UNIVERSITÄT BIELEFELD
Faculty of Technology

Programming

Applied Machine Learning

Daniel Dörr

Faculty of Technology, Bielefeld University

Recap

Pandas data structures

Series

- Container for scalar values
- 1D array
- More powerful than a “1D NumPy array”
- Allows to freely set index
- Size immutable

Data Frame

- Container for Series
- 2D array / table
- Mutability
  - Rows are immutable
  - Allows insertion of new columns

UNIVERSITÄT BIELEFELD

Machine Learning

Scikit-Learn

Applications

Machine Learning

- branch of artificial intelligence
- combination of statistics, optimization theory, CS, information th., ...

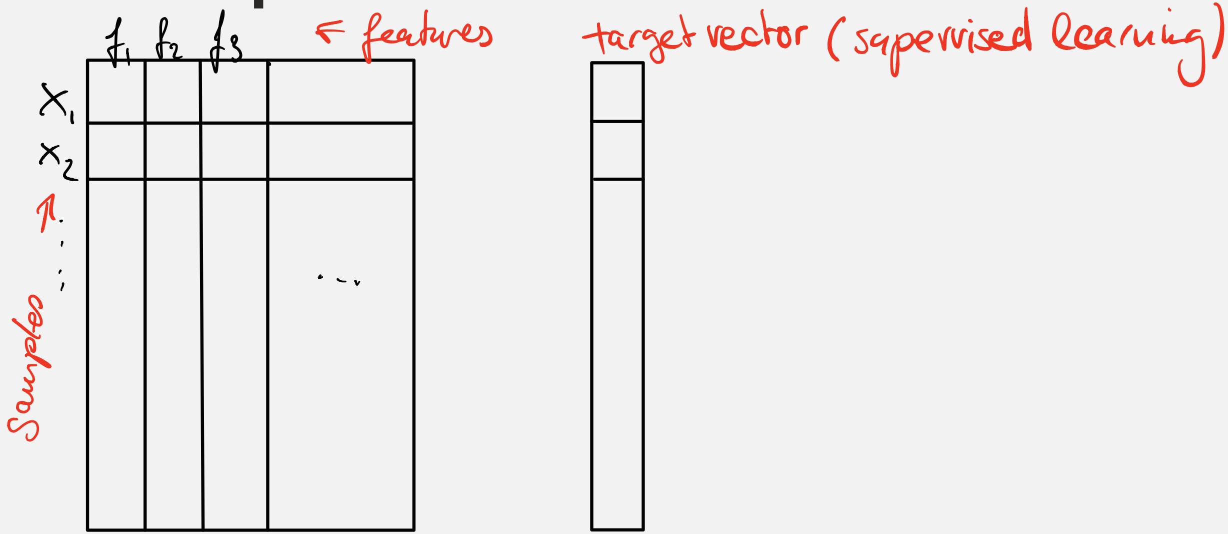
Unsupervised Learning

- Dimensionality reduction
- Clustering

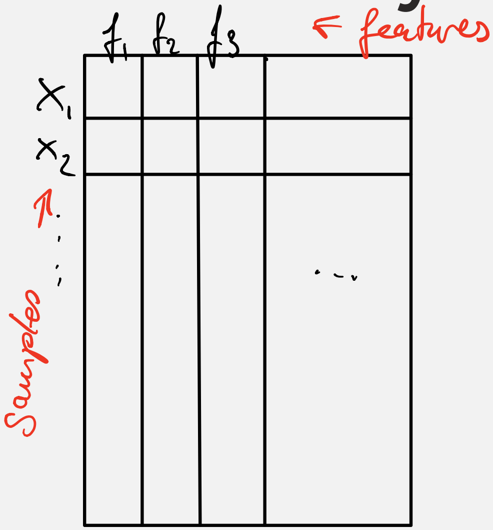
Supervised Learning

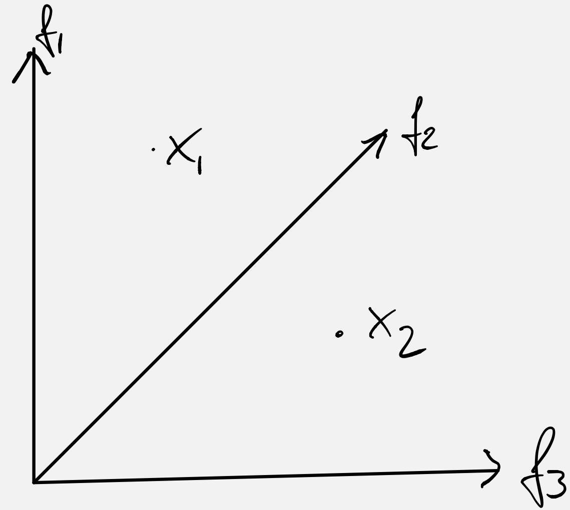
- Classification
- Regression

Data representation: feature matrix



Dimensionality reduction

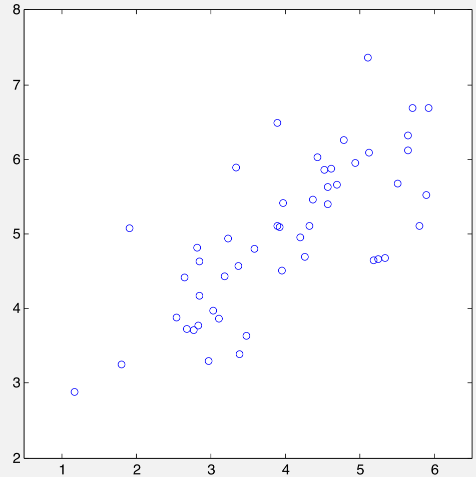




Some methods:

- Principal Component Analysis (PCA)
- Isomap

Dimensionality reduction

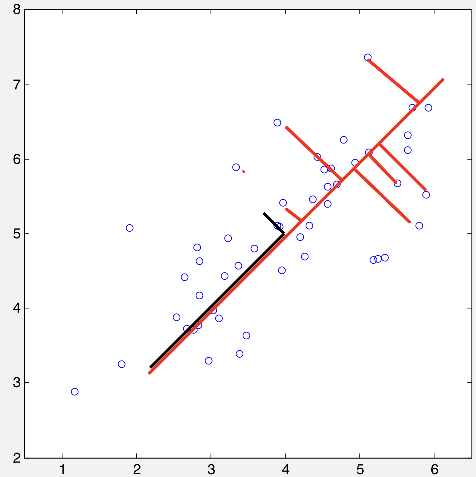


Some methods:

- Principal Component Analysis (PCA)
- Isomap

sources: Andrew Ng, ML class; `https://scikit-learn.org/stable/modules/clustering.html`

