# Supervised learning pipelines

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON


Dr. Chris Anagnostopoulos

Honorary Associate Professor

# Labeled data

- Feature variables (shorthand: `X` )
- Labels or class (shorthand: `y` )

```
credit_scoring.head(4)
```

```
  checking_status  duration  ...  foreign_worker class
0            '<0'         6  ...             yes  good
1      '0<=X<200'        48  ...             yes   bad
2   'no checking'        12  ...             yes  good
3            '<0'        42  ...             yes  good
```

# Feature engineering

- Most classifiers expect numeric features
- Need to convert string columns to numbers

Preprocess using `LabelEncoder` from `sklearn.preprocessing` :

```
le = LabelEncoder()
le.fit_transform(credit_scoring['checking_status'])[:4]
```

```
array([1, 0, 3, 1])
```

# Model fitting

- `.fit(features, labels)`
- `.predict(features)`

```
features, labels = credit_scoring.drop('class', 1), credit_scoring['class']
model_nb = GaussianNB()
model_nb.fit(features, labels)
model_nb.predict(features.head(5))
```

```
['good' 'bad' 'good' 'bad' 'good']
```

60% accuracy on first 5 examples.

# Model selection

- `.fit()` optimizes the parameters of the given model
- What about other models?

`AdaBoostClassifier` outperforms `GaussianNB` on first five data points:

```
model_ab = AdaBoostClassifier()
model_ab.fit(features, labels)
model_ab.predict(features.head(5))
numpy.array(labels[0:5])
```

```
['good' 'bad' 'good' 'good' 'bad']
['good' 'bad' 'good' 'good' 'bad']
```

# Performance assessment

Larger sample sizes $\Rightarrow$ better accuracy estimates:

```
from sklearn.metrics import accuracy_score
accuracy_score(labels, model_nb.predict(features)) # naive bayes
```

```
0.706
```

```
accuracy_score(labels, model_ab.predict(features)) # adaboost
```

```
0.802
```

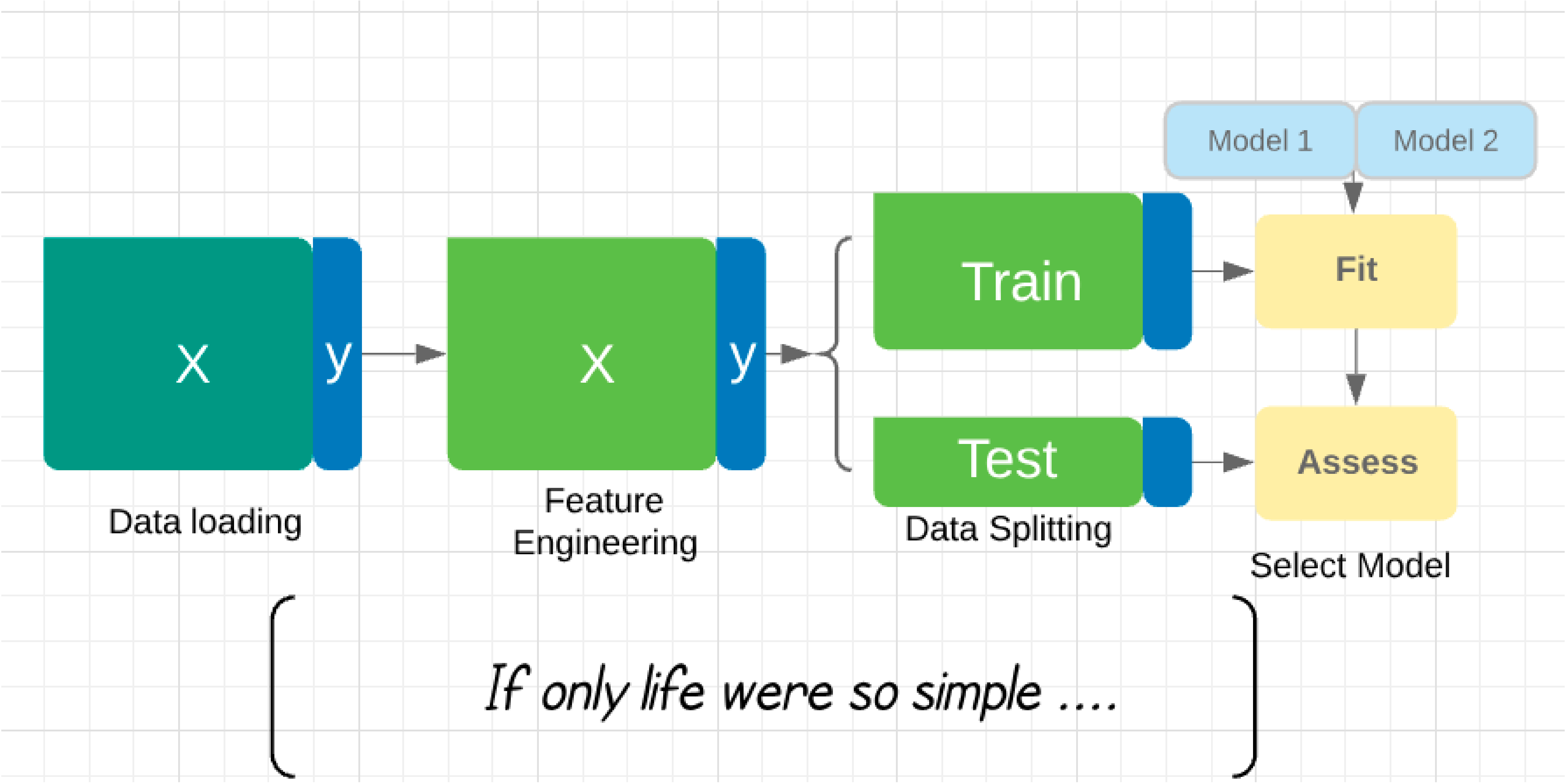
What is wrong with this calculation?

