Naive Bayes and Gaussian Bayesian Inference

Thomas Schwarz

Conditional Probability

- Given two events A and B, we define the conditional probability as

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

"probability of A given B"

- Write also as:

$$P(A \cap B) = P(A \mid B)P(B)$$

Conditional Probability

- Bayes' Theorem: An observation of extreme importance
  - Giving rise to a new way of statistics

  Theorem: $$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

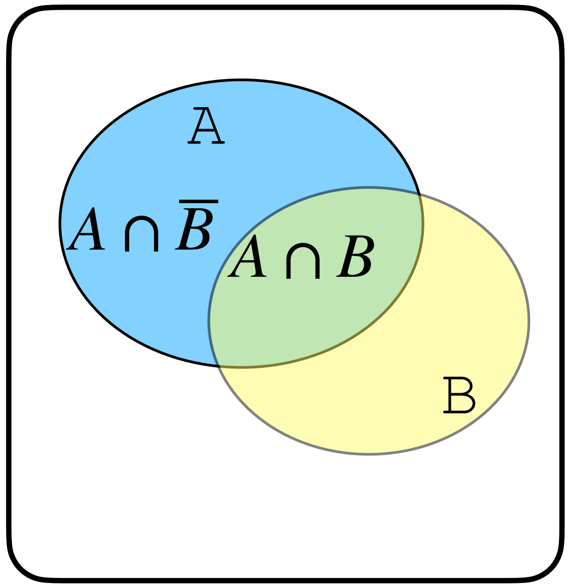
- Expresses a probability conditioned on B in one conditioned on A
- Proof:
  $$P(A|B)P(B) = P(A \cap B) = P(B \cap A) = P(B|A)P(A)$$
- Now solve for $P(A|B)$

Conditional Probability

- We can express a probability for one event in terms of another event happening or not

$$P(A) = P(A \cap B) + P(A \cap \overline{B})$$

$$= P(A \mid B)P(B) + P(A \mid \overline{B})P(\overline{B})$$



Conditional Probability

- We can expand Bayes by calculating $P(B)$ as probabilities conditioned on A

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

$$= \frac{P(B \mid A) \cdot P(A)}{P(B \cap A) + P(B \cap \bar{A})}$$

$$= \frac{P(B \mid A) \cdot P(A)}{P(B \mid A)P(A) + P(B \mid \bar{A})P(\bar{A})}$$

Conditional Probability

- Example: Medical Tests
  - An HIV test is positive. What is the probability that you have HIV?
  - Need some data: The quality of the test
    - Type 1 error: Test is negative, but there is illness
    - Type 2 error: Test is positive, but there is no illness

Conditional Probability

- Abbreviate probabilities
  - T : Test is positive
  - H : Person infected with HIV
  - Interested in $P(H|T)$. The quality of the test is expressed in terms of the opposite conditional probability.
    - Type I error probability: $P(\overline{T}|H)$
    - Type II error probability: $P(T|\overline{H})$

Conditional Probability

- We calculate

$$P(H|T) = \frac{P(T|H)P(H)}{P(T|H)P(H) + P(T|\overline{H})P(\overline{H})}$$

- Assume test has 5% type I (false positive) error probability and 1% type II (false negative) error probability:

$$P(T|\overline{H}) = 0.95$$

$$P(\overline{T}|H) = 0.99$$

- The probability still depends on the prevalence of HIV in the population

Conditional Probability

$$P(H|T) = \frac{0.99P(H)}{0.99P(H) + 0.95(1 - P(H))}$$

- Example: HIV rate in general population in the US is 13.3/100000 = 0.000,133
- After a positive test:
  - 0.000138599 (Almost no change!)
- Example 2: HIV in a high risk group in the US is 1,753.1/100000 = 0.017531
- After a positive test:
  - 0.0182557

Conditional Probability

- With these type I and type II error rates
  - the test is almost unusable at low incidence rates

Classification with Bayes

- Bayes' theorem inverts conditional probabilities
- Can use this for classification based on observations
- Idea: Assume we have observations $\overrightarrow{x}$
    - We have calculated the probabilities of seeing these observations given a certain classification
    - I.e.: for each category, we know $P(\overrightarrow{x}, c_i)$
        - Probability to observe $\overrightarrow{x}$ assuming that point lies in c_i
    - We use Bayes formula in order to calculate $P(c_i, \overrightarrow{x})$
    - And then select the category with highest probability

Classification with Bayes

- Document classification:
  - Spam detection:
    - Is email spam or ham?
  - Sentiment analysis:
    - Is a review good or bad

Classification with Bayes

- Bag of words method:
  - Model a document by only counting words
    - Restrict ourselves to non-structure = non-common words

"I love this movie! It's sweet, but with satirical humor. The dialogs are great and the adventure scenes are fun. It manages to be romantic and whimsical while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I have seen it several times and I'm always happy to see it again"

fun	1
great	2
happy	1
humor	1
love	1
recommend	1
satirical	1
sweet	1

Classification with Bayes

- There is a whole theory about recognizing key-words automatically
  - Easy out:
    - Use all words that are not common

Classification with Bayes

- Recognizing words
  - Actual documents have misspelling and grammatical forms
    - Grammatical forms less common in English but typical in other languages
      - Lemmatization: Recognize the form of the word
        - `जाओ, जाओगे, …  —> जाना`
        - `went, goes —> to go`
        - Usually difficult to automatize

Classification with Bayes

- Recognizing words
    - Stemming
        - Several methods to automatically extract the stem
            - English: Porter stemmer (1980)
            - Other languages: Can use similar ideas
            - https://www.emerald.com/insight/content/doi/10.1108/00330330610681295/full/pdf?title=the-porter-stemming-algorithm-then-and-now

Classification with Bayes

- Need to calculate the probability to observe a set of keywords given a classification
  - This is too specific:
    - There are too many sets of keywords
- First reduction:
  - Only use existence of words.

Classification with Bayes

- Want: $P(w_1, w_2, w_3, \ldots, w_n | c_i)$
  - The probability to find a certain word in documents of a certain category depends on the existence of other words.
    - E.g.: "Malicious Compliance"
  - We make now a big assumptions:
    - The probabilities of a keyword showing up are independent of each other
    - That's why this method is called "***<u>Naïve</u>* Bayes**"

Classification with Naïve Bayes

- Want:

$$P(w_1, w_2, w_3, \ldots, w_n | c_i) = P(w_1 | c_i) \times P(w_2 | c_i) \times P(w_3 | c_i) \times \ldots P(w_n | c_i)$$

- Can estimate this from a training set:
  - E.g. a set of movie reviews classified with the sentiment
  - Algorithm:

```
for document in set:
    sentiment = document.sentiment
    for word in document:
        count[word]+=1
        if sentiment=='positive':
             countPos[word]+=1
        else:
             countNeg[word]+=1
    return countPos/count, countNeg/count
```

Classification with Naïve Bayes

- This algorithm has a problem:
  - It can return a probability as zero
    - Because we use multiplication in our estimator:

$$P(w_1, w_2, w_3, \ldots, w_n \mid c_i) = P(w_1 \mid c_i) \times P(w_2 \mid c_i) \times P(w_3 \mid c_i) \times \ldots P(w_n \mid c_i)$$

    - Would create zero probabilities
  - Solution: start all counts at 1
    - No more zero probabilities

Classification with Naïve Bayes

- Result: Simple classifier
  - Example: Two categories I and II
  - Use the data set to determine $P(\text{feature} \mid C_I)$ and $P(\text{feature} \mid C_{II})$
  - Calculate
    - $P(C_I \mid \text{feature}) = \dfrac{P(\text{feature} \mid C_I)P(C_I)}{P(\text{feature} \mid C_I)P(C_I) + P(\text{feature} \mid \overline{C_I})P(\overline{C_I})}$
    - $P(C_{II} \mid \text{feature}) = \dfrac{P(\text{feature} \mid C_{II})P(C_{II})}{P(\text{feature} \mid C_{II})P(C_{II}) + P(\text{feature} \mid \overline{C_{II}})P(\overline{C_{II}})}$
  - Select the larger one as the classification
    - As the denominators are the same, just compare $P(\text{feature} \mid C_I)P(C_I)$ and $P(\text{feature} \mid C_{II})P(C_{II})$

Classification with Naïve Bayes

- Example: Use NLTK, a natural language processor
    - NLTK has several corpus (which you might have to download separately)

```
import nltk
from nltk.corpus import movie_reviews
import random
```

Classification with Naïve Bayes

- First step: Get the documents

```
documents = [(list(movie_reviews.words(fileid)), category)
              for category in movie_reviews.categories()
              for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)
train_set, test_set = featuresets[500:], featuresets[:500]
```

Classification with Naïve Bayes

- Second step: Get all "features" (important words)
- Strategy: Get a list of all words, then order it, then select the frequent ones with exception of the most frequent ones.

```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = list(all_words)[200:2000]
```

- Here is all_words:
    - `FreqDist({',': 77717, 'the': 76529, '.': 65876, 'a': 38106, 'and': 35576, 'of': 34123, 'to': 31937, "'": 30585, 'is': 25195, 'in': 21822, …})`
- Therefore, just drop the first ones.

Classification with Naïve Bayes

- Create a bag of words for each document

```
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in 
document_words)
    return features

featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[500:], featuresets[:500]
```

Classification with Naïve Bayes

- Use NLTK Naive Bayes Classifier

```
classifier = nltk.NaiveBayesClassifier.train(train_set)

print(nltk.classify.accuracy(classifier, test_set))
```

Classification with Naïve Bayes

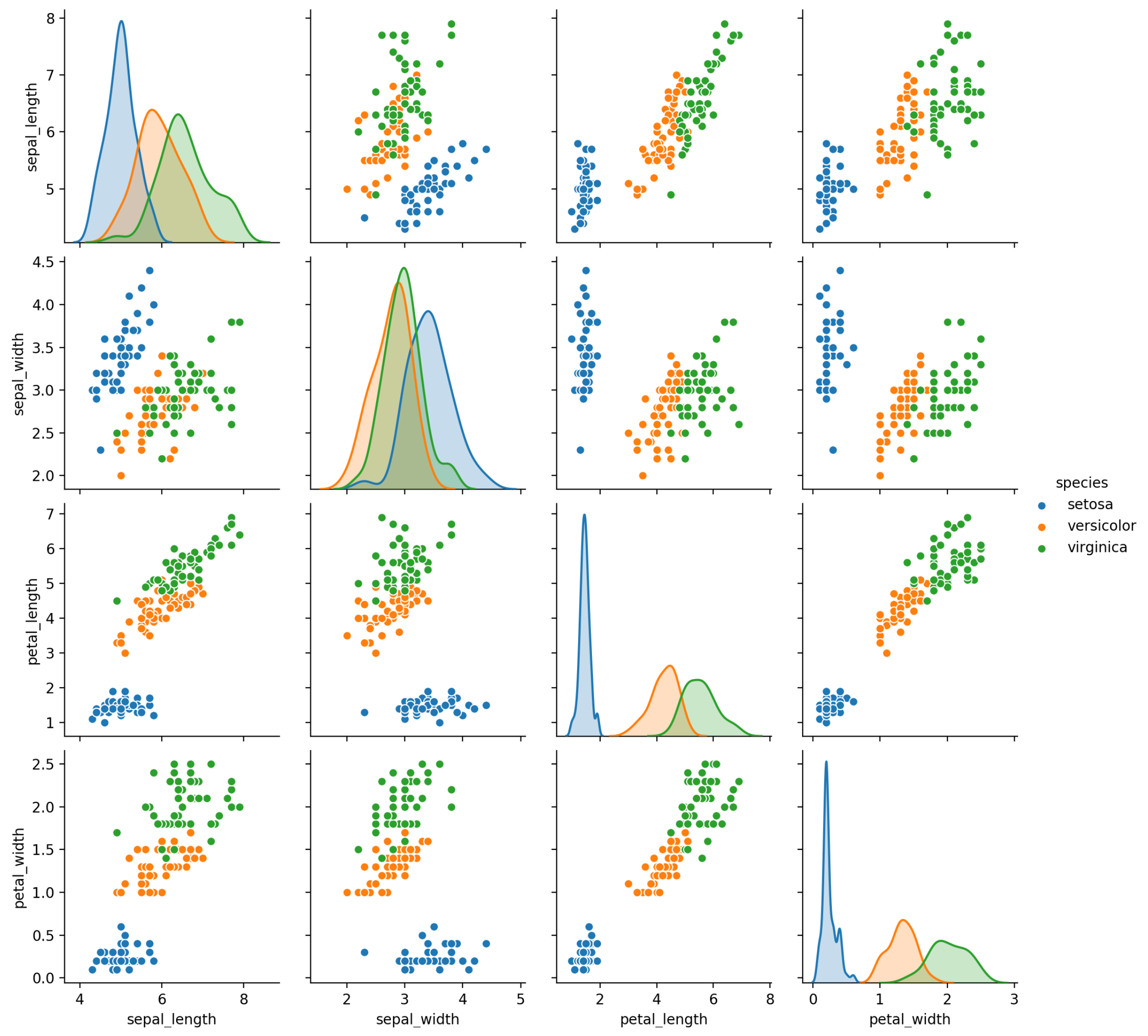
- Results: 80.2% sentiments classified correctly
- Can see how the classifier works

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
        contains(segal) = True              neg : pos    =     11.3 : 1.0
   contains(outstanding) = True             pos : neg    =      8.6 : 1.0
        contains(wasted) = True             neg : pos    =      7.3 : 1.0
         contains(mulan) = True             pos : neg    =      7.2 : 1.0
   contains(wonderfully) = True             pos : neg    =      6.3 : 1.0
```

-

Classification with Gaussian Bayes

- Continuous features
  - Assumption: Features are distributed normally
  - Example: Look again at Iris set
    - All features look normally distributed



Classification with Gaussian Naïve Bayes

- Possibility one: Disregard correlation —> Naïve
  - For each feature:
    - Calculate sample mean μ and sample standard deviation σ
    - Use these as estimators of the population mean and deviation
  - For a given feature value *x*, calculate the probability density assuming that x is in a category c
    - $P(x \mid c) \sim \mathcal{N}(\mu_c, \sigma_c)$

Classification with Gaussian Naïve Bayes

- Estimate the probability for observation $(x_1, x_2, \ldots, x_n)$ as the product of the densities

$$P((x_1, \ldots, x_n) \,|\, c_j) \sim \mathcal{N}(x_1, \sigma_{1,c_j}, \mu_{1,c_j}) \cdot \ldots \cdot \mathcal{N}(x_n, \sigma_{n,c_j}, \mu_{1,c_j})$$

- Then use Bayes formula to invert the conditional probabilities
  - This means estimating the prevalence of the categories
  - $P(c_j \,|\, (x_1, \ldots, x_n)) = \dfrac{P((x_1, \ldots, x_n) \,|\, c_j) P(c_j)}{P((x_1, \ldots, x_n))}$

Classification with Gaussian Naïve Bayes

- The denominator does not depend on the category c_j
- So, we just leave it out:
  - $P(c_j \mid (x_1, \ldots, x_n)) \sim P((x_1, \ldots, x_n) \mid c_j)P(c_j)$
- We calculate $P((x_1, \ldots, x_n) \mid c_j)P(c_j)$
  - And select the highest value

Classification with Gaussian Naïve Bayes

- Implemented in sklearn.naive_bayes
    - Example with Iris data-set

```
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB

iris = datasets.load_iris()
model = GaussianNB()
model.fit(iris.data, iris.target)
print('means', model.theta_)
print('stds', model.sigma_)

for x,t, p in zip(iris.data, iris.target, model.predict(iris.data)):
    print(x, t, p)
```

Classification with Gaussian Naïve Bayes

