DATA PROCESSING & MACHINE LEARNING WITH PYTHON

AHMED KACHKACH @KTH - 2015

Who am I?

- Ahmed Kachkach < kachkach.com >
    - Machine Learning master student @KTH.
    - Interested in all things data, Python, web dev.
    - Github & Twitter: **@halflings**
    - Email : ahmed.kachkach@gmail.com

Subject of this talk

- Manipulating data is a long process that requires different steps & tools
- Might be able to focus on one thing in a big company…
- …not an option for startups and studies/personal projects!
- This is a “quick” overview of all steps of this process

Outline

- Setup / What's special about Python / Misc.
- Fetching and cleanning (`requests, lxml, pandas`):
  - Getting data (HTTP, filesystem)
  - Scrapping/parsing data (XML, JSON)
  - Validating/Exploring the data
- Analyzing the data (`scikit-learn`)
  - Pre-processing
  - Training/Predicting with models
  - Cross-validation/evaluation
- Visualization (`matplotlib`)

PYTHON

INSTALLING PYTHON,
...AND WHY YOU
SHOULD DO IT IN THE
FIRST PLACE

Setup

- Recommended: install Canopy, a Python distribution that comes with all the libraries we're going to use (+ more!)
- Or: Download and install Python 2.7 (https://www.python.org/downloads/), probably already installed if you're on Linux or Mac OS
- Python 2.7 vs 3.x :
  2.7 is the most popular version: better compatibility, no strong reason (yet) to move to 3.x

Dependencies

- Libraries we'll need for this workshop:
  - `requests`: HTTP requests
  - `pandas`: reading/saving, organising, validating and querying datasets
  - `lxml`: parsing XML and HTML
  - `scikit-learn`: machine learning methods and utilities
  - `matplotlib`: visualizing data
- You can install them with these commands:

```
easy_install pip
pip install requests lxml scikit-learn pandas matplotlib
```

Why Python?

- Dynamic language (goods and bads)
- Strong principles, like simplicity and explicitness
- Active/open development (unlike C++, PHP, …)
- Incredibly active community (imagine something: there's a Python library for that)

Some useful features

- List comprehensions:

```
[line.rstrip().lower() for line in file if not line.startswith("#")]
```

- Useful operators:

```
map(str.upper, ["hey", "what's up?"]) # ["HEY", "WHAT'S UP?"]
any(word.startswith("s") for word in {"mloukhiya", "saykouk"}) # True
sorted(countries, key=lambda country : country.capital.size)
```

- Closures/1st class functions/Decorators

```
@functools.lru_cache(maxsize=None)
def fibonacci(num):
    …
```

FETCHING CLEANING VALIDATING DATA

IT ALL STARTS BY GETTING THE RAW DATA!

Raw data

- Data comes in all shapes and colors (or formats and encodings), often hidden quite deep.
- The first step is to extract the relevant data.

The “Hello World” of HTTP requests

```
import requests

print requests.get(“http://example.com”).text

   “<!doctype html>
   <html>
   <head>
       <title>Example Domain</title>
       . . .“
```

Communicating with APIs

```
import requests

print requests.get("https://www.googleapis.com/books/v1/volumes",
                   params={"q":"machine learning").json()['items']
```


```
 [
  {"volumeInfo": { "title": "KTH", "subtitle": "i.e. Kungliga
  Tekniska högskolan 1912-62 i.e. nittonhundra tolv till
  sextiotvå . Kungl. Tekniska Högskolan i Stockholm under 50 år", .
  . . },
  . . .
 ]
```

Parsing an HTML page

```
import lxml.html

page = lxml.html.parse("http://www.blocket.se/stockholm?q=apple")
# ^ This is probably illegal. Please don’t sue me, Blocket!

items_data = []
for el in page.getroot().find_class("item_row"):
    links = el.find_class("item_link")
    images = el.find_class("item_image")
    if links and images:
        items_data.append({"name": links[0].text,
                           "image": images[0].attrib[‘src']})
print items_data
```

