Applied Machine Learning
Week 6

Grant Schoenebeck

Some material courtesy of Andrew W. Moore. See http://www.cs.cmu.edu/~awm/tutorials.html

Outline

- Boosting
- Naïve Bayes
- Learning in High Dimensions
- Pipelines
- Unsupervised Learning: Density Estimation
- Final Project
- Kaggle Competition
- (Clustering: Impossibility)

Boosting

Boostings

- **Very high level idea:**
  - *Learn the residuals.*
    - *Learn* f *on* (X_i, Y_i)*;*
    - *let* $R_i = Y_i - f(X_i)$ *learn* f' *on* (X_i, R_i)
    - *let* $R'_i = Y_i - f(R_i)$ *learn* f'' *on* (X_i, R'_i)
    - *Etc.*
    - *Return* $f + f' + f''$
  - *Is this a good idea for linear regression?'*

Ada Boostings

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$.
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

Adaboost

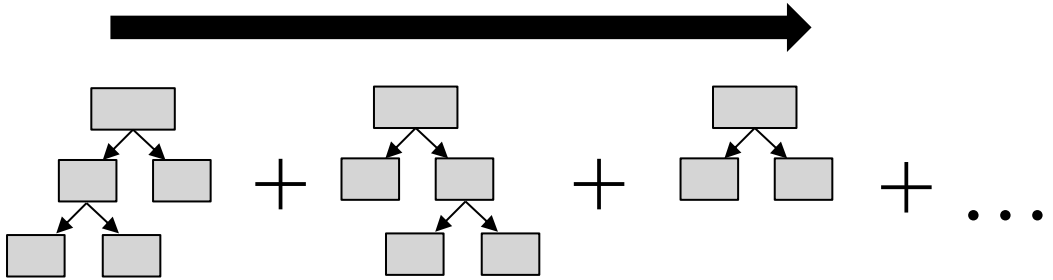
- Morally $D_t(i) \propto exp(L_{t-1}(i))$
- where $L_{t-1}(i) = \sum_{j=1}^{t-1} h_j(i) \neq Y_i$
- can be applied out of the box
- Can use gradient version: go toward new classifier a certain distance.

Gradient Boosted Trees

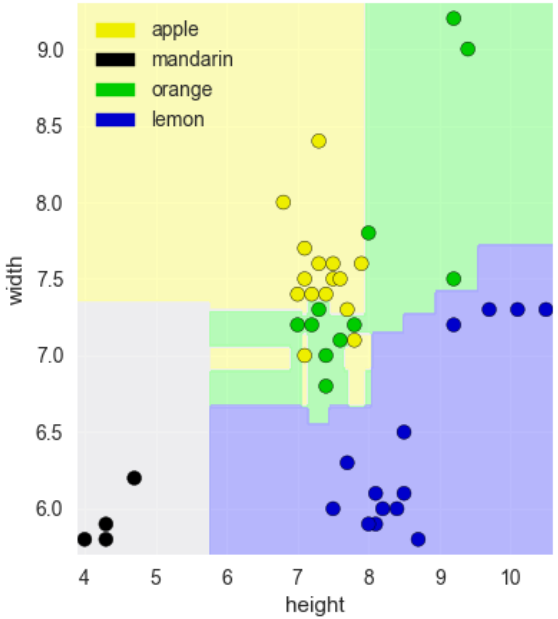
- **Algorithm:**
  - *Start with* $R_i^0 = Y_i - E[Y_i]$
  - *Learn* f^t *on* (X_i, R_i^t)
  - *Let* $R_i^{t+1} = R_i^t - \alpha f^t(X_i)$
  - *Return* $F(X_i) = E[Y_i] + \alpha \sum_t f^t(X_i)$
- **Parameters:**
  - α = *stepsize*
  - *T = number of rounds*
  - *Weak learner*

Gradient Boosted Decision Trees (GBDT)

- Training builds a series of small decision trees.
- Each tree attempts to correct errors from the previous stage.



- The learning rate controls how hard each new tree tries to correct remaining mistakes from previous round.
  - High learning rate: more complex trees
  - Low learning rate: simpler trees



GBDT decision regions on two-feature fruits dataset

GBDT: GradientBoostingClassifier
Key Parameters

- `n_estimators` and `learning_rate` interact: doubling one and halving the other should lead to little change.
- First set `n_estimators` based on performance
- Can use bagging (subsampling) as well.
- `max_depth` is typically set to a small value (e.g. 3-5) for most applications.

GBDT: Pros and Cons

Pros:

- Good accuracy
- Prediction is fast.
- Minimal but some tuning
- Handles a mixture of feature types.

Cons:

- Hard to interpret.
- learning rate and number estimators must be tuned
- Training requires significant computation.
- Not good for very high dimensional sparse features.

XGBoost

- **Similar to Gradient Boosting but:**
  - *Tree weights are regularized*
  - *Other performance optimizations*

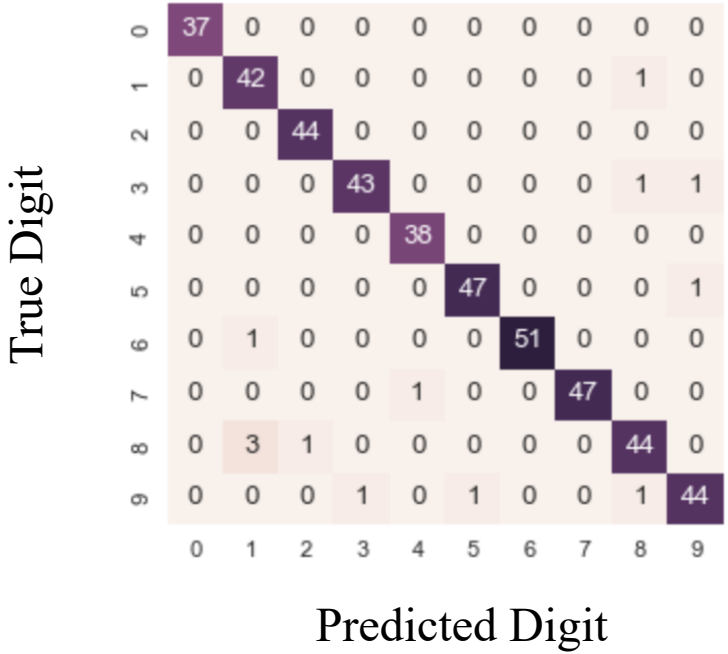
Multi-Class Logistic Regression

- **Two models:**
  - *One versus Rest*
    - *Computes X versus not X separately*
    - *Implemented by SciKitLearn*
  - *One versus One*
    - *Computes X versus Y for all Y. Takes most likely X by summing over all outputs.*
  - *Often a matter of performance?*
    - *Which is faster?*
- **Some classifiers extend to multiple classes naturally:**
  - *k-NN, decision trees*