```
means [[5.006 3.428 1.462 0.246]
 [5.936 2.77  4.26  1.326]
 [6.588 2.974 5.552 2.026]]
stds [[0.121764 0.140816 0.029556 0.010884]
 [0.261104 0.0965   0.2164   0.038324]
 [0.396256 0.101924 0.298496 0.073924]]
[5.1 3.5 1.4 0.2] 0
[4.9 3.  1.4 0.2] 0
[4.7 3.2 1.3 0.2] 0
[4.6 3.1 1.5 0.2] 0
[5.  3.6 1.4 0.2] 0
[5.4 3.9 1.7 0.4] 0
```

Classification with Gaussian Naïve Bayes

- There are a few errors:

```
[6.9 3.1 4.9 1.5] 1 2
[5.9 3.2 4.8 1.8] 1 2
[6.7 3.  5.  1.7] 1 2
[4.9 2.5 4.5 1.7] 2 1
[6.  2.2 5.  1.5] 2 1
[6.3 2.8 5.1 1.5] 2 1
```

- Caution: We did not divide the data set into a training and verification set.

Classification with Not-So-Naïve Gaussian Bayes

- We did not use correlation between features
  - If we do, use the multi-variate probability density
  - Need to estimate correlation coefficients:

$$\sigma_{k,l} = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} (x_k - \mu_k)(x_l - \mu_l)$$

  - Then use the multi-variate normal probability density

$$\text{norm}_{\mu,\Sigma}(x) = \frac{1}{(\sqrt{2\pi})^d \sqrt{|\Sigma|}} \exp\left(-\frac{(x-\mu)^{\mathsf{T}} \Sigma^{-1} (x-\mu)}{2} \right)$$

Classification with Not-So-Naïve Gaussian Bayes

- Luckily, implemented in scipy.stats

```
from scipy.stats import multivariate_normal
```

- Estimate means and correlations
- Similarly to before, estimate category by looking at the multi-variate normal density for each category and updating

```
def diagnose(tupla):
    return np.argmax(
   [multivariate_normal.pdf(tupla,mean=Gl.mu_setosa, cov=Gl.sigma_setosa),
    multivariate_normal.pdf(tupla,mean=Gl.mu_ver, cov=Gl.sigma_ver),
    multivariate_normal.pdf(tupla,mean=Gl.mu_vgc, cov=Gl.sigma_vgc)])
```

Classification with Not-So-Naïve Gaussian Bayes

- This works slightly better: three mis-classifications
  - Example:
    - Virginica features:

```
>>> get_probs((6.3, 2.8, 5.1, 1.5))
setosa 6.551299963143457e-116
versicolor 0.3895029363227387
virginica 0.2572025404570884
```

    - Versicolor and virginica probs are similar

Classification with Not-So-Naïve Gaussian Bayes

- This works slightly better: three mis-classifications
  - Example:
    - Versicolor features:

```
>>> get_probs((6.0, 2.7, 5.1, 1.6))
setosa 3.4601607892612445e-119
versicolor 0.09776449471242309
virginica 0.5656860779792
```

    - Versicolor and virginica probs are somewhat similar

Scipy.learn

- A more modern set of tools in scipy
  - Running example:
    - How to predict the newsgroup from the contents
    - Data set:
    - `from sklearn.datasets import fetch_20newsgroups`

Scipy.learn

- A set of 18846 newsgroup contributions from way back
    - Split 2/3 : 1/3 into a training set (before a certain date) and a test set (after a certain date)

```
data = fetch_20newsgroups()
print(data.target_names)
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt',
'sci.electronics', 'sci.med', 'sci.space',
'soc.religion.christian', 'talk.politics.guns',
'talk.politics.mideast', 'talk.politics.misc',
'talk.religion.misc']
```

Scipy.learn

- We do not want all of them:

```
categories = [‘talk.religion.misc',
              'soc.religion.christian','alt.atheism',
              'sci.space', 'comp.graphics']
```

- Split into training and test sets

```
train = fetch_20newsgroups(subset='train', categories = categories)
test = fetch_20newsgroups(subset='test', categories = categories)
```

Scipy.learn

- Bag Of Words uses CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
```

- We extract the Bag of Words
- To display, we make the result into a Pandas Dataframe

```
vec = CountVectorizer()
X = vec.fit_transform(train.data)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

Scipy.learn

- The result is a matrix
  - Columns by words that appear
  - Rows by document number

```
>>> df.iloc[0:15, 10300:10330]
    comm  command  commanded  ...  commercialization  commercialized  commercially
0      0        0          0  ...                  0               0             0
1      0        0          0  ...                  0               0             0
2      0        0          0  ...                  0               0             0
3      0        0          0  ...                  0               0             0
4      0        0          0  ...                  0               0             0
5      0        0          0  ...                  0               0             0
6      0        0          0  ...                  0               0             0
7      0        0          0  ...                  0               0             0
8      0        0          0  ...                  0               0             0
9      0        0          0  ...                  0               0             0
10     0        0          0  ...                  0               0             0
11     0        0          0  ...                  0               0             0
12     0        0          0  ...                  0               0             0
13     0        0          0  ...                  0               0             0
14     0        0          0  ...                  0               0             0

[15 rows x 30 columns]
```

Scipy.learn

- Get better result by dividing the words by their frequency

```
vec = TfidfVectorizer()
X = vec.fit_transform(train.data)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

Scipy.learn

- Term Frequency
  - Take raw count and divide by the number of words in the document
- Inverse Document Frequency
  - — Logarithm of (Number of Documents w. word) / (Number of Documents)
- Term-Frequency — Inverse Document Frequency (TfIDF)
  - Product of these two

Scipy.learn

- Let’s make the difference clearer

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import pandas as pd


sample = ['in the beginning of time', 'at dawn we slept',
 'this is the story’, 'beginning and end', 'frequent beginning’,
 'beginning python']

vec = CountVectorizer()
X = vec.fit_transform(sample)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
print(df)

vec = TfidfVectorizer()
X1 = vec.fit_transform(sample)
df1 = pd.DataFrame(X1.toarray(), columns=vec.get_feature_names())
print(df1)
```

Scipy.learn

- CountVectorizer

```
   and  at  beginning  dawn  end  frequent  ...  slept  story  the  this  time  we
0    0   0          1     0    0         0  ...      0      0    1     0     1   0
1    0   1          0     1    0         0  ...      1      0    0     0     0   1
2    0   0          0     0    0         0  ...      0      1    1     1     0   0
3    1   0          1     0    1         0  ...      0      0    0     0     0   0
4    0   0          1     0    0         1  ...      0      0    0     0     0   0
5    0   0          1     0    0         0  ...      0      0    0     0     0   0

[6 rows x 16 columns]
```

Scipy.learn

- TfIdfVectorizer

```
        and   at  beginning  dawn  ...       the      this      time   we
0  0.000000  0.0   0.295730   0.0  ...  0.408763  0.000000  0.498483  0.0
1  0.000000  0.5   0.000000   0.5  ...  0.000000  0.000000  0.000000  0.5
2  0.000000  0.0   0.000000   0.0  ...  0.427903  0.521823  0.000000  0.0
3  0.652057  0.0   0.386839   0.0  ...  0.000000  0.000000  0.000000  0.0
4  0.000000  0.0   0.510227   0.0  ...  0.000000  0.000000  0.000000  0.0
5  0.000000  0.0   0.510227   0.0  ...  0.000000  0.000000  0.000000  0.0

[6 rows x 16 columns]
```

Scipy.learn

- CountVectorizer and TfidfVectorizer generate **sparse** matrices
    - Storage is compressed

Scipy.learn

- Multinomial Bayes is in sklearn
  - `from sklearn.naive_bayes import MultinomialNB`
- sklearn has a pipeline constructor
  - Combines feature extraction with training multinomial NB

```
from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(train.data, train.target)
labels = model.predict(test.data)
```

Scipy.learn

- To measure success:
  - Use a confusion matrix
    - For the test set: Show how often group elements are predicted to belong to another group
    - Fictitious example: Can a NN distinguish cats and dogs

	actual	
	dog	cat
predicted dog	1023	245
predicted cat	134	1183

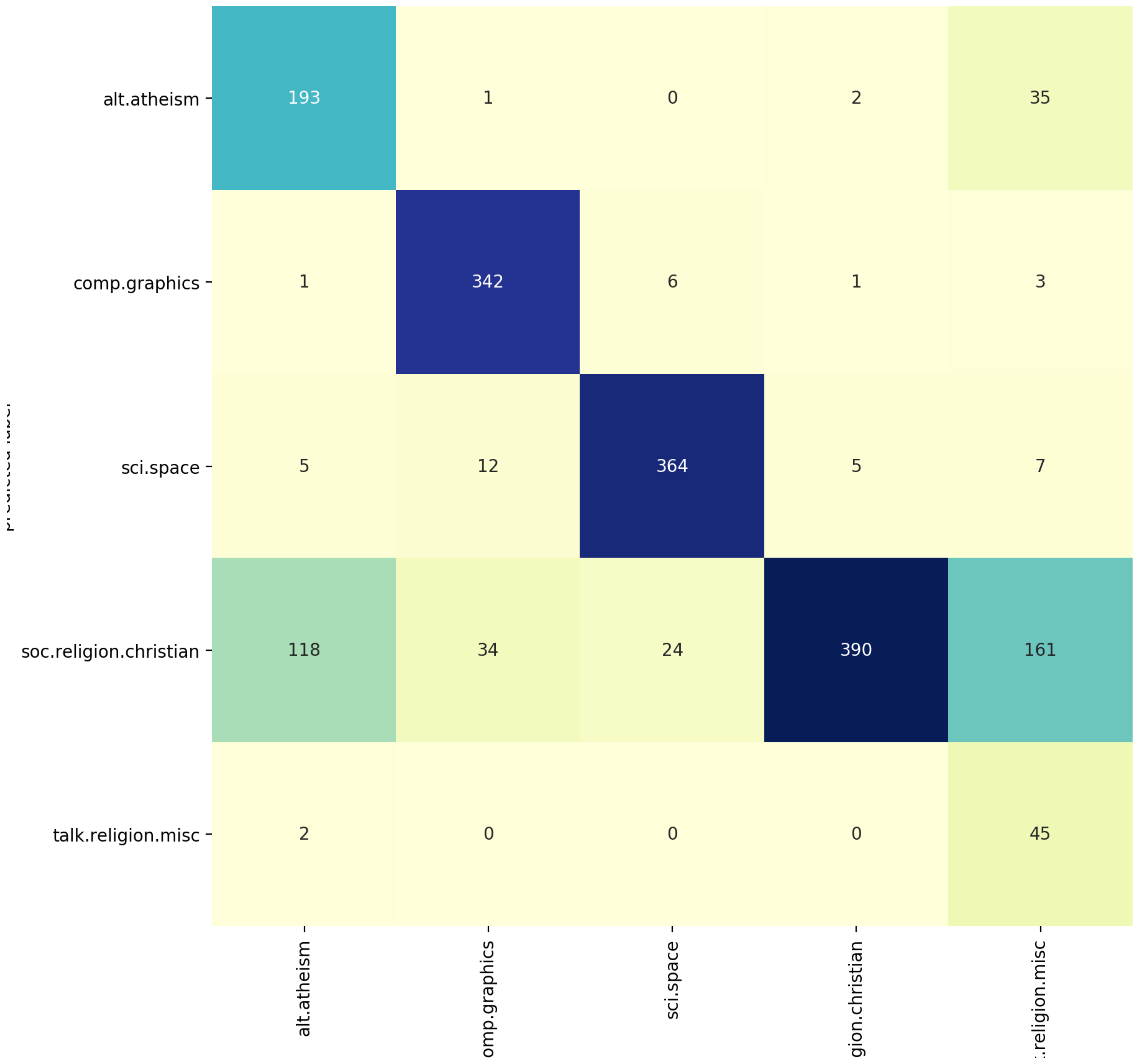
Scipy.learn

- Can find confusion matrix

```
from sklearn.metrics import confusion_matrix
```

- Import pyplot and seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt

mat = confusion_matrix(test.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=train.target_names,
            yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.show()
```


alt.atheism
comp.graphics
sci.space
soc.religion.christian
talk.religion.misc
193 1 0 2 35
1 342 6 1 3
5 12 364 5 7
118 34 24 390 161
2 0 0 0 45
alt.atheism
omp.graphics
sci.space
gion.christian
.religion.misc

Dimensionality Reduction

Thomas Schwarz, SJ

Introduction

- Real life problems need to learn from many features
- This can cause problem
  - Excursion
    - Why our intuition is wrong about high dimensional data