Dimensionality reduction



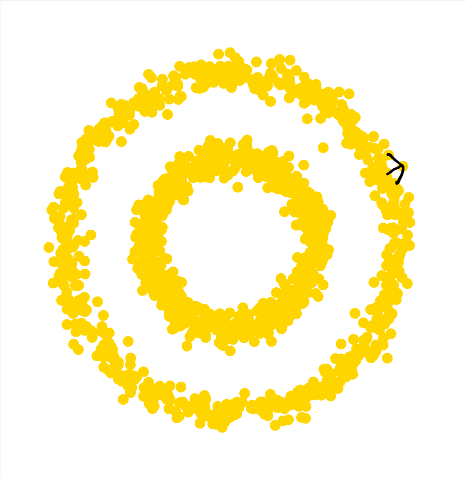
Some methods:

- Principal Component Analysis (PCA)
- Isomap

sources: Andrew Ng, ML class; `https://scikit-learn.org/stable/modules/clustering.html`

UNIVERSITÄT BIELEFELD

Dimensionality reduction



Some methods:

- Principal Component Analysis (PCA)
- Isomap

sources: Andrew Ng, ML class; `https://scikit-learn.org/stable/modules/clustering.html`

UNIVERSITÄT BIELEFELD

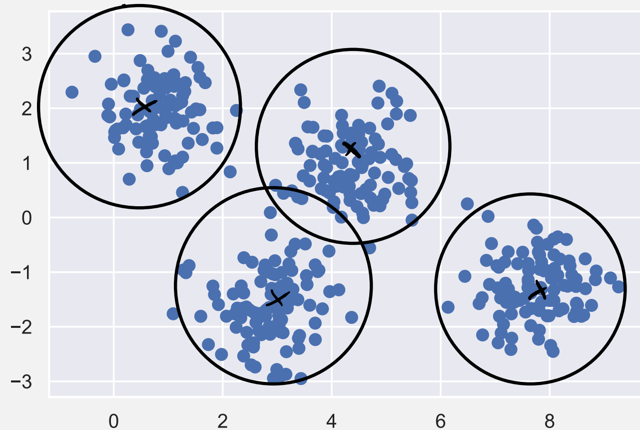
Clustering

- unsupervised
- assigns (cluster) labels to data points

Some methods:

- K-means :
- Gaussian Mixture Models (GMM)
- Spectral Clustering

Clustering



× cluster center (centroid)

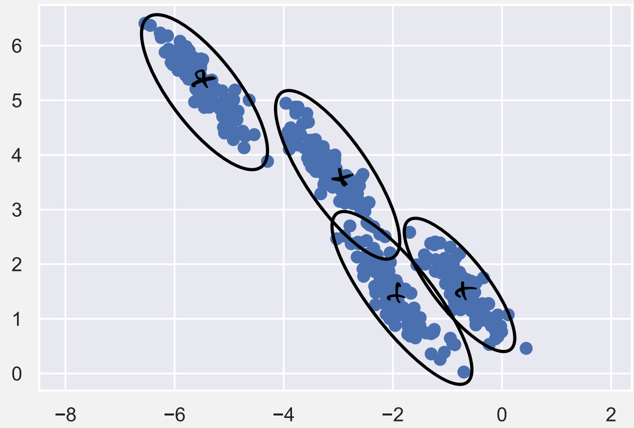
Some methods:

- K-means : Each point is assigned to the cluster with the nearest cluster center
- Gaussian Mixture Models (GMM)
- Spectral Clustering

sources: Jake VanderPlas, Python Data Science Handbook; `https://scikit-learn.org/stable/modules/clustering.html`

UNIVERSITÄT
BIELEFELD

Clustering

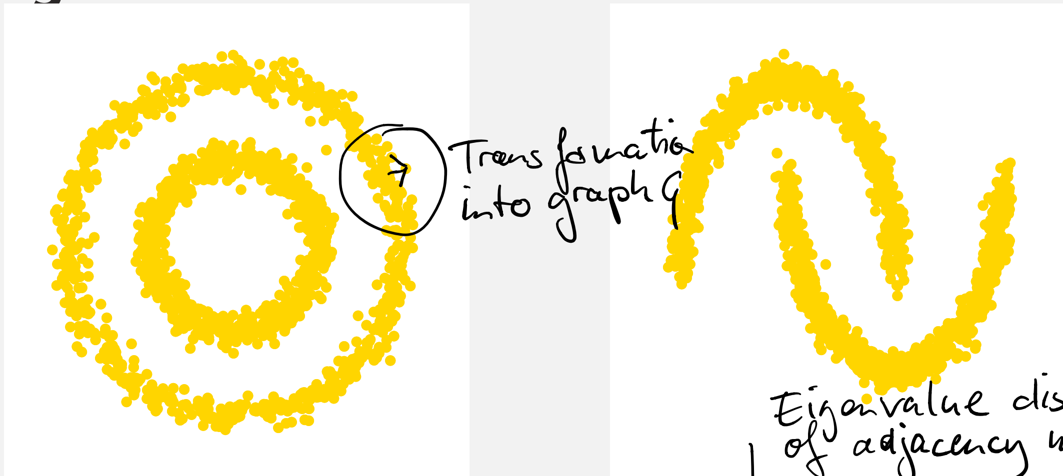


Some methods:

- K-means
- Gaussian Mixture Models (GMM)
- Spectral Clustering

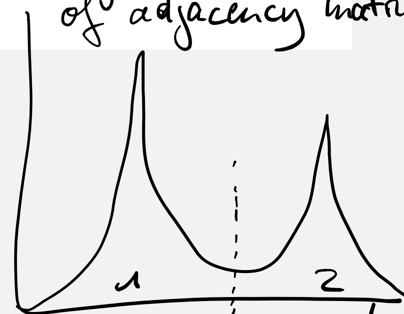
sources: Jake VanderPlas, Python Data Science Handbook; `https://scikit-learn.org/stable/modules/clustering.html`

Clustering



Transformation into graph G

Eigenvalue distribution of adjacency matrix of G



components

Some methods:

- K-means
- Gaussian Mixture Models (GMM)
- Spectral Clustering

dimension reduction

sources: Jake VanderPlas, Python Data Science Handbook; https://scikit-learn.org/stable/modules/clustering.html

UNIVERSITÄT BIELEFELD

Classification

- supervised ML

classification ≠ clustering

unsupervised

Some methods:

- Naive Bayes
- Decision Trees

UNIVERSITÄT BIELEFELD

Classification

- Relies on Bayes' theorem (relationship of conditional probabilities)
- Idea: assign label L to sample based on probability

$$P(L \mid \text{features}) = \frac{P(\text{features} \mid L) \cdot P(L)}{P(\text{features})}$$