# Overfitting and data splitting

> **Overfitting**: a model will always perform better on the data it was trained on than on unseen data.

Train on `X_train, y_train`, assess accuracy on `X_test, y_test`:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
GaussianNB().fit(X_train, y_train).predict(X_test)
```


Model 1
Model 2
X
y
X
y
Train
Test
Fit
Assess
Data loading
Feature Engineering
Data Splitting
Select Model
If only life were so simple ....

# So, what is this course about?

1. Scalable ways to tune your pipeline.
2. Making sure your predictions are relevant by involving domain experts.
3. Making sure your model continues to perform well over time.
4. Fitting models when you don't have enough labels.

# Could you have prevented the mortgage crisis?

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

# Model complexity and overfitting

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

Dr. Chris Anagnostopoulos
Honorary Associate Professor

# What is model complexity?

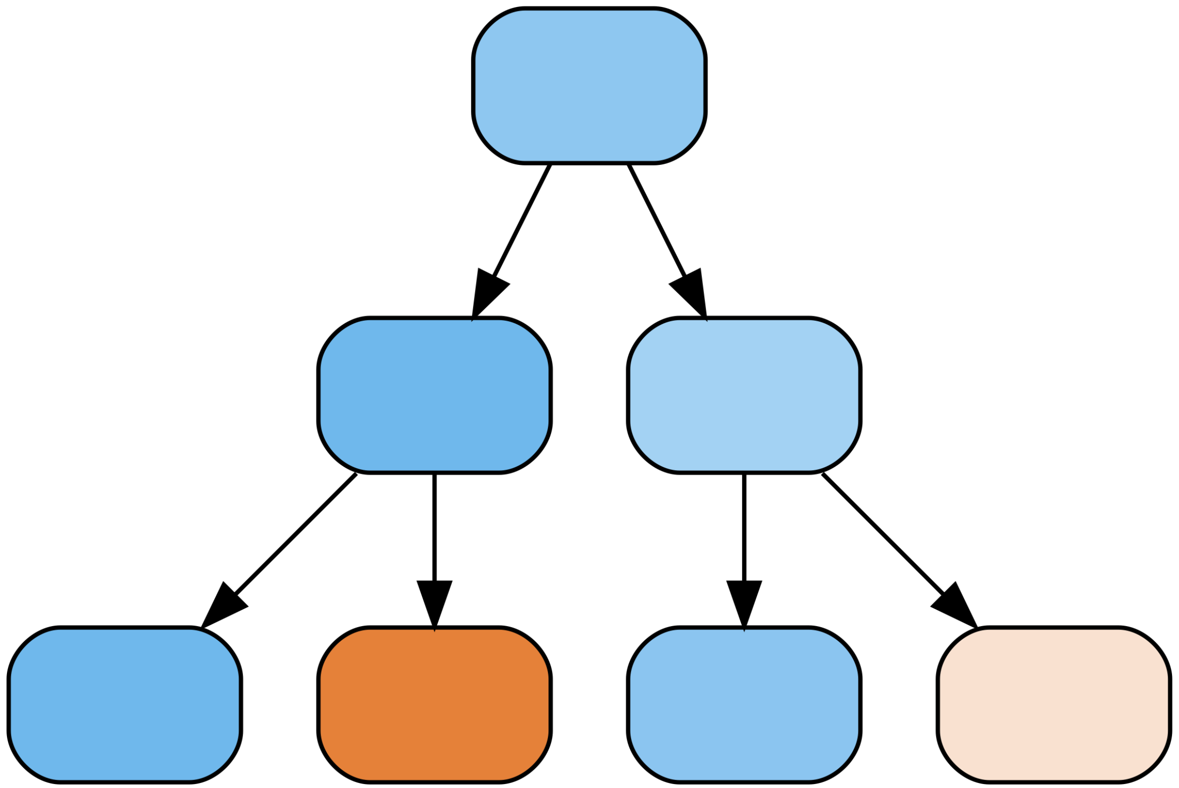
`RandomForestClassifier()` takes additional arguments, like `max_depth` :

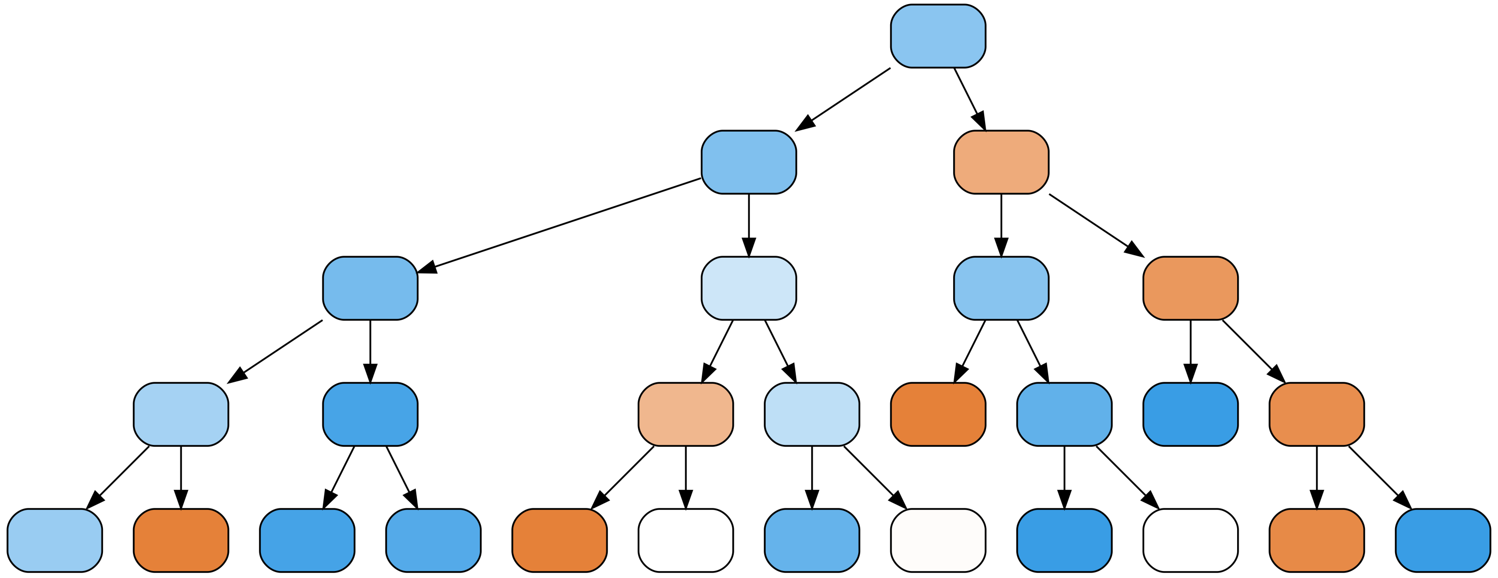
```
help(RandomForestClassifier)
```

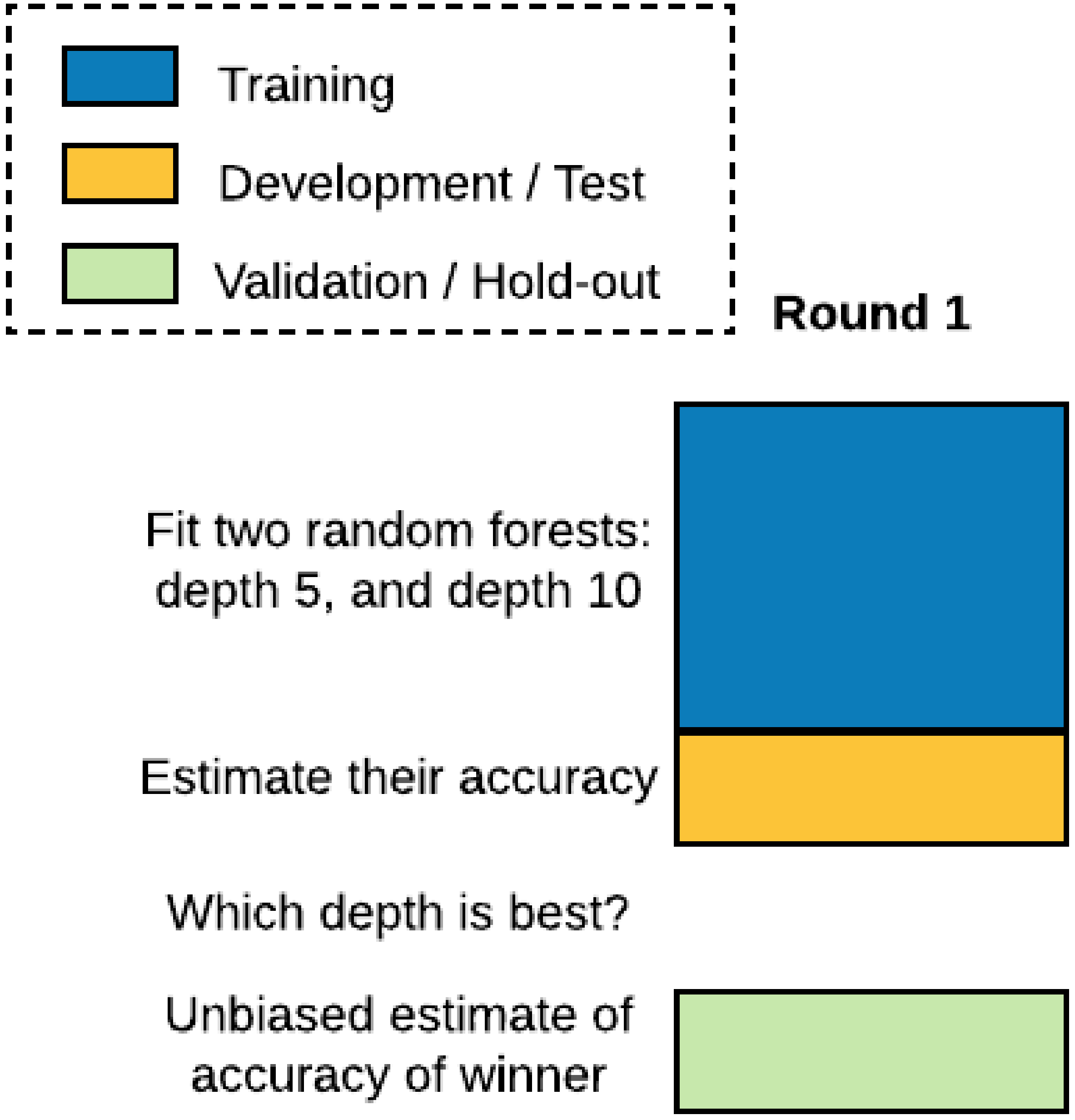
```
Help on class RandomForestClassifier in module sklearn.ensemble.forest:
...
 |  max_depth : integer or None, optional (default=None)
 |      The maximum depth of the tree. If None, then nodes are expanded until
 |      all leaves are pure or until all leaves contain less than
 |      min_samples_split samples.
```

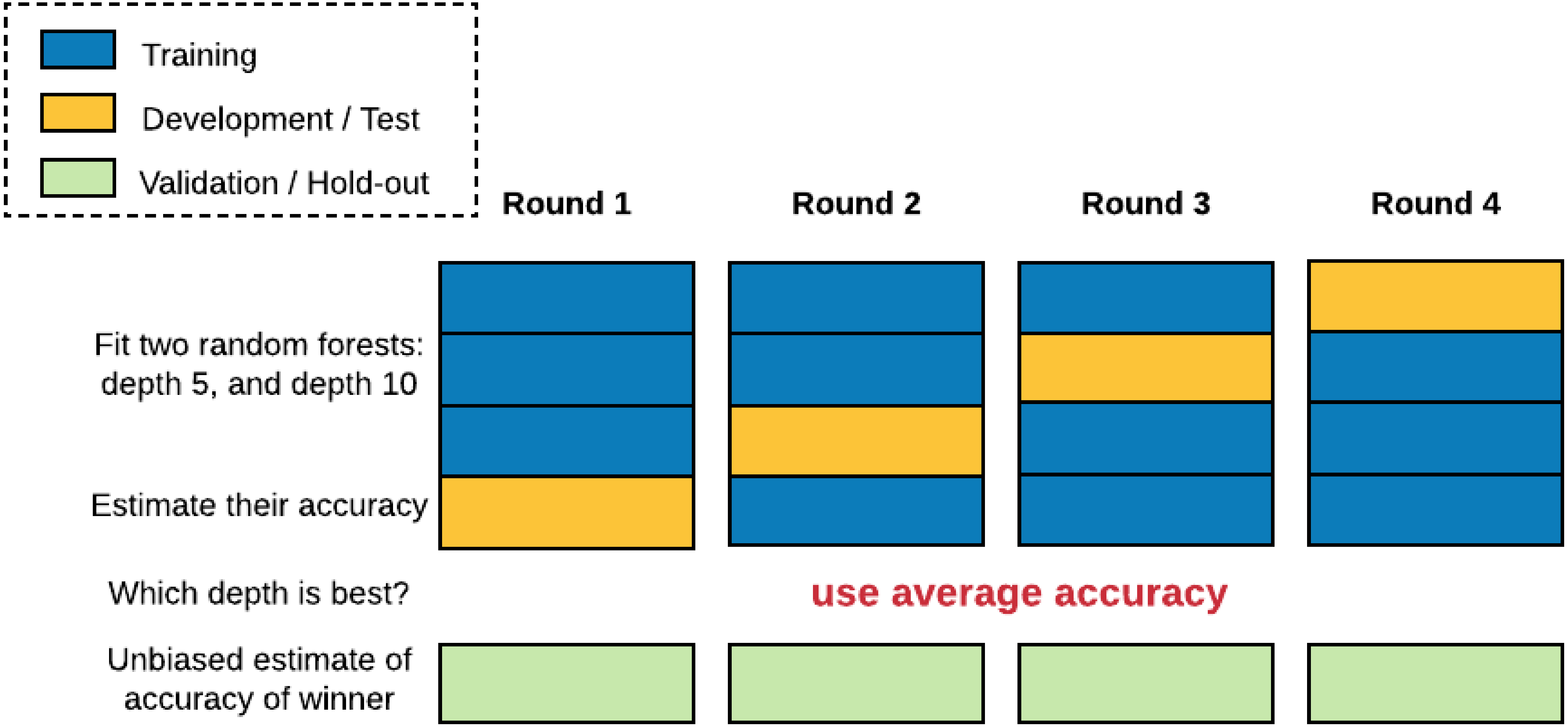
```python
m2 = RandomForestClassifier(
    max_depth=2)
m2.fit(X_train, y_train)