Multi-Class Evaluation

- Confusion matrices are especially useful
  - *Classification report*
- Multi-label classification: each instance can have multiple labels
  - *Can train on each label independently*
- Overall evaluation metrics are averages across classes
  - *How to average?*
  - *The size of classes is important*

Multi-Class Confusion Matrix



Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

Micro-average:

- Each instance has equal weight.
- Largest classes have most influence

1. Aggregate outcomes across all classes
2. Compute metric with aggregate outcomes

Micro-average accuracy:
4 / 9 = **0.44**

Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

Macro-average:

- Each class has equal weight.

1. Compute metric within each class
2. Average resulting metrics across classes

Class	Accuracy
orange	1/5 = 0.20
lemon	1/2 = 0.50
apple	2/2 = 1.00

Macro-average accuracy:
(0.20 + 0.50 + 1.00) / 3 = **0.57**

Macro-Average vs Micro-Average

- **Classes all same size?**
- **If some larger classes:**
  - *Weight toward largest classes, use micro-averaging.*
  - *Weight toward smaller classes, use macro-averaging.*
- **micro-average << macro-average examine performance on large classes.**
- **macro-average << micro-average examine performance on small classes.**

Naïve Bayes Classifiers

Naïve Bayes Classifiers

- Highly efficient learning and prediction.
- Make a strong assumption that $Pr[X|Y] = \prod_i Pr[X^i|Y]$
  - *When might this be true?*
  - *When might this fail?*
- Examples:
  - *Words in spam*
  - *Words in topic "Fisheries"*
  - *Peer grading scores*

Naïve Bayes Classifiers: a simple, probabilistic classifier family

- Assume that features are conditionally independent.
- $Pr[Y|X] = Pr[X|Y]\frac{Pr[Y]}{Pr[X]} = \prod_i Pr\left[X^i|Y\right]\frac{Pr[Y]}{Pr[X]} \propto \prod_i Pr\left[X^i|Y\right] Pr[Y]$
- Use that $Pr[X|Y] = \prod_i Pr\left[X^i|Y\right]$
  - *When might this be true?*
  - *When might this fail?*
- $f(Y|X) \equiv \prod_i Pr\left[X^i|Y\right] Pr[Y]$ unnormalized posterior
- $Pr[Y|X] = \frac{f(Y|X)}{f(Y|X)+f(\neg Y|X)}$
- So just need to know likelihood of X^i in Y versus $\neg$Y

Naïve Bayes Classifiers: a simple, probabilistic classifier family

- $\frac{Pr[Y|X]}{Pr[\neg Y|X]} \propto \prod_i \frac{Pr[X^i|Y]}{Pr[X^i|\neg Y]} \frac{Pr[Y]}{Pr[\neg Y]}$
- So just need to know likelihood of X^i‘s in Y versus ¬Y

Features

- What does $\frac{Pr[X^i|Y]}{Pr[X^i|Y']}$ mean?
  - *How much more likely is word X^i to appear in spam versus non-spam?*
  - *How likely is a paper to be about fisheries compared to AML if it has the word "net" in it?*
  - *How much more likely is a movie to be rated a 3 stars by Ebert if it is good versus if it is bad?*
- $Pr[X^i|Y]$ is
  - *Number of times "Free" is in a spam email/number of words in an email.*
  - *Typically regularized in SciKit Learn:*

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Example

X_1	X_2	Y	Count
0	0	0	84
1	0	0	5
0	1	0	10
1	1	0	1
0	0	1	67
1	0	1	30
0	1	1	2
1	1	1	1

X_1 = does "free" appear
X_2 = does "yesterday" appear
Y = IsSpam

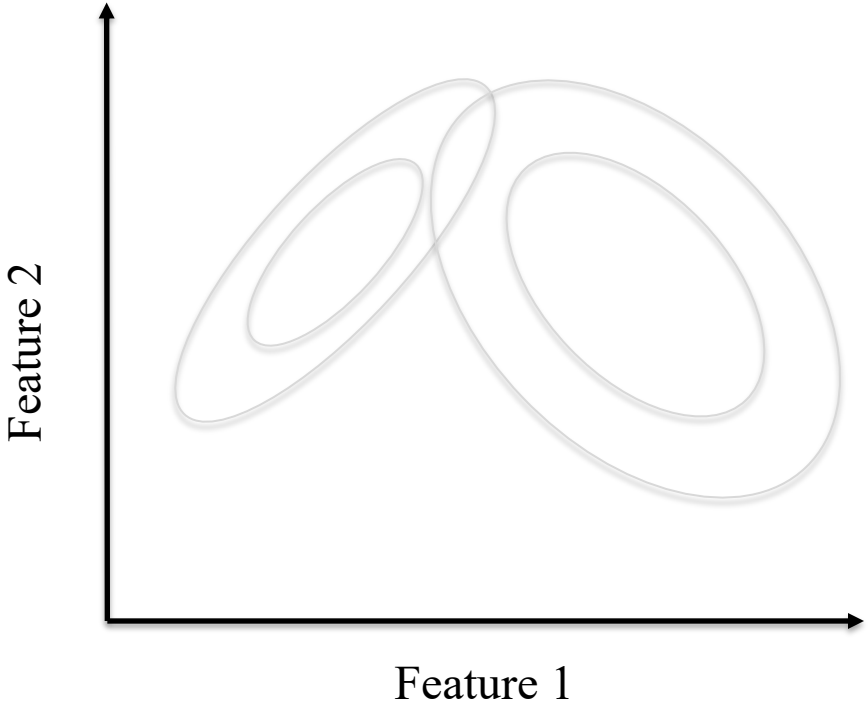
Questions:
1) What is Pr[Y = 1| X = (1, 1)] ?