More advanced HTML/XML parsing

```
import lxml.html

page = lxml.html.parse(“http://www.blocket.se/stockholm?q=apple”)

# number of links in the page
print len(page.xpath(‘//a'))
# products' images
print page.xpath(‘//img[contains(@class, “item_image”)]/@src’)
```

Reading local data

```
import pandas

df = pandas.read_csv('sample.csv')

# Display the DataFrame
print df

# DataFrame's columns
print df.columns

# Values of a given column
print df['Model']
```

Processing/Validating data with Pandas

```
df = pandas.read_csv('sample.csv')

# Any missing values?
print df['Price']
print df['Description']

# Fill missing prices by a linear interpolation
df['Description'] = df['Description'].fillna("No
description is available.")
df['Price'] = df['Price'].interpolate()

print df
```

Exploring/Visualizing data

```
df = pandas.read_csv('sample2.csv')
# This table has 3 columns: Office, Year, Sales
print df.columns
# It's really easy to query data with Pandas:
print df[(df['Office'] == 'Stockholm') & (df['Sales'] > 260)]
# It's also easy to do aggregations...
aggregated_sales = df.groupby('Year').sum()
print aggregated_sales
# ... and generate plots
aggregated_sales.plot(kind='bar')

plt.show()
```

MACHINE LEARNING: ANALYZING THE DATA

WE HAVE THE DATA, NOW LET'S MAKE SOMETHING OF IT!

PART 1: PRE-PROCESSING

Pre-processing data

Pre-processing is often a vital step to change our data into a representation usable by our ML models.

Among the most common steps:

- Feature extraction & Vectorization
- Scaling/Normalization
- Feature selection/Dimensionality reduction

Feature extraction

Raw data comes in multiple shapes:

- Image
- Text
- Structured data (database table, dictionary, etc.)

We need to extract relevant features from this data.

Example: text documents

Converting text documents to a vector representation using TF-IDF:

```
from sklearn import feature_extraction

corpus = ['Cats really are great.',
          'I like cats but I still prefer dogs.',
          'Dogs are the best.',
          'I like trains']

tfidf = feature_extraction.text.TfidfVectorizer()

print tfidf.fit_transform(corpus)
print tfidf.get_feature_names()
```

Vectorization

Your features may be in various forms:

- Numerical variables (ex: weight)
- Categorical variables (ex: country of origin)
- Boolean variables (ex: active account)

We have to represent all these variables in the vector space model to train our models.

Example: DictVectorizer

Transforming key->value pairs to vectors:

```
from sklearn import feature_extraction

data = [{"weight": 60., "sex": 'female', "student": True},
        {"weight": 80.1, "sex": 'male', "student": False},
        {"weight": 65.3, "sex": 'male', "student": True},
        {"weight": 58.5, "sex": 'female', "student": False}]

vectorizer = feature_extraction.DictVectorizer(sparse=False)

vectors = vectorizer.fit_transform(data)
print vectors
print vectorizer.get_feature_names()
```

Normalization

Many models are sensitive to the scale of the input data, so it's often a good idea to normalize the data we feed it:

(each sample is normalized independently)