m2.estimators_[0]
```

```python
m4 = RandomForestClassifier(
    max_depth=4)
m4.fit(X_train, y_train)

m4.estimators_[0]
```






Training
Development / Test
Validation / Hold-out
Round 1
Fit two random forests:
depth 5, and depth 10
Estimate their accuracy
Which depth is best?
Unbiased estimate of
accuracy of winner


Training
Development / Test
Validation / Hold-out
Round 1
Round 2
Round 3
Round 4
Fit two random forests:
depth 5, and depth 10
Estimate their accuracy
Which depth is best?
use average accuracy
Unbiased estimate of
accuracy of winner

# Cross-validation

Assess accuracy using `cross_val_score()` :

```
from sklearn.model_selection import cross_val_score

cross_val_score(RandomForestClassifier(), X, y)
```

```
array([0.7218 , 0.7682, 0.7866])
```

```
numpy.mean(cross_val_score(RandomForestClassifier(), X, y))
```

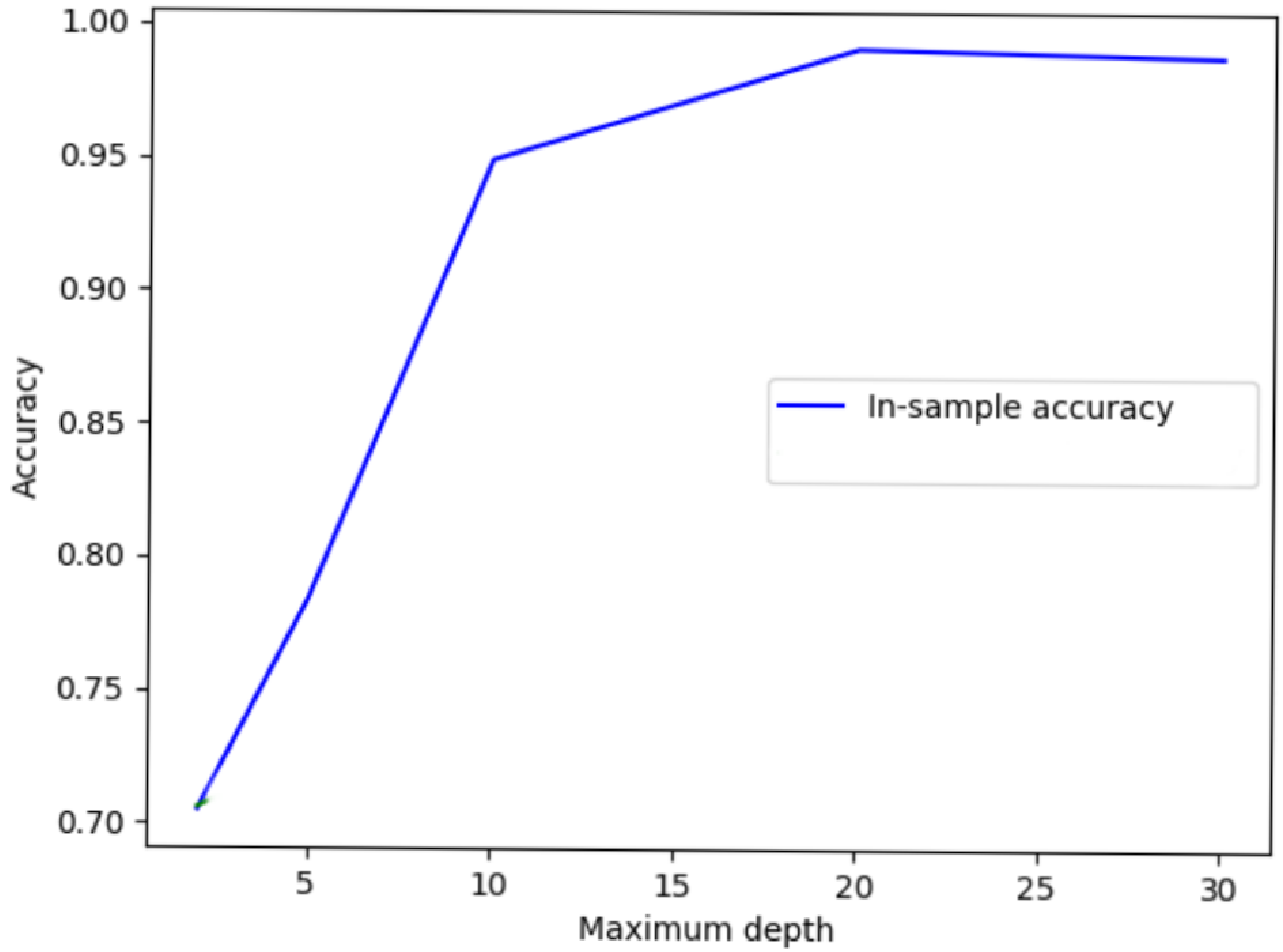
```
0.7589
```

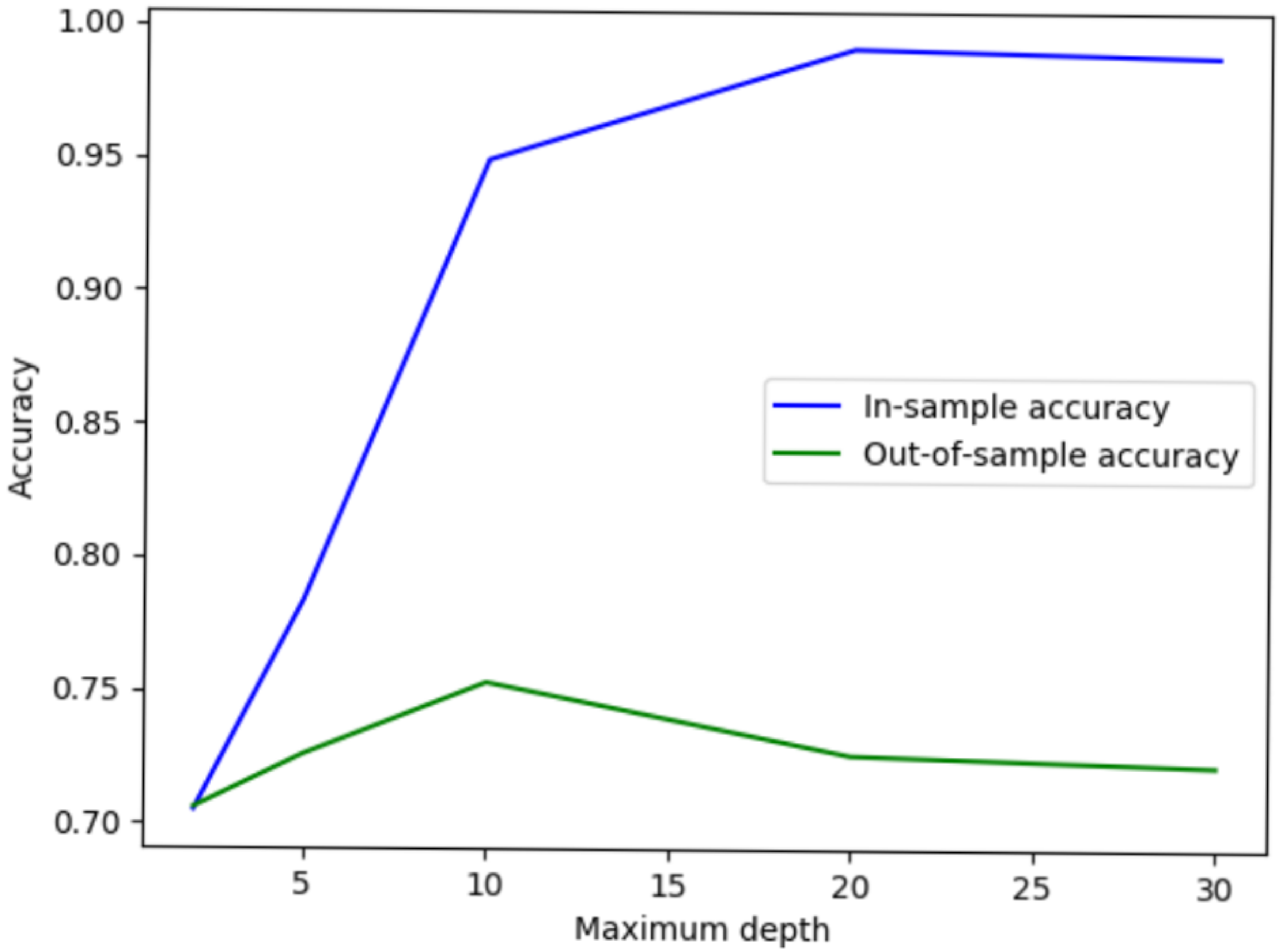
# Tuning model complexity

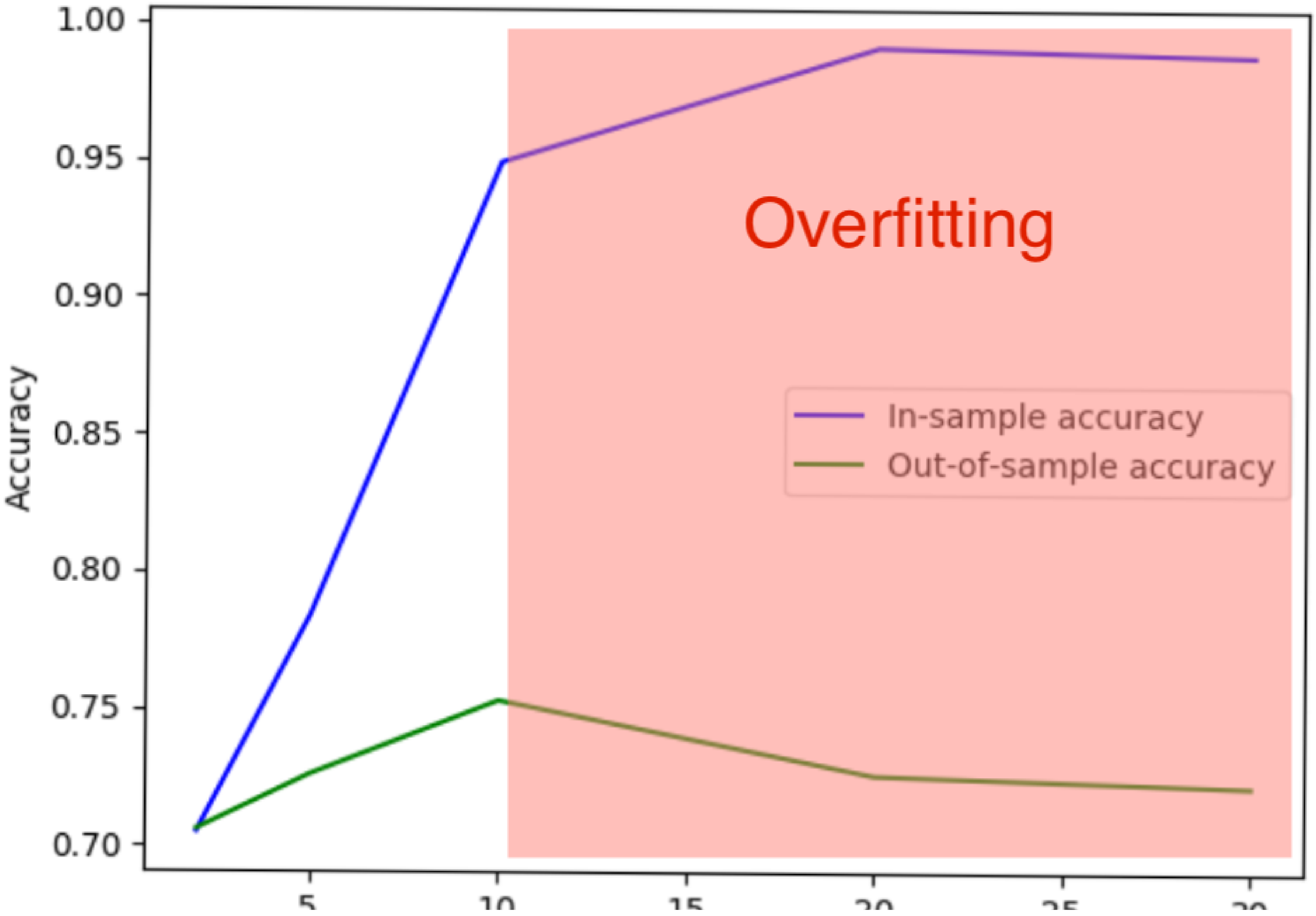
Tune the tree depth using `GridSearchCV()` :