2) Using Naïve Bayes, what is
Pr[Y = 1| X = (1, 1)] ?

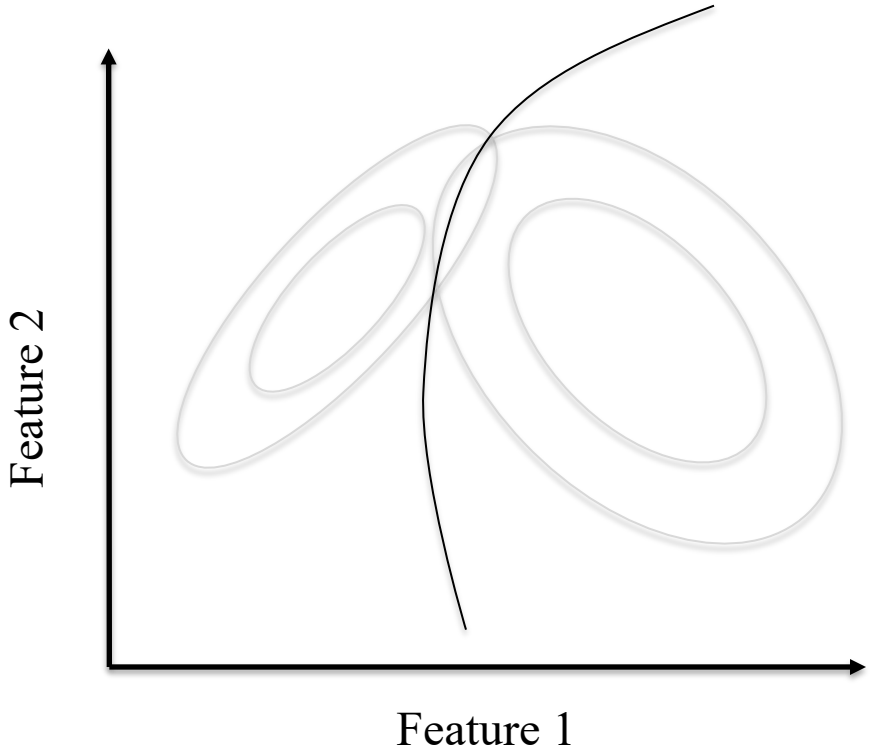
Naïve Bayes classifier types

- **Bernoulli: binary features (e.g. word presence/absence)**
- **Multinomial: discrete features (e.g. word counts)**
- **Gaussian: continuous/real-valued features**
  - *Statistics computed for each class:*
    - *For each feature: mean, standard deviation*

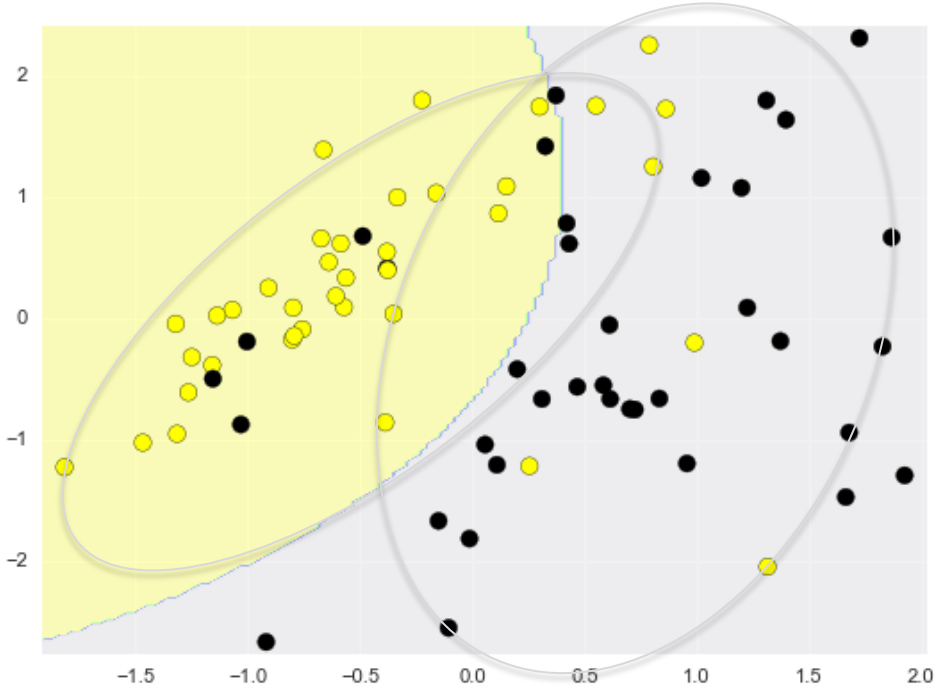
Gaussian Naïve Bayes classifier



Gaussian Naïve Bayes classifier



Gaussian Naïve Bayes classifier



Naïve Bayes Classifiers

- **Parameters:**
  - *Smoothing parameter α (in multinomial)*
- **Evaluation:**
  - *Probabilities are meaningful*

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Naïve Bayes Classifiers

- Highly efficient
- Typically less generalization than more sophisticated learners.
- A good baseline.
- Can be used with boosting.

Naïve Bayes classifiers: Pros and Cons

Pros:

- Easy to understand
- Simple, efficient parameter estimation
- Works well with high-dimensional data
- Often useful as a baseline comparison against more sophisticated methods

Cons:

- Accuracy is not great
- Confidence estimates for predictions are inaccurate.