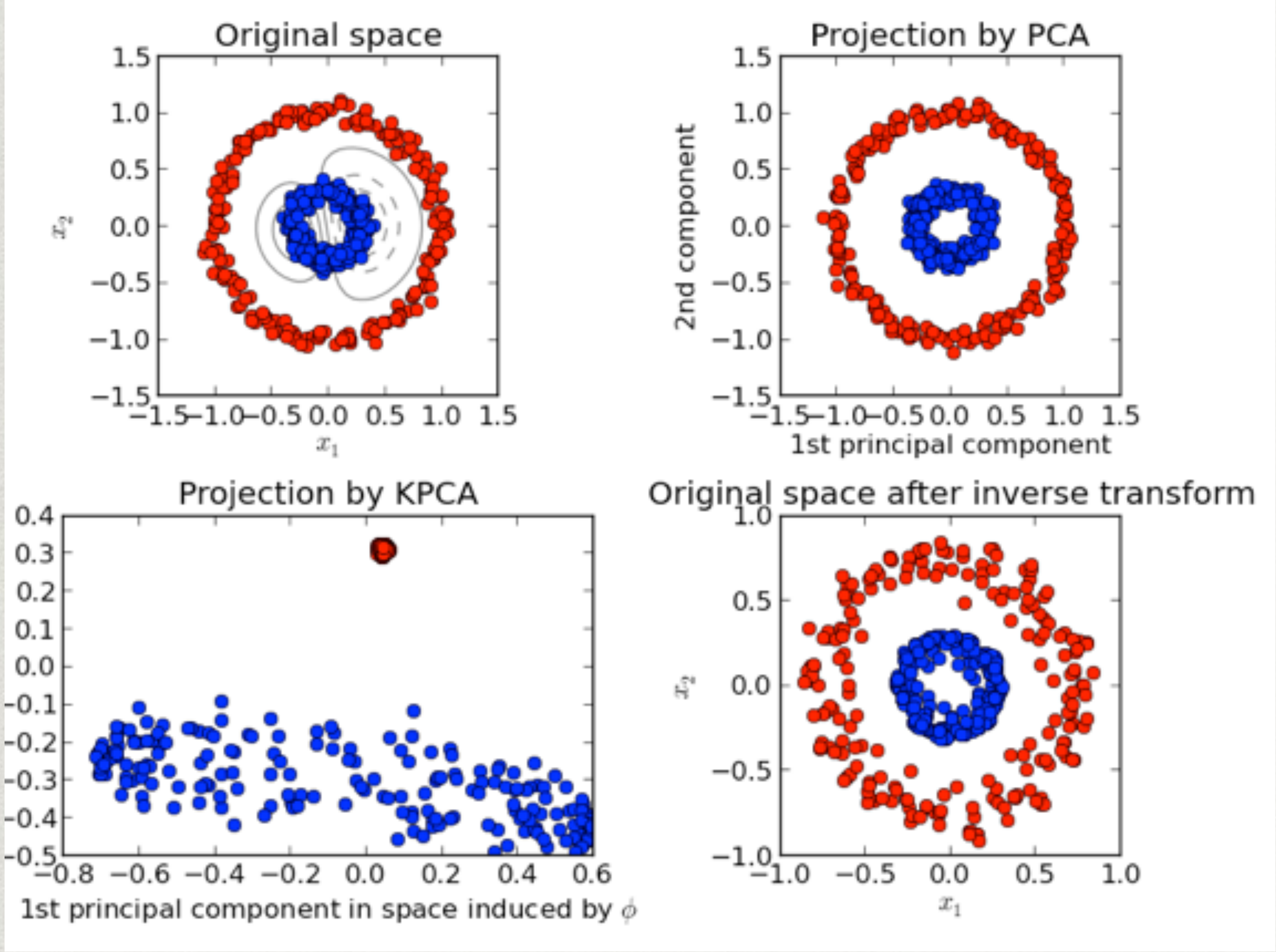
```
from sklearn import preprocessing
data = [[10., 2345., 0., 2.],
        [3., -3490., 0.1, 1.99],
        [13., 3903., -0.2, 2.11]]
print preprocessing.normalize(data)
```

Dimensionality reduction

Many features can be invariant or heavily correlated with other features. A dimensionality reduction algorithm (like PCA) can help us get better performance and faster training/predictions.

