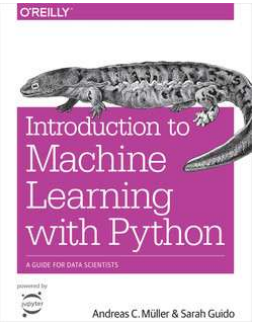
Applied Machine Learning
Grid Search

BSc course Informatiekunde 2026
https://staff.fnwi.uva.nl/a.visser/education/AML

Arnoud Visser
Intelligent Robotics Lab & Computer Vision Lab
Informatics Institute
Universiteit van Amsterdam
A.Visser@uva.nl

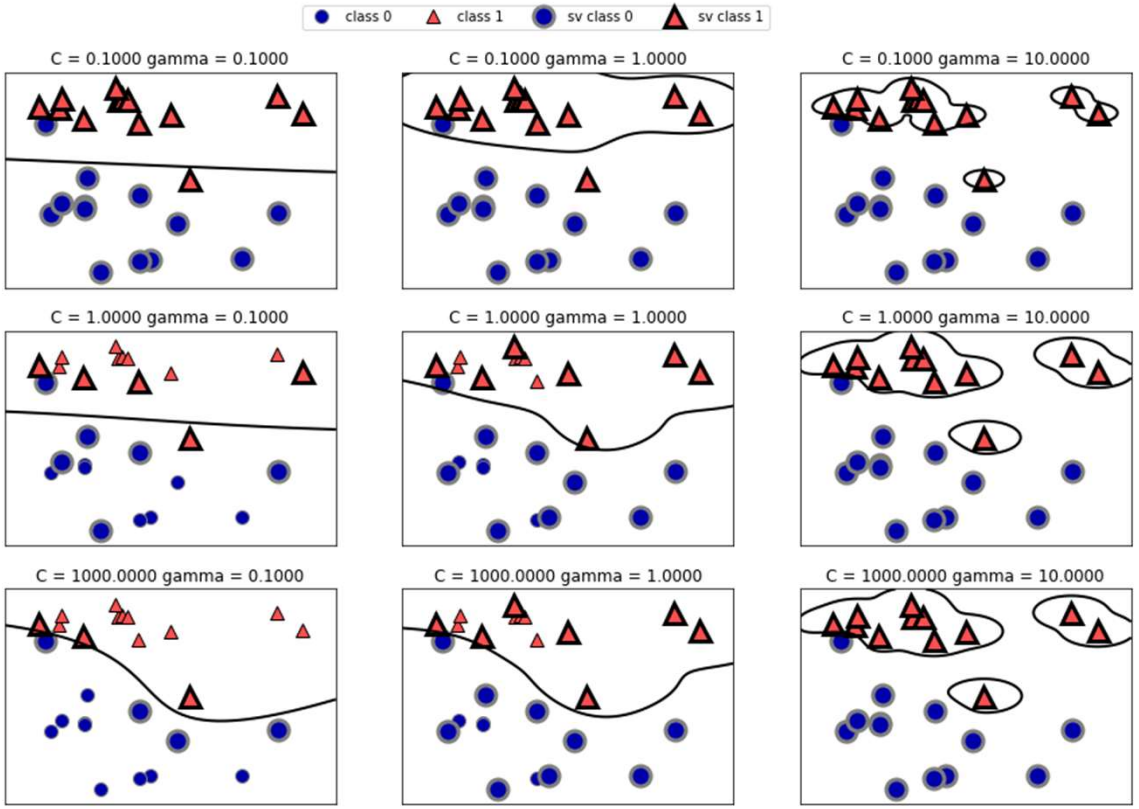
Illustrations courtesy of Maarten Marx, Sarah Guido, Yolanda Hagar, and many others.

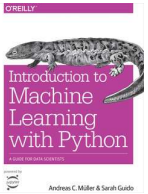


Section 5.2

Complexity of the model

- The complexity of Support Vector Machines depends on C & gamma, which have to be tuned together

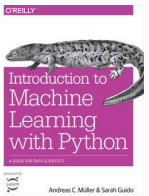




Complexity of the model

- The complexity of Support Vector Machines depends on C & gamma, which have to be tuned together

	C = 0.001	C = 0.01	...	C = 10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	...	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	...	SVC(C=10, gamma=0.01)
...	...	...	...	...
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	...	SVC(C=10, gamma=100)



Complexity of the model

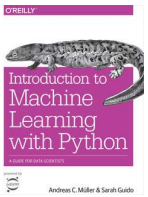
- The complexity of Support Vector Machines depends on C & gamma, which have to be tuned together

```
# naive grid search implementation
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
print("Size of training set: {}   size of test set: {}".format(
      X_train.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

Data Leakage

```
Size of training set: 112   size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```

overfitting



Complexity of the model

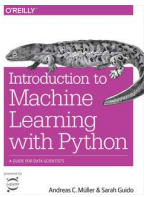
- Test data was used to adjust the parameters

```
# correct grid search implementation
from sklearn.svm import SVC
X_trainval, X_test, y_trainval, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(
    X_trainval, y_trainval, random_state=1)
print("Size of training set: {}   size of validation set: {}   size of test set:"
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))
best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

# rebuild a model on the combined training and validation set,
# and evaluate it on the test set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

Secure an extra test set

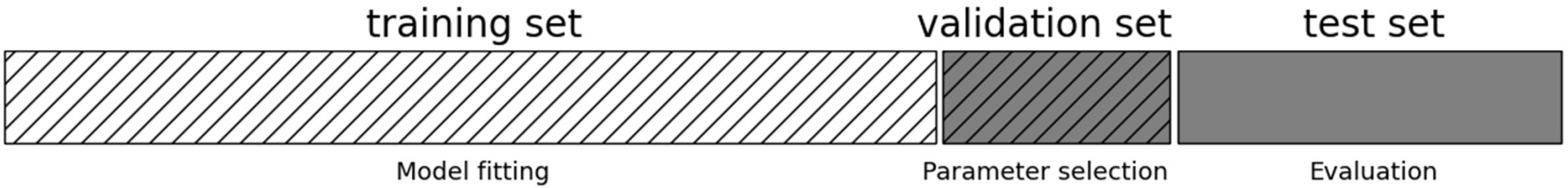
```
Size of training set: 84
size of validation set: 28
size of test set: 38