Pipelines and Column Transforms

Doing things right

Pipelines

- Can compose transforms together
- Each transform must have .fit and .transform method except the final one just needs .fit
- Useful with cross_val_score and GridSearchCV
  - *Why?*
- Can access parameters of pipelir

Column Transform

- Can apply transform to specified columns
  - *(e.g. One-hot encoding).*

```
param_grid = [
    {'classifier': [SVC()], 'preprocessing': [StandardScaler(), None],
     'classifier__gamma': [0.001, 0.01, 0.1, 1, 10, 100],
     'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100]},
    {'classifier': [RandomForestClassifier(n_estimators=100)],
     'preprocessing': [None], 'classifier__max_features': [1, 2, 3]}]
```

Learning in high dimensions

High Dimensions

Are not intuitive

- If choose two random points in $[0,1]^n$ how close would they usually be?
- If choose 1000 random points in $[0,1]^{100}$ how close would they usually be?
  - *(Good example for Kernel density estimation)*
- How many points would you need so that, on average, the nearest point was of distance ½?

High Dimensions

Are not intuitive

- If choose two random points in $[0,1]^n$ how close would they usually be? $O(\sqrt{n})$
- If choose 1000 random points in $[0,1]^{100}$ how close are the closest 2 points? (about 2.78)
  - *(Good example for Kernel density estimation)*
- How many points would you need so that, on average, the nearest point was of distance ½?
  - *(a lot!) For each point to have another within distance ½, would need ~ 3^100 points.*

Final Project

Final Project

- Groups of 2-3
- Proposal Due Nov 11
  - *Can start now!*
- Poster session Friday Dec 6th from 11-2pm.
- Final report due Monday Dec 10th

Final Project Proposal

- Motivation: what problem are you tackling?
- Methods: What machine learning techniques do you expect to apply?
- Datasets: what dataset(s) do you intend to use? What if any preprocessing will they require?
- Evaluation: what experiments do you intend to run? How is "success" defined for your project?
- Computing: The computing resources you're likely to need.
- If you can provide an example of a prior related project or research on your problem, that would be helpful (to us, and you).

Kaggle Competition

- **Teams:**
  - *Up to 4, no private sharing outside teams.*
- **Timeline:**
  - *Start date: Monday 10/21*
  - *First evaluation: Sunday 11/03 06:59pm*
  - *Final evaluation: Sunday 11/10 06:59pm*
- **Rubric:**
  - *200 points in total*
  - *Successful submission: + 80 points*
  - *Achieve 0.8 accuracy: + 40 points*
  - *Surpass the benchmark on leaderboard: + 20 points*
  - *Ranking grade: + 60 * 2 / $\log_2$(2 + rank)*
    - *The winner team gets 216 points!*
  - *Bonus for top 3 teams at the first evaluation: + 10 points*

Review

Topics

Classifiers

- K-NN
- Logistic regression
- SVM
  - *Linear*
  - *Kernalized*
- Decision Trees
  - *Random Forests*
  - *Gradient Boosted Trees*
- Multiclass
  - *One versus rest/all*
  - *One versus one*

Regressors

- K-NN
- Ordinary Least Squares
  - *Ridge, Lasso, Robust*
- SVM
- Regression Trees
  - *Random Forests*
  - *Gradient Boosted Trees*

For Each Learner

- Intuition for how it works
  - *What types of features will it need*
- Key Parameters
  - *Function*
  - *How to tune*
- Evaluation/Transparency
- Advantages/Disadvantages
  - *When is it likely to do well/poorly*

Concepts

- ML Process
- Data Exploration
- Preprocessing
  - *Feature Expansion/Kernaliziation*
  - *Normalization*
  - *One-hot Encoding*
  - *Dealing with Missing Data*
- Over/Under Fitting
- K-Fold Cross Validation
- Data Leakage
  - *Feature*
  - *Test set*
- Evaluation Metrics
  - *Confusion Matrix,*
  - *Accuracy, Recall, Precision, F-1*
  - *ROC*
  - *RSS*
  - *Feature Importance*
  - *Looking at Residuals*
- Interim/Post Processing
  - *Regularization*
  - *Ensembles*
    - *Boosting*
    - *Bagging*
  - *Calibration*

How to study?

- **1) Review labs and PSs**
  - *Make sure you understand concepts*
  - *Look at book, ask on Piazza, or come to OH if something is confusing.*
- **2) Practice test is like a validation set, don't use up too soon.**
- **3) Review slides:**
  - *More information than you need to know.*

Additional Resources

- **Nearly all these topics are covered in course text: Introduction to Machine Learning with Python**
- **A few exceptions:**
  - *Robust Regression:*
    - *Slides*
  - *Missing data*
    - *covered in other course text: Deep Learning in Python*
  - *Bagging/Boosting*
    - *Slides/book/Wikipedia*

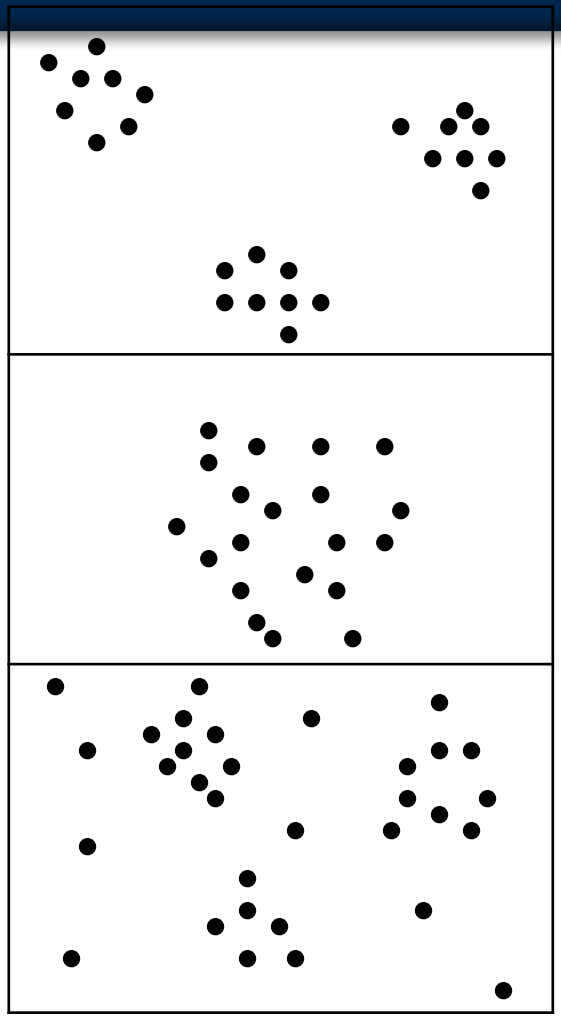
Unsupervised Learning

Introduction: Unsupervised Learning

- Operates on datasets without labels.
- Goal: find interesting structure

Applications:

- Density estimation.
- Visualization.
- Compress and summarize the data.
- Extract features for supervised learning.
- Discover important clusters or outliers.