```
from sklearn.model_selection import GridSearchCV
param_grid = {'max_depth':[5,10,20]}
grid = GridSearchCV(RandomForestClassifier(), param_grid)
grid.fit(X,y)
grid._best_params
```

```
{'max_depth': 10}
```


1.00
0.95
0.90
0.85
0.80
0.75
0.70
Accuracy
In-sample accuracy
5
10
15
20
25
30
Maximum depth


1.00
0.95
0.90
0.85
0.80
0.75
0.70
Accuracy
5
10
15
20
25
30
Maximum depth
In-sample accuracy
Out-of-sample accuracy


1.00
0.95
0.90
0.85
0.80
0.75
0.70
Accuracy
5
10
Overfitting
In-sample accuracy
Out-of-sample accuracy

# More complex is not always better!

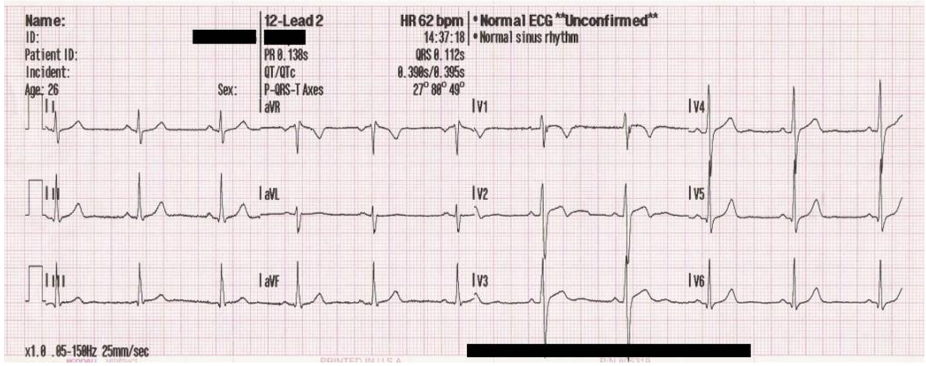
DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

# Feature engineering and overfitting

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON

**Dr. Chris Anagnostopoulos**
Honorary Associate Professor

# Feature extraction from non-tabular data