Best score on validation set: 0.96
Best parameters:  {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92
```

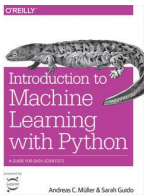
Test on untouched test set



Three fold split

- Keep one part safe until the end!





Three fold split

- Keep one part safe until the end!

D.2 Selection of Wetland areas for Sentinel-2

Four training areas:

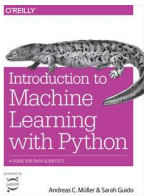
- *Oostvaarderplassen*
- *Loosdrechtse Plassen*
- *Gendtste Polder*
- *Land van Saeftinghe*

One Validation area:

- *Lauwersmeer*

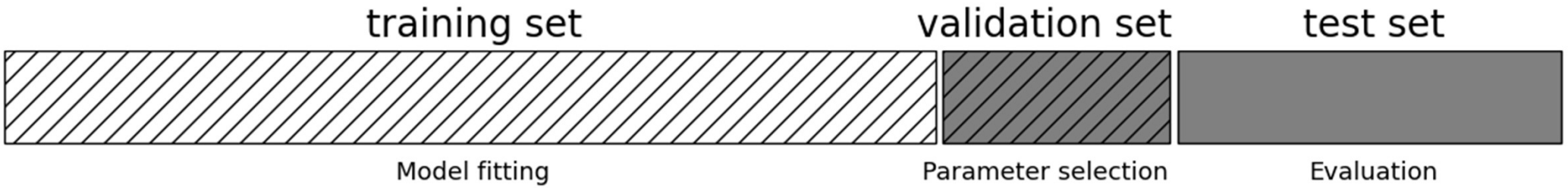
One Testing area:

- *Biesbosch*

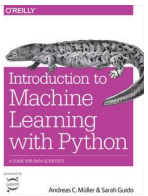


Three fold split

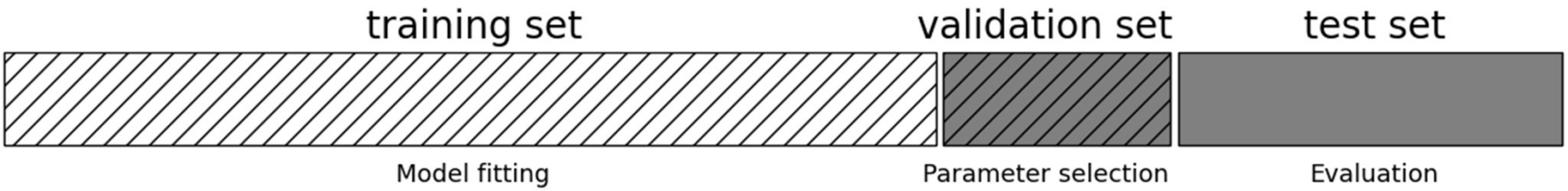
- End-result depends on the split!



- Cross-validation

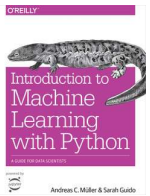


Three fold split & Cross validation



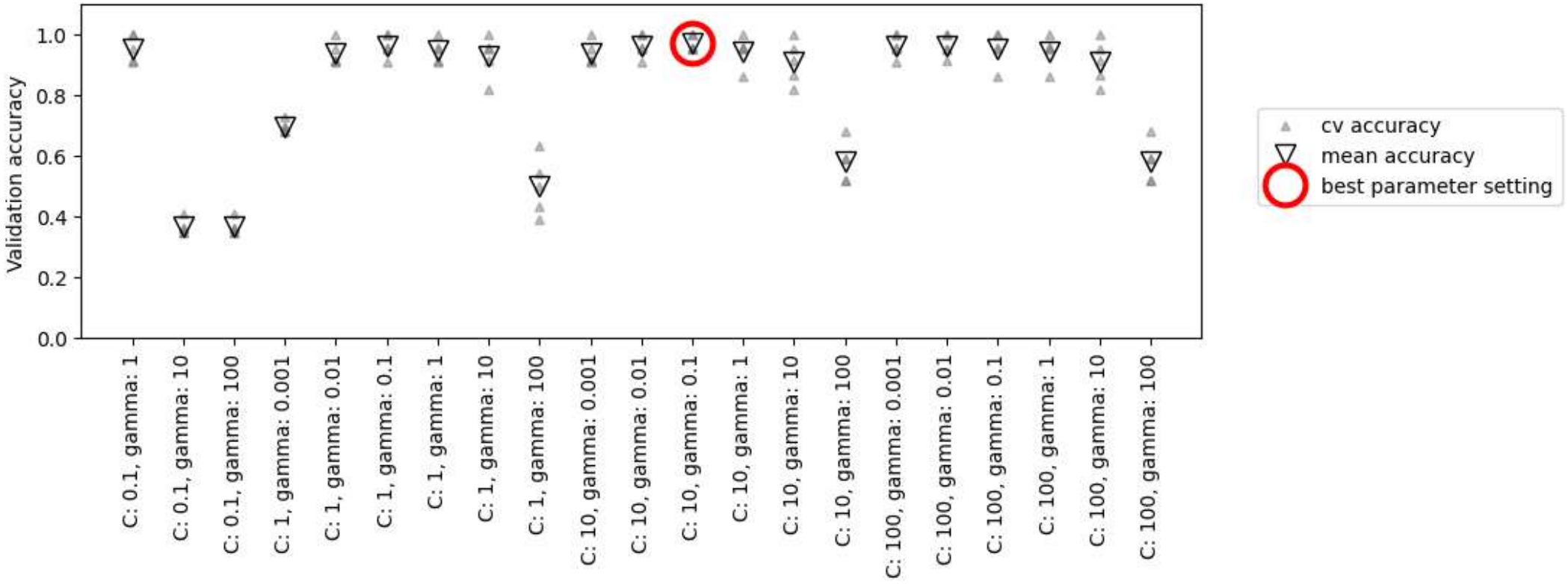