Kernel PCA

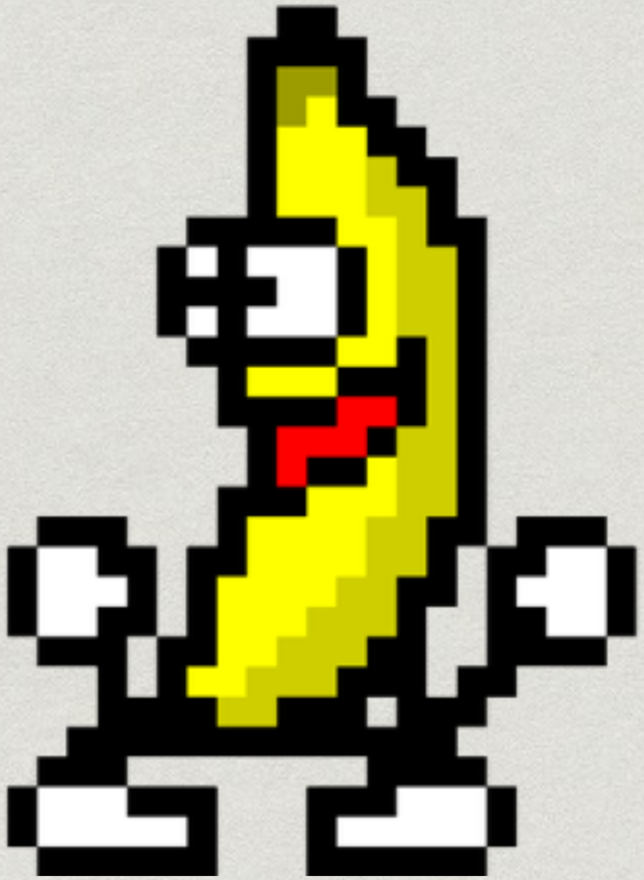
Kernel PCA can be used for non-linear decompositions



PART 2: TRAINING & USING MODELS

Let the fun begin!

We finally have some clean, relevant and structured data. Let's train some models!