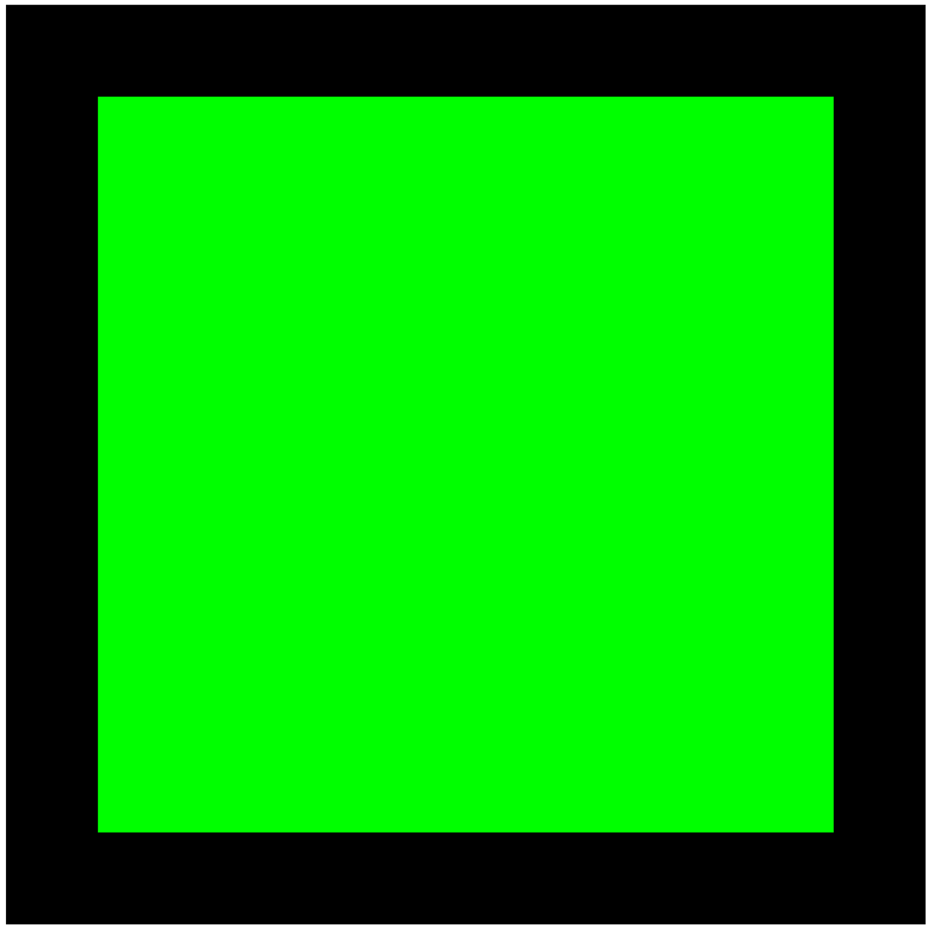
High-Dimensional Data

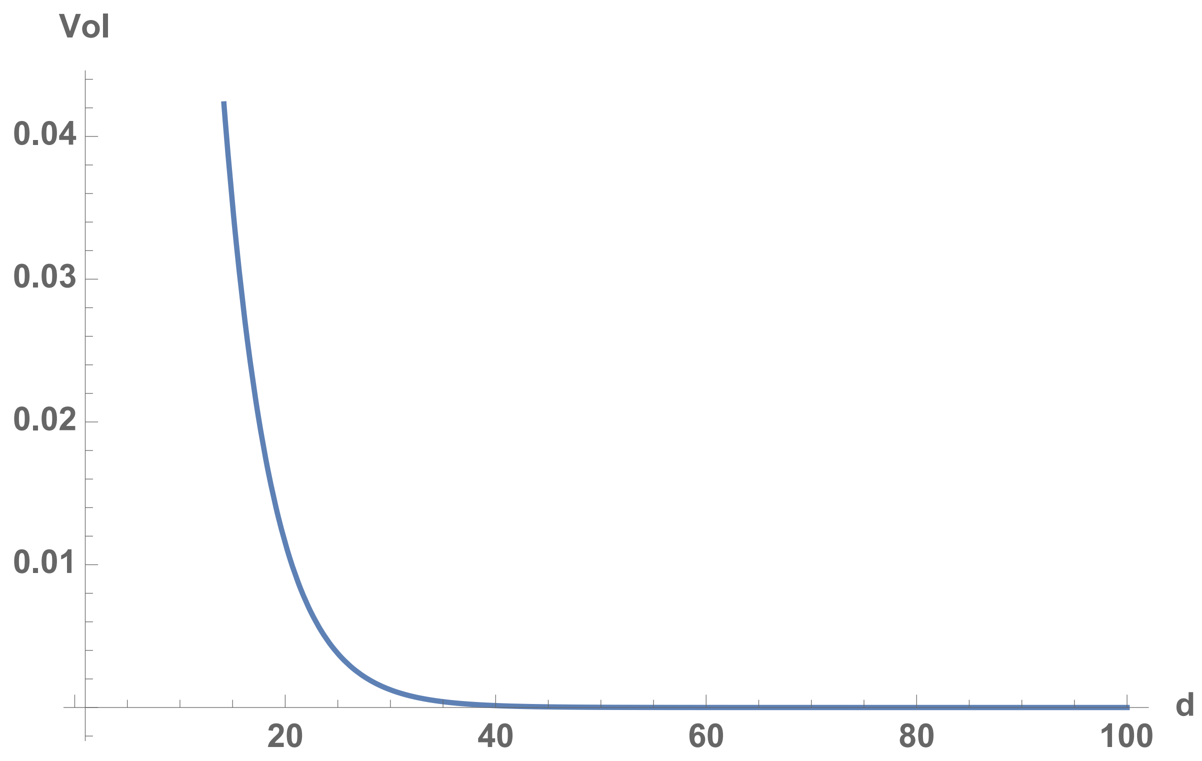
- As we increase the number of (numerical) dimensions:
  - Our intuition is faulty
  - Look at this using random numbers

High Dimensional Data

- Intuition 1: Random points tend to be close to the center
    - Hypercube volumes
        - Volume of a hypercube that stays 0.1 away from the edge

$$0.8^d$$



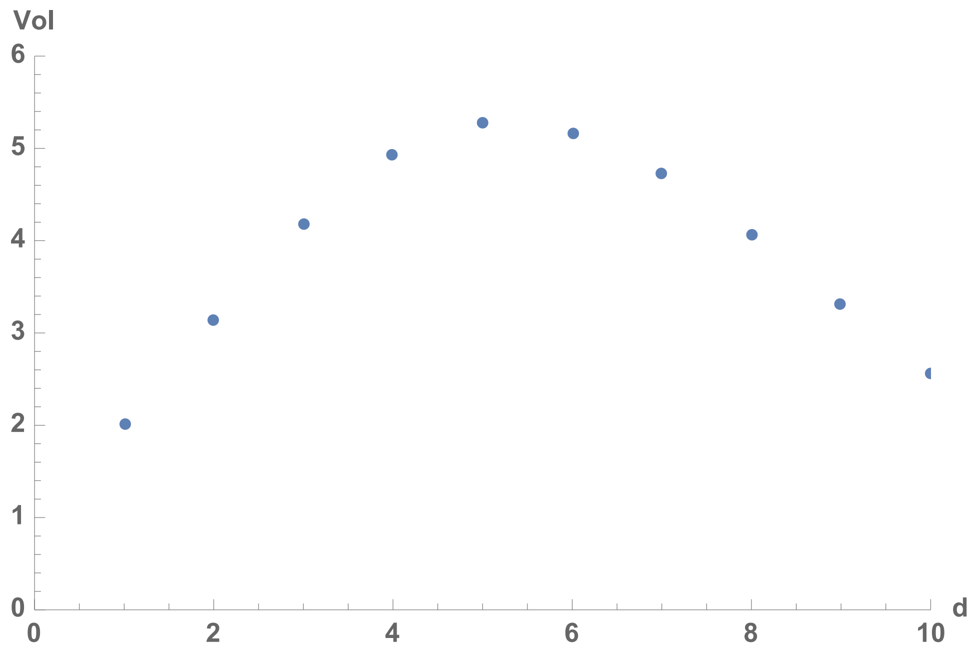


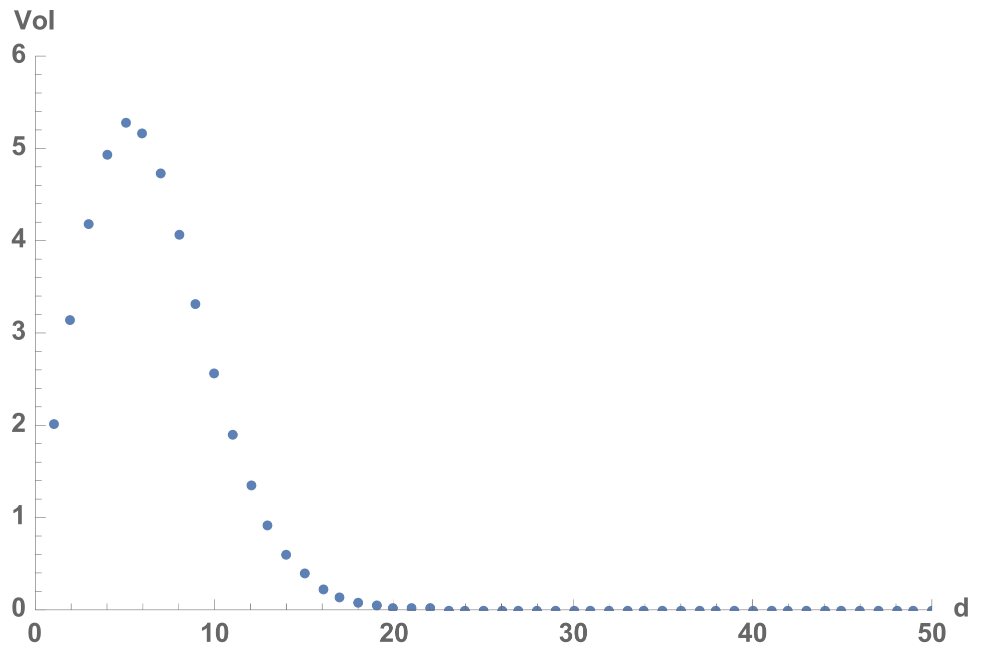
High-Dimensional Data

- Hyper-ball of radius *r*
- Volume is

$$r^d\left(\frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}\right)$$

- Maximum volume for dimension 5

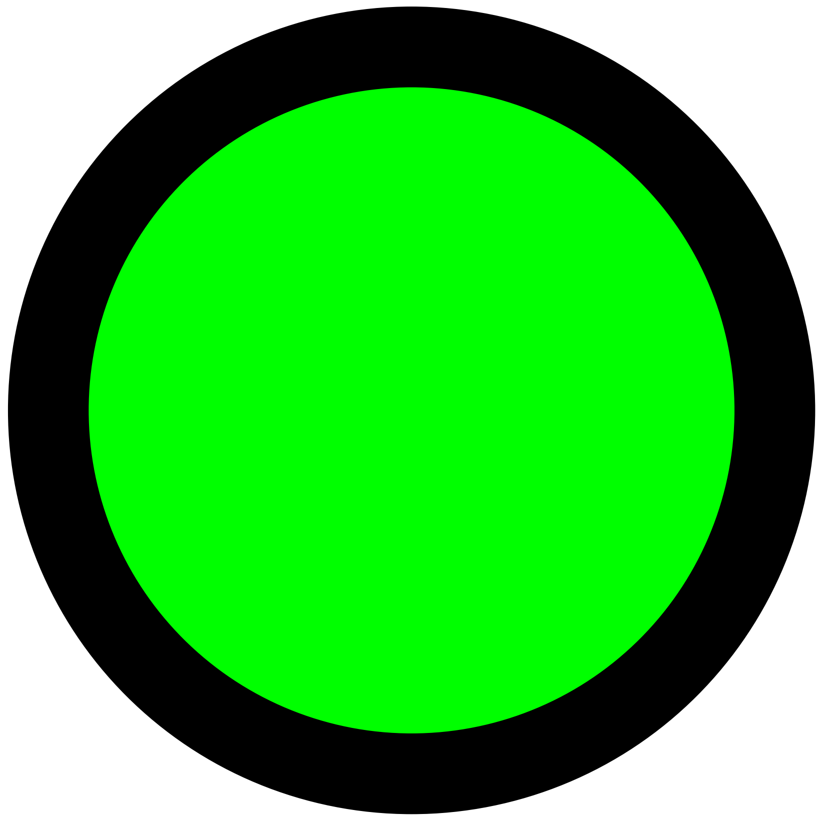


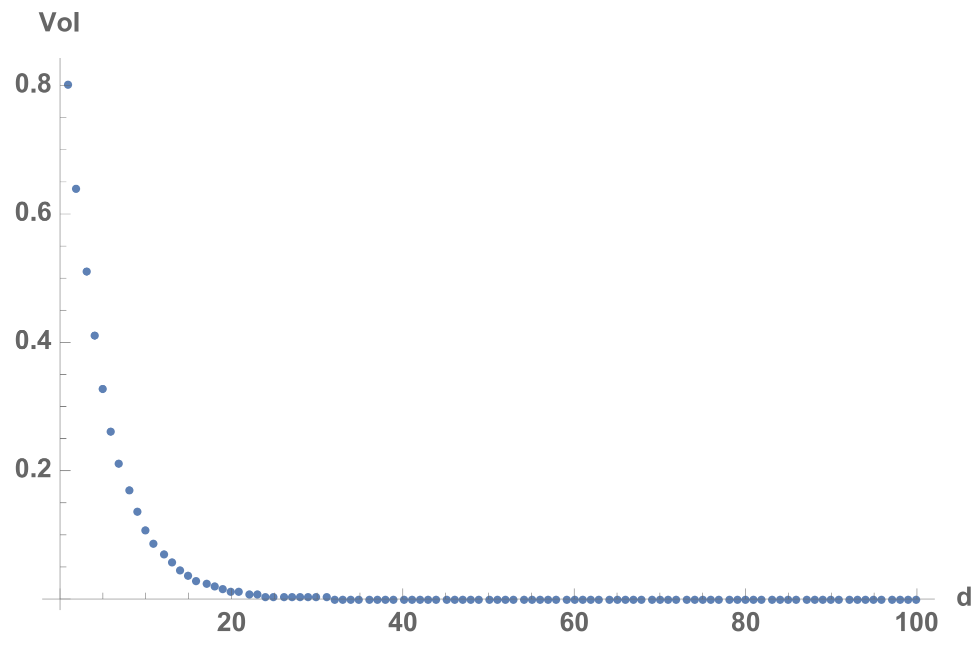


High-Dimensional Data

- Proportion of points in a unit hyper-ball that are 0.1 away from the edge is

$$\frac{0.4^d}{0.5^d}$$

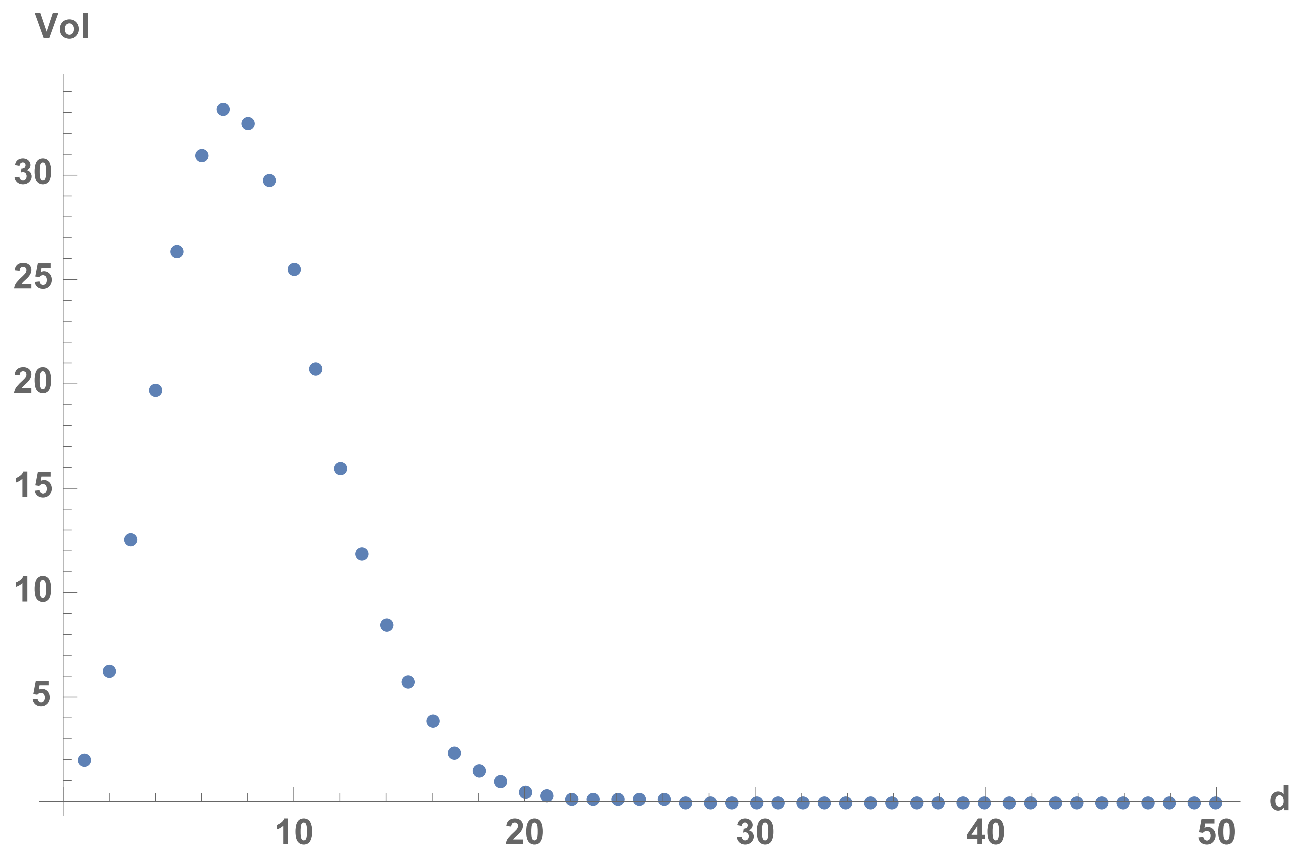




High-Dimensional Data

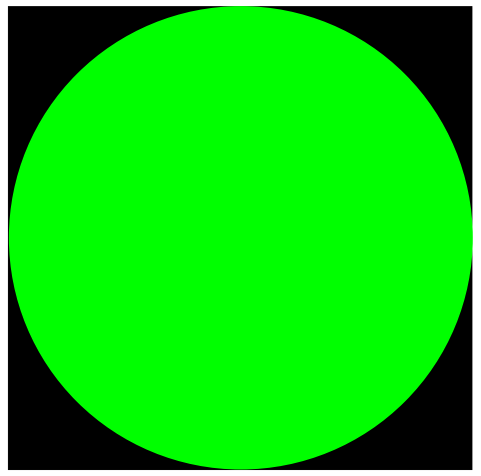
- Surface Area of the hypersphere in dimension *d*

