DUT STID, Université de la Côte d'Azur

Data mining

Introduction à l'apprentissage automatique

Prof. Dario Malchiodi

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

UNIVERSITÉ CÔTE D'AZUR

In [165]:

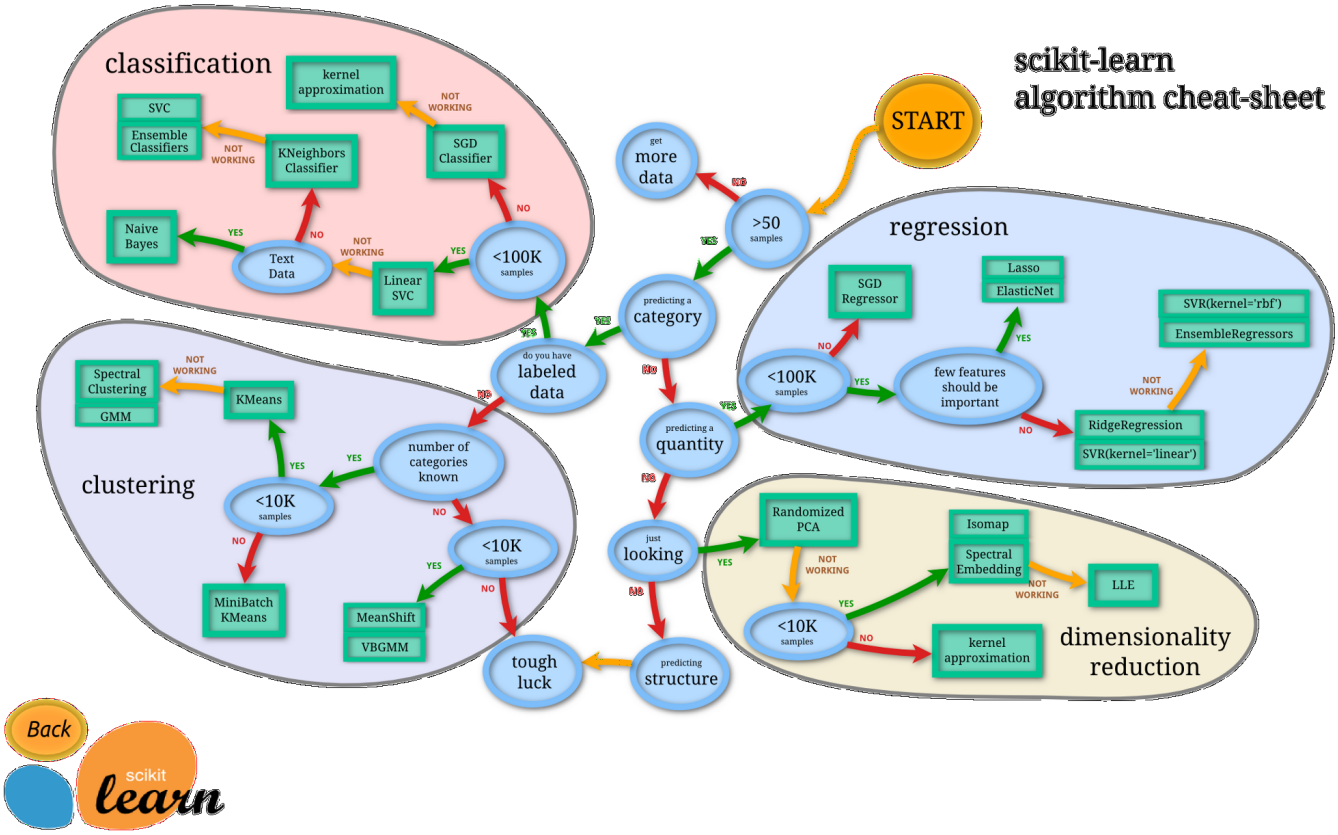
```
print('Hello world')
```

```
Hello world
```

Introduction à l'apprentissage automatique

Sources:

- J. VanderPlas, Python Data Science Handbook, https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.0 Introducing-Scikit-Learn.ipynb (https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.(Introducing-Scikit-Learn.ipynb)



Data Representation in Scikit-Learn

In [177]:

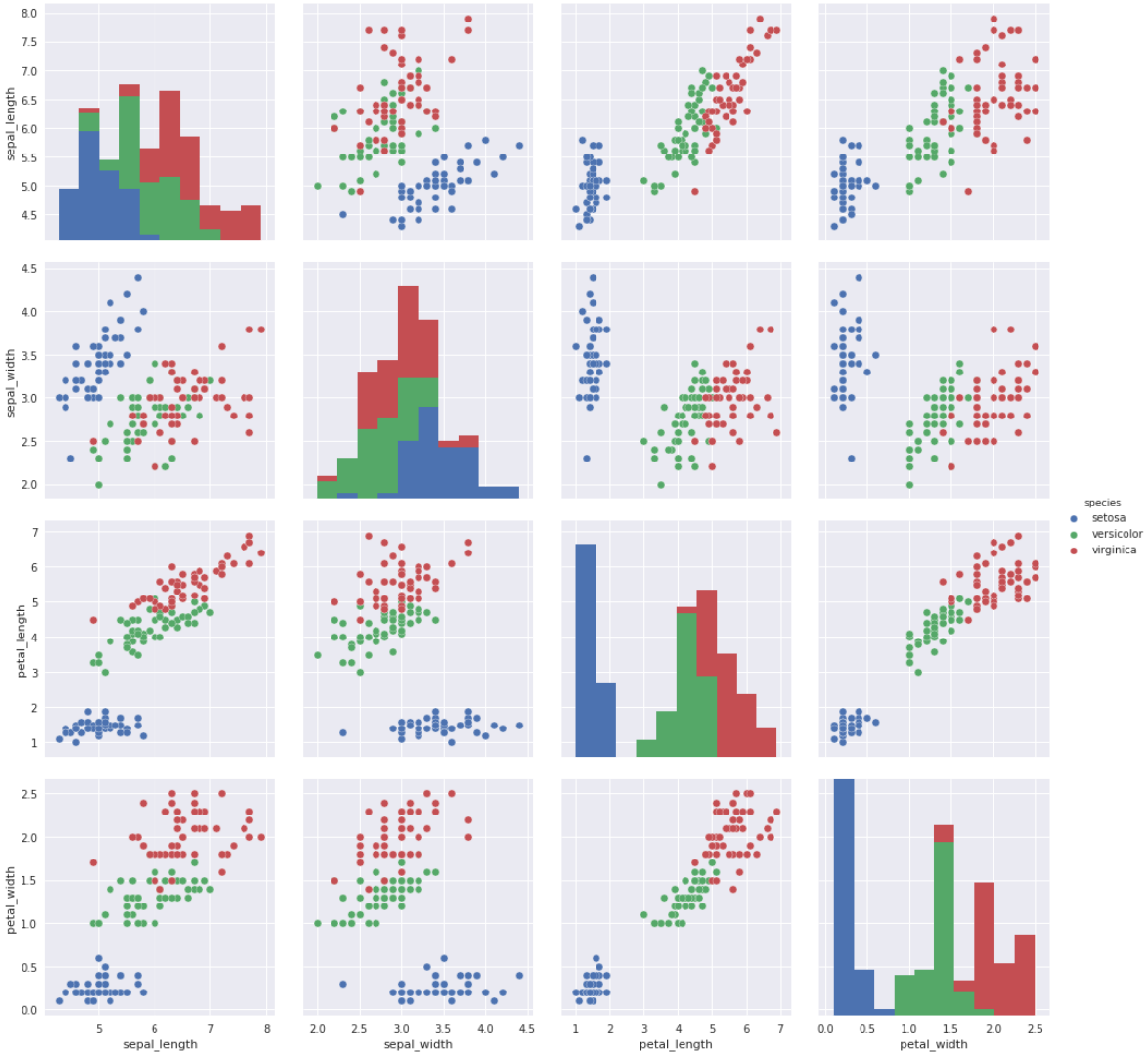
```
import seaborn as sns
sns.set()
iris = sns.load_dataset('iris')
iris.head()
```

