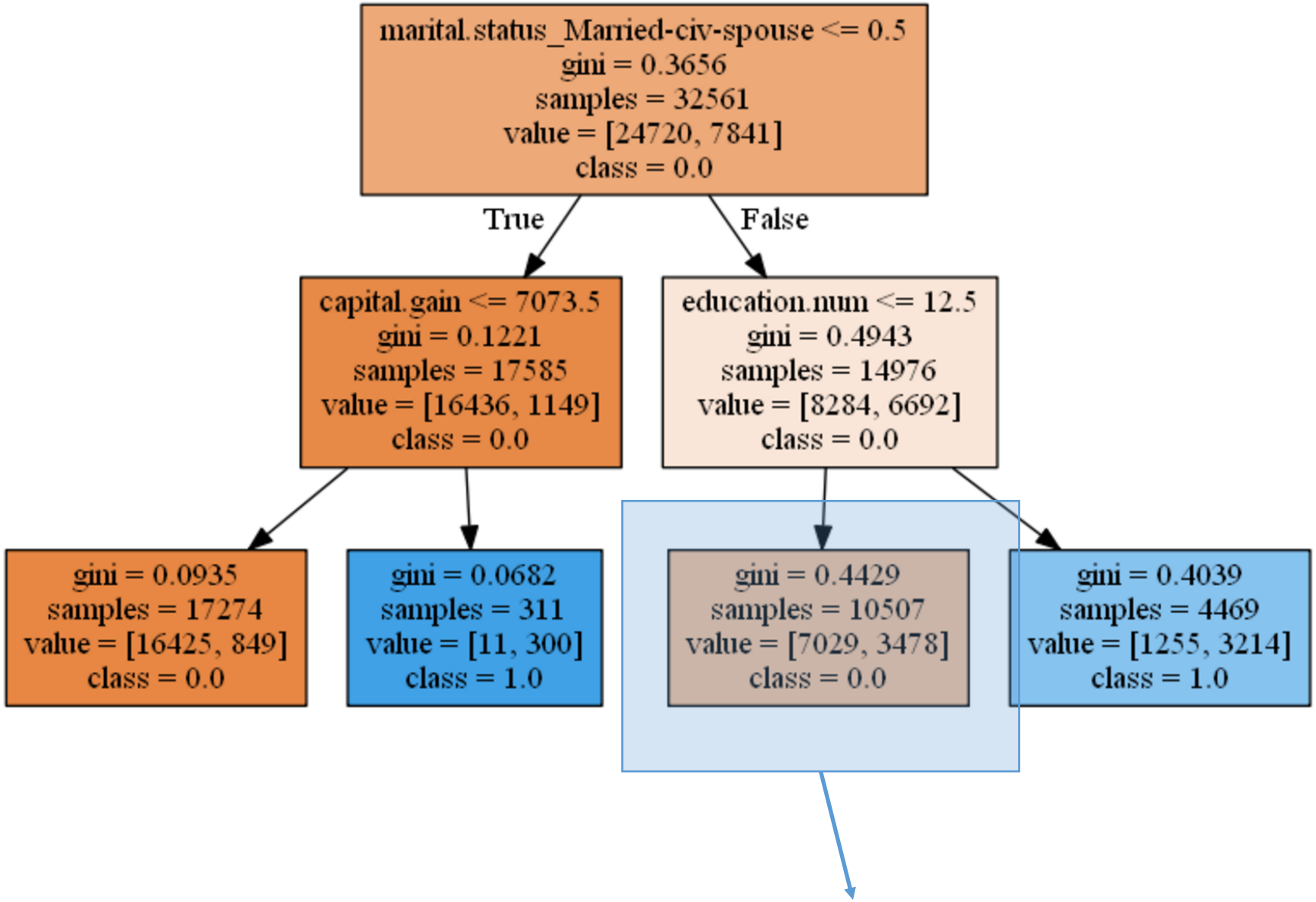




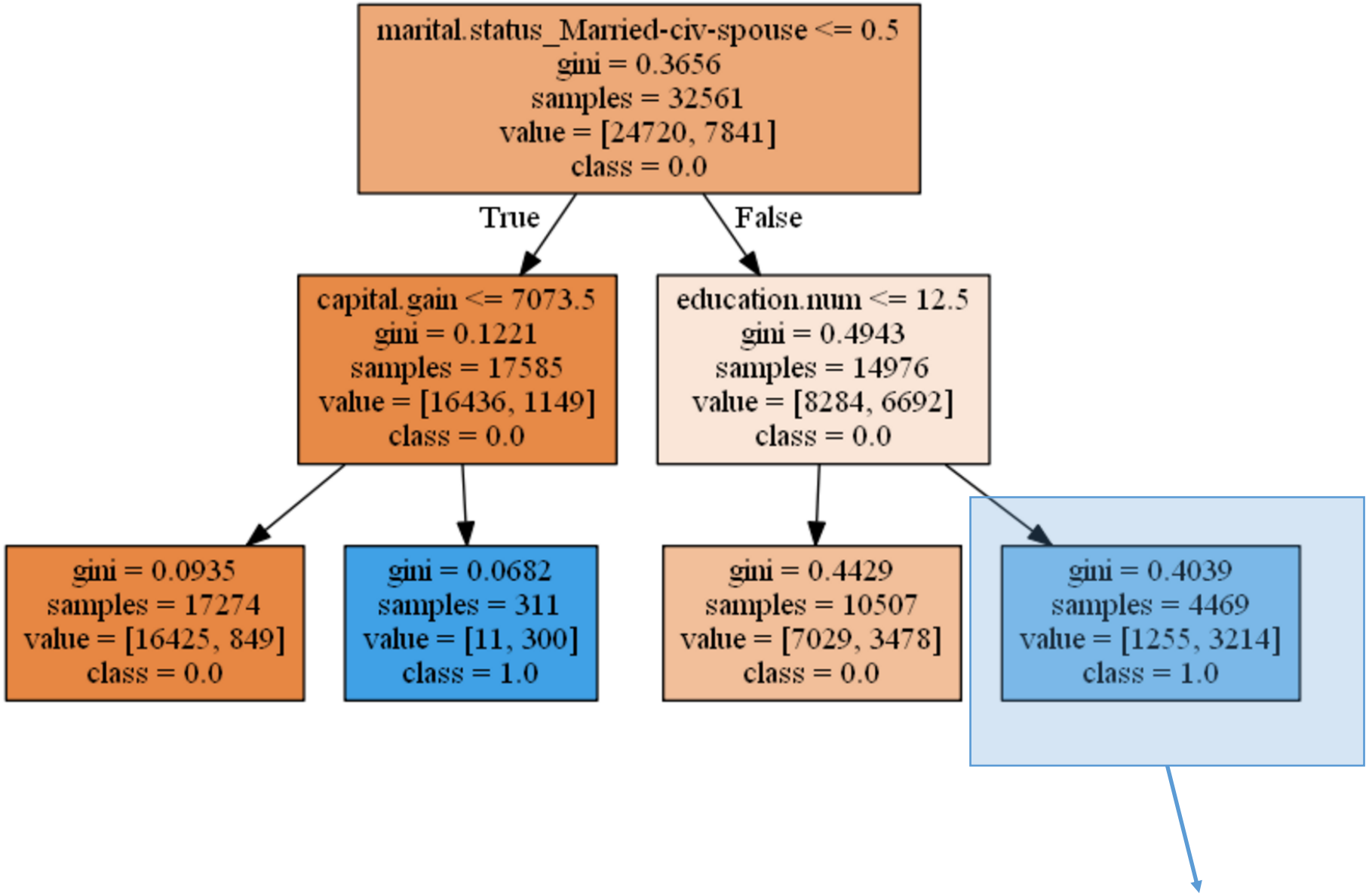
Those not married civilly and with low capital gain have low income