Classification

Given labeled feature vectors, we can identify which class a new datapoint should belong to using some of these methods:

- Naive Bayes
- Decision trees / Random forest
- SVM
- KNN

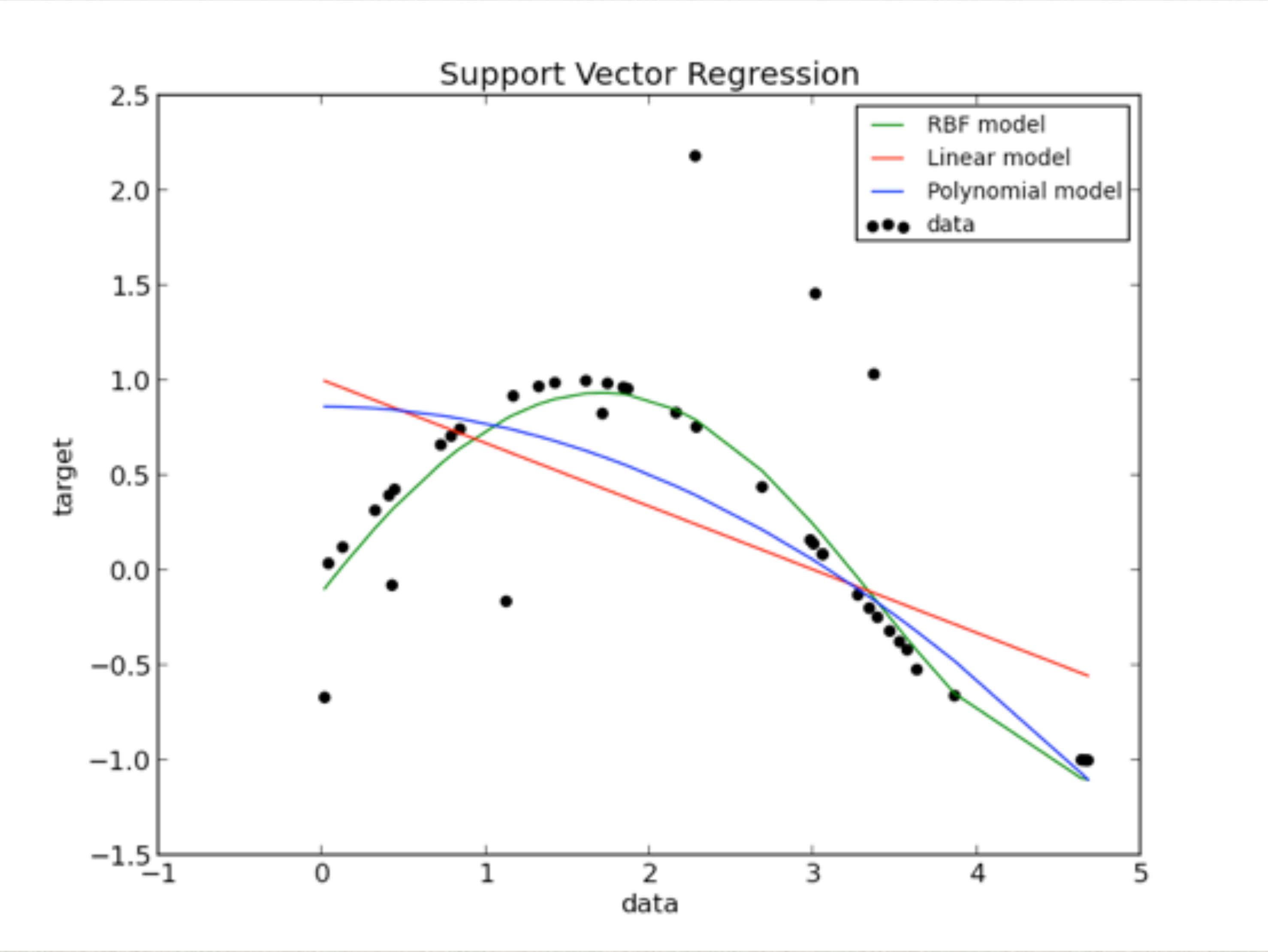
… and many others!