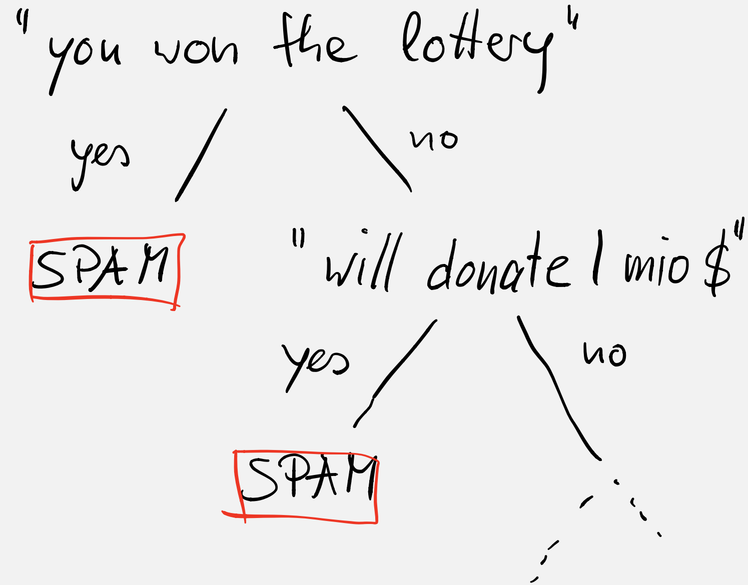
obtained in training

Some methods:

- Naive Bayes
- Decision Trees

Classification

Spam-mail classifier



Some methods:

- Naive Bayes
- Decision Trees

UNIVERSITÄT BIELEFELD

Regression

Some methods:

- Linear regression, ridge regression, Lasso regression

  $Y = a + bX + cX^2 + \ldots$
- Multiple regression

  X is a vector (ex: 12 temp. measurement/year in GISS)
- Multivariate regression

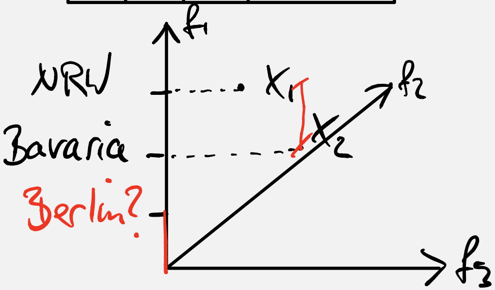
  Y is a matrix (outcome is not a single point but a plane or higher-dim. object)

Feature representation

← features

	f_1	f_2	f_3	
x_1	NRW			
x_2	Bavaria			
⋮				...

↑ samples



Categorical features
(e.g. federal states)

NRW − Bavaria = Berlin? No!

Solution:

	f_1					f_2	f_3
	NRW	Bavaria	Berlin	...			
x_1	1	0	0	0	...		
x_2	0	1	0	0	...		
					...		

Text features

terms

documents

I	may	region	Petersburgh	sister
10	6	3	2	2
⋮	⋮	⋮	⋮	⋮

← word count vector

Problem: Frequent words often not characteristic of the document

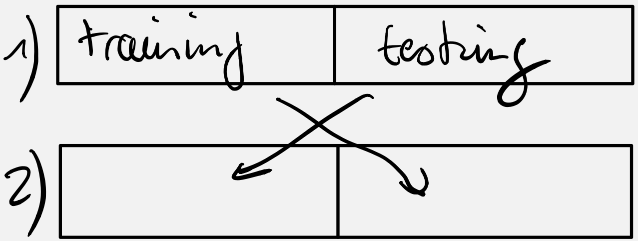
Solution: Normalization

e.g. term frequency - inverse document frequency (tf-idf)

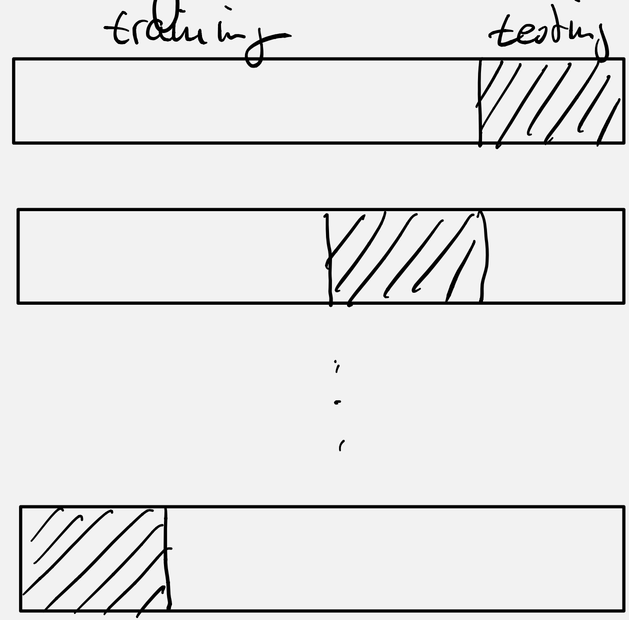
→ weights terms antiproportional to their frequency in document

Cross-validation

Data set:



More general



Summarize evaluation:
- mean
- min
- whisker
- ...

Quiz

- Assign the following methods to their categories:
  - Naive Bayes
  - Kmeans
  - PCA
  - Decision Tree
  - Gaussian Mixture Models
  - Isomap
  - Spectral Clustering

- *True* or *false?*
  - Cross validation can only be performed on labeled data
  - Gaussian Mixture Models assumes that data points follow a normal distribution
  - Categorical feature must be transformed priot to machine learning analysis

Quiz

- Assign the following methods to their categories:
  - Naive Bayes — Classification
  - Kmeans — Clustering
  - PCA — Dimensionality red.
  - Decision Tree — Classification
  - Gaussian Mixture Models — Clustering
  - Isomap — Dimensionality red.
  - Spectral Clustering — Clustering

- *True* or *false?*
  - Cross validation can only be performed on labeled data — true
  - Gaussian Mixture Models assumes that data points follow a normal distribution — true
  - Categorical feature must be transformed priot to machine learning analysis — true

Machine Learning

Scikit-Learn

Applications

The Estimator API

Estimators of the Scikit-Learn package share a common API.