```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters,
        # train an SVC
        svm = SVC(gamma=gamma, C=C)
        # perform cross-validation
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # compute mean cross-validation accuracy
        score = np.mean(scores)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
# rebuild a model on the combined training and validation set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
```

Test on untouched test set

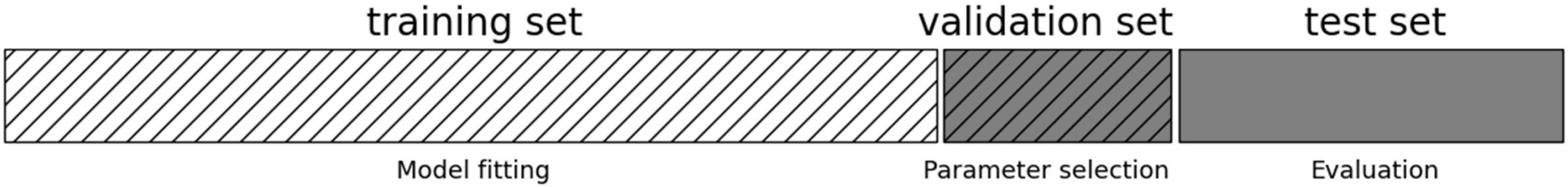


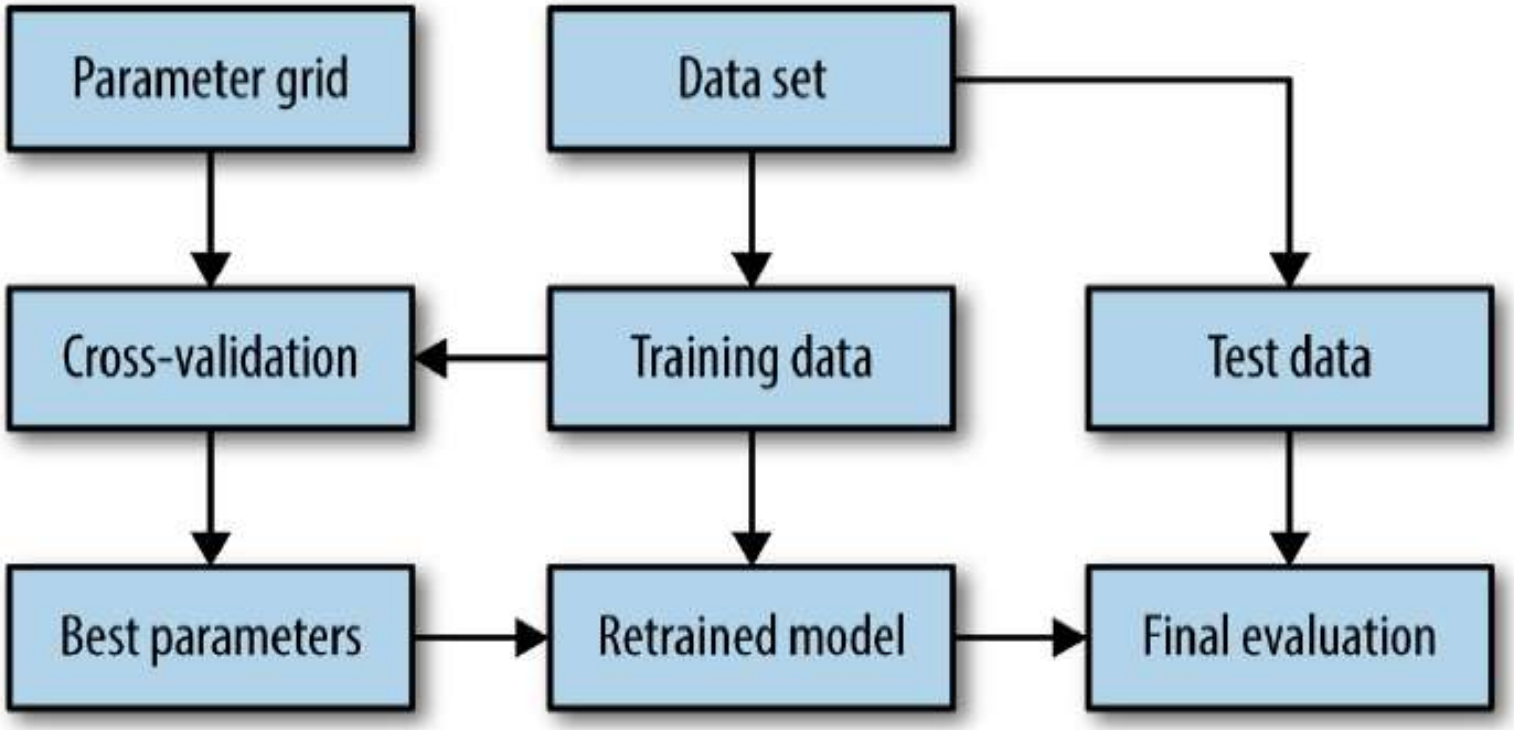
Three fold split & Cross validation

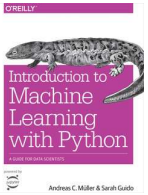
```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters,
        # train an SVC
        svm = SVC(gamma=gamma, C=C)
        # perform cross-validation
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # compute mean cross-validation accuracy
        score = np.mean(scores)