```
arrhythmias.head()
```

|   | age | sex | height | weight | ... | chV6_TwaveAmp | chV6_QRSA | chV6_QRSTA | class |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 75 | 0 | 190 | 80 | ... | 2.9 | 23.3 | 49.4 | 0 |
| 1 | 56 | 1 | 165 | 64 | ... | 2.1 | 20.4 | 38.8 | 0 |
| 2 | 54 | 0 | 172 | 95 | ... | 3.4 | 12.3 | 49.0 | 0 |
| 3 | 55 | 0 | 175 | 94 | ... | 2.6 | 34.6 | 61.6 | 1 |
| 4 | 75 | 0 | 190 | 80 | ... | 3.9 | 25.4 | 62.8 | 0 |

# Label encoding for categorical variables

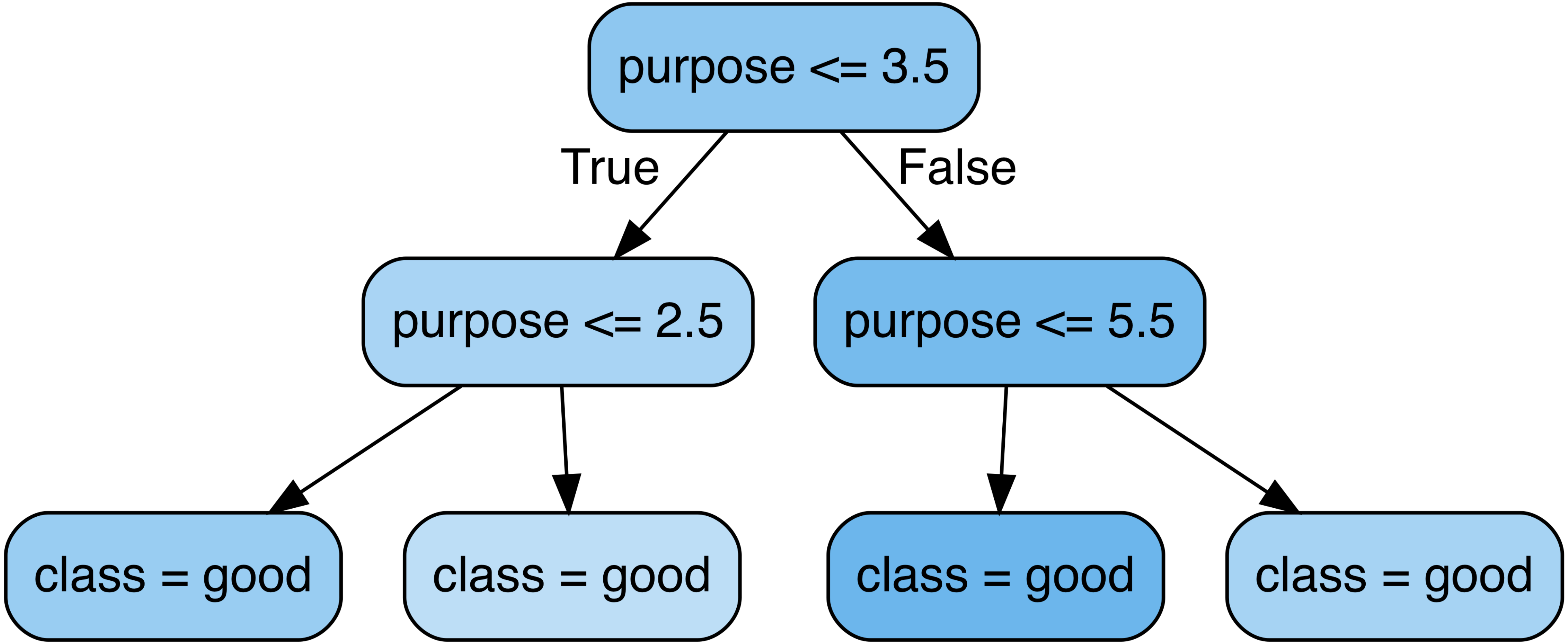
```
numpy.unique(credit_scoring['purpose'])
```

```
array(['business', 'buy_domestic_appliance', 'buy_furniture_equipment',
       'buy_new_car', 'buy_radio_tv', 'buy_used_car', 'education',
       'other', 'repairs', 'retraining'], dtype=object)
```

```
numpy.unique(LabelEncoder().fit_transform(credit_scoring['purpose']))
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

# Label encoding for categorical variables



# One hot encoding for categorical variables

```
pd.get_dummies(credit_scoring['purpose']).iloc[1]
```

```
purpose_business                   0
purpose_buy_domestic_appliance     0
purpose_buy_furniture_equipment    0
purpose_buy_new_car                0
purpose_buy_radio_tv               1
purpose_buy_used_car               0
purpose_education                  0
purpose_other                      0
purpose_repairs                    0
purpose_retraining                 0
```