Use of estimators:

- Choose model (Estimator)
- Choose model hyperparameters
- Instantiate model with hyperparameters
- Call `fit()` to train the model on a given data set
- Apply model to new data:
  - Supervised learning: call `predict()`
  - Unsupervised learning: call `transform()` or `predict()` (depending on the estimator)

The Estimator API

Supervised learning

In [1]:

```
from sklearn.linear_model import LinearRegression

# further packages that are necessary for the analysis
import matplotlib.pylab as plt
import pandas as pd
import numpy as np

temp_data = pd.read_csv('course_material_07/Temp_global-mean-monthly.csv', header=
0)
temp_data.head()
```

Out[1]:

	1881	1882	1883	1884	1885	1886	1887	1888	1889	1890	...	2010	2011	2012	2013	2014	2015	2016	2017	2018
0	-0.19	0.17	-0.28	-0.12	-0.58	-0.42	-0.70	-0.33	-0.07	-0.41	...	0.75	0.52	0.50	0.71	0.76	0.85	1.17	1.02	0.82
1	-0.13	0.15	-0.36	-0.07	-0.32	-0.49	-0.55	-0.35	0.18	-0.45	...	0.83	0.49	0.49	0.63	0.55	0.90	1.37	1.14	0.85
2	0.04	0.05	-0.12	-0.35	-0.25	-0.42	-0.34	-0.40	0.07	-0.39	...	0.91	0.65	0.57	0.67	0.79	0.96	1.36	1.16	0.90
3	0.06	-0.16	-0.17	-0.39	-0.41	-0.27	-0.33	-0.19	0.10	-0.29	...	0.84	0.65	0.71	0.56	0.81	0.77	1.12	0.94	0.90
4	0.07	-0.14	-0.17	-0.34	-0.44	-0.23	-0.29	-0.21	0.00	-0.39	...	0.76	0.52	0.77	0.62	0.85	0.79	0.96	0.90	0.83

5 rows × 139 columns

Step 1: create model

In [2]:
```
model = LinearRegression(fit_intercept=True)
model
```

Out[2]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

Step 2: train model (fitting)

In [3]:

```
X = temp_data.columns.to_numpy(dtype=int)[:, np.newaxis]
Y = temp_data.iloc[0]

# train model by calling 'fit' function
model.fit(X, Y)

print('Intercept and slope of regression line:')
print(model.intercept_, model.coef_)
```

```
Intercept and slope of regression line:
-15.984774118593323 [0.00821887]
```

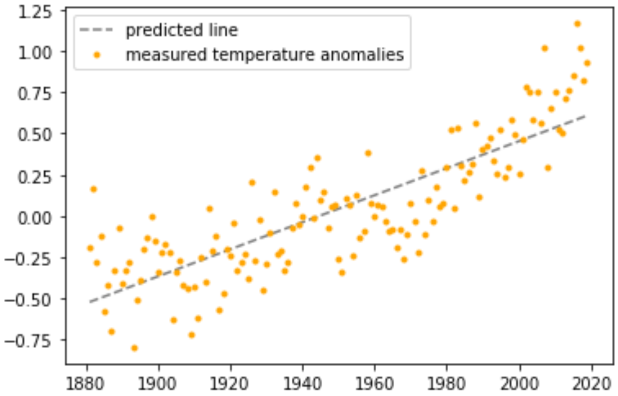
Step 3: use model to make predictions

In [4]:

```
Yhat = model.predict(X)
```

Visualize outcome

In [5]:
```
plt.plot(X.flatten(), Yhat.flatten(), '--', color='gray')
plt.plot(X.flatten(), Y.to_numpy().flatten(), '.', color='orange')
ls = plt.legend(('predicted line', 'measured temperature anomalies'))
```



Model evaluation: Sci-kit learn provides a range of metrics to evaluate model predictions, including the R2 score:

In [6]:
```
from sklearn.metrics import r2_score

r2_score(Y, Yhat)
```

Out[6]: 0.686079358654462

Unsupervised learning

Generate some data

In [7]:
```
from sklearn.datasets import make_blobs
X, Y = make_blobs(n_samples=400, centers=4, cluster_std=0.60, random_state=0)
X = X[:, ::-1] # flip axes for better plotting
```

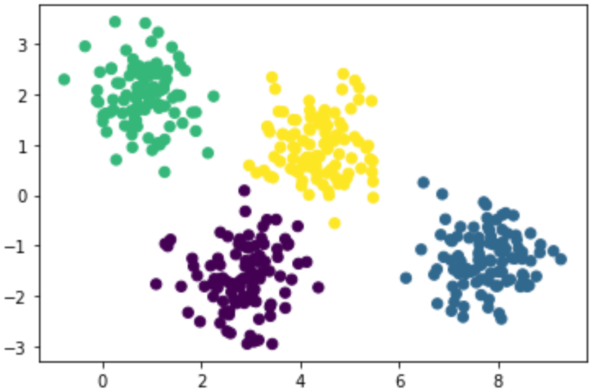
Model, fit, predict!

In [8]:

```
from sklearn.cluster import KMeans

model = KMeans(4, random_state=0)
model.fit(X)
Yhat = model.predict(X)
```

Plot outcome of the clustering

In [9]:
```
plt.scatter(X[:, 0], X[:, 1], c=Yhat, s=40, cmap='viridis');
```



Feature representation

In [10]:
```
demogrphx = pd.read_table('course_material_07/12111-04-01-4-B_processed3.tsv',
                          header=0)
demogrphx.head()
```

Out[10]:

	FederalState	Age	PopulationMale	PopulationFemale	PopulationTotal
0	Schleswig-Holstein	0	11132	10400	21532
1	Schleswig-Holstein	1	11504	10360	21864
2	Schleswig-Holstein	2	11733	11067	22800
3	Schleswig-Holstein	3	12214	11147	23361
4	Schleswig-Holstein	4	12142	10945	23087

Vectorizing categorical features

Transforming DataFrames with OneHotEncoder

In [11]:

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# construct "model"
categorical_trans = OneHotEncoder(handle_unknown='ignore')
column_trans = ColumnTransformer(
    transformers=[('FederalState', categorical_trans, [0])], remainder='passthrough')
# "fit" model
column_trans.fit(demogrphx)
# transform data
column_trans.transform(demogrphx)
# alternatively, do both steps at once:
trans_data = column_trans.fit_transform(demogrphx)
# transform outcome into a DataFrame for better display
pd.DataFrame(trans_data.toarray()).head()
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
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	1.0	0.0	0.0	11132.0	10400.0	21532.0
1	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	1.0	0.0	1.0	11504.0	10360.0	21864.0
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	1.0	0.0	2.0	11733.0	11067.0	22800.0
3	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	1.0	0.0	3.0	12214.0	11147.0	23361.0
4	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	1.0	0.0	4.0	12142.0	10945.0	23087.0

Transforming data with the DictVectorizer

Let's create some dictionary data for this example:

In [12]:
```
data_dict = demogrphx.to_dict(orient='records')
data_dict[:3]
```

Out[12]:
```
[{'FederalState': 'Schleswig-Holstein',
  'Age': 0,
  'PopulationMale': 11132,
  'PopulationFemale': 10400,
  'PopulationTotal': 21532},
 {'FederalState': 'Schleswig-Holstein',
  'Age': 1,
  'PopulationMale': 11504,
  'PopulationFemale': 10360,
  'PopulationTotal': 21864},
 {'FederalState': 'Schleswig-Holstein',
  'Age': 2,
  'PopulationMale': 11733,
  'PopulationFemale': 11067,
  'PopulationTotal': 22800}]
```