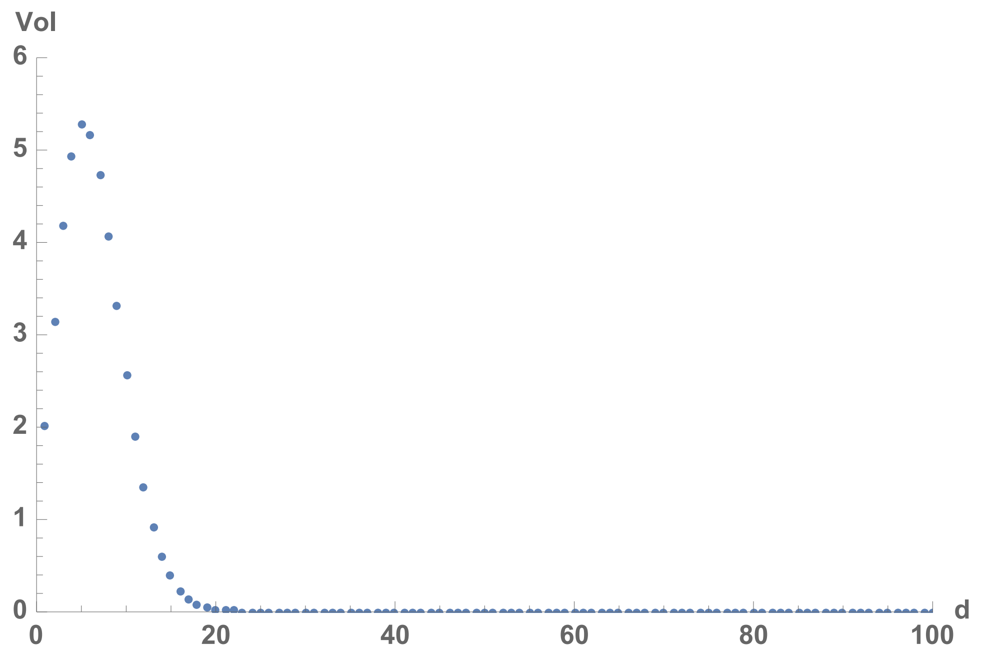
$$r^{d-1}\left(\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}\right)$$



High-Dimensional Data

- Hypersphere inscribed a hypercube
- Ratio of volume of hypersphere over volume of hypercube goes quickly to zero





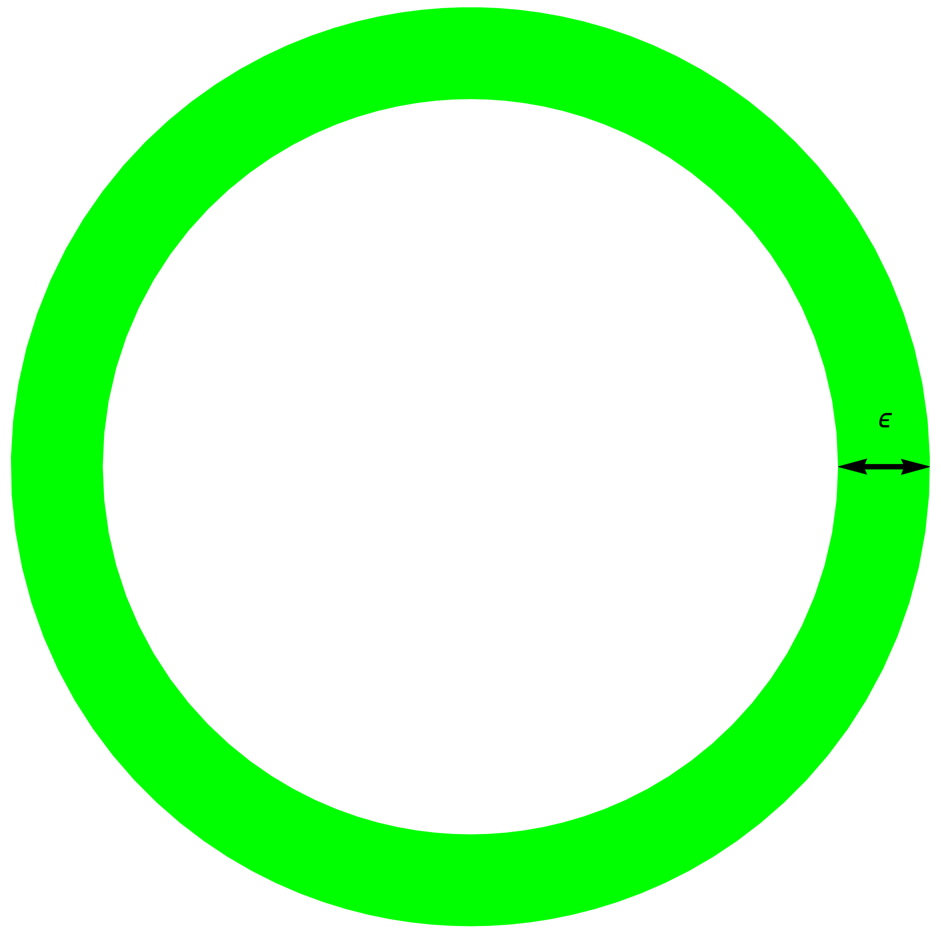
High-Dimensional Data

- Volume of a thin shell of width ε is

$$1 - (1 - \frac{\epsilon}{r})^d$$

- with

$$\lim_{d\to\infty} (1 - (1 - \frac{\epsilon}{r})^d = 1$$



High-dimensional Data

- Assume a multi-variate normal distribution centered around the origin and without covariances
- Probability density is given by

$$f(\overrightarrow{x}) = \frac{1}{\sqrt{2\pi}^d} \exp\left(- \frac{\overrightarrow{x}^T \overrightarrow{x}}{2} \right)$$

- Peak density is $f(\overrightarrow{0}) = \frac{1}{\sqrt{2\pi}^d}$

High-dimensional Data

- Set of points with *α* of the peak density is given by

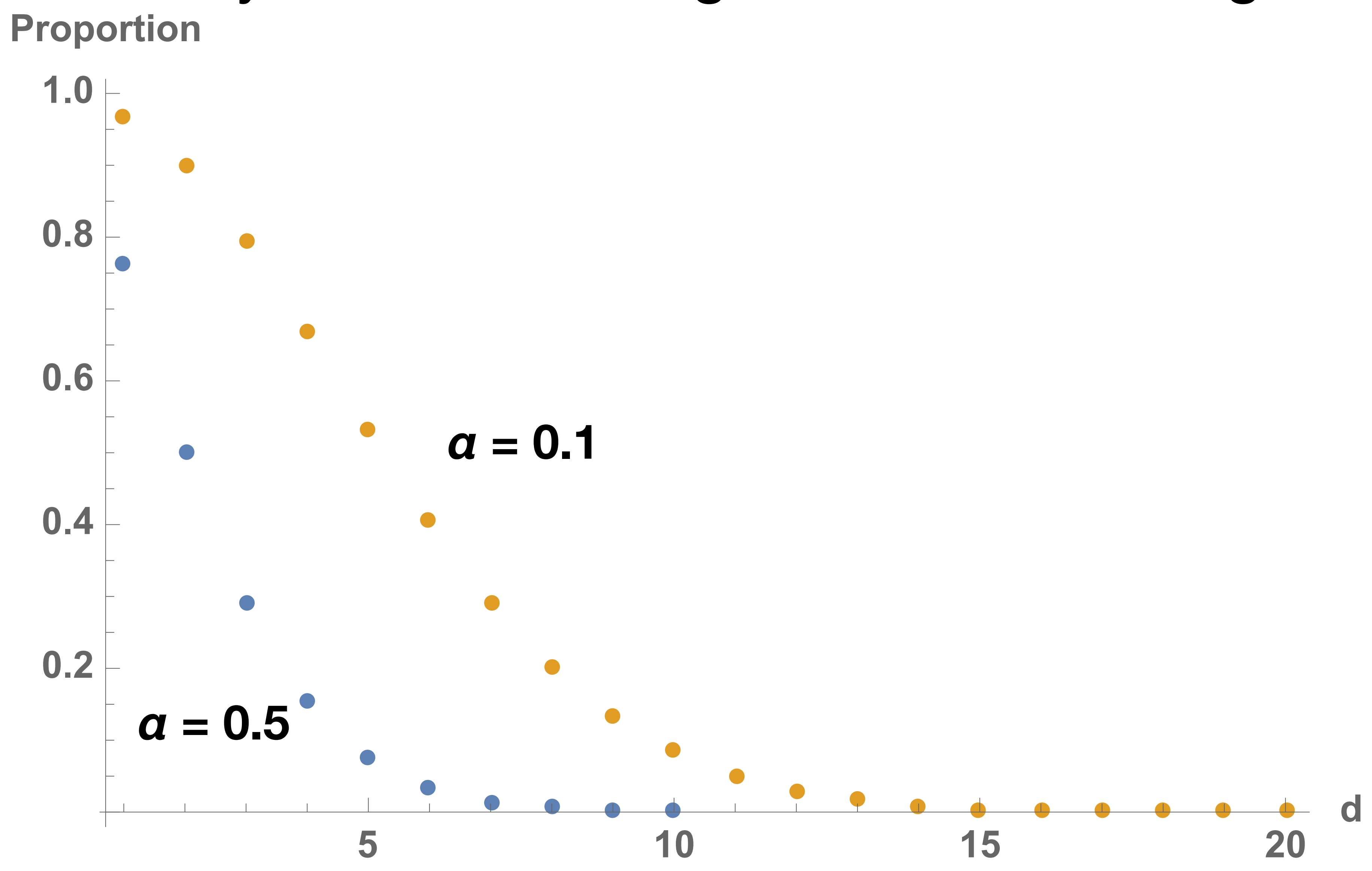
$$\frac{f(\overrightarrow{x})}{f(\overrightarrow{0})} \geq \alpha \quad \Leftrightarrow \quad \exp(-\frac{\overrightarrow{x}^T\overrightarrow{x}}{2}) \geq \alpha \quad \Leftrightarrow \quad \frac{\overrightarrow{x}^T\overrightarrow{x}}{2} \leq -\log_e(\alpha)$$

- Since $\overrightarrow{x}^T\overrightarrow{x}$ follows a χ^2 distribution, we can calculate the probability of a random point being within *α* of the peak density to be

$$F_{\chi^2_d}(-2\ln\alpha)$$

High Dimensional Data

- This proportion goes quickly to zero. Mass of the probability distribution migrates to the tail region



Consequences

- Almost all of the data is close to a boundary
- Almost all of the data is in a corner, whereas the center is empty
  - I.e. there are no more "typical" data points by looking at typical values in a random world

Consequences

- K-means algorithm
  - Determine the category of a new data point by choosing the closest *k* points to the new data point
  - Assign the majority of the categories of these *k* points to the new point
- Very simple but quite effective at learning categories
- But in high dimensions, all data points tend to be wide apart.

Consequences

- Naive or not naive Bayes
  - Estimate the distribution of points in a category with a multi-variate normal distribution (with or without correlations)
  - Categorize a new point according to the probabilities according these distributions
    - The category for which the probability of the point is the highest
      - Tends to be quite good until all probabilities are low
    - High dimensionality: expect many more close decisions (that have a higher failure quota)

Consequences

- Luckily, real world data does **not** look like random data
- But we still need to reduce dimensionality

Introduction

- Feature selection
  - Data sets contain often large numbers of features
    - Some of the features depend on each other
    - Selecting features
      - makes current classification fast
      - can generalize better from training to general data
    - This even works with Neural Networks

Introduction

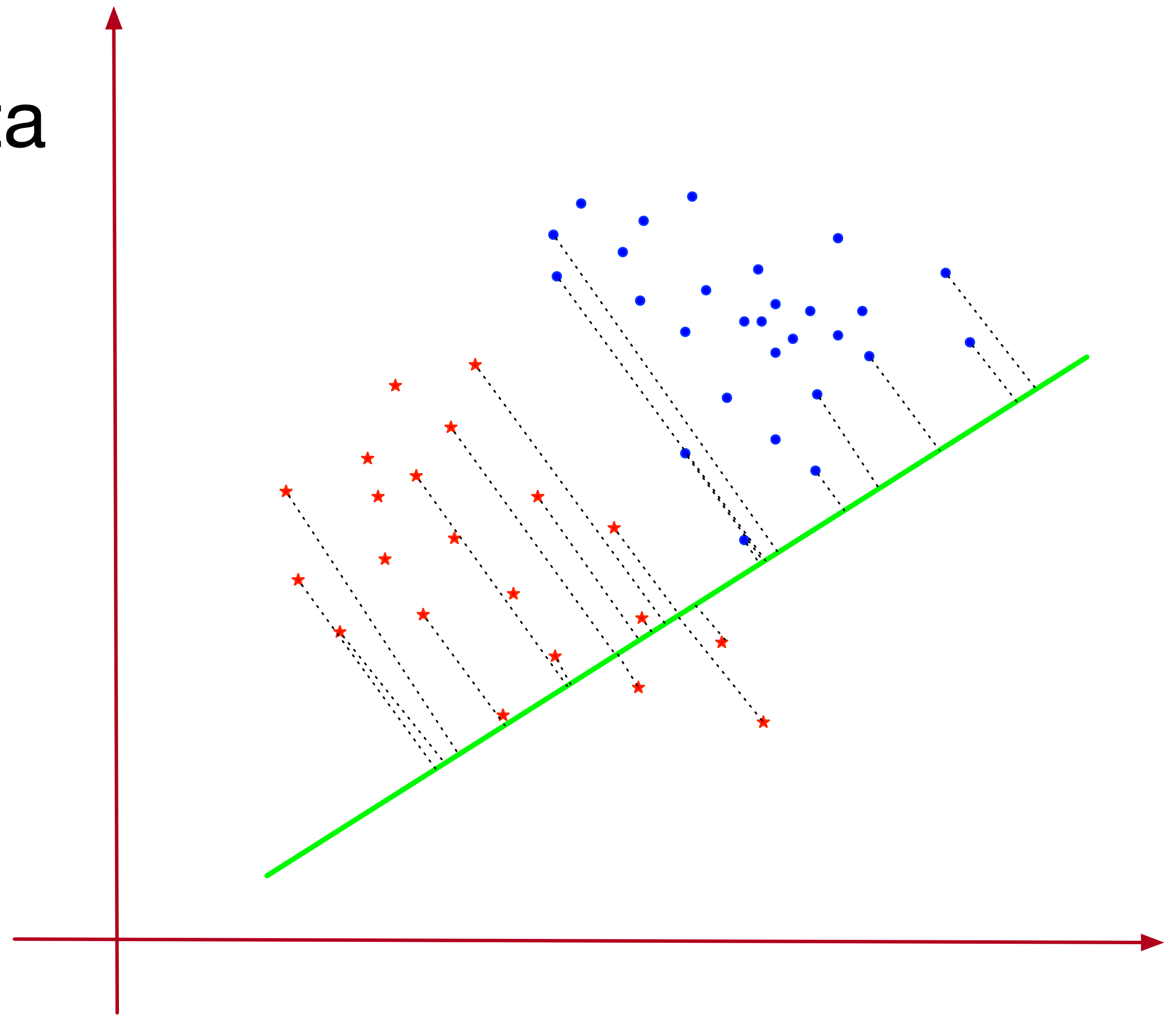
- Feature Combination:
  - Generate artificial features by combining features
    - Then do away with (some of the) old features