Example: SVM

```
from sklearn import datasets
from sklearn import svm

iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

# Training the model
clf = svm.SVC(kernel='rbf')
clf.fit(X, y)

# Doing predictions
new_data = [[4.85, 3.1], [5.61, 3.02], [6.63,  3.13]]
print clf.predict(new_data)
```

Regression

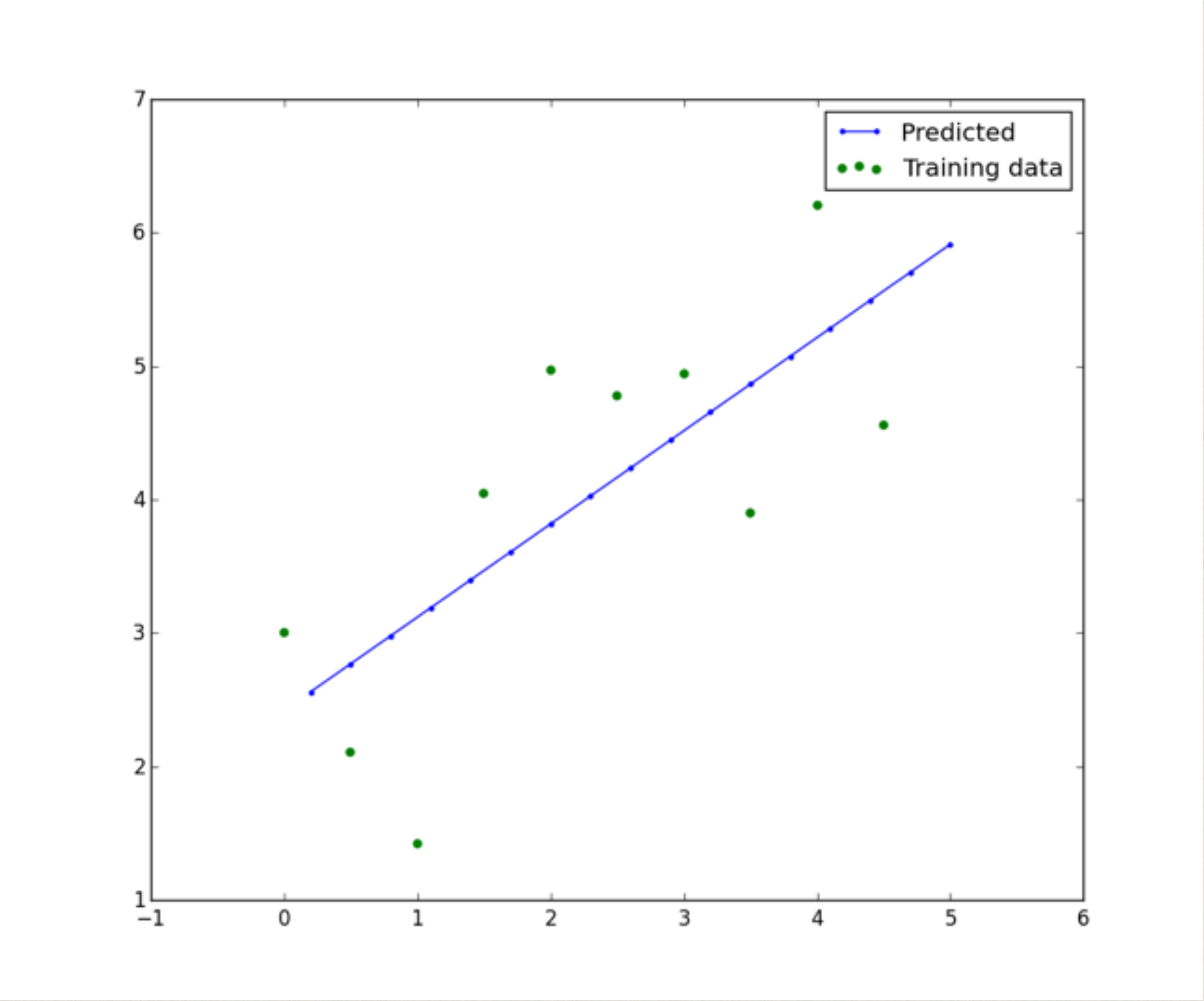
Given a series of inputs and their target output, we want to be able to predict the output of a new series of inputs.