        # if we got a better score, store the score and parameters
```



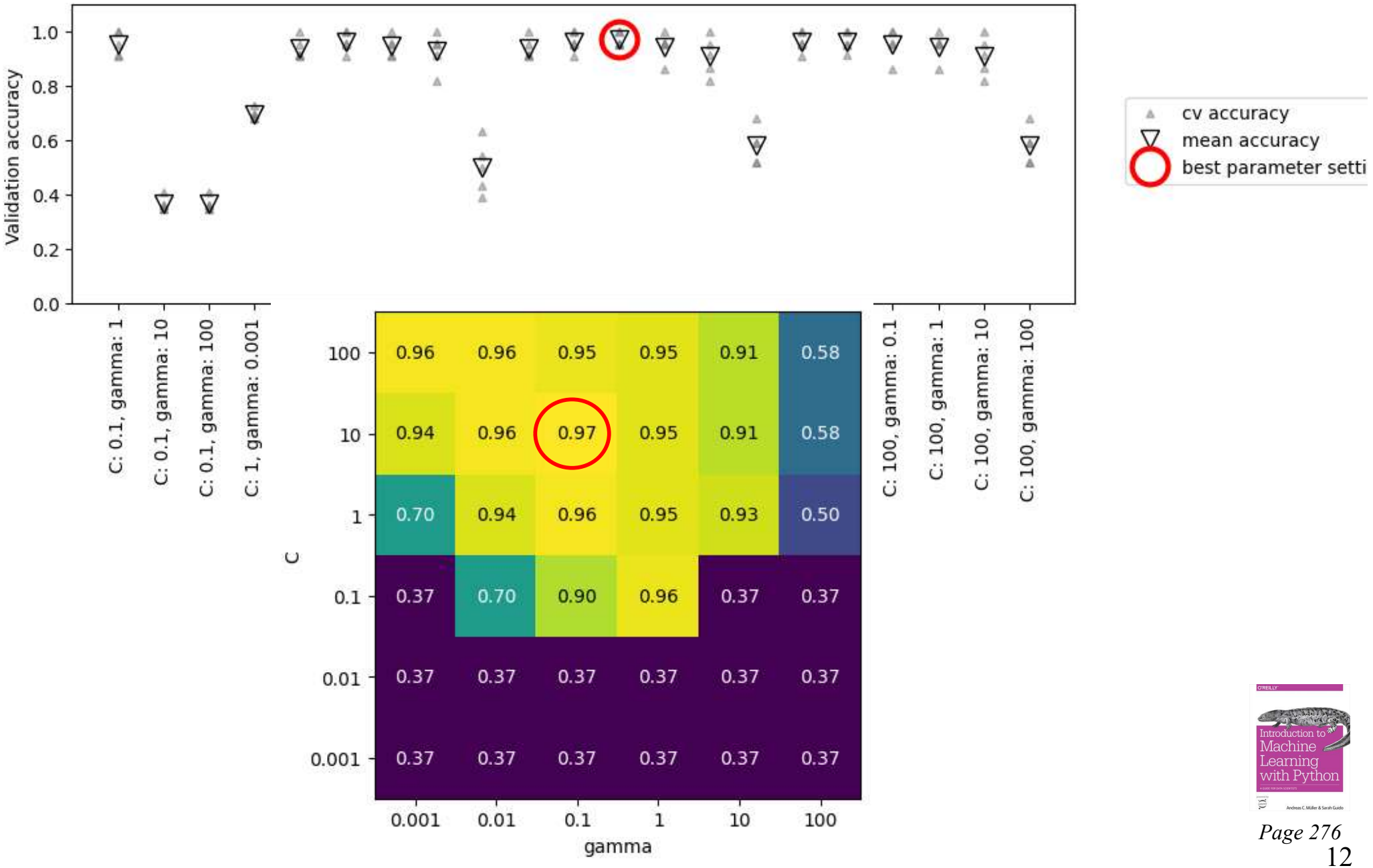
Three fold split



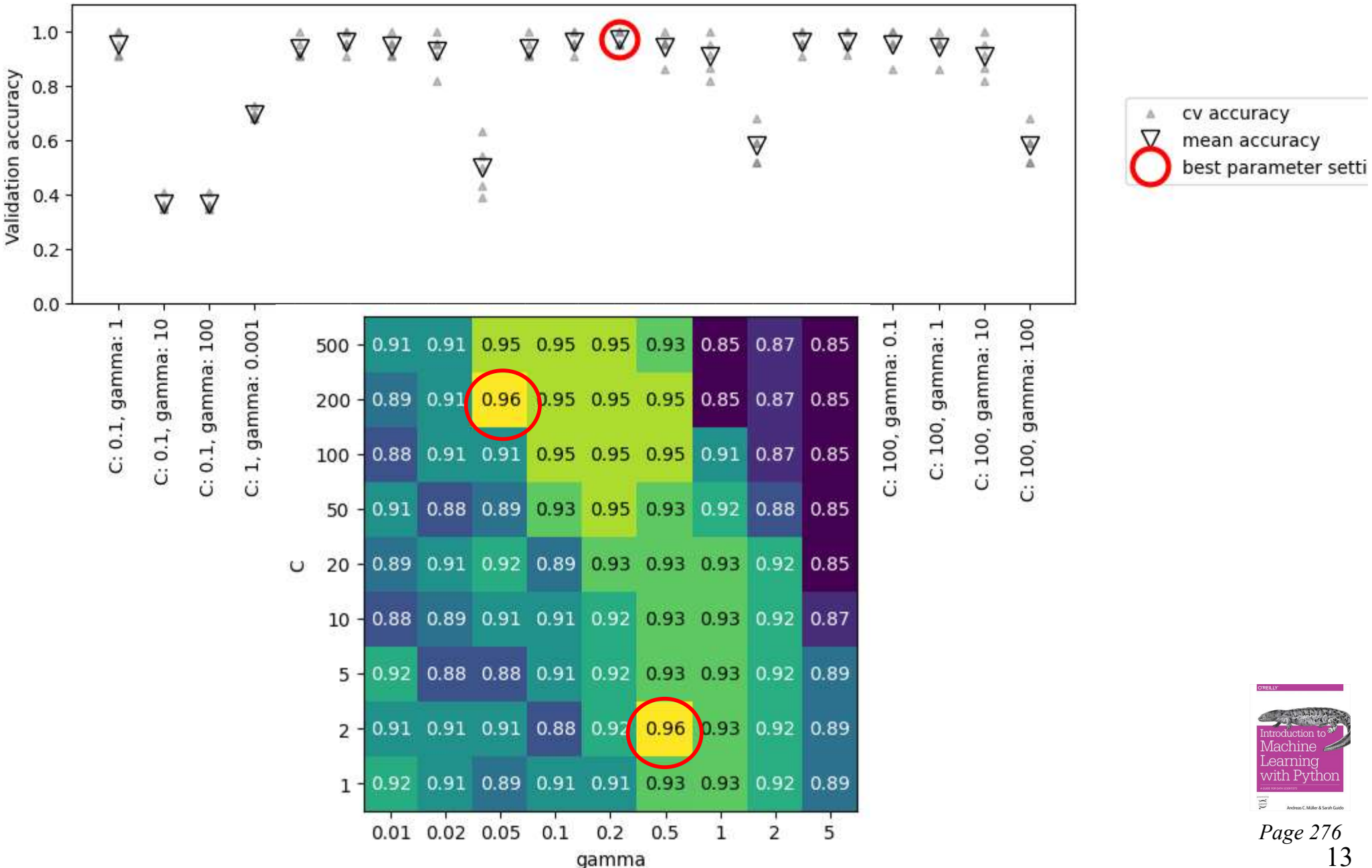




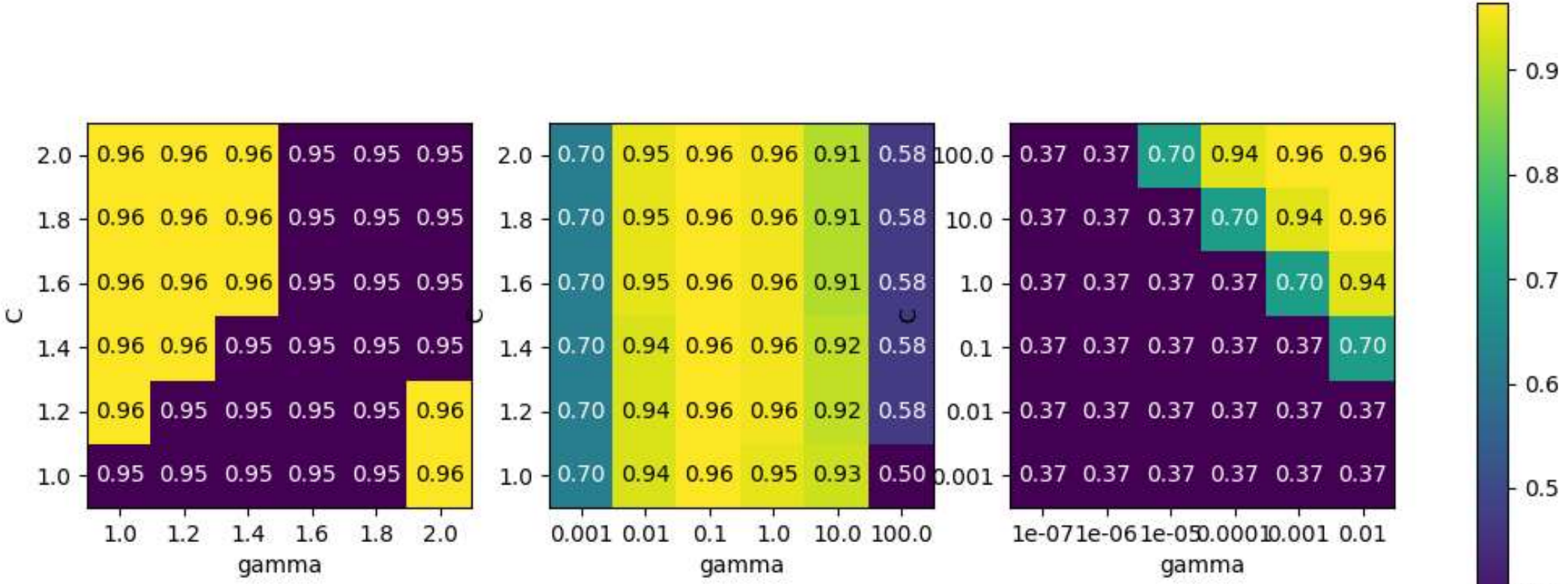
Parameter Grid



Parameter Grid



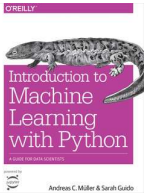
Bad choices for Parameter Grid



Not enough change

Change in only one dimension

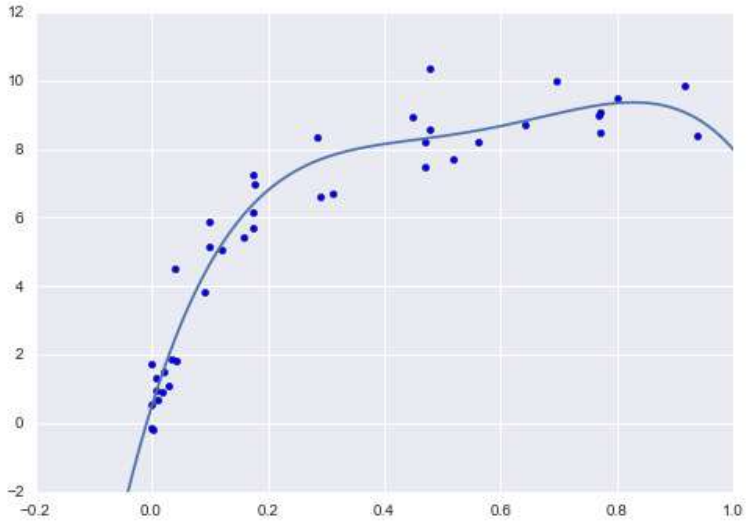
Unused lower corner



Searches over spaces which are not Grids