Introduction

- Clustering:
  - Automatic clustering
    - Groups similar data points
      - Often allows fewer features to be used

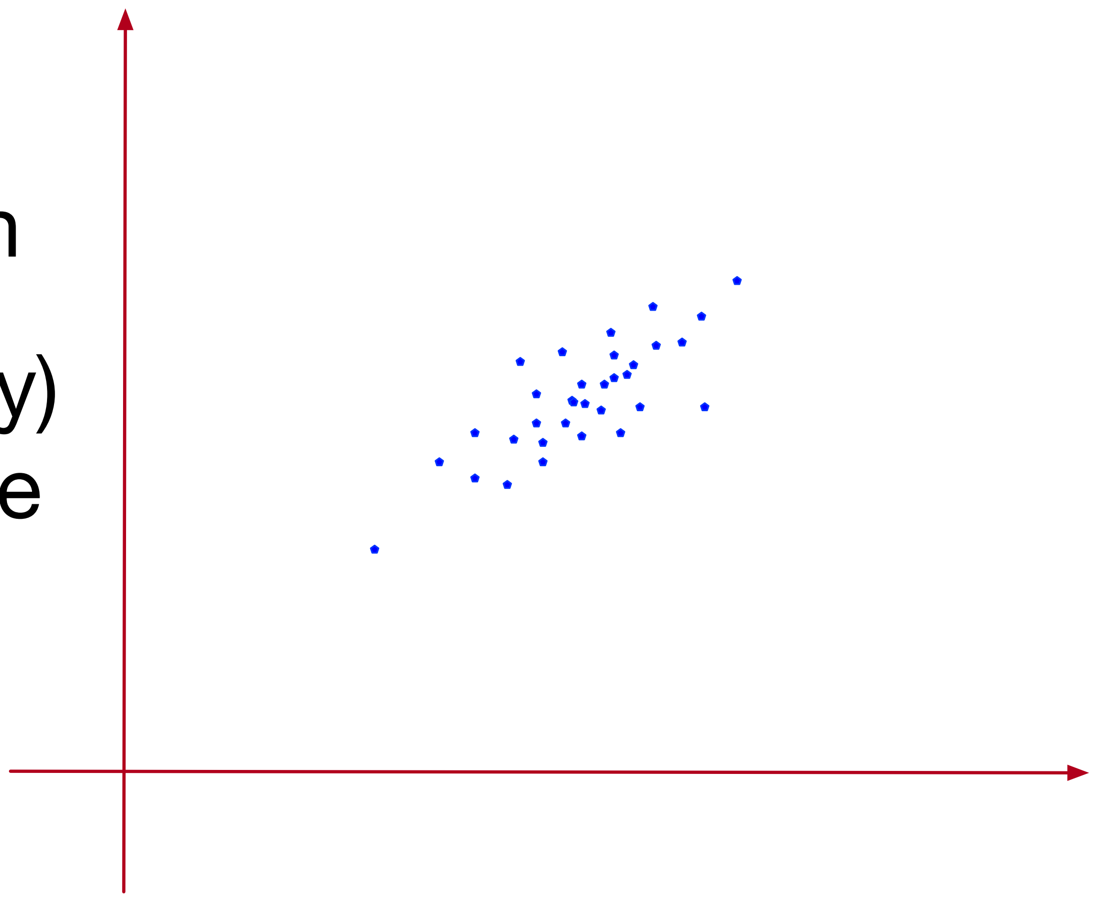
Introduction

- Automatic dimensionality reduction:
  - Project 2-dimensional data set on a single line
  - Projections separates the two data sets
  - Can use a ***single, combined*** feature for classification
    - Linear Discriminant Analysis



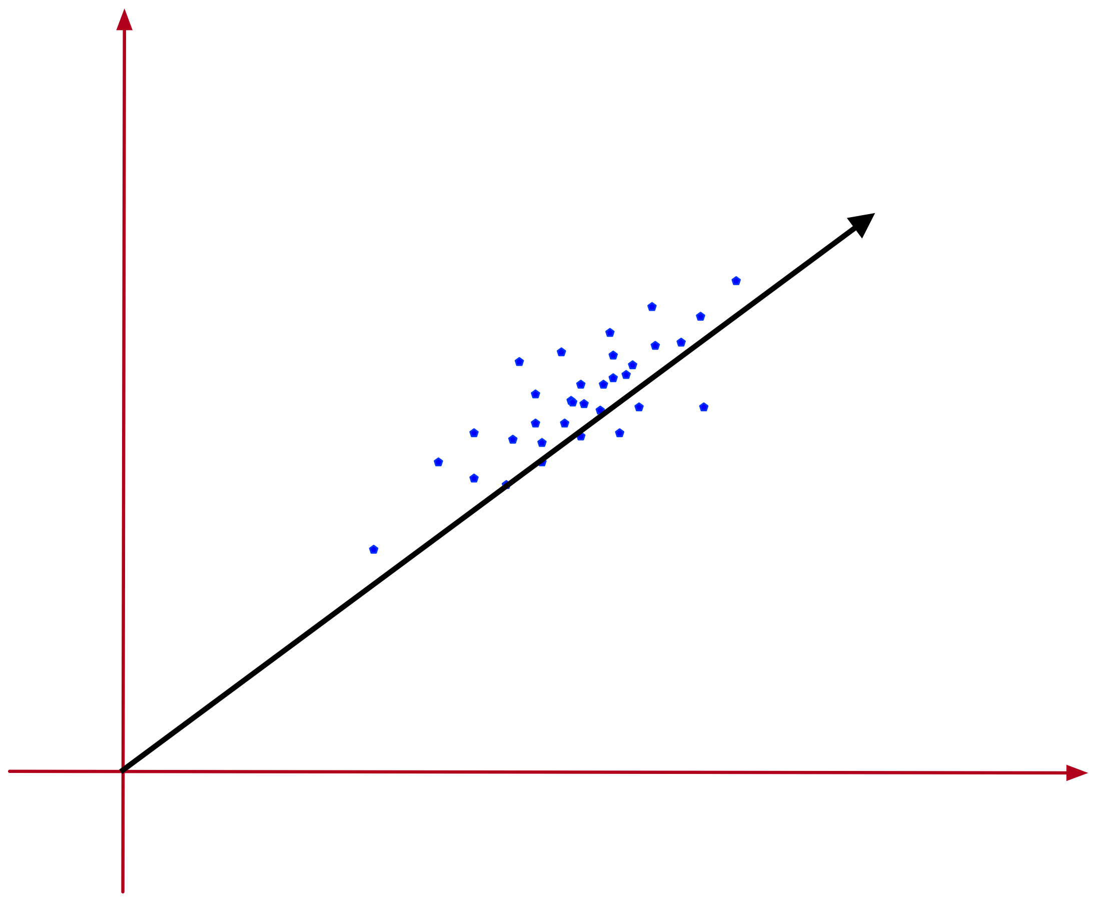
Introduction

- Two-dimensional data set
  - Spread around one dimension
  - Combine the two features (x, y) into one that has almost all the variance
    - Principal component analysis



Principal Component Analysis

- Goal:
  - Find the one direction in which the data sets varies most



Principal Component Analysis

- Given a set of U of data points with d numerical attributes
- Write as an $n \times d$ matrix

$$\mathbf{D} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ x_{3,1} & x_{3,2} & \dots & x_{3,d} \\ & \dots & & \\ x_{n,1} & x_{n,2} & \dots & x_{n,d} \end{pmatrix}$$

Principal Component Analysis

- Each data point is a linear combination of standard basis

$$\mathbf{x}_i = \sum_{j=1}^{d} x_{i,j}\mathbf{e_j}$$

- Dimensionality reduction:
  - Replace standard basis with another orthogonal matrix
  - Weight of data should be concentrated in a few dimensions

Principal Component Analysis

- Assume $(\mathbf{u}_i \,|\, i \in \{1,\ldots,d\})$ is such a basis
    - Then $\mathbf{u}_i \cdot \mathbf{u}_j = \delta_{i,j}$
    - Actually, any d vectors of length one with this property are a basis

Proof: If $\sum_{i=1}^{d} \alpha_i \mathbf{u}_i = 0$, then

$$0 = \mathbf{u}_j \cdot \sum_{i=1}^{d} \alpha_i \mathbf{u}_i = \alpha_j$$

Principal Component Analysis

- Write the vectors in an orthonormal basis as column vectors of a matrix

$$\mathbf{U} = \begin{pmatrix} | & | & \dots & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_d \\ | & | & \dots & | \end{pmatrix}$$

Then: $\mathbf{u}_i^t \mathbf{u}_j = \delta_{i,j}$ implies:

$$\mathbf{U}^T \mathbf{U} = \mathbf{1}_d$$

Principal Component Analysis

A feature vector $\mathbf{x}$ is a linear combination $\mathbf{x} = \sum_{i=1}^{d} \alpha_i \mathbf{u}_i$.

Write: $\mathbf{a} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$

Then $\mathbf{x} = \mathbf{a} \cdot \mathbf{U}^t$ or equivalently $\mathbf{x}^t = \mathbf{U} \cdot \mathbf{a}^t$

Principal Component Analysis

$U = (\frac{1}{\sqrt{2}}, 0, \frac{-1}{\sqrt{2}}), (\frac{1}{\sqrt{6}}, \frac{\sqrt{2}}{\sqrt{3}}, \frac{1}{\sqrt{6}}), (\frac{1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$ is an orthonormal basis of $\mathbb{R}^3$

Matrix is $\mathbf{U} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{3}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{pmatrix}$

Principal Component Analysis

Column vector $(2,1,3)^t$ is a linear combination of the column vectors of $\mathbf{U}$.

Use $\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} = \mathbf{U} \cdot \mathbf{a}^t$

Multiply with $\mathbf{U}^t$

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{6}} & \frac{\sqrt{2}}{\sqrt{3}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} = \mathbf{U}^t \cdot \mathbf{U} \cdot \mathbf{a} = \mathbf{a}^t$$

Obtain: $(-\frac{3}{\sqrt{2}} + \sqrt{2}, 2\sqrt{\frac{2}{3}} + \sqrt{\frac{3}{2}}, \frac{1}{\sqrt{3}} + \sqrt{3}) = \mathbf{a}$

Principal Component Analysis: Projection on Subspace

Write a data point as $\mathbf{x} = \sum_{i=1}^{d} \alpha_i \mathbf{u}_i$.

Assume that we have ordered the basis by importance

We select only the first r components:

Write: $\mathbf{U}_r = \begin{pmatrix} | & | & \dots & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_r \\ | & | & \dots & | \end{pmatrix}$

Then set $\pi_r(\mathbf{x}) = \sum_{i=1}^{r} \alpha_i \mathbf{u}_i = \mathbf{U}_r \cdot (\alpha_1, \alpha_2, \dots, \alpha_r)^t$

Principal Component Analysis: Projection on Subspace

Since $\mathbf{a}^t = \mathbf{U}^t \cdot \mathbf{x}$, it follows $\quad \pi_r(\mathbf{a}^t) = \mathbf{U}_r^t \mathbf{x}^t$ and

$$\pi_r(\mathbf{x}^t) = \mathbf{U}_r \pi_r(\mathbf{a}^t) = \mathbf{U}_r \mathbf{U}_r^t \mathbf{x}^t$$

$\Pi_r = \mathbf{U}_r \mathbf{U}_r^t$ is called the projection matrix since

(a) $\Pi_r \cdot \Pi_r = \mathbf{U}_r \mathbf{U}_r^t \mathbf{U}_r \mathbf{U}_r^t = \mathbf{U}_r \mathbf{U}_r^t$

(b) $\Pi_r^t = (\mathbf{U}_r \mathbf{U}_r^t)^t = \mathbf{U}_r^{t\,t} \mathbf{U}^t = \mathbf{U}_r \mathbf{U}_r^t \Pi_r = \mathbf{U}_r \mathbf{U}_r^t$

Principal Component Analysis: Projection on Subspace

Example (continued): Project on the first two coordinates with respect to U

$$\mathbf{U}_r = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \\ 0 & \frac{\sqrt{2}}{\sqrt{3}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \end{pmatrix}$$

Principal Component Analysis: Projection on Subspace

Then we calculate the projection matrix

$$\Pi_2 = \mathbf{U}_2\mathbf{U}_2^t = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} & \frac{-1}{3} \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{-1}{3} & \frac{1}{3} & \frac{2}{3} \end{pmatrix}$$

Principal Component Analysis: Projection on Subspace

Projection of $\mathbf{x}^t = (2,1,3)$ is

$$\Pi_2(\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}) = \begin{pmatrix} \frac{2}{3} \\ \frac{7}{3} \\ \frac{5}{3} \end{pmatrix}$$

Principal Component Analysis: Projection on Subspace

- Now we know how to project
  - Need to find the best orthonormal matrix for the projection

Single Principal Component Analysis

- There are infinitely many choices of orthonormal bases
- Start out with reduction to a single dimension
- First step: Center the data set
    - By subtracting the mean of the data set
- Therefore: **The mean of the data set is now zero**

Single Principal Component Analysis

- If we reduce to a single dimension, than the partial basis is given by a **single** vector $\mathbf{u}$.
- *Optimality criterion:* Projection maximizes the variance

Single Principal Component Analysis

$$\text{var}(\{\mathbf{u}^t\mathbf{x}_i \mid i \in \{1,\ldots,n\}\}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}^t\mathbf{x}_i - \mathbf{u}^t(\overline{\mathbf{x}}))^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}^t\mathbf{x}_i)^2 \qquad \text{(Average is zero)}$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}^t\mathbf{x}_i)(\mathbf{u}^t\mathbf{x}_i)^t$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbf{u}^t\mathbf{x}_i\mathbf{x}_i^t\mathbf{u}$$

$$= \mathbf{u}^t\left(\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^t\right)\mathbf{u} = \mathbf{u}^t\Sigma\mathbf{u}$$

Single Principal Component Analysis

- Therefore: $\mathbf{u}^t \Sigma \mathbf{u} \longrightarrow \max$ subject to $\mathbf{u}^t \mathbf{u} = 1$

- Use Lagrange multiplier λ and now maximize

$$J(\mathbf{u}) := \mathbf{u}^t \Sigma \mathbf{u} - \lambda(\mathbf{u}^t \mathbf{u} - 1)$$

- So, we differentiate:

$$\frac{\delta}{\delta \mathbf{u}} J(\mathbf{u}) = 2\Sigma \mathbf{u} - 2\lambda$$

Single Principal Component Analysis

- Result: Maximum obtained if $\Sigma \mathbf{u} = \lambda \mathbf{u}$
- With other words: $\mathbf{u}$ has to be an eigenvector of Σ with eigenvalue λ.
- And to maximize, we want the eigenvector with the largest eigenvalue
-

Single Principal Component Analysis

- Turns out that finding the maximum eigenvector and eigenvalue is quite simple:
  - Write any non-zero vector as a combination of eigen-vectors
  - Then repeatedly apply the matrix, but always normalize the product
  - The coefficient corresponding to the largest eigenvalue gets more and more magnified
  - And in the limit, the product will be the eigenvector corresponding to the largest eigenvalue

Single Principal Component Analysis

- Another goodness criterion:
  - Minimize the sum of squares of the differences between projected values and original values of the feature vector
  - Error is

$$||\mathbf{x} - \Pi_1(\mathbf{x})||^2 = (\mathbf{x} - \Pi_1(\mathbf{x}))^t(\mathbf{x} - \Pi_1(\mathbf{x}))$$