Out[177]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [178]:

```
%matplotlib inline
sns.pairplot(iris, hue='species', size=3.5);
```



In [179]:

```
X_iris = iris.drop('species', axis=1)
X_iris.shape
```

Out[179]:

```
(150, 4)
```

In [180]:

```
y_iris = iris['species']
y_iris.shape
```

Out[180]:

```
(150,)
```

API d'éstimation de Scikit-Learn

- *Consistency*: All objects share a common interface drawn from a limited set of methods, with consistent documentation.
- *Inspection*: All specified parameter values are exposed as public attributes.
- *Limited object hierarchy*: Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas `DataFrame` s, SciPy sparse matrices) and parameter names use standard Python strings.
- *Composition*: Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.
- *Sensible defaults*: When models require user-specified parameters, the library defines an appropriate default value.

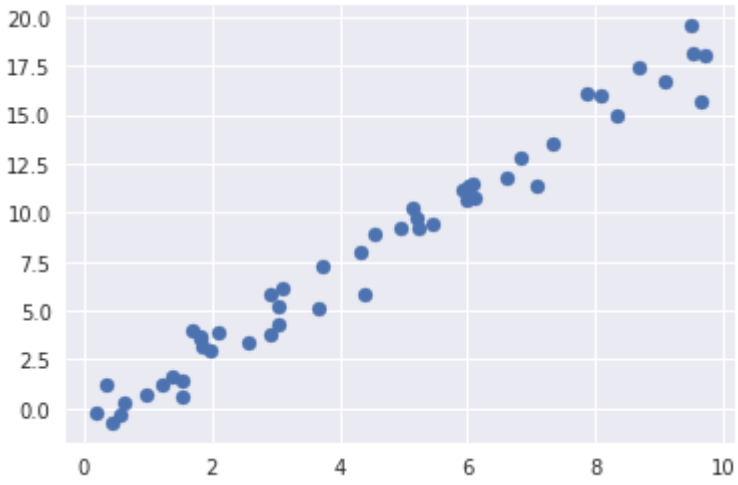
Essentiel de l'API

1. Choisir une classe de modèles (en important la classe corréspondante de Scikit-Learn).
2. Choisir les valeurs pour les hyperparamètres (en créant un objet de la classe spécifiant les valeurs).
3. Organiser les données en créant une matrice de *features* et un tableau de *labels*.
4. Adapter le modèle aux données (entraîner le modèle) en invocant la méthode `fit()` sur l'objet.
5. Utiliser le modèle sur des données nouveaux:
    - Apprentissage supervisé: prediction des étiquettes de nouveaux données en invocant la méthode `predict()` .
    - Apprentissage unsupervisé: effectuer une transformation ou une inférence en invocant les méthodes `transform()` et `predict()` .

Apprentissage supervisé: régression linéaire

In [181]:

```
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(42)
x = 10 * rng.rand(50)
y = 2 * x - 1 + rng.randn(50)
plt.scatter(x, y);
```



In [182]:

```
rng.rand(3)
```

Out[182]:

```
array([0.892559  , 0.53934224, 0.80744016])
```

1. Choisir une classe de modèles

In [183]:

```
from sklearn.linear_model import LinearRegression
```

2. Choisir les hyperparamètres

In [184]:

```
model = LinearRegression(fit_intercept=True)
model
```

Out[184]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=
False)
```

3. Construire *features* et *labels*

In [185]:

```
X = x[:, np.newaxis]
X.shape
```

Out[185]:

```
(50, 1)
```

In [186]:

```
# np.array([[el] for el in x])
```

In [187]:

```
(X[0], x[0])
```

Out[187]:

```
(array([3.74540119]), 3.745401188473625)
```

4. Adapter le modèle aux données

In [188]:

```
model.fit(X, y)
```

Out[188]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=
False)
```

In [189]:

```
model.coef_
```

Out[189]:

```
array([1.9776566])
```

In [190]:

```
model.intercept_
```

Out[190]:

```
-0.9033107255311164
```

5. Effectuer des prédiction sur nouveaux données

In [191]:

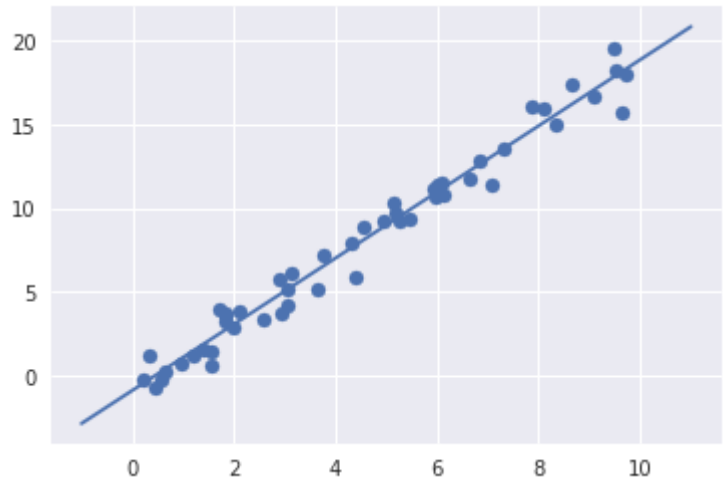
```
xfit = np.linspace(-1, 11)
```

In [192]:

```
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
```

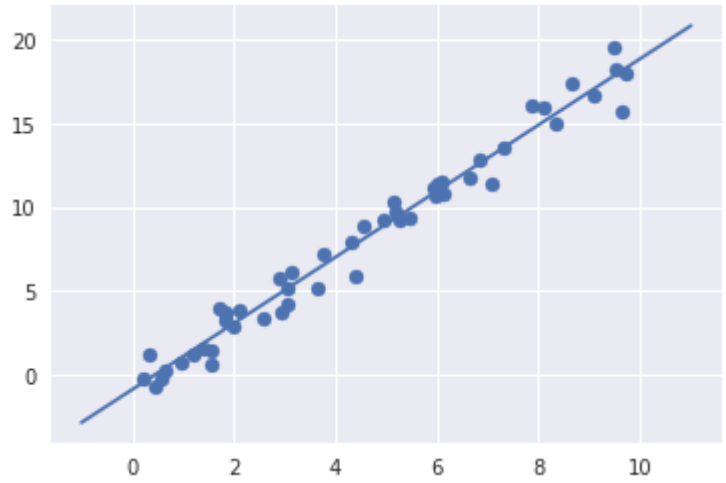
In [193]:

```
plt.scatter(x, y)
plt.plot(xfit, yfit);
```



In [28]:

```
xfit = np.array([-1, 11])
Xfit = xfit[:, np.newaxis]
yfit = model.predict(Xfit)
plt.scatter(x, y)
plt.plot(xfit, yfit);
```



Apprentissage supervisé: Naive Bayes

In [194]:

```
from sklearn.model_selection import train_test_split
```

In [195]:

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X_iris, y_iris,
                                                random_state=1,
                                                train_size=.6, test_size=.4)
```

In [196]:

```
from sklearn.naive_bayes import GaussianNB # 1. choose model class
model = GaussianNB()                       # 2. instantiate model
model.fit(Xtrain, ytrain)                  # 3. fit model to data
y_prediction = model.predict(Xtest)        # 4. predict on new data
```

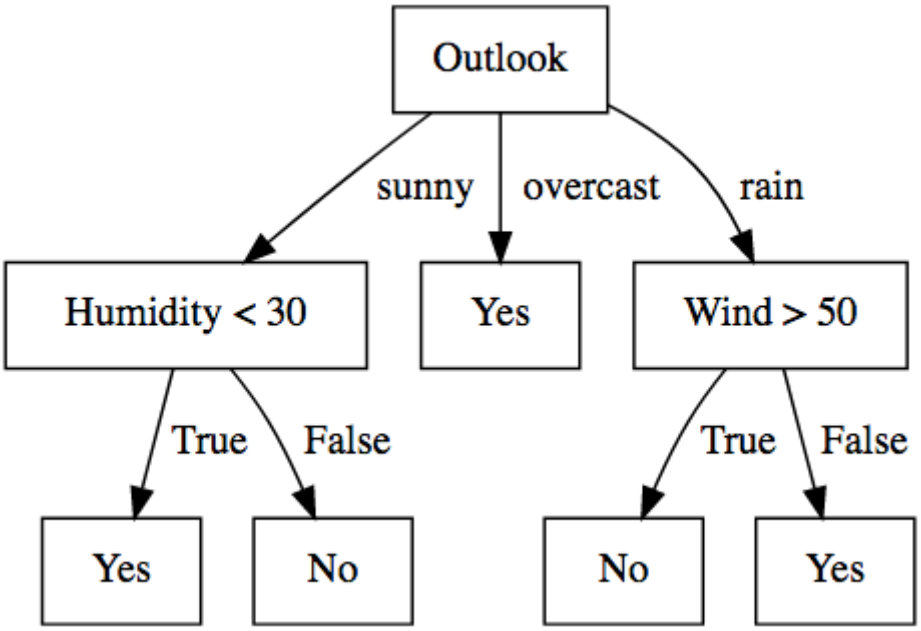
In [199]:

```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_prediction)
```

Out[199]:

0.95

Apprentissage supervisé: arbres de décision



In [200]:

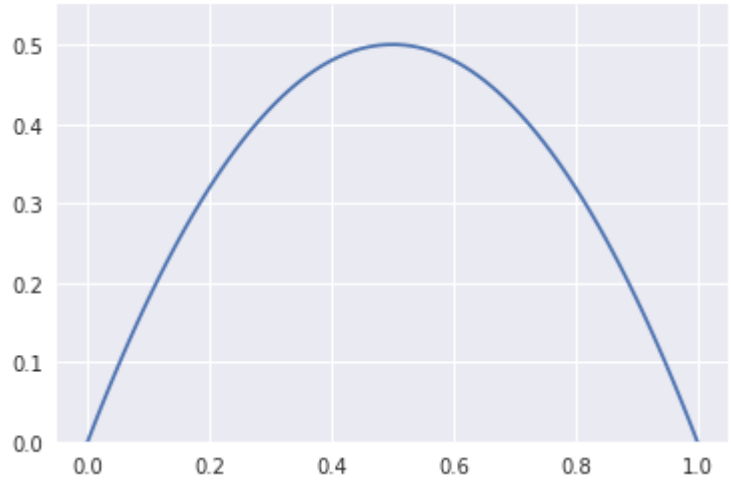
```
heroes = pd.read_csv('heroes.csv', sep=';', index_col=0)
heroes.head()
```

Out[200]:

Name	Identity	Birth place	Publisher	Height	Weight	Gender	First appearance	Eye color	Ha col
A-Bomb	Richard Milhouse Jones	Scarsdale, Arizona	Marvel Comics	203.21	441.95	M	2008.0	Yellow	N Ha
Abraxas	Abraxas	Within Eternity	Marvel Comics	NaN	NaN	M	NaN	Blue	Blac
Abomination	Emil Blonsky	Zagreb, Yugoslavia	Marvel Comics	203.04	441.98	M	NaN	Green	N Ha
Adam Monroe	NaN	NaN	NBC - Heroes	NaN	NaN	M	NaN	Blue	Blor
Agent 13	Sharon Carter	NaN	Marvel Comics	173.41	61.03	F	NaN	Blue	Blor

In [201]:

```
def gini_2_val(f):
    return 1 - f**2 - (1-f)**2

x = np.arange(0, 1.01, .01)
y = list(map(gini_2_val, x))

plt.plot(x, y)
plt.ylim((0, 0.55))
plt.show()
```



In [202]:

```
def gini(series):
    return 1 - (sum(series.value_counts(normalize=True)
                    .map(lambda f: f**2)))

publisher = heroes[heroes['Publisher'].isin(['Marvel Comics', 'DC Comics'])]['Publi
eye_color = heroes[pd.notnull(heroes['Eye color'])]['Eye color']
hair_color = heroes[pd.notnull(heroes['Hair color'])]['Hair color']

print(gini(publisher))
print(gini(eye_color))
print(gini(hair_color))
print(gini(heroes.index))
```

```
0.45945353273979217
0.77232789326401
0.8370723950922644
0.9985654125595816
```

In [203]:

```
good_guys = heroes.loc[['Wonder Woman',
                        'Aquaman',
                        'Cyborg',
                        'Flash II']]
bad_guys = heroes.loc[['Black Manta',
                       'Penguin',
                       'Joker',
                       'Deathstroke',
                       'Bizarro']]
all_guys = pd.concat([good_guys, bad_guys])

features = ['Height', 'Weight', 'Gender', 'First appearance',
            'Hair color', 'Eye color', 'Strength', 'Intelligence']
X = all_guys[features]
X
```

Out[203]:

Name	Height	Weight	Gender	First appearance	Hair color	Eye color	Strength	Intelligence
Wonder Woman	183.13	74.74	F	1941.0	Black	Blue	100.0	high
Aquaman	185.71	146.96	M	1941.0	Blond	Blue	85.0	high
Cyborg	198.12	173.81	M	1980.0	Black	Brown	55.0	good
Flash II	183.41	88.32	M	1956.0	Blond	Blue	50.0	high
Black Manta	188.12	92.78	M	1967.0	No Hair	Black	30.0	good
Penguin	157.89	79.13	M	1941.0	Black	Blue	10.0	good
Joker	196.07	86.91	M	1940.0	Green	Green	10.0	high
Deathstroke	193.87	101.98	M	1980.0	White	Blue	30.0	good
Bizarro	191.00	155.57	M	1958.0	Black	Black	95.0	moderate

In [204]:

```
Y = pd.concat([pd.DataFrame(['good guy'] * len(good_guys),
                            index=good_guys.index),
               pd.DataFrame(['bad guy'] * len(bad_guys),
                            index=bad_guys.index)])
```

In [205]:

```
Y[X['Strength'] <= 40]
```

Out[205]:

Name	0
Black Manta	bad guy
Penguin	bad guy
Joker	bad guy
Deathstroke	bad guy

In [206]:

```
Y[X['Strength'] > 40]
```

Out[206]:

Name	0
Wonder Woman	good guy
Aquaman	good guy
Cyborg	good guy
Flash II	good guy
Bizarro	bad guy

In [207]:

```
freq = Y[X['Strength'] <= 40][0].value_counts(normalize=True)
freq_bad = freq['bad guy']
gini_left = gini_2_val(freq_bad)
gini_left
```

Out[207]:

```
0.0
```

In [208]:

```
freq = Y[X['Strength'] > 40][0].value_counts(normalize=True)
freq_bad = freq['bad guy']
gini_right = gini_2_val(freq_bad)
gini_right
```

Out[208]:

```
0.31999999999999984
```

In [209]:

```
weight_left = len(Y[X['Strength'] <= 40]) / len(Y)
weight_right = len(Y[X['Strength'] > 40]) / len(Y)
gini_left * weight_left + gini_right * weight_right
```

Out[209]:

```
0.1777777777777777
```

In [210]:

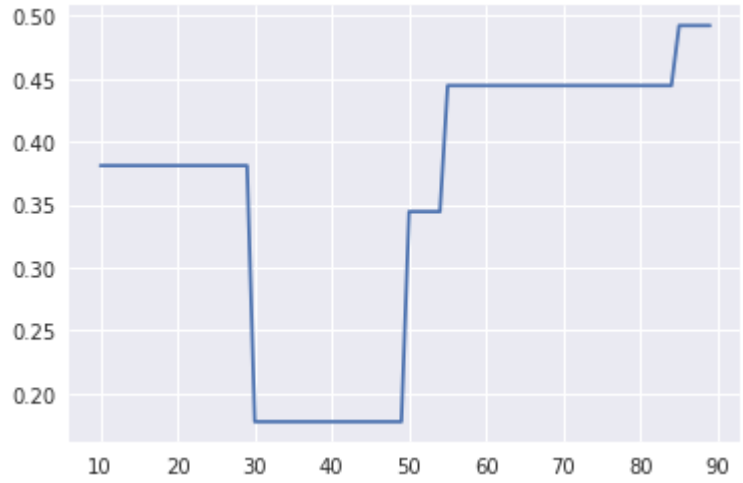
```
def split_value(attribute, value, index):
    freq = (Y[X[attribute] <= value])[0].value_counts(normalize=True)
    freq_bad = freq['bad guy']
    index_left = index(freq_bad)
    weight_left = len(Y[X[attribute] <= value]) / len(Y)
    freq = (Y[X[attribute] > value])[0].value_counts(normalize=True)
    freq_bad = freq['bad guy']
    index_right = index(freq_bad)
    weight_right = len(Y[X[attribute] > value]) / len(Y)
    return index_left * weight_left + index_right * weight_right

split_value('Strength', 40, gini_2_val)
```

Out[210]:

```
0.1777777777777777
```

In [211]:

```
plt.plot(range(10,90),
         list(map(lambda v: split_value('Strength', v, gini_2_val),
                  range(10,90))))
plt.show()
```



In [212]:

```
from sklearn.preprocessing import LabelEncoder


gender_encoder = LabelEncoder()
gender_encoder.fit(all_guys['Gender'])

eye_col_encoder = LabelEncoder()
eye_col_encoder.fit(all_guys['Eye color'])

hair_col_encoder = LabelEncoder()
hair_col_encoder.fit(all_guys['Hair color'])

intelligence_encoder = LabelEncoder()
_ = intelligence_encoder.fit(all_guys['Intelligence'])
```

In [213]:

```
all_guys['Gender'] = gender_encoder.transform(all_guys['Gender'])
all_guys['Eye color'] = eye_col_encoder.transform(all_guys['Eye color'])
all_guys['Hair color'] = hair_col_encoder.transform(all_guys['Hair color'])
all_guys['Intelligence'] = intelligence_encoder.transform(all_guys['Intelligence'])
```

In [214]:

```
X = all_guys[features]
X
```

Out[214]:

Name	Height	Weight	Gender	First appearance	Hair color	Eye color	Strength	Intelligence
Wonder Woman	183.13	74.74	0	1941.0	0	1	100.0	1
Aquaman	185.71	146.96	1	1941.0	1	1	85.0	1
Cyborg	198.12	173.81	1	1980.0	0	2	55.0	0
Flash II	183.41	88.32	1	1956.0	1	1	50.0	1
Black Manta	188.12	92.78	1	1967.0	3	0	30.0	0
Penguin	157.89	79.13	1	1941.0	0	1	10.0	0
Joker	196.07	86.91	1	1940.0	2	3	10.0	1
Deathstroke	193.87	101.98	1	1980.0	4	1	30.0	0
Bizarro	191.00	155.57	1	1958.0	0	0	95.0	2

In [215]:

```
from sklearn import tree

clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)

predictions = clf.predict([X.loc[name] for name in X.index])
predictions
```

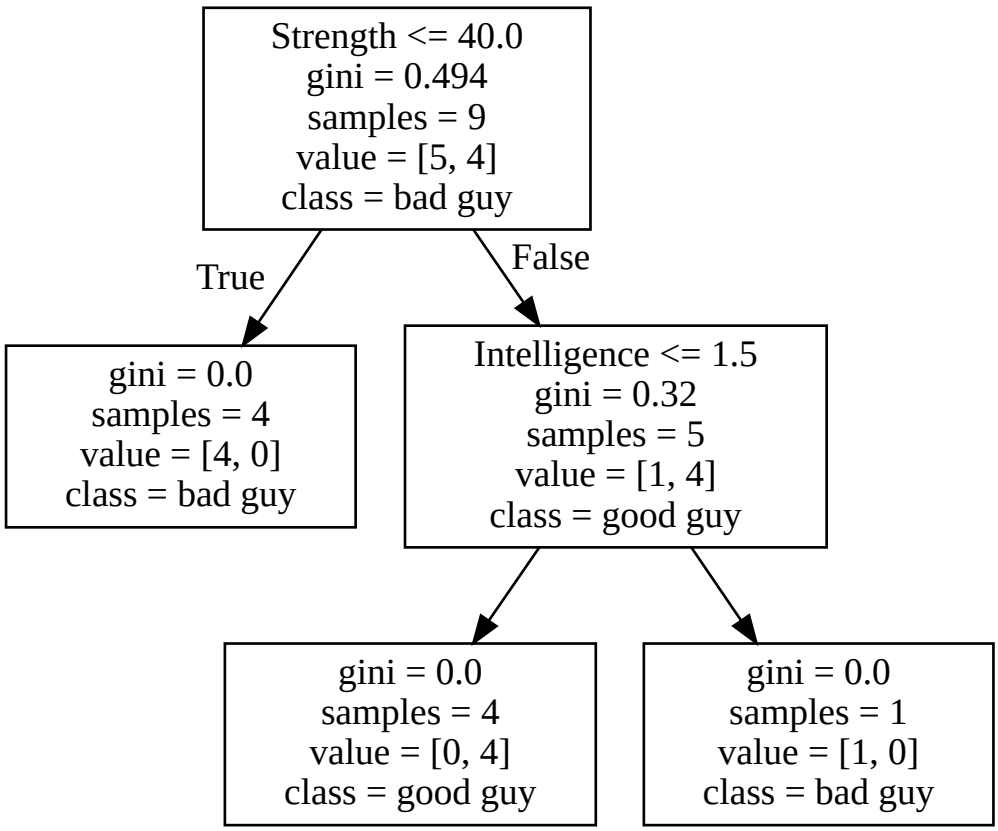
Out[215]:

```
array(['good guy', 'good guy', 'good guy', 'good guy', 'bad guy',
       'bad guy', 'bad guy', 'bad guy', 'bad guy'], dtype=object)
```

In [216]:

```
import graphviz

graphviz.Source(tree.export_graphviz(clf, out_file=None,
                                     class_names=['bad guy', 'good guy'],
                                     feature_names=features))
```

Out[216]:



In [163]:

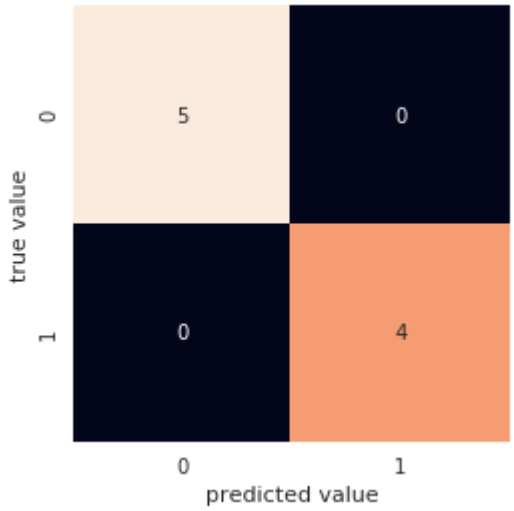
```
pd.DataFrame(np.array((X.index, Y[0], predictions)))
```

Out[163]:

	0	1	2	3	4	5	6	7	8
0	Wonder Woman	Aquaman	Cyborg	Flash II	Black Manta	Penguin	Joker	Deathstroke	Bizarro
1	good guy	good guy	good guy	good guy	bad guy	bad guy	bad guy	bad guy	bad guy
2	good guy	good guy	good guy	good guy	bad guy	bad guy	bad guy	bad guy	bad guy

In [162]:

```
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(Y, predictions)

sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```



In [164]:

```
def filter(obj):
    transformed_obj = obj['Height':'Intelligence']
    transformed_obj['Gender'] = gender_encoder.transform([transformed_obj['Gender']
    transformed_obj['Eye color'] = eye_col_encoder.transform([transformed_obj['Eye
    transformed_obj['Hair color'] = hair_col_encoder.transform([transformed_obj['Ha
    transformed_obj['Intelligence'] = intelligence_encoder.transform([transformed_o
    return transformed_obj

clf.predict([filter(heroes.loc['Professor X'])])
```

Out[164]:

```
array(['bad guy'], dtype=object)
```

Apprentissage unsupervisé: PCA

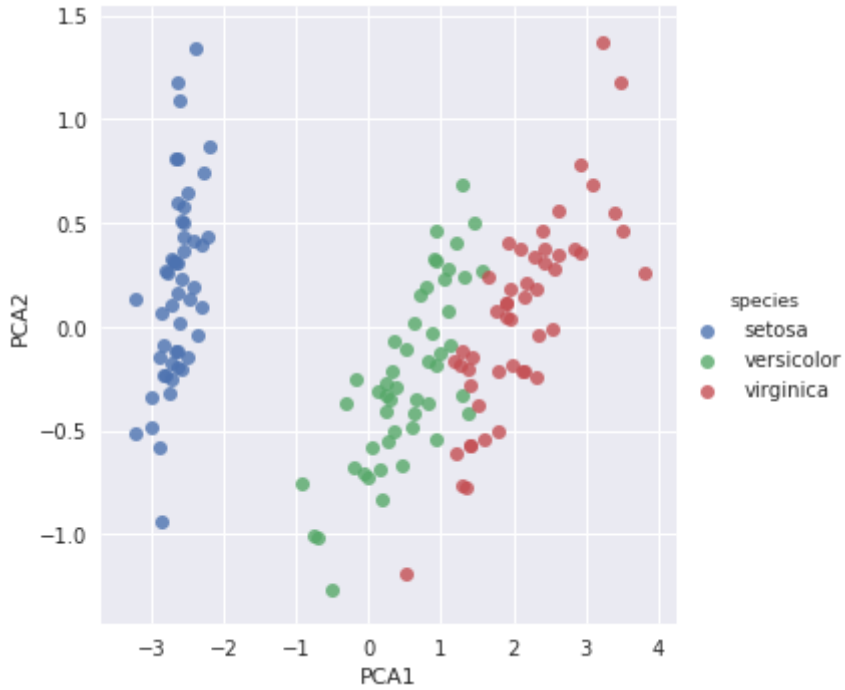
In [217]:

```
from sklearn.decomposition import PCA  # 1. Choose the model class
model = PCA(n_components=2)            # 2. Instantiate the model with hyperparamet
model.fit(X_iris)                      # 3. Fit to data. Notice y is not specified!
X_2D = model.transform(X_iris)         # 4. Transform the data to two dimensions
```

Now let's plot the results. A quick way to do this is to insert the results into the original Iris `DataFrame` , and use Seaborn's `lmplot` to show the results:

In [218]:

```
iris['PCA1'] = X_2D[:, 0]
iris['PCA2'] = X_2D[:, 1]
sns.lmplot("PCA1", "PCA2", hue='species', data=iris, fit_reg=False);
```



We see that in the two-dimensional representation, the species are fairly well separated, even though the PCA algorithm had no knowledge of the species labels! This indicates to us that a relatively straightforward classification will probably be effective on the dataset, as we saw before.

Aprentissage unsupervisé: partitionnement

In [106]:

```
from sklearn.cluster import KMeans  # 1. Choose the model class
model = KMeans(n_clusters=3)        # 2. Instantiate the model with hyperparameters
model.fit(X_iris)                   # 3. Fot to data. Notice y is not specified!
y_kmeans = model.predict(X_iris)    # 4. Determine cluster labels
```

In [107]:

```
iris['cluster'] = y_kmeans
sns.lmplot("PCA1", "PCA2", data=iris, hue='species',
           col='cluster', fit_reg=False);
```

In [108]:

```
import pandas as pd

label_to_cluster = {}
for cluster in range(3):
    label = pd.Series([y_iris[i] for i in range(len(y_gmm)) if y_gmm[i]==cluster]).
    label_to_cluster[label] = cluster

label_to_cluster
```

Out[108]:

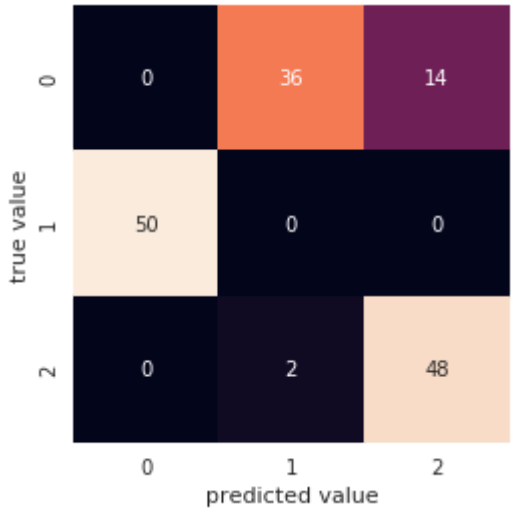
```
{'virginica': 0, 'setosa': 1, 'versicolor': 2}
```

In [110]:

```
y_labels = [label_to_cluster[label] for label in y_iris]
```

In [114]:

```
mat = confusion_matrix(y_labels, y_kmeans)

sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```



Aprentissage unsupervisé: partitionnement

In [79]:

```
from sklearn.mixture import GaussianMixture      # 1. Choose the model class
model = GaussianMixture(n_components=3)          # 2. Instantiate the model with hy
model.fit(X_iris)                                # 3. Fit to data.
y_gmm = model.predict(X_iris)                    # 4. Determine cluster labels
```

In [81]:

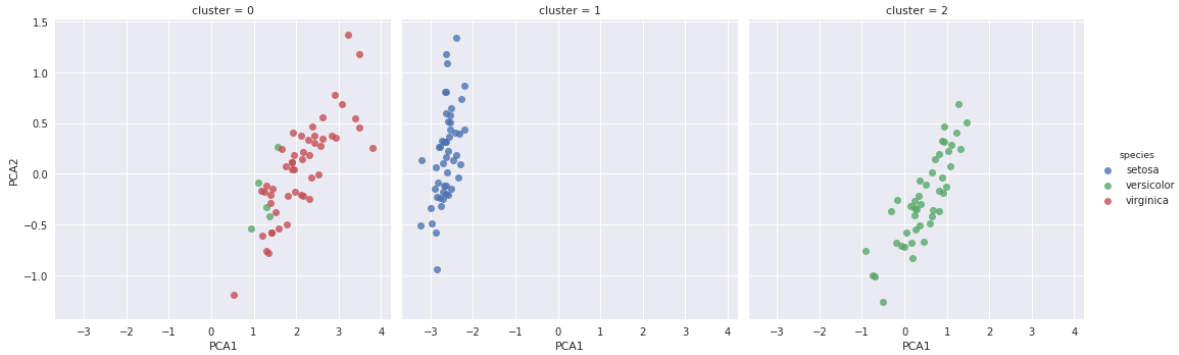
```
y_gmm
```

Out[81]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 0, 2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2,
2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [80]:

```
iris['cluster'] = y_gmm
sns.lmplot("PCA1", "PCA2", data=iris, hue='species',
           col='cluster', fit_reg=False);
```



In [101]:

```
label_to_cluster = {}
for cluster in range(3):
    label = pd.Series([y_iris[i] for i in range(len(y_gmm)) if y_gmm[i]==cluster]).
    label_to_cluster[label] = cluster

label_to_cluster
```

Out[101]:

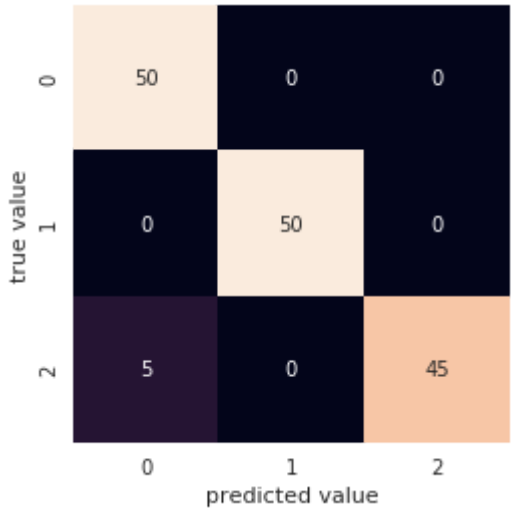
```
{'virginica': 0, 'setosa': 1, 'versicolor': 2}
```

In [103]:

```
y_labels = [label_to_cluster[label] for label in y_iris]
```

In [104]:

```
mat = confusion_matrix(y_labels, y_gmm)

sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```



Application: Exploration de chiffres manuscrites

In [115]:

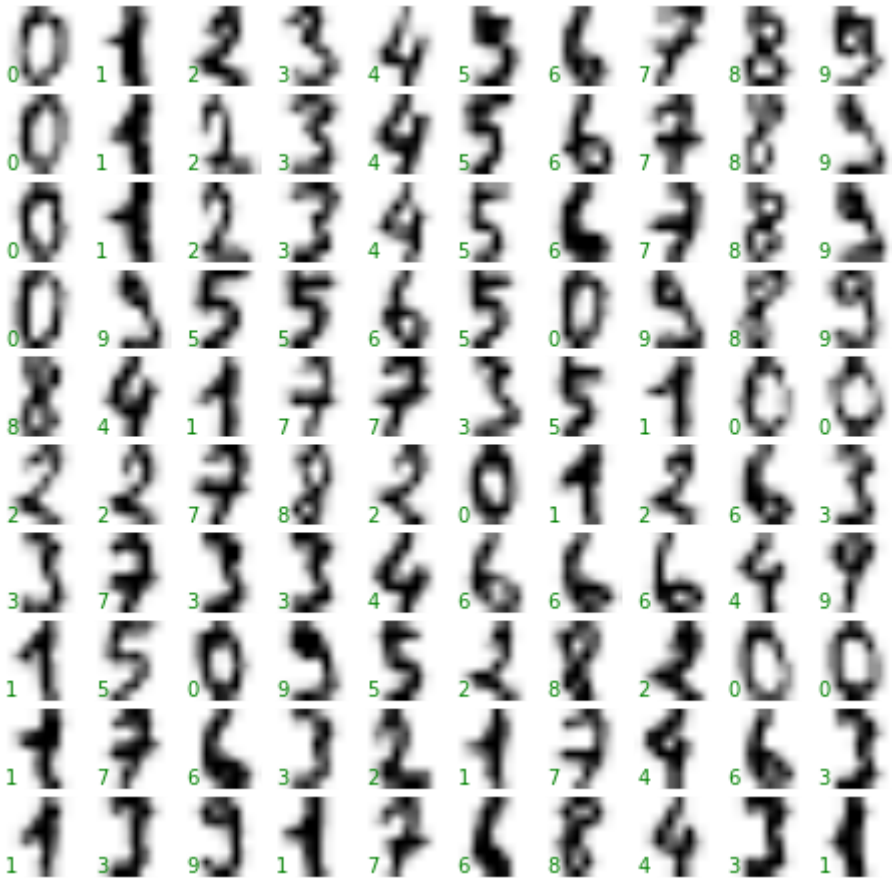
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

Out[115]:

```
(1797, 8, 8)
```

In [124]:

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

for i, ax in enumerate(axes.flat):
    ax.imshow(digits.images[i], cmap='Greys', interpolation='bilinear') # nearest,
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```



In [121]:

```
X = digits.data
X.shape
```

Out[121]:

```
(1797, 64)
```

In [122]:

```
y = digits.target
y.shape
```

Out[122]:

```
(1797,)
```

In [127]:

```
model = PCA(n_components=2)
model.fit(digits.data)
digits_2D = model.transform(digits.data)
digits_2D.shape
```