Example: LinearClassifier

```
import numpy as np
from sklearn import linear_model

def f(x):
    return x + np.random.random() * 3.

X = np.arange(0, 5, 0.5)
X = X.reshape((len(X), 1))
y = map(f, X)

clf = linear_model.LinearRegression()
clf.fit(X, y)

new_X = np.arange(0.2, 5.2, 0.3)
new_X = new_X.reshape((len(new_X), 1))
new_y = clf.predict(new_X)
```

Example: LinearClassifier

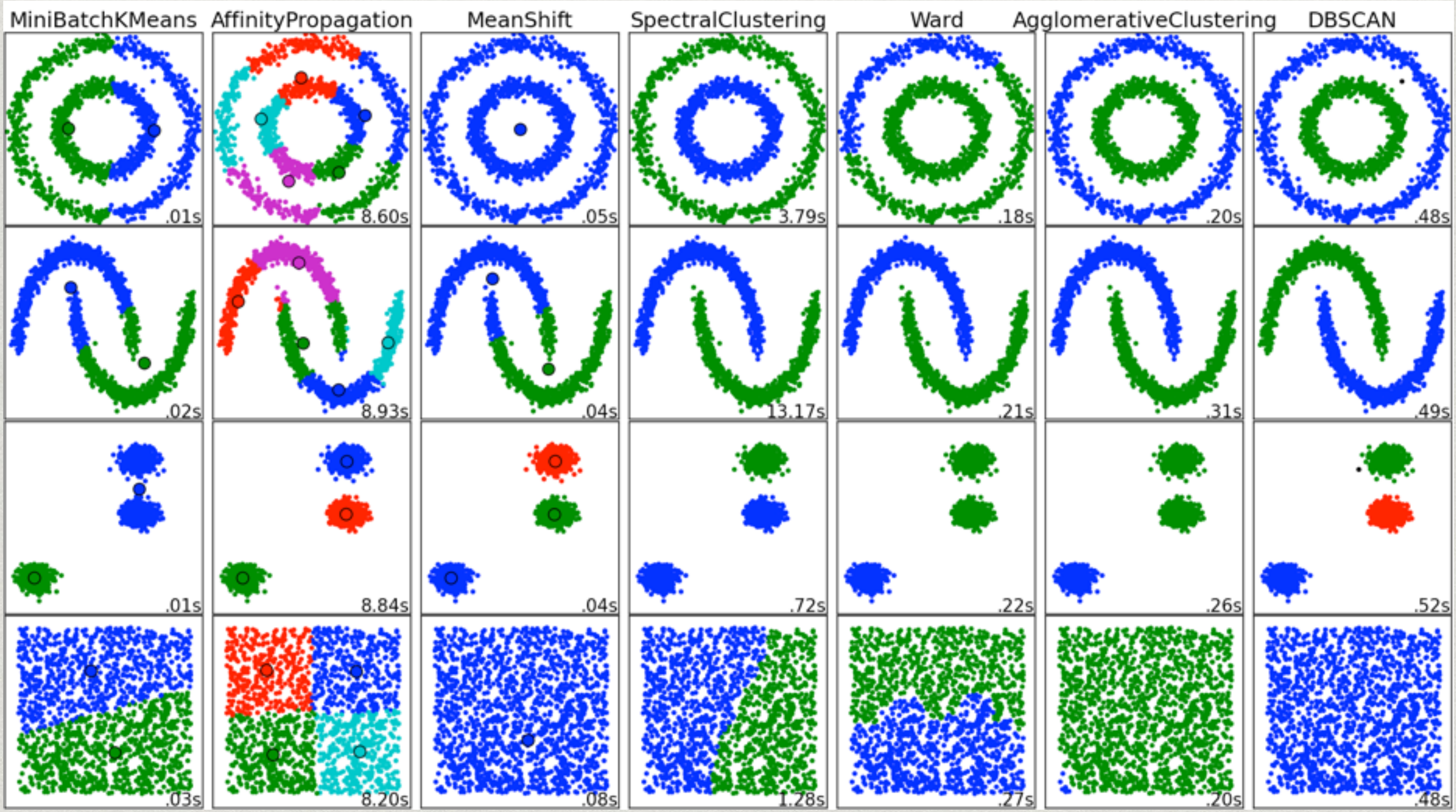


Clustering

Grouping similar data-points together.

Can be either with a known number of clusters (KMeans, Hierarchical clustering, …) or an unknown number of clusters (Mean-shift, DBScan, …).

Comparison of clustering techniques



Example: DBSCAN

```
from sklearn.cluster import DBSCAN
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Generate sample data
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=200, centers=centers, cluster_std=0.4,
                            random_state=0)
X = StandardScaler().fit_transform(X)

# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
labels = db.labels_

plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.show()
print labels
```

MINI-PART 3: MODEL EVALUATION

“No free hunch”

Looking at your program’s output and saying **“Mmh that looks about right”** isn’t a sane way to evaluate your models.

scikit-learn makes it extremely easy to do systematic model evaluation.

Integrated model evaluation

- Most scikit-learn classifiers have a **`score`** function that takes a list of inputs and the target outputs.
- Scoring functions let you calculate some of these values:
  - accuracy
  - precision/recall
  - mean absolute error / mean squared error

Cross-validation

```
from sklearn import svm, cross_validation, datasets

iris = datasets.load_iris()
X, y = iris.data, iris.target

model = svm.SVC()
print cross_validation.cross_val_score(model, X, y, scoring='precision')
print cross_validation.cross_val_score(model, X, y, 
scoring='mean_squared_error')
```

VISUALIZE AND EXPLORE DATA

SCATTER PLOTS, GRAPHS, HEAT-MAPS AND OTHER FANCY THINGS.

Matplotlib

The “go-to” plotting library in Python.

Integrated with most scientific/data libraries (pandas, scikit-learn, etc.)