# Keyword encoding for categorical variables

```
from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer()
credit_scoring['purpose'] = credit_scoring['purpose'].apply(
    lambda s: ' '.join(s.split('_')), 0)
dummy_matrix = vec.fit_transform(credit_scoring['purpose']).toarray()
pd.DataFrame(dummy_matrix, columns=vec.get_feature_names()).head()
```

```
     appliance  business  buy  car  ...   repairs  retraining  tv  used
0            0         0    1    0  ...         0           0   1     0
1            0         0    1    0  ...         0           0   1     0
2            0         0    0    0  ...         0           0   0     0
3            0         0    1    0  ...         0           0   0     0
```

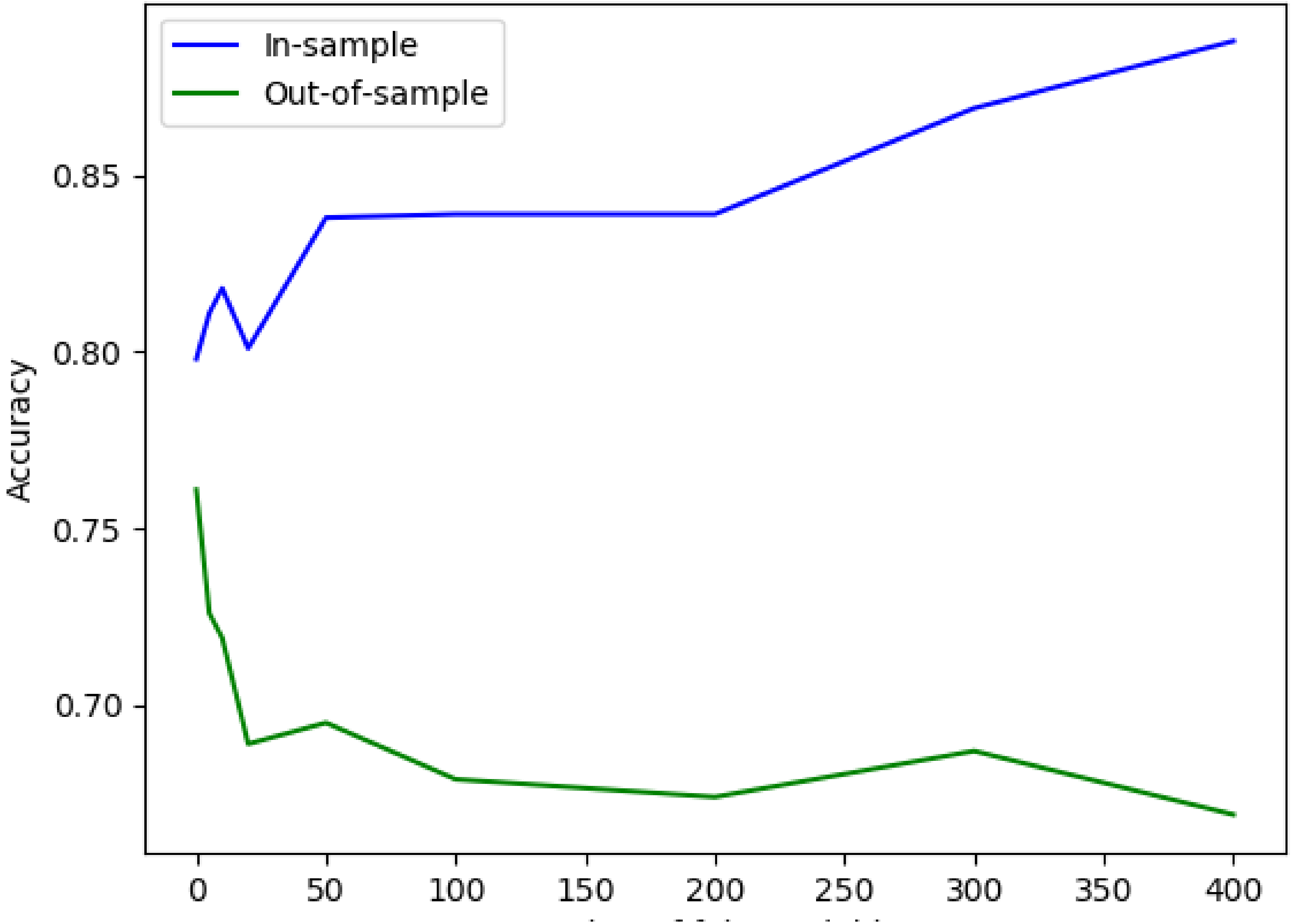
# Dimensionality and feature engineering

Categorical variables in `credit` :

- Label encoding: 1 column.
- One-hot encoding: 10 columns.
- Keyword encoding: 15 columns.

ECG features in `arrhythmias` :

- Over 250 features


In-sample
Out-of-sample
Accuracy
0.85
0.80
0.75
0.70
0
50
100
150
200
250
300
350
400

# Feature selection

```
from np.random import uniform
fakes = pd.DataFrame(
    uniform(low=0.0, high=1.0, size=n * 100).reshape(X.shape[0], 100),
    columns=['fake_' + str(j) for j in range(100)]
)
X_with_fakes = pd.concat([X, fakes], 1)
```

# Feature selection

```
from sklearn.feature_selection import chi2, SelectKBest
sk = SelectKBest(chi2, k=20)
which_selected = sk.fit(X_with_fakes, y).get_support()
X_with_fakes.columns[which_selected]
```

```
['checking_status', 'duration', 'credit_history', 'purpose',
    'credit_amount', 'savings_status', 'installment_commitment',
    'personal_status', 'property_magnitude', 'age', 'other_payment_plans',
    'job', 'own_telephone', 'fake_28', 'fake_46', 'fake_51', 'fake_60',
    'fake_83', 'fake_95', 'fake_99']
```

# Tradeoffs everywhere!

DESIGNING MACHINE LEARNING WORKFLOWS IN PYTHON