In [13]:

```
from sklearn.feature_extraction import DictVectorizer

# create "model"
vec = DictVectorizer(sparse=False, dtype=int)

# apply fit/transform to data_dict
trans_dict = vec.fit_transform(data_dict)

# transform outcome into a DataFrame for better display
pd.DataFrame(trans_dict, columns=vec.get_feature_names()).head()
```

Out[13]:

	Age	FederalState=Baden-Wuerttemberg	FederalState=Bayern	FederalState=Berlin	FederalState=Brandenburg	FederalState=Bremen	FederalState
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0
3	3	0	0	0	0	0	0
4	4	0	0	0	0	0	0

In [14]:
```
vec.get_feature_names()
```

Out[14]:
```
['Age',
 'FederalState=Baden-Wuerttemberg',
 'FederalState=Bayern',
 'FederalState=Berlin',
 'FederalState=Brandenburg',
 'FederalState=Bremen',
 'FederalState=Hamburg',
 'FederalState=Hessen',
 'FederalState=Mecklenburg-Vorpommern',
 'FederalState=Niedersachsen',
 'FederalState=Nordrhein-Westfalen',
 'FederalState=Rheinland-Pfalz',
 'FederalState=Saarland',
 'FederalState=Sachsen',
 'FederalState=Sachsen-Anhalt',
 'FederalState=Schleswig-Holstein',
 'FederalState=Thueringen',
 'PopulationFemale',
 'PopulationMale',
 'PopulationTotal']
```

Vectorizing text features

Some toy example:

In [15]:
```
raw_texts = [
    'When you have seen more of this country, I am afraid you will think you have
 overrated Hartfield.',
    'His cold politeness, his ceremonious grace, were worse than anything.',
    'He left her house yesterday, but where he is gone, or whether he is still in
 town, I do not know; for WE of course can make no inquiry.']
```

In [16]:
```
from sklearn.feature_extraction.text import CountVectorizer

# create "model"
vec = CountVectorizer()

# apply fit/transform to raw_texts
trans_text = vec.fit_transform(raw_texts)

# transform outcome into a DataFrame for better display
pd.DataFrame(trans_text.toarray(), columns=vec.get_feature_names())
```

Out[16]:

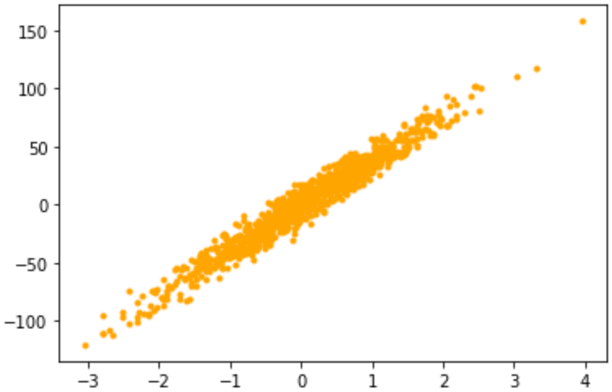
	afraid	am	anything	but	can	ceremonious	cold	country	course	do	...	town	we	were	when	where	whether	will	wo
0	1	1	0	0	0	0	0	1	0	0	...	0	0	0	1	0	0	1	0
1	0	0	1	0	0	1	1	0	0	0	...	0	0	1	0	0	0	0	1
2	0	0	0	1	1	0	0	0	1	1	...	1	1	0	0	1	1	0	0

3 rows × 47 columns

Separating test- and training data

Generate example date with Sci-kit learn data generator

In [17]:
```
from sklearn.datasets import make_regression

X, Y, coef = make_regression(n_samples = 1000, n_features = 1, noise=8, coef=True,
random_state=1)

ls = plt.plot(X, Y, '.', color='orange')
```

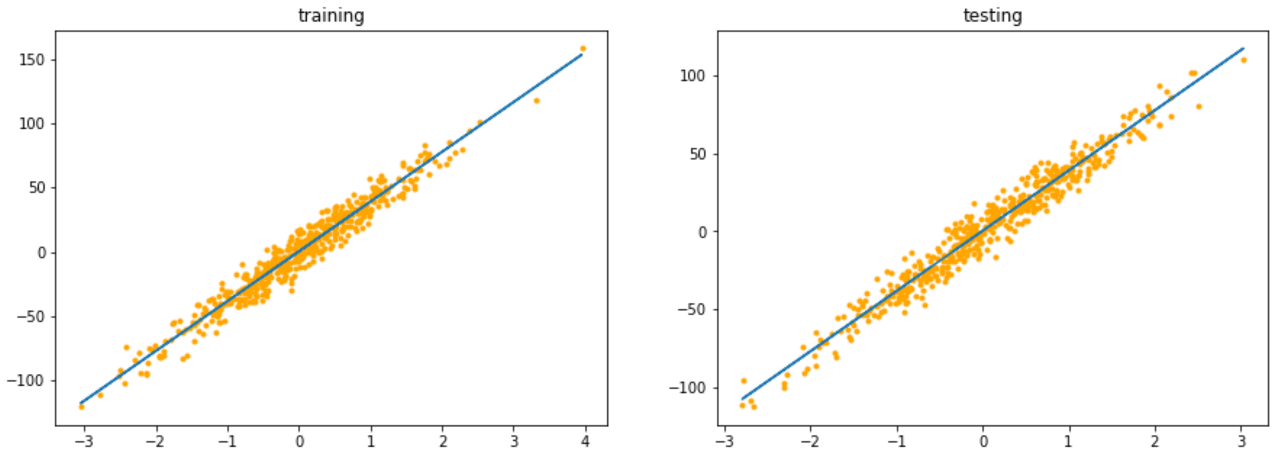


Splitting data randomly into training and testing dataset:

In [18]:
```
from sklearn.model_selection import train_test_split

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.5, random_state=
1)
```

In [19]:

```
# create simple linear regression model and train on training data
model = LinearRegression(fit_intercept=True)
model.fit(Xtrain, Ytrain)
# visualize training and testing data
plt.figure(figsize=(15, 5))
plt.subplot(121)
plt.plot(Xtrain, Ytrain, '.', color='orange')
plt.plot(Xtrain, model.predict(Xtrain))
plt.title('training')
plt.subplot(122)
plt.plot(Xtest, Ytest, '.', color='orange')
plt.plot(Xtest, model.predict(Xtest))
title = plt.title('testing')
```



Cross-validation

In [20]:

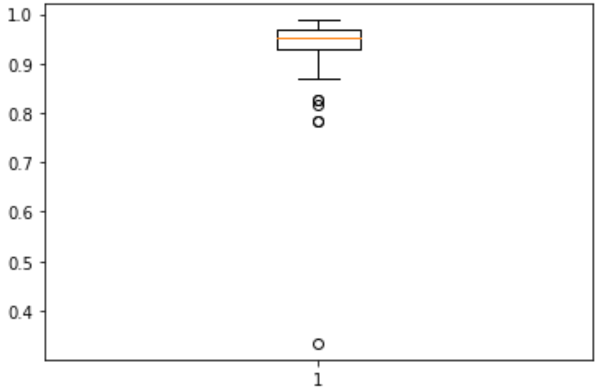
```
from sklearn.model_selection import cross_val_score

model = LinearRegression(fit_intercept=True)
cross_val_score(model, X, Y, cv=5, scoring='r2')
```