Single Principal Component Analysis

$$\sum_{i=1}^{n} ||\mathbf{x}_i - \Pi_1(\mathbf{x} - i)||^2$$

$$= \sum_{i=1}^{n} (\mathbf{x}_i - \Pi_1(\mathbf{x}_i)^t(\mathbf{x}_i - \Pi_1(\mathbf{x}_i))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - 2\mathbf{x}_i^t\Pi_1(\mathbf{x}_i) + \Pi_1(\mathbf{x})^t\Pi_1(\mathbf{x}))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - 2(\mathbf{u}^t\mathbf{x}_i)(\mathbf{x}_i^t\mathbf{u}) + (\mathbf{u}^t\mathbf{x}_i)(\mathbf{u}^t\mathbf{x}_i)\mathbf{u}^t\mathbf{u})$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - 2(\mathbf{u}^t\mathbf{x}_i)(\mathbf{x}_i^t\mathbf{u}) + (\mathbf{u}^t\mathbf{x}_i)(\mathbf{u}^t\mathbf{x}_i))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - (\mathbf{u}^t\mathbf{x}_i)(\mathbf{x}_i^t\mathbf{u}))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - \sum_{i=1}^{n} (\mathbf{u}^t\mathbf{x}_i\mathbf{x}_i^t\mathbf{u})$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - \mathbf{u}^t(\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^t)\mathbf{u}$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - \mathbf{u}^t(\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^t)\mathbf{u}$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - n\mathbf{u}^t\Sigma\mathbf{u}$$

Single Principal Component Analysis

- This means:
    - In order to minimize the sum of squared errors,
    - Need to minimize the projected variance
- Our two criteria are the **same**

Dual Principal Component Analysis

- We can redo our calculation for two dimensions
- Calculate just as before the minimum variance
- Obtain: minimum variance is the sum of the two largest eigenvalues
- Need to pick the two eigenvectors with the two largest eigenvalues

PCA in Python

- Part of sklearn.decomposition
    - Import bunch of modules

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

- Create random, but skewed data set

```
rng = np.random.RandomState(2020716)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
```

PCA in Python

- Here is some code to draw a vector

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=1,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)
```

PCA in Python

- Calculate the PCA (with two components)

  -
    ```
    pca = PCA(n_components=2)
    pca.fit(X)

    print(pca.components_)
    print(pca.explained_variance_)
    ```

PCA in Python

- First component has almost all the variance:

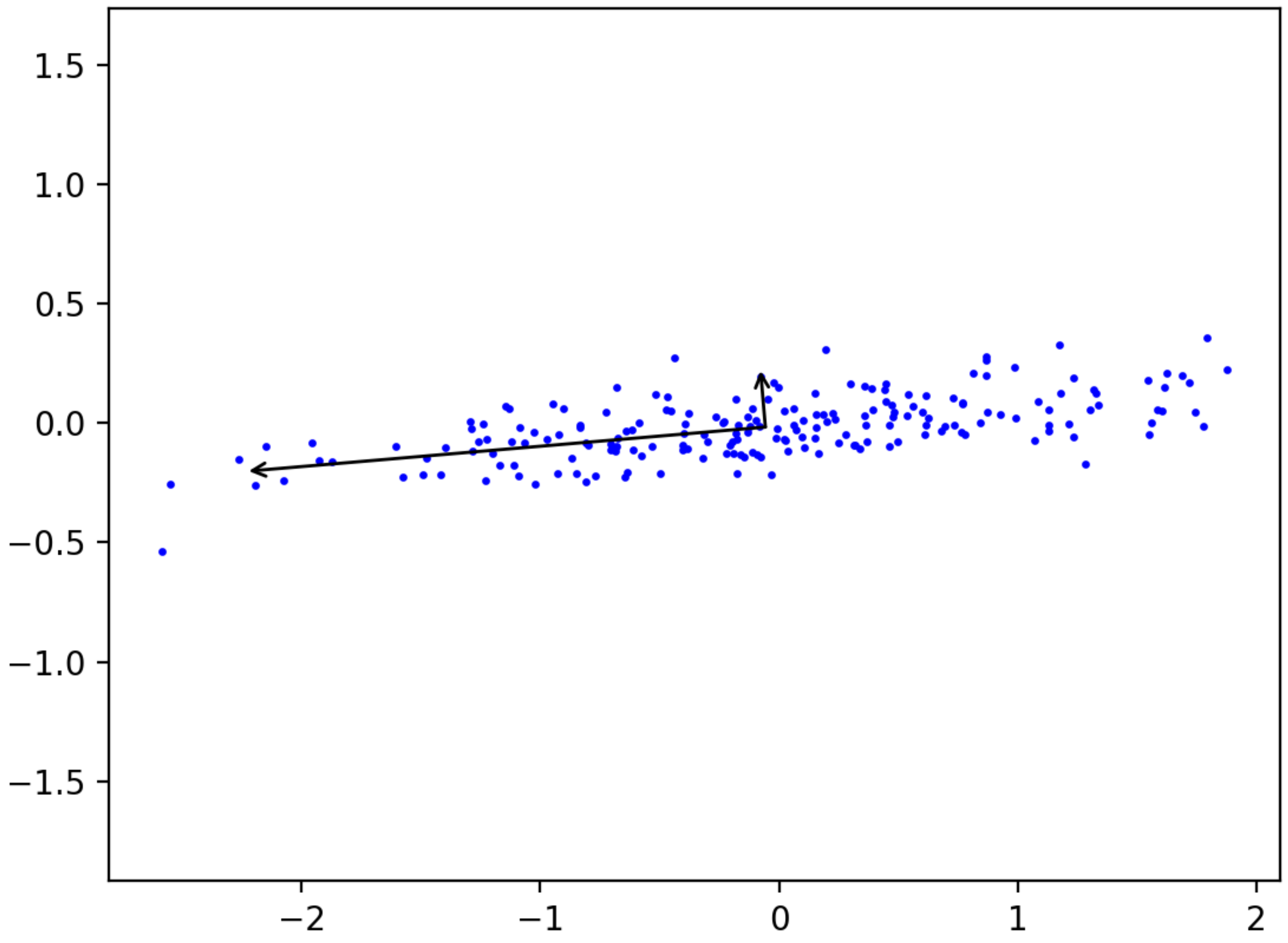
```
[[-0.99638832 -0.08491358]
 [-0.08491358  0.99638832]]
[0.89143208 0.01057402]
```

PCA in Python

- Draw everything:
-

```
plt.scatter(X[:, 0], X[:, 1], s=2, c='blue')
for length, vector in zip(pca.explained_variance_,
pca.components_):
    v = vector * 2.3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)

plt.axis('equal')
plt.show()
```

PCA in Python

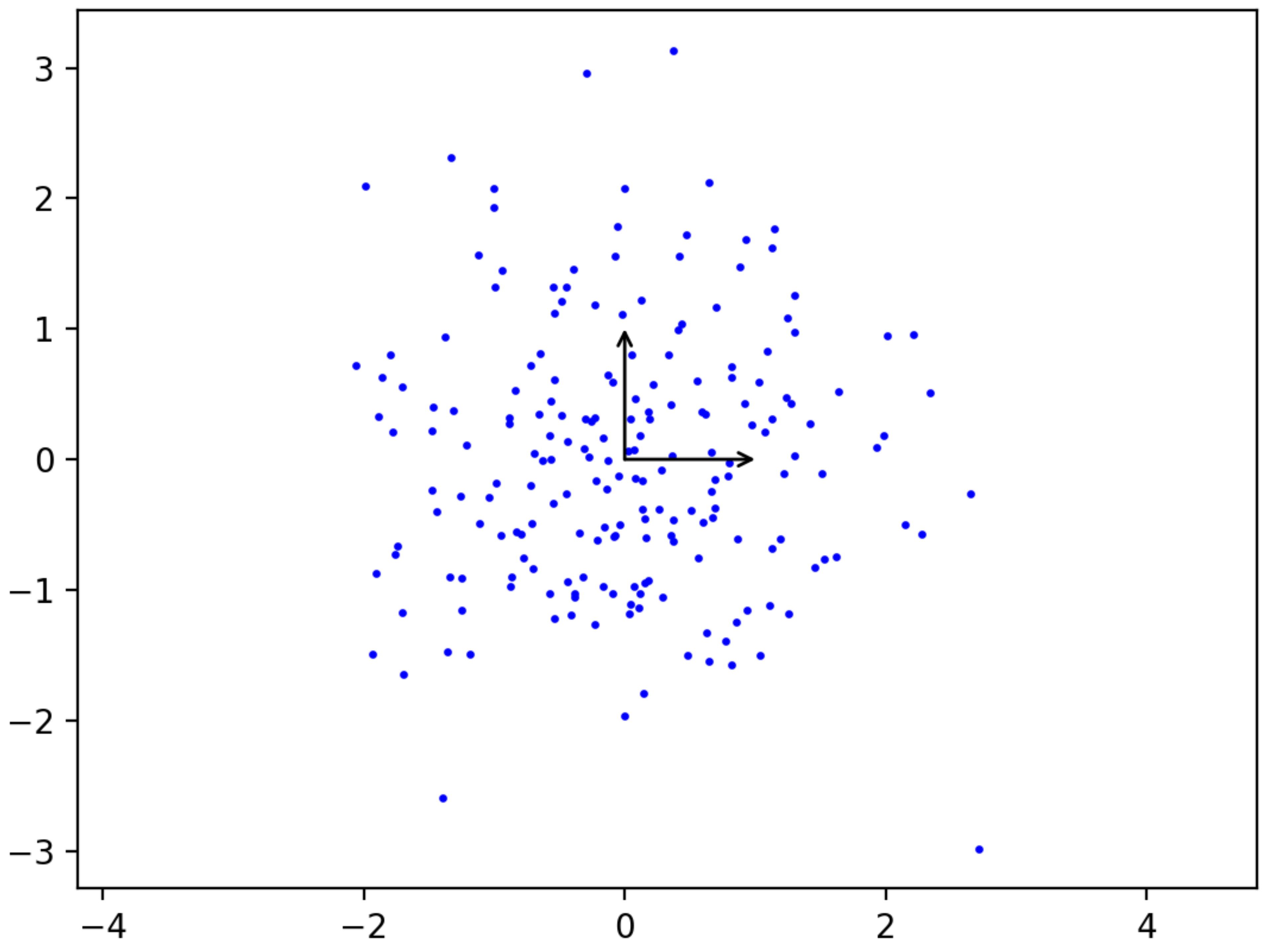


PCA in Python

- Can express data points in the new coordinates:
-

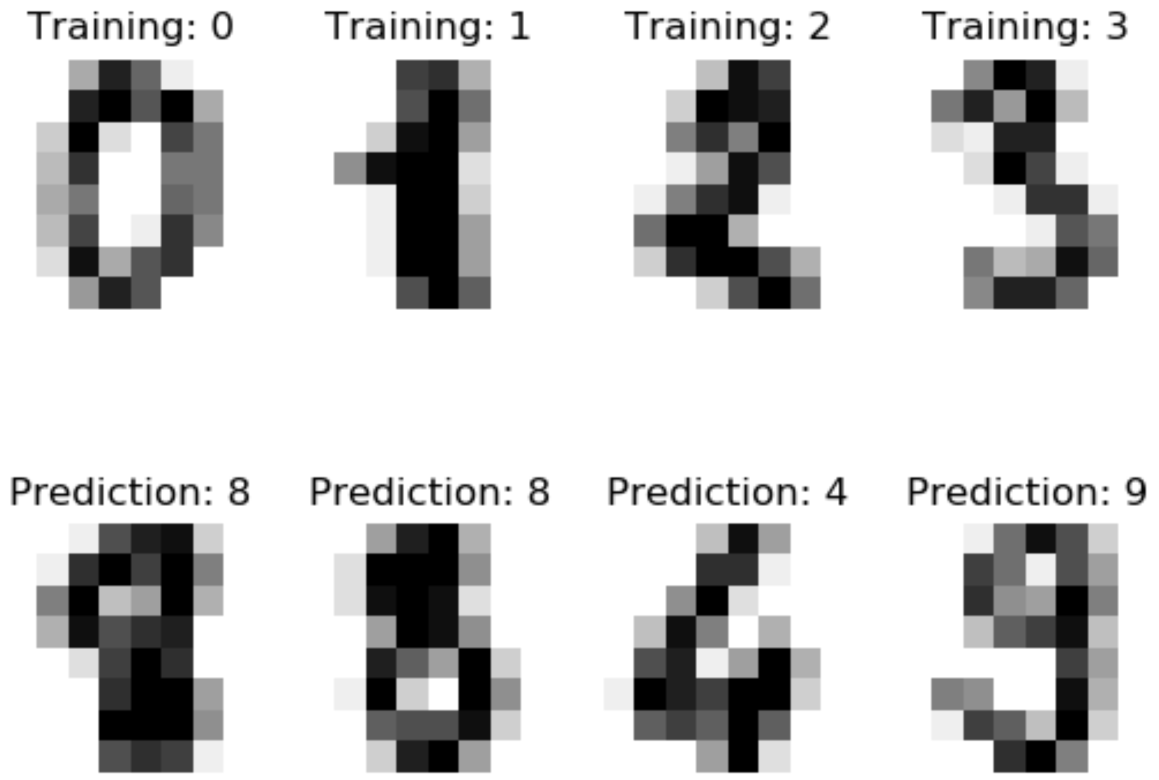
```
pca = PCA(n_components=2, whiten=True)
pca.fit(X)

X_pca = pca.transform(X)
```

PCA in Python



PCA in Python

- Sklearn has the digit data-set
  - Used for learning how to recognize digits for post-office automation, etc



PCA in Python

- Images have 64 pixels with gray values

```
from sklearn.datasets import load_digits

digits = load_digits()
```

```
>>> digits.data.shape
(1797, 64)
```

PCA in Python

- Can use PCA to lower dimension to two

```
pca = PCA(2)
projected = pca.fit_transform(digits.data)
```