```
from sklearn.grid_search import GridSearchCV

param_grid = {'polynomialfeatures__degree': np.arange(21),
              'linearregression__fit_intercept': [True, False],
              'linearregression__normalize': [True, False]}

grid = GridSearchCV(PolynomialRegression(), param_grid, cv=7)

Grid.fit(X,y)
grid.best_params_
```

```
{'linearregression__fit_intercept': False,
 'linearregression__normalize': True,
 'polynomialfeatures__degree': 4}
```

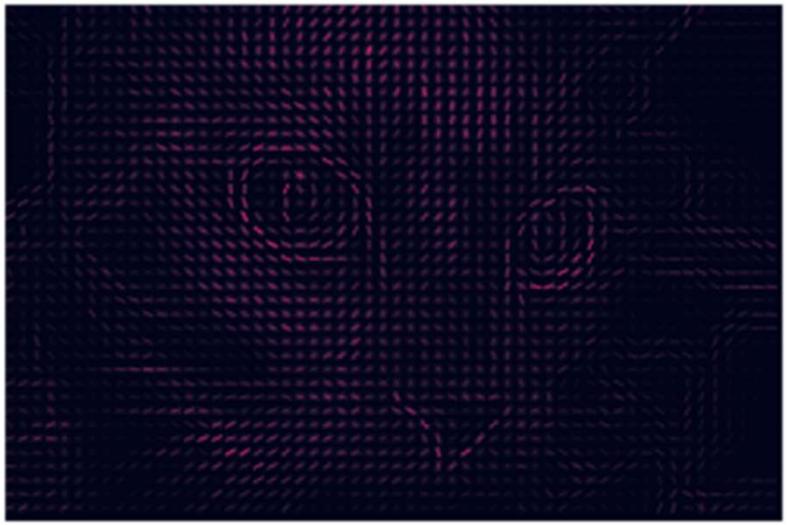


Works also for Regression

Page 374

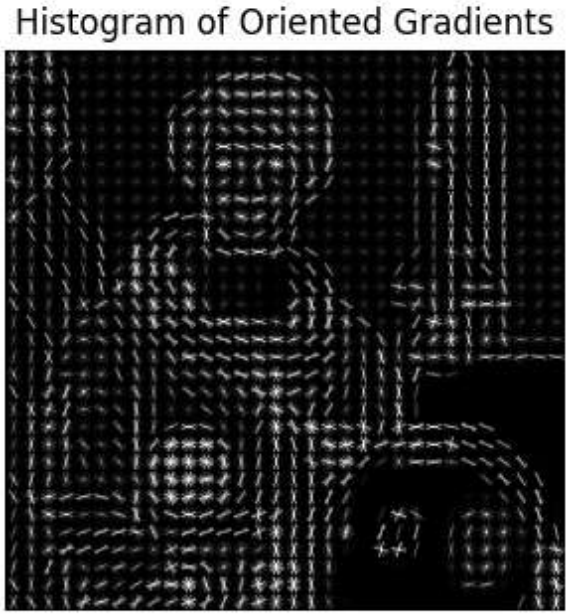
GridSearch in Practice

```
from skimage import data, color, feature
import skimage.data

image = color.rgb2gray(data.chelsea())
hog_vec, hog_vis = feature.hog(image, visualize=True)

fig, ax = plt.subplots(1, 2, figsize=(12, 6),
                       subplot_kw=dict(xticks=[], yticks=[]))
ax[0].imshow(image, cmap='gray')
ax[0].set_title('input image')

ax[1].imshow(hog_vis)
ax[1].set_title('visualization of HOG features');
```



GridSearch in Practice

```
from skimage import data, color, feature
import skimage.data

image = color.rgb2gray(data.astronaut())
hog_vec, hog_vis = feature.hog(image, visualize=True, orientations=8,
    pixels_per_cell=(16, 16),
    cells_per_block=(1, 1) )
```



A Simple Face Detector

Using these HOG features, we can build up a simple facial detection algorithm with any Scikit-Learn estimator; here we will use a linear support vector machine. The steps are as follows:

1. Obtain a set of image thumbnails of faces to constitute "positive" training samples.
2. Obtain a set of image thumbnails of non-faces to constitute "negative" training samples.
3. Extract HOG features from these training samples.
4. Train a linear SVM classifier on these samples.
5. For an "unknown" image, pass a sliding window across the image, using the model to evaluate whether that window contains a face or not.
6. If detections overlap, combine them into a single window.

Step 1

Obtain a set of image thumbnails of faces to constitute "positive" training samples.

```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people()
positive_patches = faces.images
positive_patches.shape
```

```
(13233, 62, 47)
```

This gives us a sample of 13,000 face images to use for training.

Step 2

Obtain a set of image thumbnails of non-faces to constitute "negative" training samples.

```
images = [data.camera(), data.text(), data.coins(), data.moon(),
data.page(), data.clock(), color.rgb2gray(data.immunohistochemistry()),
color.rgb2gray(data.chelsea()), color.rgb2gray(data.coffee()),
color.rgb2gray(data.hubble_deep_field())].

negative_patches = np.vstack([extract_patches(im, 1000, scale)
                              for im in images for scale in [0.5, 1.0, 2.0]])
negative_patches.shape
```

```
(30000, 62, 47)
```

This gives us a sample of 30,000 face images to use for training.

Step 3

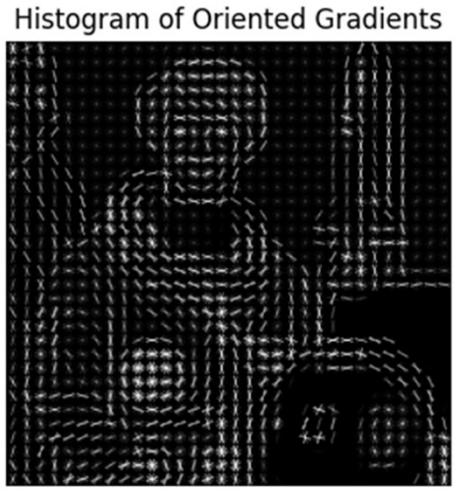
Extract HOG features from these training samples.