Sometimes easy

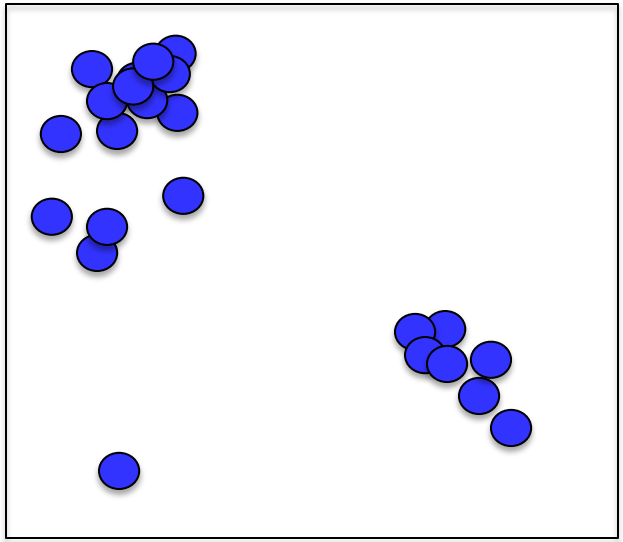
Sometimes impossible

and sometimes
in between

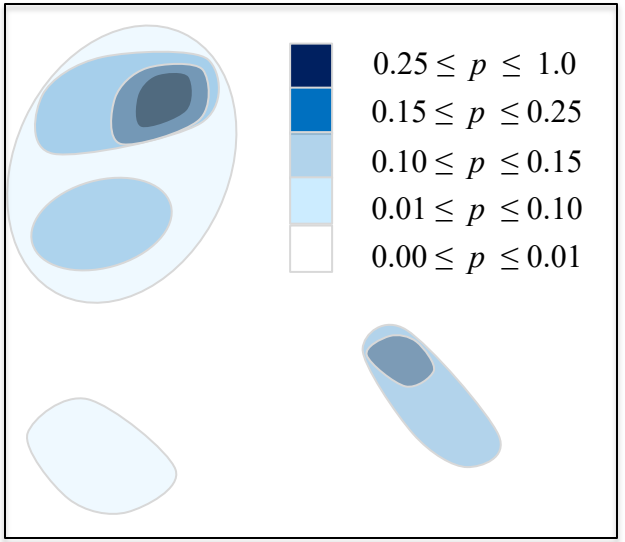
Two major types of unsupervised learning methods

- **Transformations**
  - *Processes that extract or compute information*
- **Clustering**
  - *Find groups in the data*
  - *Assign every point in the dataset to one of the groups*