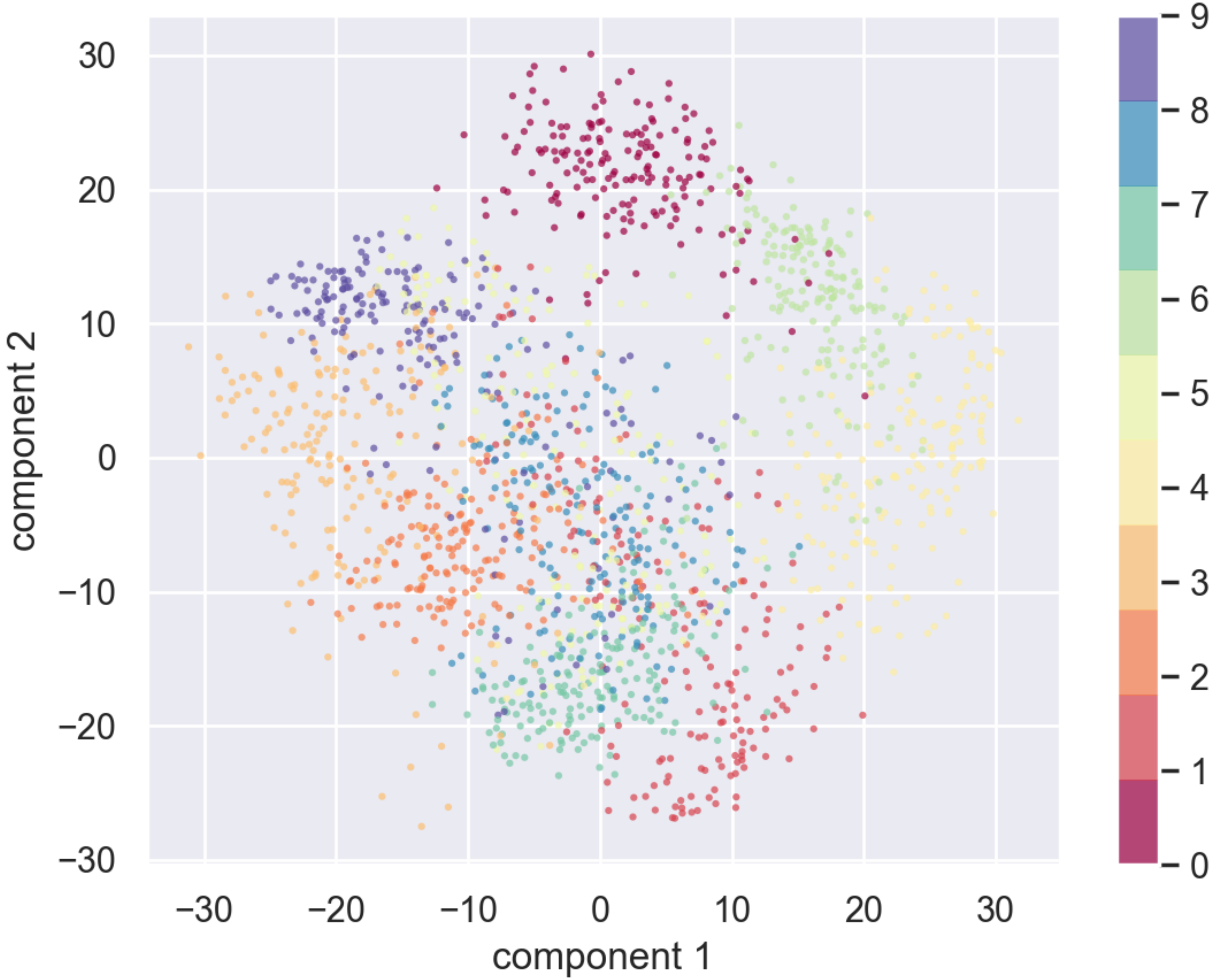
PCA in Python

- And display with the Spectral colormap

```
plt.scatter(projected[:, 0],
            projected[:, 1],
            s=5,
            c=digits.target,
            edgecolor='none',
            alpha=0.7,
            cmap=plt.cm.get_cmap('Spectral', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();

plt.show()
```

PCA in Python

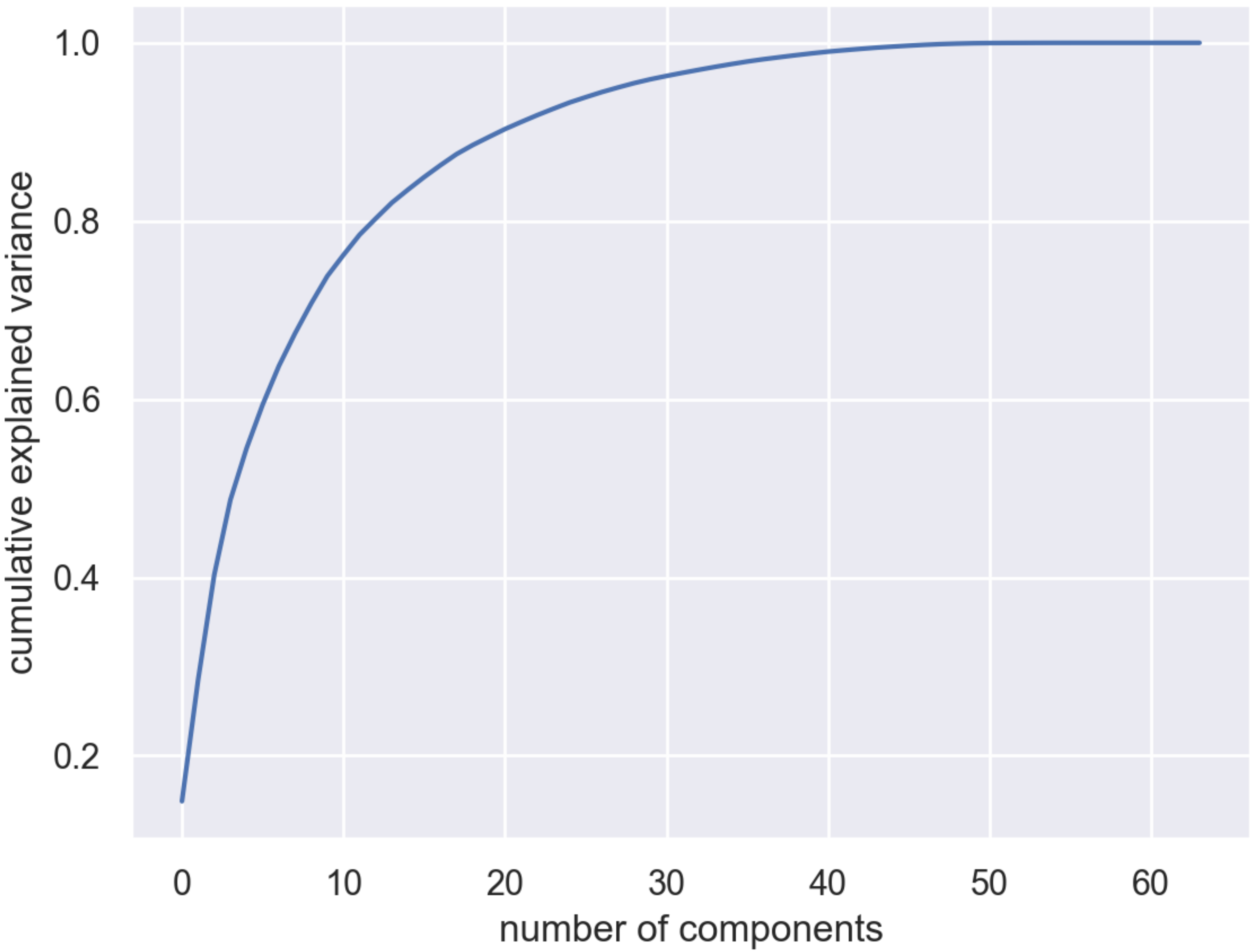
- Result shows that two features already give a decent classification:


component 1
component 2
−30
−20
−10
0
10
20
30
0
1
2
3
4
5
6
7
8
9

PCA in Python

- We can calculate the complete orthonormal base
    - And decide how many features we might need by looking at the total explained variance

```
pca = PCA().fit(digits.data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')

plt.show()
```

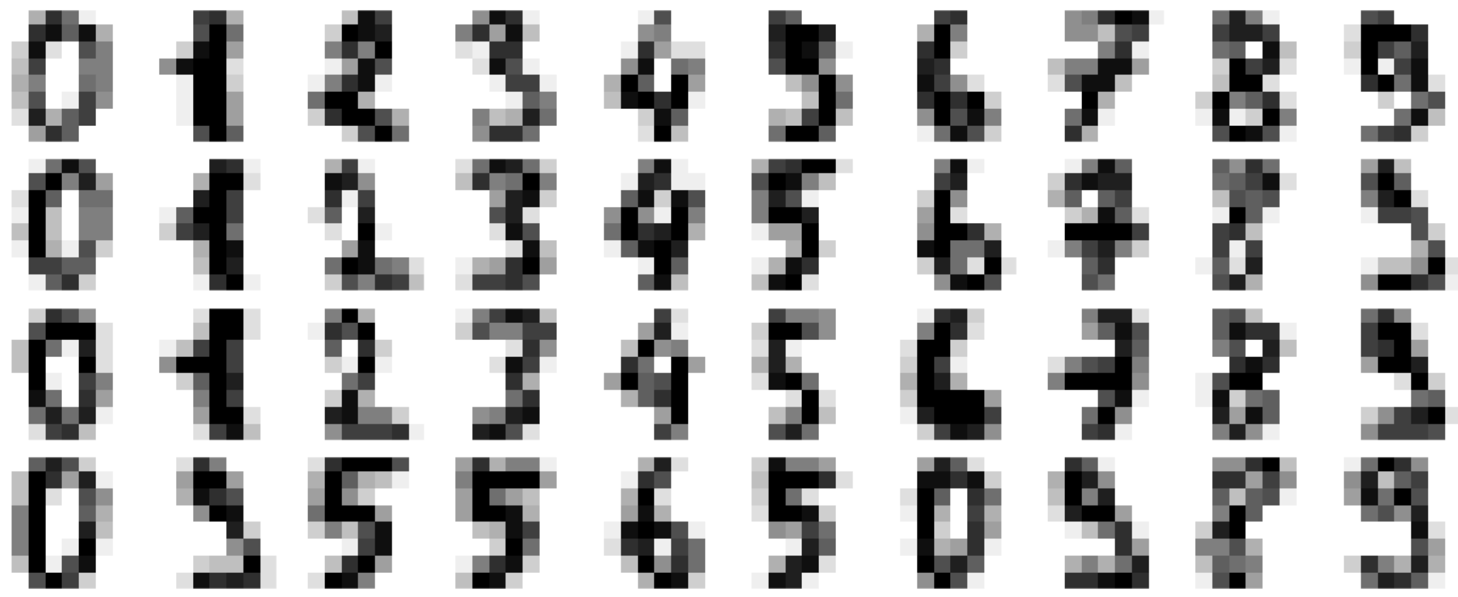

cumulative explained variance
number of components
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
60

PCA in Python

- Can also use this to filter noise:
    - Data will live primarily in the most important components

PCA in Python

- Example:
    - Use some digits from the data set

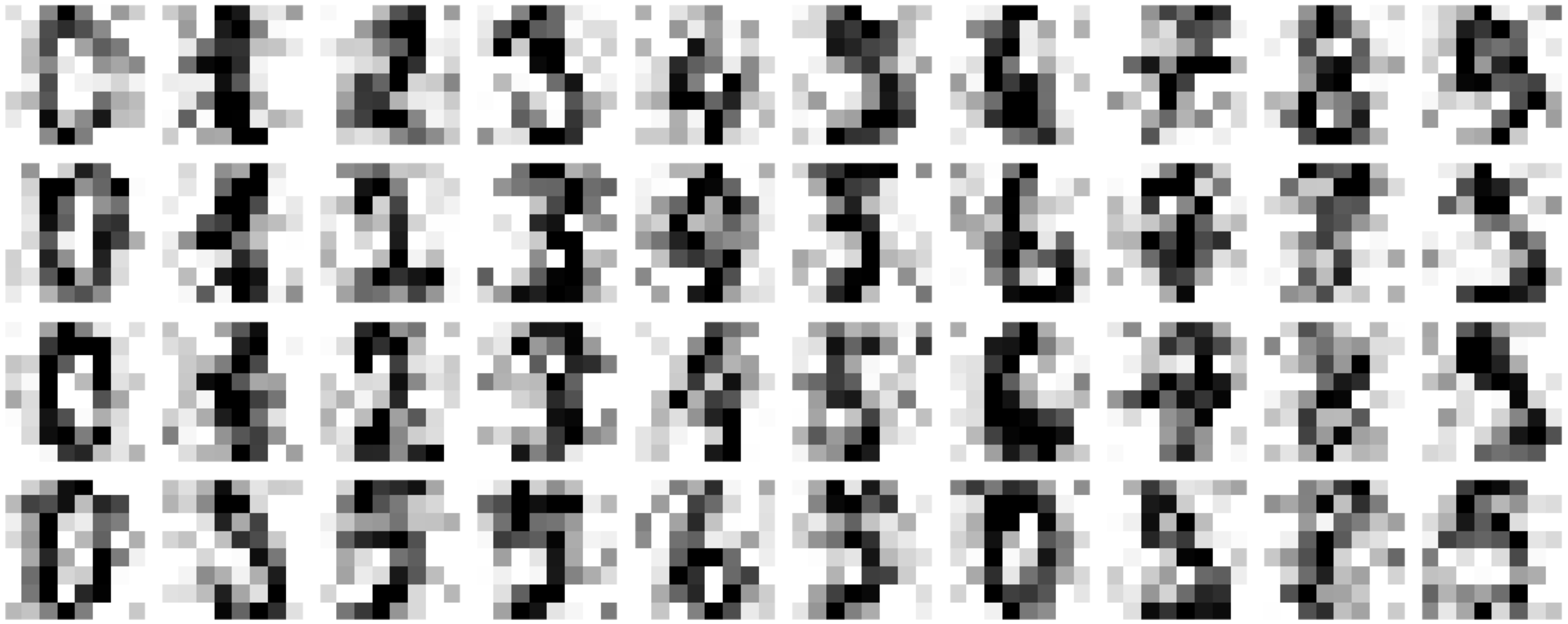


PCA in Python

- Now add some noise

```
np.random.seed(42)
noisy = np.random.normal(digits.data, 4)
plot_digits(noisy)
```

PCA in Python



PCA in Python

- Take the noisy set
  - Use enough components to obtain 50% explained variance

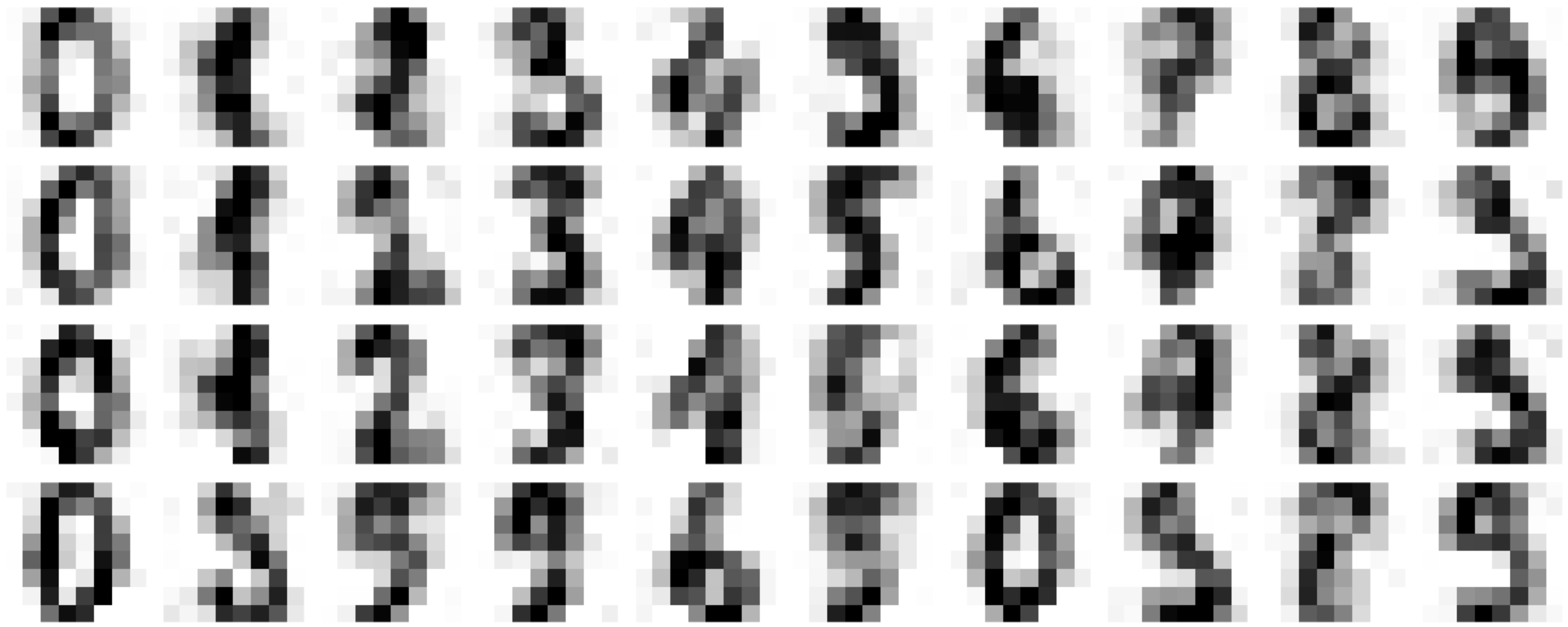
```
pca = PCA(0.50).fit(noisy)
print(pca.n_components_)
```