```
from itertools import chain
X_train = np.array([feature.hog(im)
                    for im in chain(positive_patches,
                                    negative_patches)])
y_train = np.zeros(X_train.shape[0])
y_train[:positive_patches.shape[0]] = 1

X_train.shape
```

```
(43233, 1215)
```

This gives us a sample of 43,000 training samples in 1,215 dimensions.

64 features



1215 features

Page 509

Step 4

Train a linear SVM classifier on these samples.

```
from sklearn.svm import LinearSVC
from sklearn.grid_search import GridSearchCV

grid = GridSearchCV(LinearSVC(), {'C': [1.0, 2.0, 4.0, 8.0]})
grid.fit(X_train, y_train)
grid.best_score_
```

```
0.98878
```

For such a high-dimensional binary classification task, a Linear support vector machine is a good choice.

```
model = grid.best_estimator_
model.fit(X_train, y_train)
```

Step 5

For an "unknown" image, pass a sliding window across the image,

```
test_image = skimage.data.astronaut()
test_image = skimage.color.rgb2gray(test_image)
test_image = skimage.transform.rescale(test_image, 0.5)
test_image = test_image[:160, 40:180]

plt.imshow(test_image, cmap='gray')
plt.axis('off');
```

```
indices, patches = zip(*sliding_window(test_image))
patches_hog = np.array([feature.hog(patch) for patch in patches])
patches_hog.shape
```

```
(1911, 1215)
```

This gives us a sample of ~1900 patches in 1,215 dimensions.

Step 6

Using the model to evaluate whether that window contains a face or not.

```
labels = model.predict(patches_hog)
labels.sum()
```

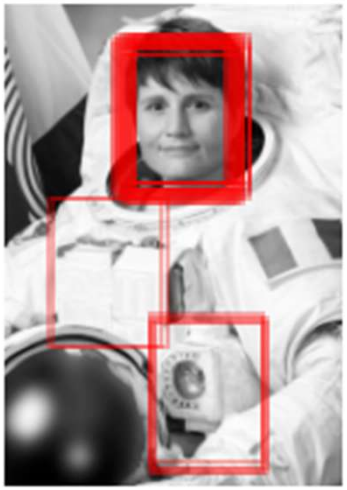
```
33.0
```

If detections overlap, combine them into a single window.

Generalize

Use a European astronaut.

```
test_image = skimage.io.imread('../Samantha_Cristoforetti.jpg')
test_image = skimage.color.rgb2gray(test_image)
test_image = skimage.transform.rescale(test_image, 0.25)
test_image = test_image[60:260, 40:180]
```



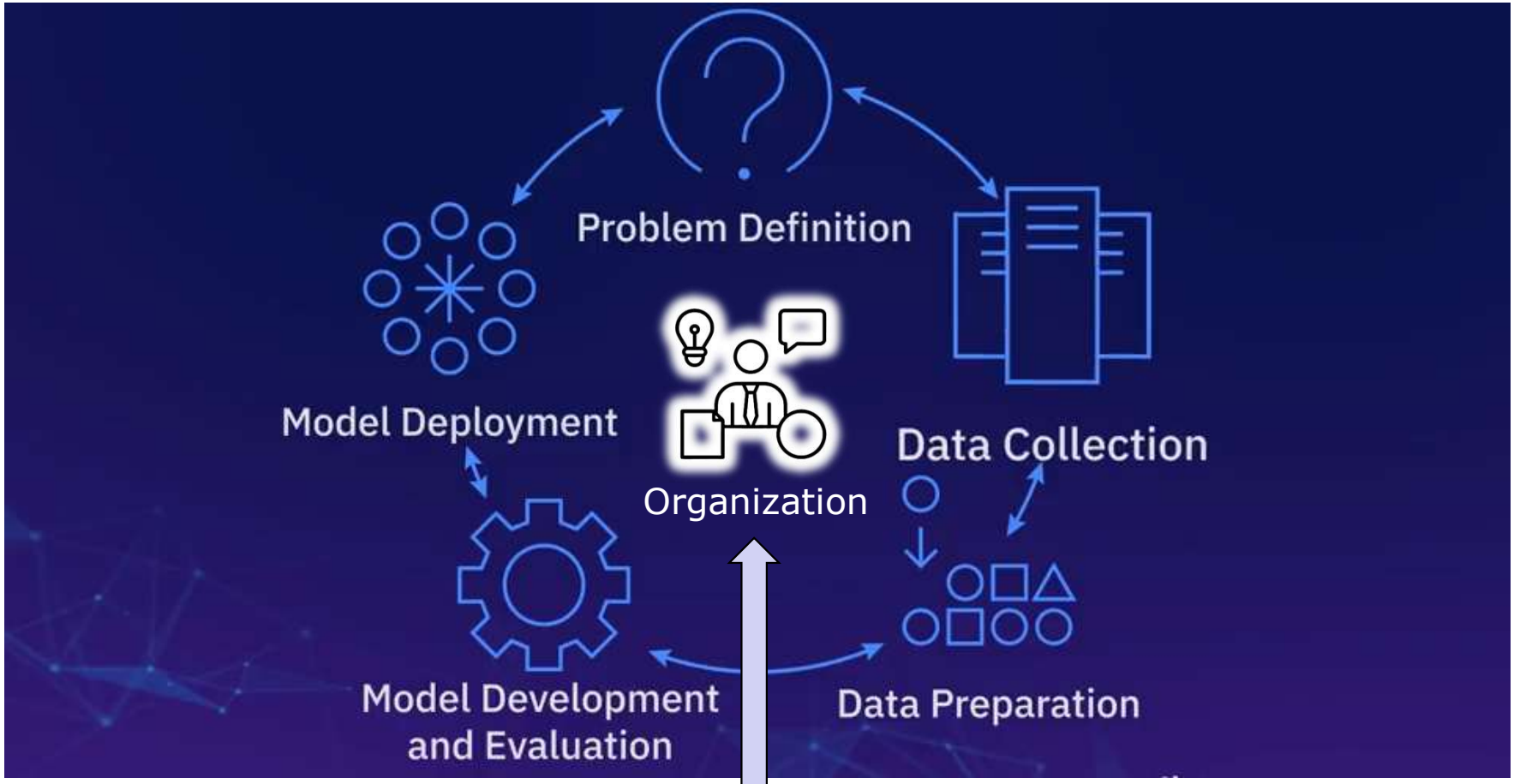
Generalize

Use another Italian.