Out[20]:

```
array([0.95546461, 0.95096678, 0.96044884, 0.95728812, 0.95819694])
```

In [21]:

```
eval_result = cross_val_score(model, X, Y, cv=100, scoring='r2')
bplt = plt.boxplot(eval_result)
```



Quiz

- Assign the following methods to their categories:
  - Naive Bayes
  - Kmeans
  - PCA
  - Decision Tree
  - Gaussian Mixture Models
  - Isomap
  - Spectral Clustering

- *True* or *false?*
  - Cross validation can only be performed on labeled data
  - Gaussian Mixture Models assumes that data points follow a normal distribution
  - Categorical feature must be transformed priot to machine learning analysis

Quiz

- Assign the following methods to their categories:
  - Naive Bayes — Classification
  - Kmeans — Clustering
  - PCA — Dimensionality red.
  - Decision Tree — Classification
  - Gaussian Mixture Models — Clustering
  - Isomap — Dimensionality red.
  - Spectral Clustering — Clustering

- *True* or *false?*
  - Cross validation can only be performed on labeled data — true
  - Gaussian Mixture Models assumes that data points follow a normal distribution — true
  - Categorical feature must be transformed priot to machine learning analysis — true

Machine Learning

Scikit-Learn

Applications

The Estimator API

Estimators of the Scikit-Learn package share a common API.

Use of estimators:

- Choose model (Estimator)
- Choose model hyperparameters
- Instantiate model with hyperparameters
- Call `fit()` to train the model on a given data set
- Apply model to new data:
  - Supervised learning: call `predict()`
  - Unsupervised learning: call `transform()` or `predict()` (depending on the estimator)

UNIVERSITÄT BIELEFELD

Quiz

- *True* or *false?*
  - The basic steps are *model*, *fit*, *predict/transform*
  - `LinearRegression.coef_` returns slope and intercept of line
  - Scikit-Learn can generate artificial datasets
  - Scikit-Learn doesn't provide real world data sets
  - transformers uses the `predict()` to transform data.
- Explain the function of the following estimators:
  - `OneHotEncoder`
  - `ColumnTransformer`
  - `DictVectorize`
  - `CountVectorizer`

Quiz

- *True* or *false?*
  - The basic steps are *model*, *fit*, *predict/transform* — true
  - `LinearRegression.coef_` returns slope and intercept of line — false
  - Scikit-Learn can generate artificial datasets — true
  - Scikit-Learn doesn't provide real world data sets — false
  - transformers uses the `predict()` to transform data. — false
- Explain the function of the following estimators:
  - `OneHotEncoder` Transforms one categorical feature with *n* possible values into *n* binary features
  - `ColumnTransformer` Transforms all columns of a `DataFrame`
  - `DictVectorize` Transforms `dict` with categorical variables into numeric features
  - `CountVectorizer` Tokenizes strings and constructs word count frequency matrix

Machine Learning

Scikit-Learn

Applications

Applications

The ”digits” data set

Source: [https://jakevdp.github.io/PythonDataScienceHandbook/05.02-introducing-scikit-learn.html (https://jakevdp.github.io/PythonDataScienceHandbook/05.02-introducing-scikit-learn.html)](https://jakevdp.github.io/PythonDataScienceHandbook/05.02-introducing-scikit-learn.html)

We'll use Scikit-Learn's data access interface and take a look at this data:

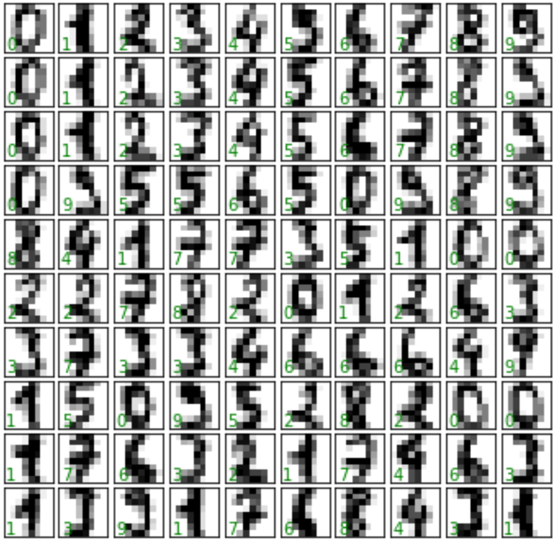
In [22]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.images.shape
```

Out[22]:
```
(1797, 8, 8)
```

The images data is a three-dimensional array: 1,797 samples each consisting of an 8 × 8 grid of pixels. Let's visualize the first hundred of these:

In [23]:

```
fig, axes = plt.subplots(10, 10, figsize=(6, 6),
                         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='binary', interpolation='nearest')
    ax.text(0.05, 0.05, str(digits.target[i]),
            transform=ax.transAxes, color='green')
```



In [24]:
```
X = digits.data
X.shape
```

Out[24]:
```
(1797, 64)
```

In [25]:
```
y = digits.target
y.shape
```

Out[25]:
```
(1797,)
```

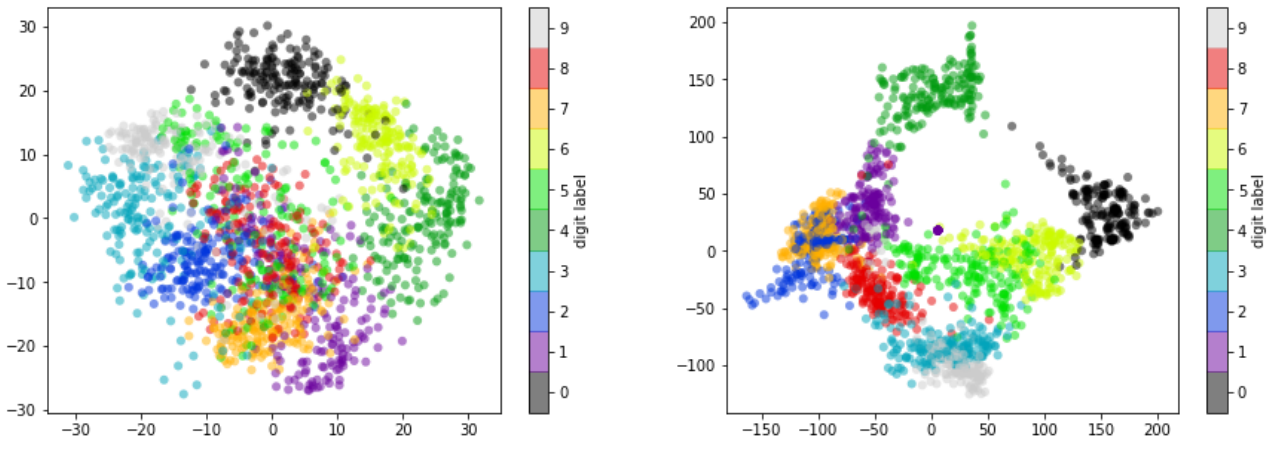
We see here that there are 1,797 samples and 64 features.

Visualizing the parameter space

In [26]:

```
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap

# dimensionality reduction with PCA
pca = PCA(n_components=2)
pca.fit(digits.data)
pca_projected = pca.transform(digits.data)

# dimensionality reduction with Isomap
iso = Isomap(n_components=2)
iso.fit(digits.data)
iso_projected = iso.transform(digits.data)
```

In [27]:
```
plt.figure(figsize=(15, 5))

# plot PCA projection
plt.subplot(121)
plt.scatter(pca_projected[:, 0], pca_projected[:, 1], c=digits.target,
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);

# plot Isomap projection
plt.subplot(122)
plt.scatter(iso_projected[:, 0], iso_projected[:, 1], c=digits.target,
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
plt.colorbar(label='digit label', ticks=range(10))
plt.clim(-0.5, 9.5);
```



Classification

Let's apply a classification algorithm to the digits. We will split the data into a training and testing set, and fit a Gaussian naive Bayes model:

In [28]:
```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=0)
```

In [29]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(Xtrain, ytrain)
y_model = model.predict(Xtest)
```

Now that we have predicted our model, we can gauge its accuracy by comparing the true values of the test set to the predictions:

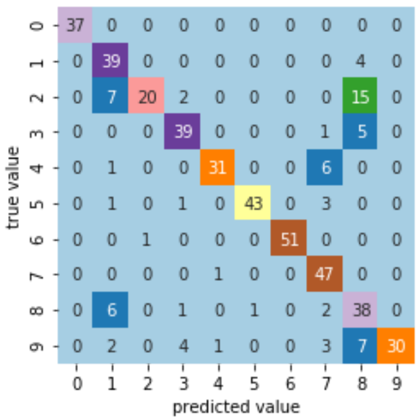
In [30]:
```
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)
```

Out[30]:
```
0.8333333333333334
```

With even this extremely simple model, we find about 83% accuracy for classification of the digits! However, this single number doesn't tell us where we've gone wrong—one nice way to do this is to use the confusion matrix, which we can compute with Scikit-Learn and plot with Seaborn:

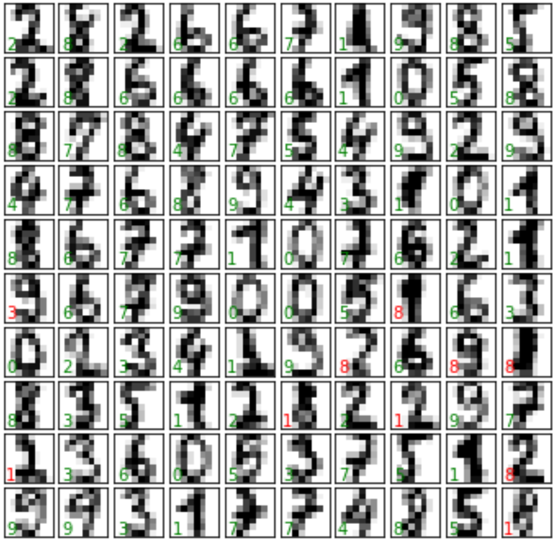
In [31]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

mat = confusion_matrix(ytest, y_model)

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



In [32]:

```python
fig, axes = plt.subplots(10, 10, figsize=(6, 6),
                         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='nearest')
    ax.text(0.05, 0.05, str(y_model[i]),
            transform=ax.transAxes,
            color='green' if (ytest[i] == y_model[i]) else 'red')
```



Text

Data retrieval and extraction

In this example, we will use the Wordnet lemmatizer from NLTK to preprocess text. The `lemmatizeText` function is a slight variation from that discussed at the beginning of Lecture 5.

In [33]:

```
from nltk.stem.wordnet import WordNetLemmatizer
import nltk

def lemmatizeText(text):
    uni2wn = {'ADJ': nltk.corpus.wordnet.ADJ, 'NOUN': nltk.corpus.wordnet.NOUN,
              'VERB': nltk.corpus.wordnet.VERB, 'ADV' : nltk.corpus.wordnet.ADV }
    stop_words = set(nltk.corpus.stopwords.words("english"))
    # instantiate word-net lemmatizer
    wnl = WordNetLemmatizer()
    # initialize result list of lemmatized words
    lemmatized_words = list()
    for s in nltk.tokenize.sent_tokenize(text):
        words = nltk.tokenize.word_tokenize(s)
        tagged_text = nltk.pos_tag(words, tagset='universal')
        for w, p in tagged_text:
            if p in uni2wn:
                lem_w = wnl.lemmatize(w.lower(), uni2wn[p])
                if lem_w.isalnum() and lem_w not in stop_words:
                    lemmatized_words.append(lem_w)
    return lemmatized_words
```

We will compare exerpts of corpora from the Gutenberg library that have been made available through NLTK's corpus database. The objective of this toy application is to answer the simple question: "How similar are the texts to each other?"

In [34]:
```
from os.path import basename

# paths and names of the Gutenberg corpora
gutenberg_corpora_paths = nltk.corpus.gutenberg.abspaths()
corpora_names = list(map(lambda x: basename(x).split('.')[0], gutenberg_corpora_pa
ths))

raw_texts = list()
# read the first 10000 characters of each Gutenberg corpus
for p in gutenberg_corpora_paths:
    with open(p, encoding='latin2') as f:
        raw_texts.append(f.read(10000))
```

We will use the `CountVectorizer` to extract a feature matrix from the text, but specify the lemmatizeText function as `analyzer`. The `analyzer` takes care of tokenization and the extraction of terms (features) of each text.