Those not married civilly and with high capital gain have high income



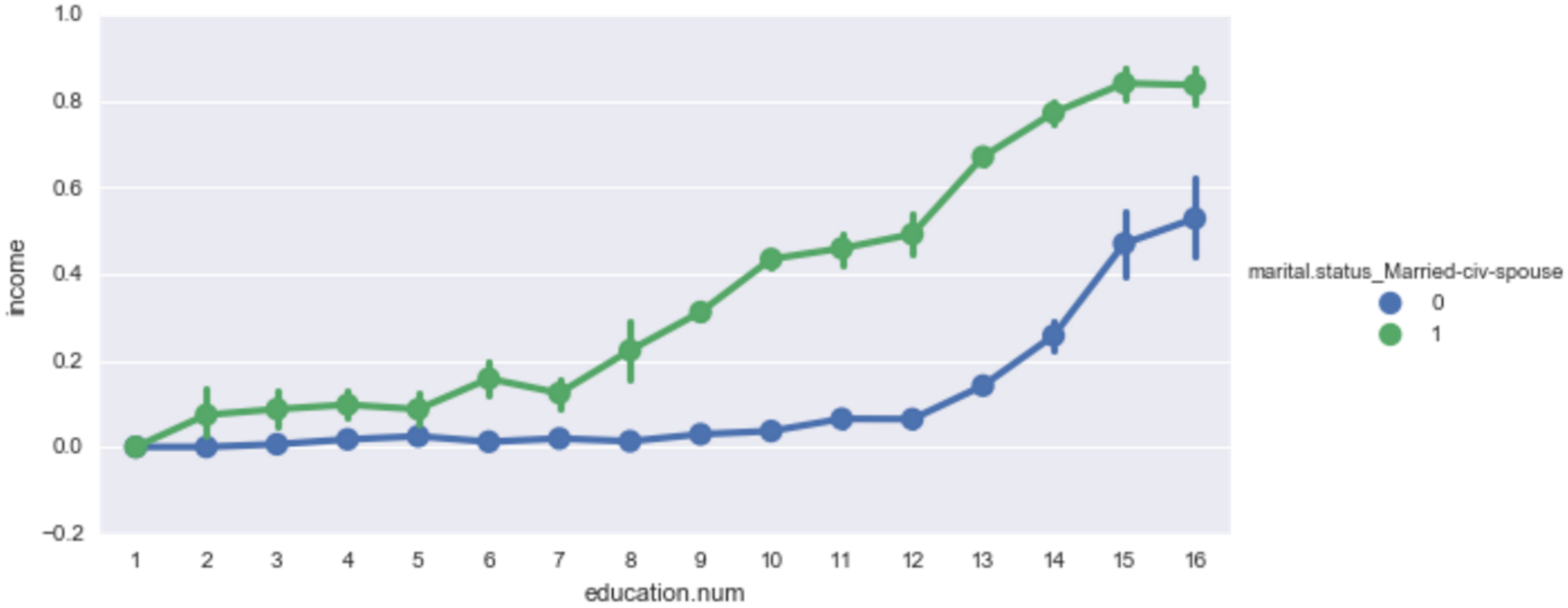
Those married civilly and with low education have low income