Transformations: Density Estimation



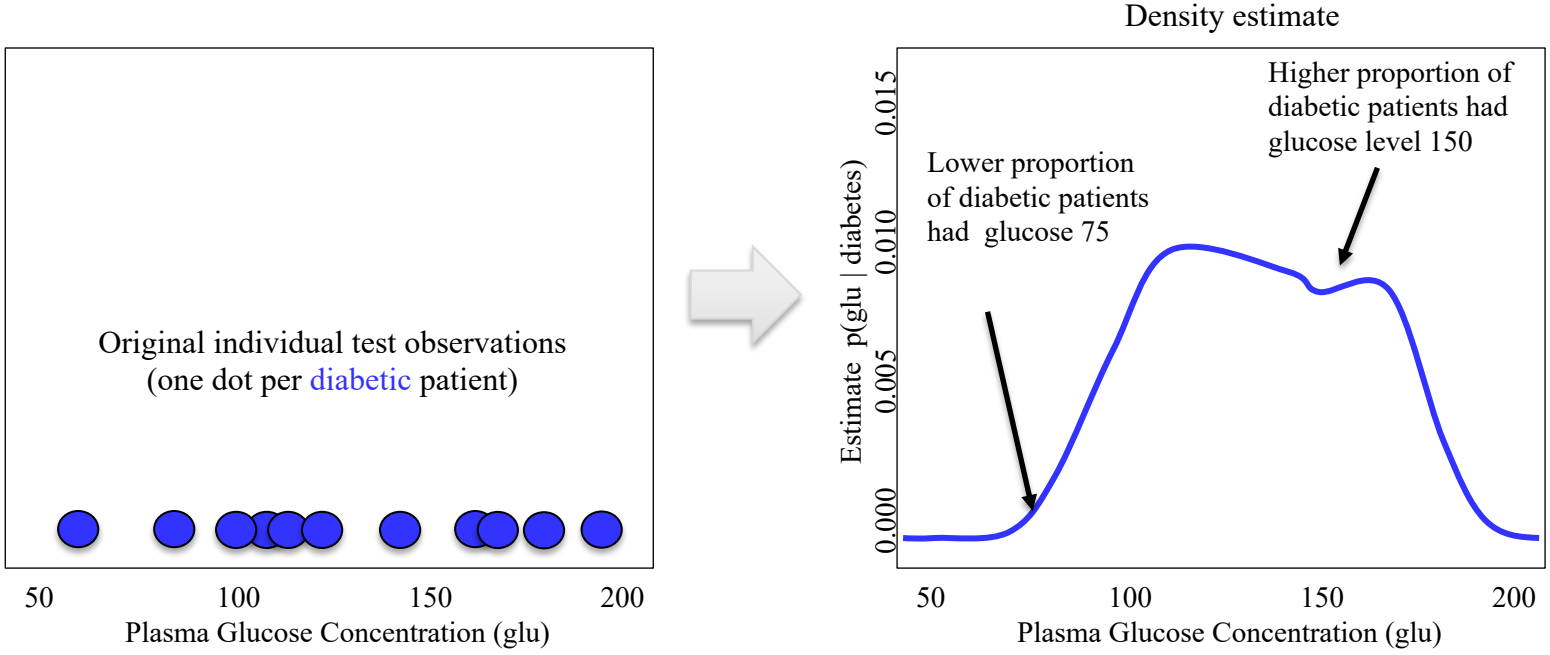
Individual measurements



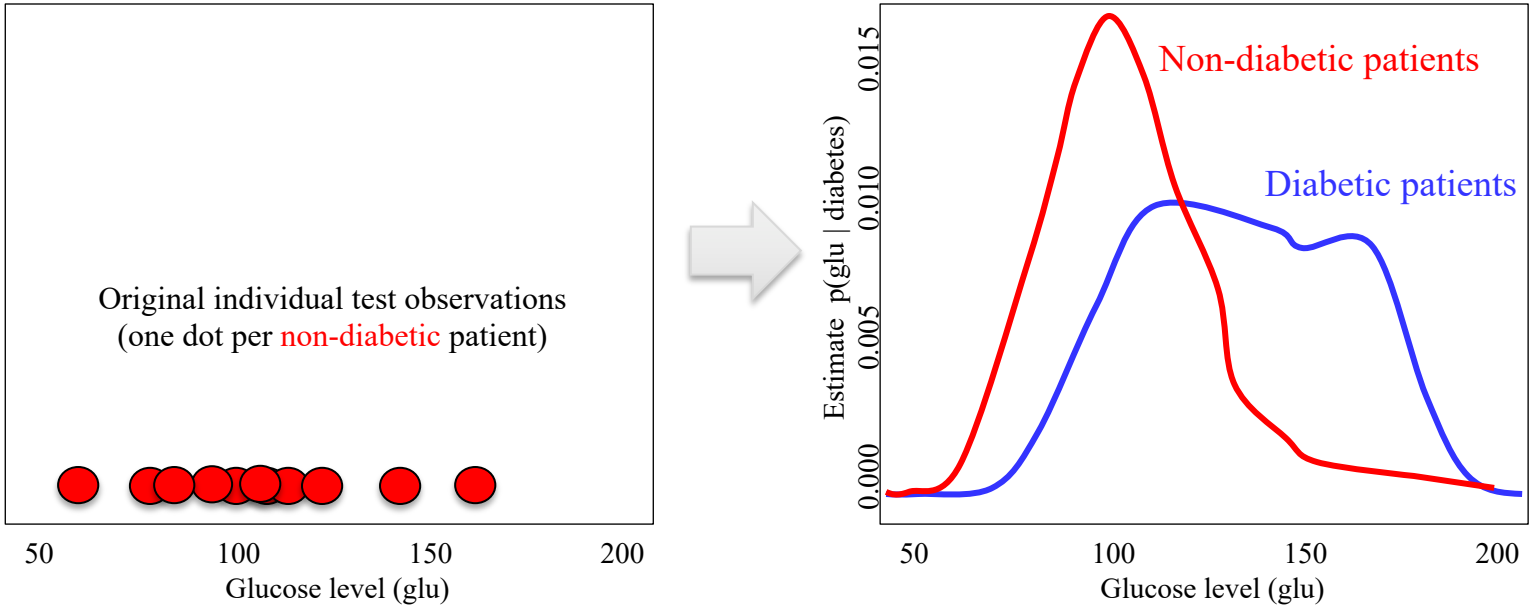
Density estimate

(Estimated probability p of observing a measurement at a given location)

Density Estimation Example



Density Estimation Example

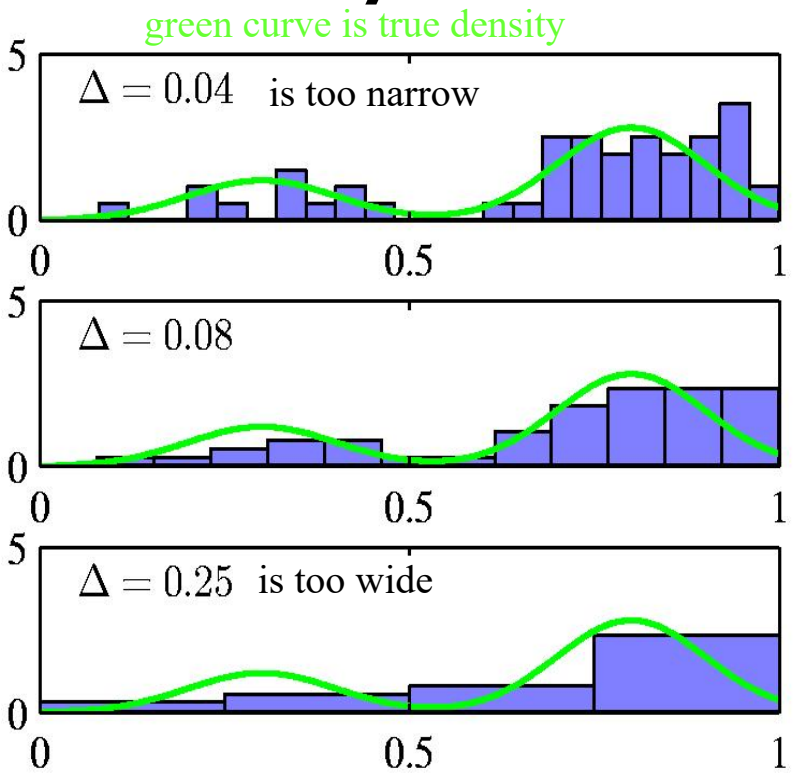


Histograms as density models

Low dimensional data:
use a histogram as a density model.

- **How wide should the bins be? (width=regularizer)**
- **Do we want the same bin-width everywhere?**
- **Do we believe the density is zero for empty bins?**

$$p_i = \frac{n_i}{N \Delta_i}$$



Histogram Pros/Cons

- Simple to compute/interpret
- High-dimensional data is tricky!
  - *The number of bins is exponential in the dimensionality of the dataspace*
  - *Either big bins or lots of zero counts (or adapt local bin-width to the density) ->*
- Discontinuities at the bin boundaries.