In [35]:
```
from sklearn.feature_extraction.text import CountVectorizer

# create "model"
vec = CountVectorizer(analyzer=lemmatizeText)

# apply fit/transform to raw_texts
trans_text = vec.fit_transform(raw_texts)

# transform outcome into a DataFrame for better display
pd.DataFrame(trans_text.toarray(), columns=vec.get_feature_names()).head()
```

Out[35]:

	ability	able	abolishes	abroad	absence	absurd	abundance	abundantly	abyss	accept	...	ynch	yoake	yon	yond	yonder	y
0	0	2	0	0	1	0	0	0	0	1	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	2	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

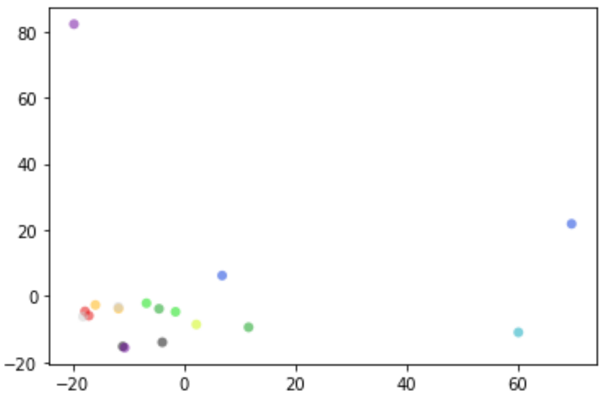
5 rows × 4358 columns

The outcome is a feature matrix with 4358 features, corresponding to 4358 dimensions of the text's parameter space.

Data Visualization

We will now reduce these to two dimensions. This allows us to visualize the data and obtain an understanding on similarity of the texts to each other: close vicinity of points indicates similarity.

In [36]:
```
# dimensionality reduction with PCA
pca = PCA(n_components=2)
pca_projected = pca.fit_transform(trans_text.toarray())

plt.figure(figsize=(6, 4))
pc = plt.scatter(pca_projected[:, 0], pca_projected[:, 1],
            c=list(range(pca_projected.shape[0])),
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
```



Text normalization with TF-IDF

We will use the *term frequency-inverse document frequency* (tf-idf) normalization to remove the frequent words bias.

In [37]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# create "model"
vec = TfidfVectorizer(analyzer=lemmatizeText)
# apply fit/transform to raw_texts
trans_text = vec.fit_transform(raw_texts)

# transform outcome into a DataFrame for better display
pd.DataFrame(trans_text.toarray(), columns=vec.get_feature_names()).head()
```

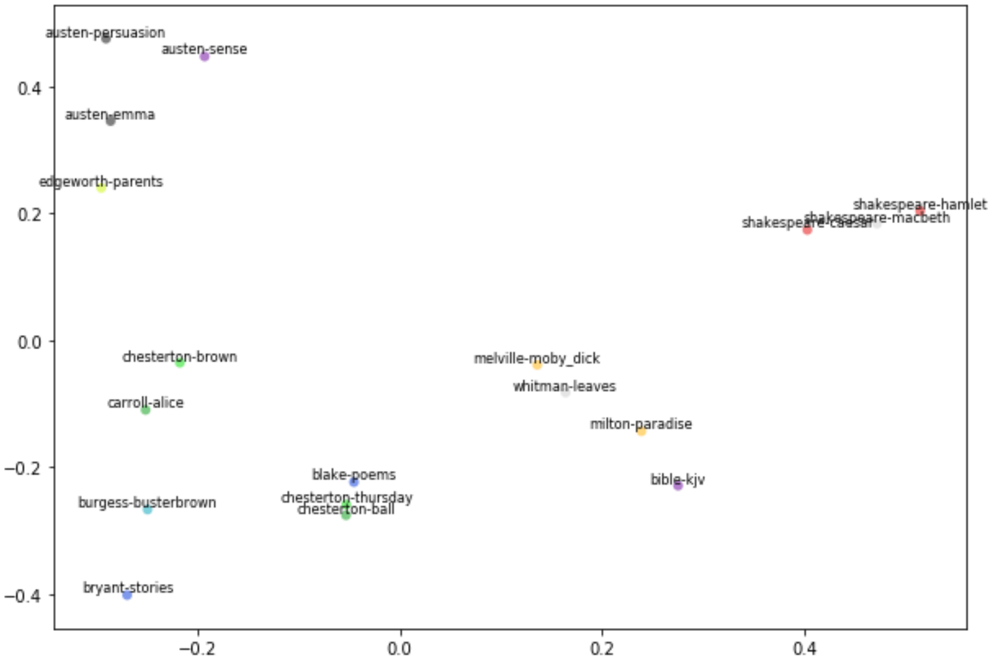
Out[37]:

	ability	able	abolishes	abroad	absence	absurd	abundance	abundantly	abyss	accept	...	ynch	yoake	yon	yond	yo
0	0.000	0.051529	0.0	0.0	0.02316	0.0	0.0	0.00000	0.0	0.025765	...	0.0	0.0	0.0	0.0	0.0
1	0.000	0.000000	0.0	0.0	0.00000	0.0	0.0	0.00000	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.0
2	0.028	0.000000	0.0	0.0	0.00000	0.0	0.0	0.00000	0.0	0.024508	...	0.0	0.0	0.0	0.0	0.0
3	0.000	0.000000	0.0	0.0	0.00000	0.0	0.0	0.03842	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.0
4	0.000	0.000000	0.0	0.0	0.00000	0.0	0.0	0.00000	0.0	0.000000	...	0.0	0.0	0.0	0.0	0.0

5 rows × 4358 columns

In [38]:

```
pca = PCA(n_components=2)
pca_projected = pca.fit_transform(trans_text.toarray())
plt.figure(figsize=(10, 7))
plt.scatter(pca_projected[:, 0], pca_projected[:, 1],
            c=list(range(pca_projected.shape[0])),
            edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))

for i, label in enumerate(corpora_names):
    plt.text(pca_projected[i, 0], pca_projected[i, 1], label, fontsize=8,
             horizontalalignment='center', verticalalignment='bottom')
```



Quiz

- In which order does function `train_test_split` return test/train data?
  - `Xtrain, Ytrain, Xtest, Ytrain`
  - `Xtest, Ytest, Xtrain, Ytrain`
  - `Xtrain, Xtest, Ytrain, Ytest`
  - `Xtest, Xtrain, Ytest, Ytrain`
- What data is stored in
  - `digits.images`
  - `digits.data`
  - `digits.target`

Quiz

- In which order does function `train_test_split` return test/train data?
  - `Xtrain, Ytrain, Xtest, Ytrain`
  - `Xtest, Ytest, Xtrain, Ytrain`
  - `Xtrain, Xtest, Ytrain, Ytest` ✔
  - `Xtest, Xtrain, Ytest, Ytrain`
- What data is stored in
  - `digits.images` bitmap data of all images
  - `digits.data` feature matrix
  - `digits.target` labels (ground truth digits)

Recap

Summary

- Machine Learning
  - Dimensionality reduction
  - Clustering
  - Classification
  - Regression
- Scikit-Learn
  - Estimator API
  - Feature representation
  - Crossvalidation
- Applications
  - Handwritten digits dataset
  - Text comparison

What comes next?

- Have a look at the Jupyter Notebook of this lecture
- Further reading about Pandas: Chapter 5 of the "Python Data Science Handbook":
  `https://jakevdp.github.io/PythonDataScienceHandbook/`
- Have a look at the in-depth analyses that are provided in the handbook

Next lecture: Distributed computing and data bases