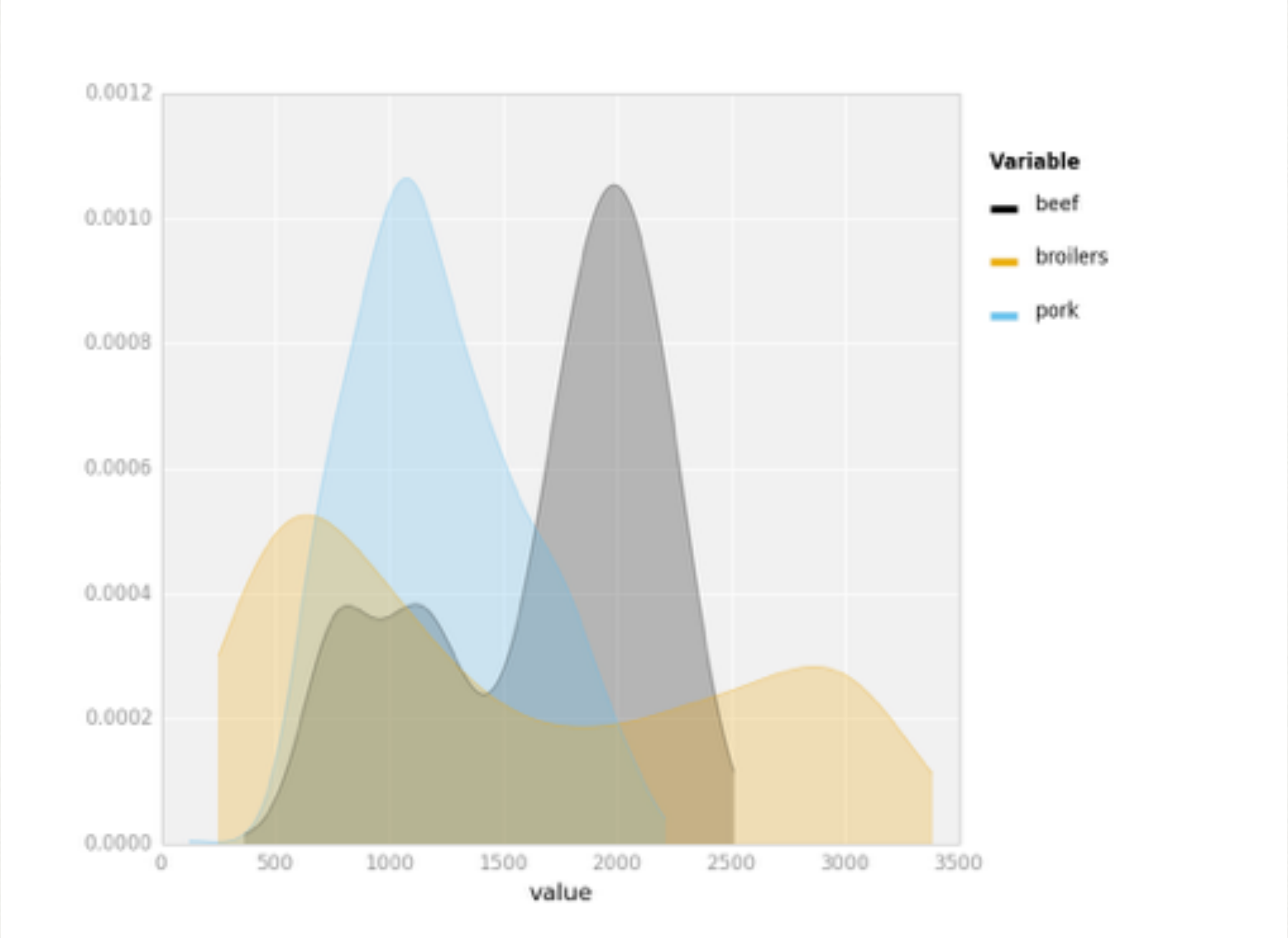
Easy to use, can be used to create various plots and offers a high level of customizability, but graphs are pretty “ugly” by default and are hard to integrate for web use.

Bokeh

Simple API, more diverse plots, allows plotting interactive graphs that can be shared on the web (using **D3.js**)

Example: http://bokeh.pydata.org/en/latest/docs/gallery/texas.html

ggplot



Fancier plots, similar to R's ggplot2.

THAT'S ALL FOLKS!

THANKS FOR YOUR ATTENTION.

AHMED KACHKACH < KACHKACH.COM >
@ HALFLINGS