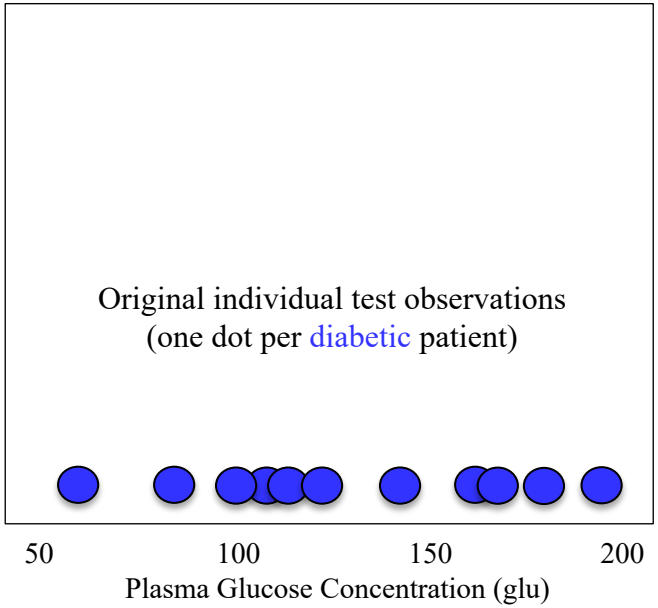
Kernel density estimators

Sum over masses centered at datapoints:

- **Each point contributes density of 1/N**

Example with points $x_1 \ldots x_n$ and Gaussian masses:

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(\frac{\| \mathbf{x} - \mathbf{x}_i \|^2}{2\sigma^2}\right)$$



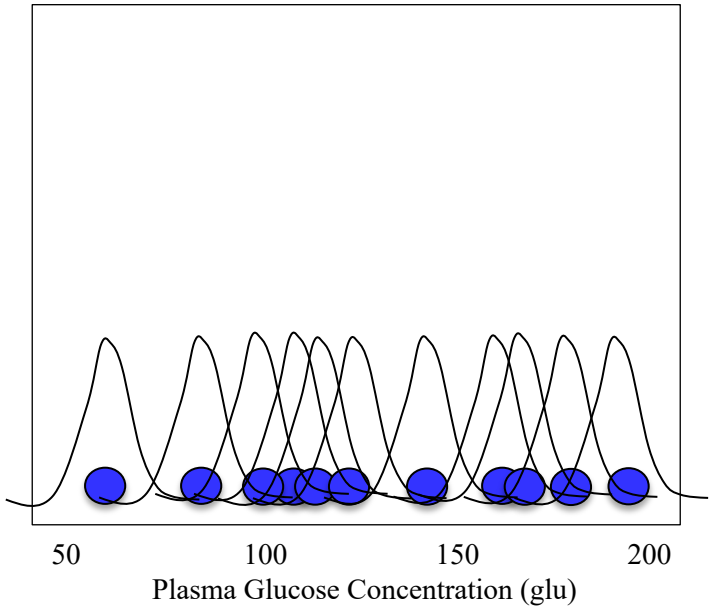
Kernel density estimators

Sum over masses centered at datapoints:

- **Each point contributes density of 1/N**

Example with points $x_1 \ldots x_n$ and Gaussian masses:

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(\frac{\| \mathbf{x} - \mathbf{x}_i \|^2}{2\sigma^2}\right)$$



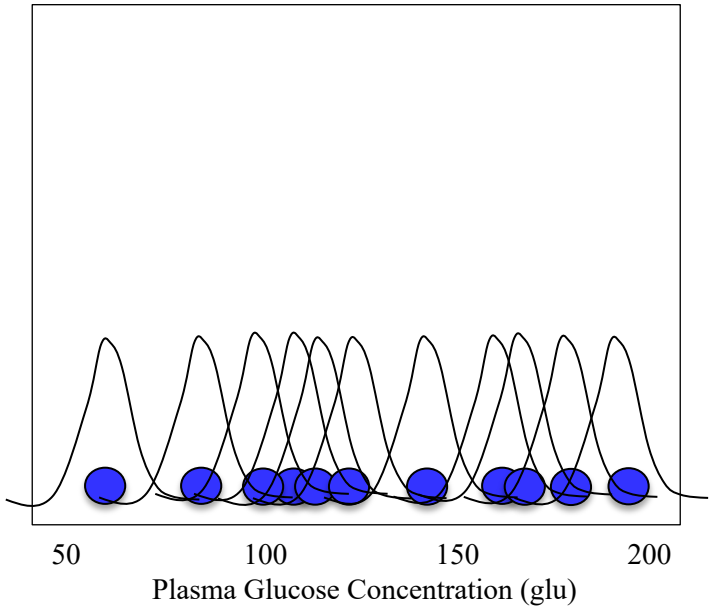
Kernel density estimators

Sum over masses centered at datapoints:

- **Each point contributes density of 1/N**

Example with points $x_1 \ldots x_n$ and Gaussian masses:

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(\frac{\| \mathbf{x} - \mathbf{x}_i \|^2}{2\sigma^2}\right)$$

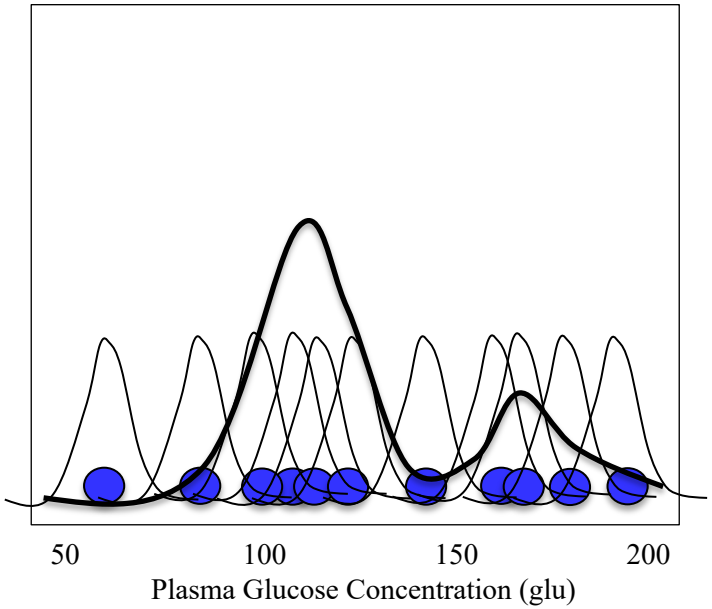


Kernel density estimators

Use regions centered on the datapoints

- Allow the regions to overlap.
- Let each individual region contribute a total density of 1/N
- Use regions with soft edges to avoid discontinuities (e.g. Gaussians)
- For a sample of points $x_1 \ldots x_n$