```
test_image = skimage.io.imread('../Giovanni.jpg')
test_image = skimage.color.rgb2gray(test_image)
test_image = test_image[60:260, 40:180]
```

Introduction to Machine Learning

Building a predictive model is a circular process.

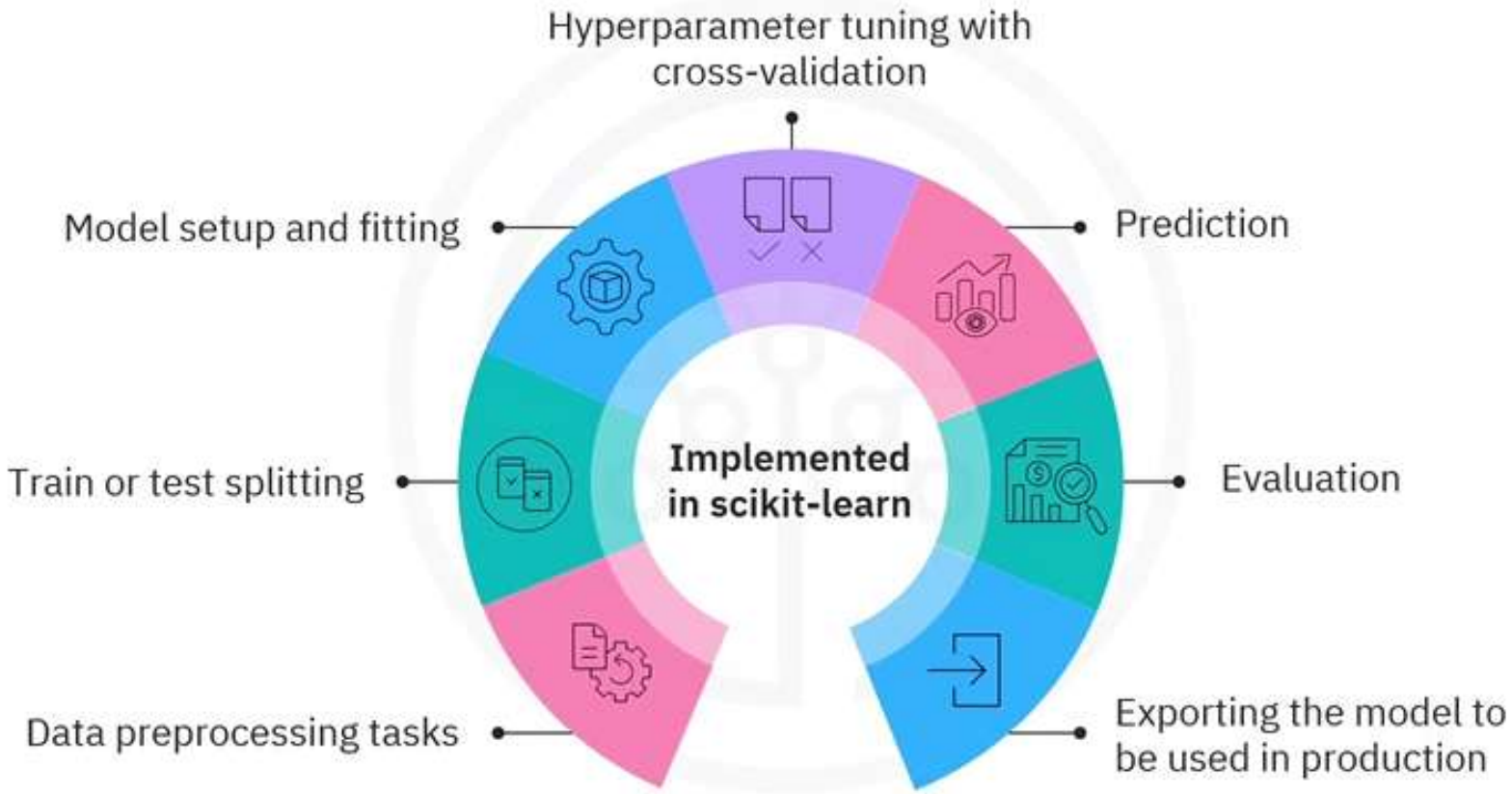


Expertise

- Depends on the organization
- Interpret the information in the right way

Introduction to Machine Learning

Building a predictive model is a circular process.

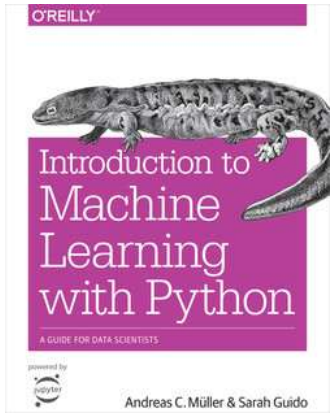


Joseph Santarcangelo & Joss Grossman, Machine Learning with Python, IBM Skills network, 2025

Introduction to Machine Learning

- 5.2 Grid Search

- Strengths:
  - Essential to make a ‘fair comparison’ between algorithms
  - Make your algorithm ready for deployment
- Weakness:
  - Can be computationally challenging
  - Danger of Data leakage



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O’Reilly Media, October 2016

Conclusion

Learning outcomes of this course covered today

- Most work is in Data Preparation and Model Evaluation
- Beter do Model Evaluation the correct way!