Those married civilly and with high education have high income

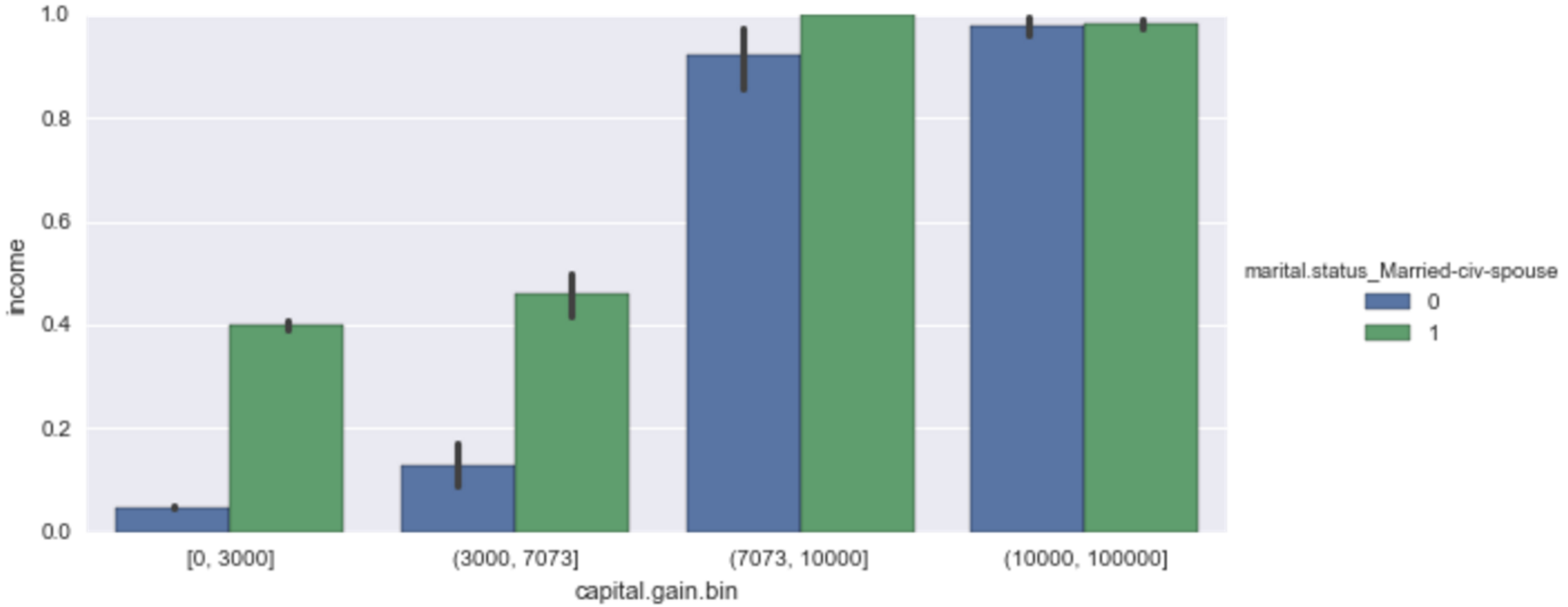
```
sns.factorplot(x='education.num',y='income',data=df2,
               hue='marital.status_Married-civ-spouse', kind='point',aspect=2)
```

```
<seaborn.axisgrid.FacetGrid at 0xdcd4dd8>
```



Education.num = 12.5 is a good threshold to separate low income from high income, but it is more effective for those married civilly (right part of the tree)

```
df2['capital.gain.bin']=pd.cut(df2['capital.gain'],bins=[0,3000,7073,10000, 100000],
                               include_lowest=True)
sns.factorplot(x='capital.gain.bin',y='income',data=df2,
               hue='marital.status_Married-civ-spouse', kind='bar',aspect=2)
df2.drop('capital.gain.bin',axis=1,inplace=True)
```



Capital.gain = 7073 is a good threshold to separate low income from high income, but it is more effective for those NOT married civilly (left part of the tree)