Out[127]:

```
(1797, 2)
```

In [129]:

```
plt.scatter(digits_2D[:, 0], digits_2D[:, 1], c=digits.target,
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('coolwarm', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);
```

In [45]:

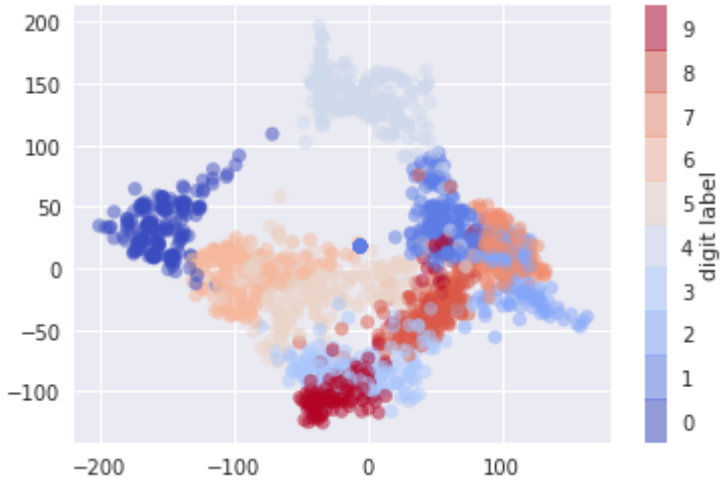
```
from sklearn.manifold import Isomap
iso = Isomap(n_components=2)
iso.fit(digits.data)
data_projected = iso.transform(digits.data)
data_projected.shape
```

Out[45]:

```
(1797, 2)
```

In [49]:

```
plt.scatter(data_projected[:, 0], data_projected[:, 1], c=digits.target,
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('coolwarm', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);
```



Classification des chiffres: Naive Bayes

In [130]:

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
```

In [131]:

```
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

In [132]:

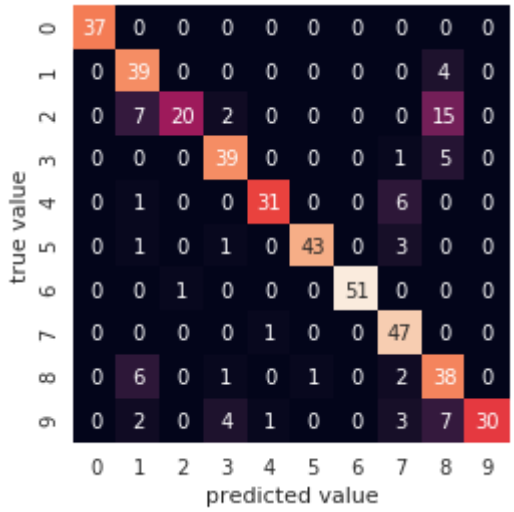
```
accuracy_score(ytest, y_model)
```

Out[132]:

```
0.8333333333333334
```

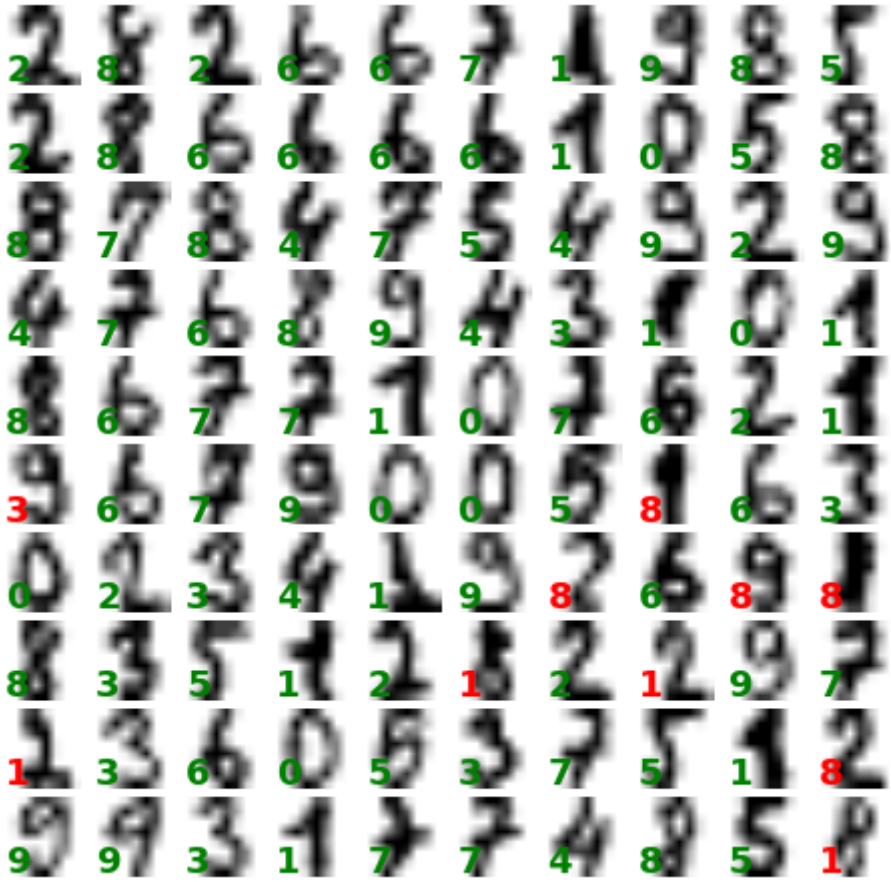
In [133]:

```
mat = confusion_matrix(ytest, y_model)

sns.heatmap(mat, square=True, annot=True, cbar=False)
plt.xlabel('predicted value')
plt.ylabel('true value');
```



In [137]:

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))

test_images = Xtest.reshape(-1, 8, 8)

for i, ax in enumerate(axes.flat):
    ax.imshow(test_images[i], cmap='binary', interpolation='bilinear')
    ax.text(0.05, 0.05, str(y_model[i]),
            transform=ax.transAxes,
            color='green' if (ytest[i] == y_model[i]) else 'red',
            weight='bold',
            size=18)
```



Y a plus de choses!

- holdout répété
- validation croisée
- un tas d'autres modèles (résaux de neurones, simples ou profondes, SVM, ...)

In [ ]:

```
```