  - Need 12 components in this case

PCA in Python

- Then display the data of only the highest 12 components

```
components = pca.transform(noisy)
filtered = pca.inverse_transform(components)
plot_digits(filtered)

plt.show()
```



PCA : Eigenfaces

- There is a set of faces of important people in sklearn

```
from sklearn.datasets import fetch_lfw_people
sns.set()


faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld'
'George W Bush' 'Gerhard Schroeder' 'Hugo Chavez'
'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

PCA : Eigenfaces

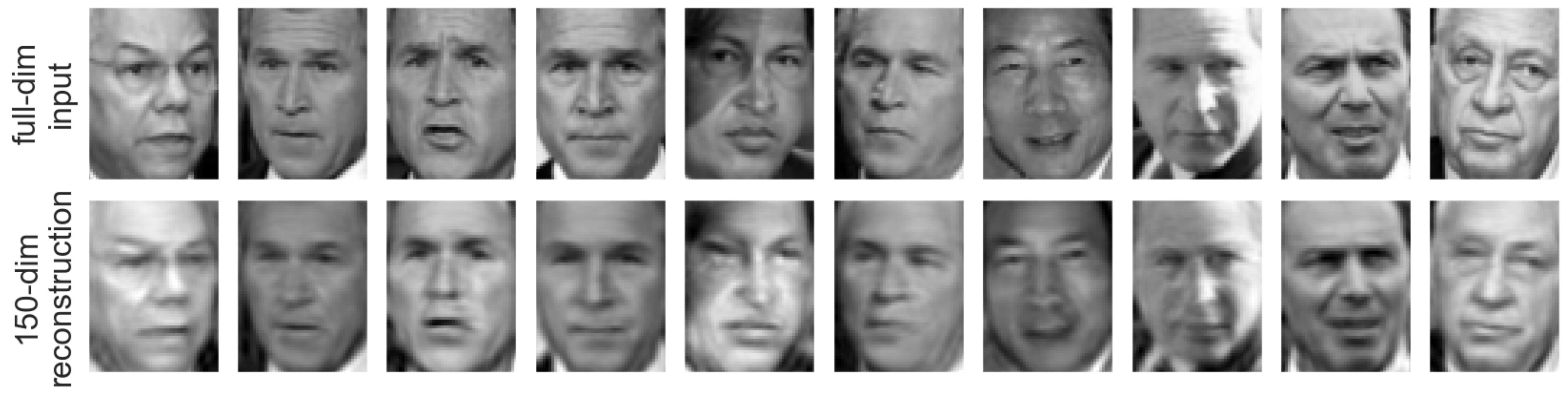
- There is a randomized version of PCA that approximates
  - This is necessary because of the size of the data set

```
pca = PCA(n_components=150,
          svd_solver = 'randomized',
          whiten=True
          )
pca.fit(faces.data)
```

```
pca = PCA(n_components=150, svd_solver = 'randomized',
whiten=True)
pca.fit(faces.data)
components = pca.transform(faces.data)
projected = pca.inverse_transform(components)

fig, ax = plt.subplots(2, 10, figsize=(10, 2.5),
    subplot_kw={'xticks':[], 'yticks':[]},
    gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    ax[0, i].imshow(faces.data[i].reshape(62, 47),
cmap='binary_r')
    ax[1, i].imshow(projected[i].reshape(62, 47),
cmap='binary_r')

ax[0, 0].set_ylabel('full-dim\ninput')
ax[1, 0].set_ylabel('150-dim\nreconstruction');

plt.show()
```

PCA : Eigenfaces

- With about 150 components, the features of the faces are retained



Linear Discriminant Analysis

- Idea:
    - Estimate mean and variance for each category
    - Assumes same covariances
    - Calculates (like PCA) an affine transformation

Linear Discriminant Analysis

- Import LDA:

```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA
```

- Read data & divide

```
iris = pd.read_csv('Iris.csv',
index_col=0).drop(columns='Species')
X_train, X_test, y_train, y_test = train_test_split(
                                   iris,
                                   50*[0]+50*[1]+50*[2],
                                   test_size=0.2,
                                   random_state=0)
```

Linear Discriminant Analysis

- Reset

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- Train with two dimensions:

```
lda = LDA(n_components=2)
lda.fit(X_train, y_train)

for i in range(len(X_test)):
    print(lda.predict([X_test[i]])[0], y_test[i])
```

Linear Discriminant Analysis

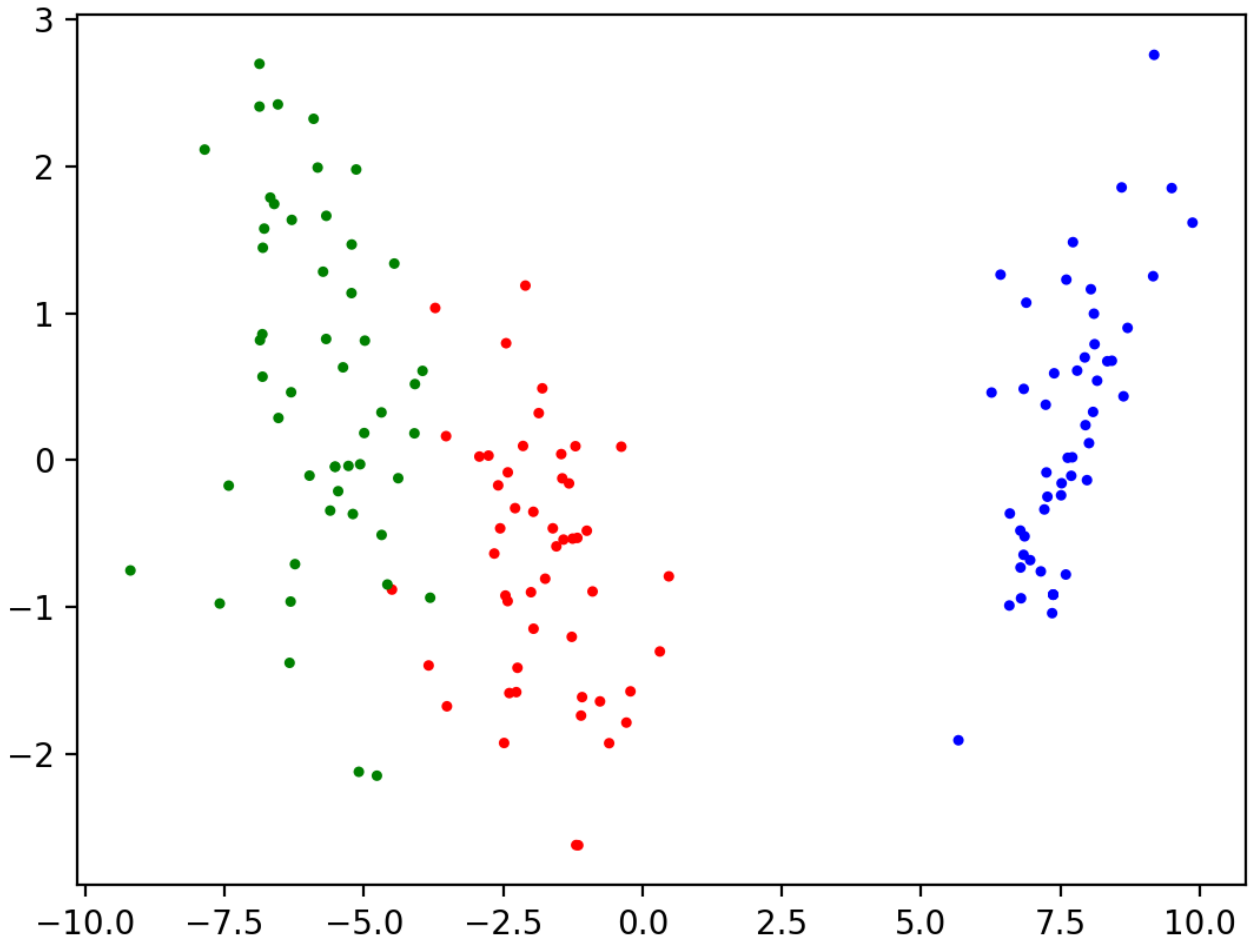
- Results is 100%

Linear Discriminant Analysis

- Show transformation for LDA:

```
transX = lda.fit_transform(iris, 50*[0]+50*[1]+50*[2])

cmap = colors.ListedColormap(['b','r','g'])
plt.scatter(transX[:, 0], transX[:, 1], s=3,
            c=50*[0]+50*[1]+50*[2], cmap = cmap )
plt.show()
```

Linear Discriminant Analysis



Final Example

- Kaggle has a penguins data set on three types of penguins on three islands in the antarctic ocean
- After downloading
  - Read data into a Pandas Dataframe
    - Need to get rid of NA columns

```
penguins_df = pd.read_csv('../SVM/penguins.csv')
penguins_df.dropna(axis=0, how='any',inplace=True)
```

Final Example

- Let's prepare the data for SVM
    - Take 'Adelie' and 'Chinstrap' as the target categories
        - Restrict to only those data

```
penguins_df = penguins_df.loc[
   penguins_df['species'].isin(['Adelie','Chinstrap'])]
```

- Get the labels from the species column

```
labels = np.array(penguins_df['species'])
```

- Restrict to numerical columns

```
penguins_df=penguins_df[['bill_length_mm',
'bill_depth_mm', 'flipper_length_mm',
'body_mass_g']].astype(float)
```

Final Example

- Make the labels numerical

```
labels[labels=='Adelie']=0
labels[labels=='Chinstrap']=1
labels = labels.astype(int)
```

- And make the features into a numpy array

```
features=np.array(penguins_df)
```

Final Example

- Create training and test data (70% / 30% split)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    features, labels, test_size=0.3)
```

Final Example

- Use SVM (better vary C)

```
clf = svm.SVC(kernel='linear', C=0.5)
clf.fit(X_train, y_train)
```

- Determine accuracy

```
from sklearn import metrics

y_pred = clf.predict(X_test)
print("Accuracy:",  metrics.accuracy_score(y_test,
y_pred))
print(clf.coef_)
```

Final Example

- Result varies between 95% and 100% accuracy based on values for C
    - C=0.5 gives the best results

Final Example

- Principal component analysis
    - PCA only affects the features
    - Vary the dimensions

```
from sklearn.decomposition import PCA

pca = PCA(n_components=4)
pca.fit(features)
```

Final Example

- Now print out the results

```
pca_df = pd.DataFrame(pca.components_,
             columns=list(penguins_df.columns))

print(pca.components_)
print(pca.explained_variance_)
print(pca_df)
```

Final Example

- The data frame results (for N=4):

```
   bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g
0        0.004003      -0.001154           0.015195     0.999876
1       -0.319278       0.086848          -0.943542     0.015717
2        0.941265       0.144495          -0.305190     0.001036
3       -0.109847       0.985686           0.127891    -0.000366
```

- These results show that PCA selects basically the coordinates
    - And body_mass followed by flipper-length are the most important components

Final Example

- Linear Discriminant Analysis

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- Train

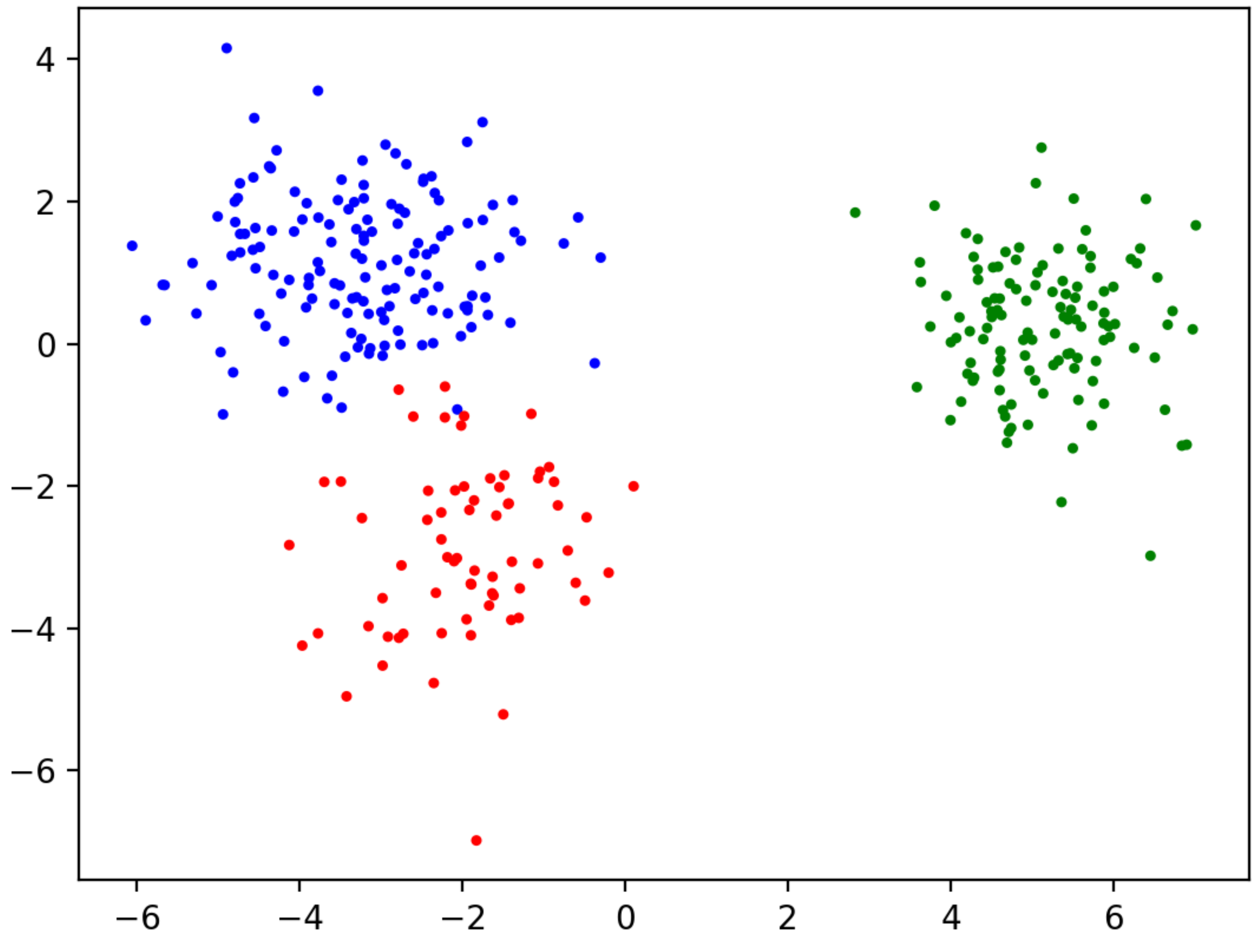
```
lda = LDA(n_components=2)
lda.fit(X_train, y_train)
```

Final Example

- Display

```
transX = lda.fit_transform(features, labels)

cmap = colors.ListedColormap(['b','r','g'])
plt.scatter(transX[:, 0], transX[:, 1], s=5,
            c=labels, cmap = cmap )
plt.show()
```

Final Example



4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6