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(\frac{\| \mathbf{x} - \mathbf{x}_i \|^2}{2\sigma^2}\right)$$



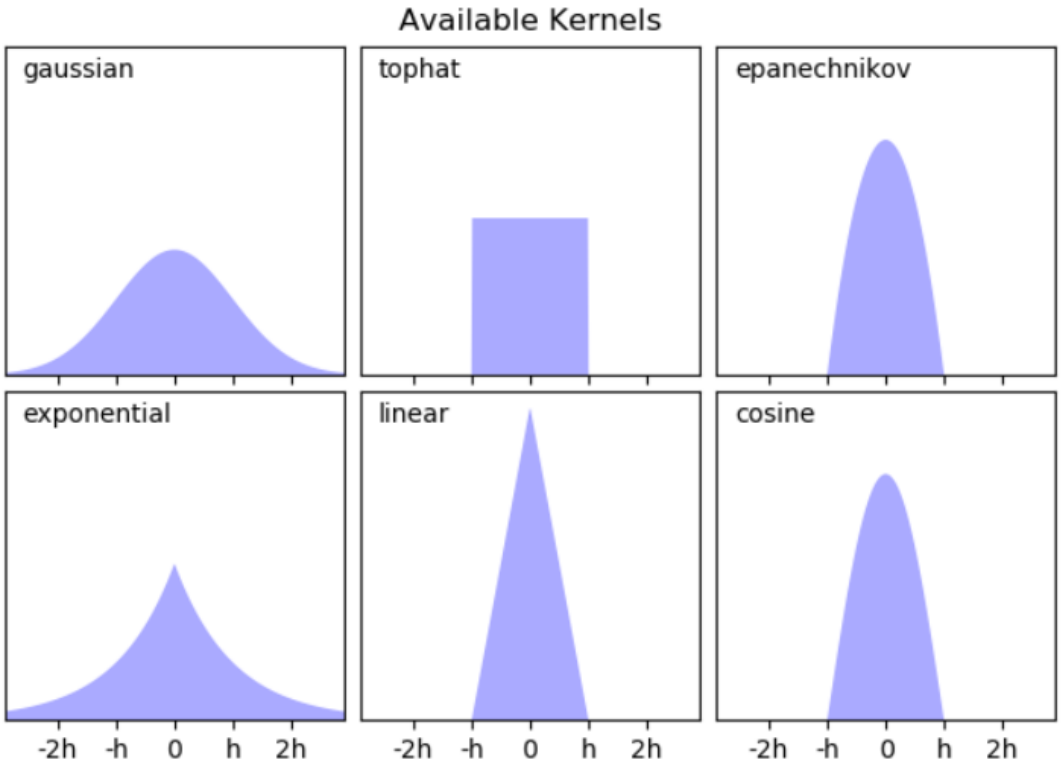
Kernel functions

- A kernel is a positive function K(*x*; *h*) that's controlled by the bandwidth parameter *h*.
- The density estimate at a point *y* within a given group of points $\{x_i\}$ is given by:

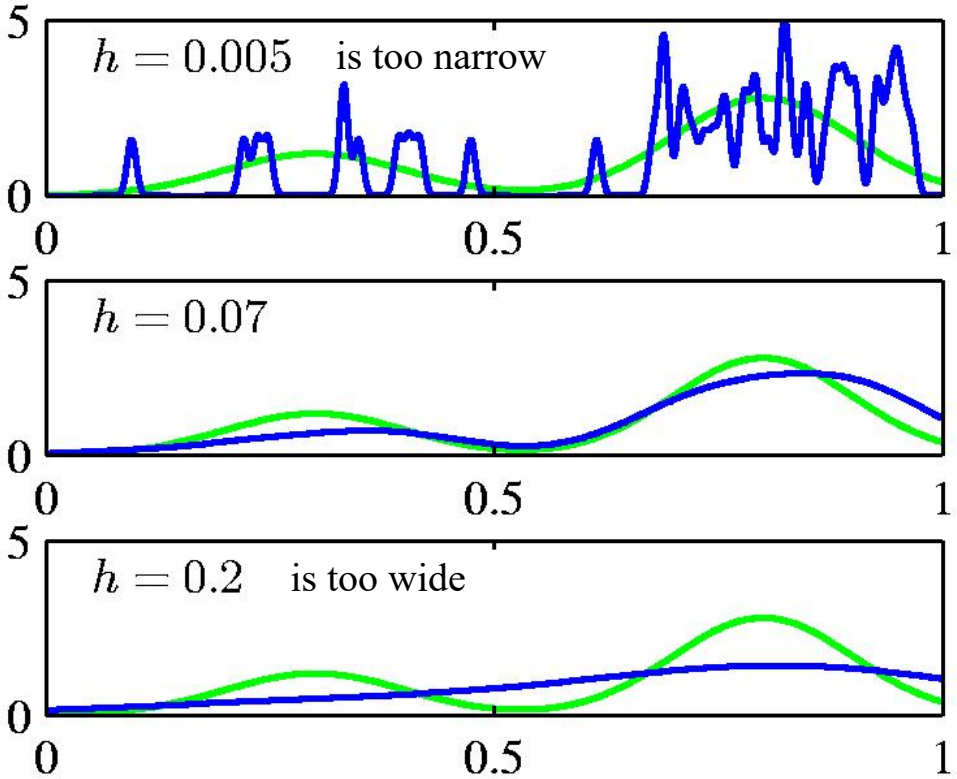
$$\rho_K(y) = \sum_{i=1}^{N} K((y - x_i)/h)$$

- The bandwidth acts as a *smoothing* parameter
  - *LARGE bandwidth -> very smooth (high-bias) density estimator*
  - *small bandwidth -> very bumpy (high-variance) density estimator*

Kernel functions



The density modeled by a kernel density estimator

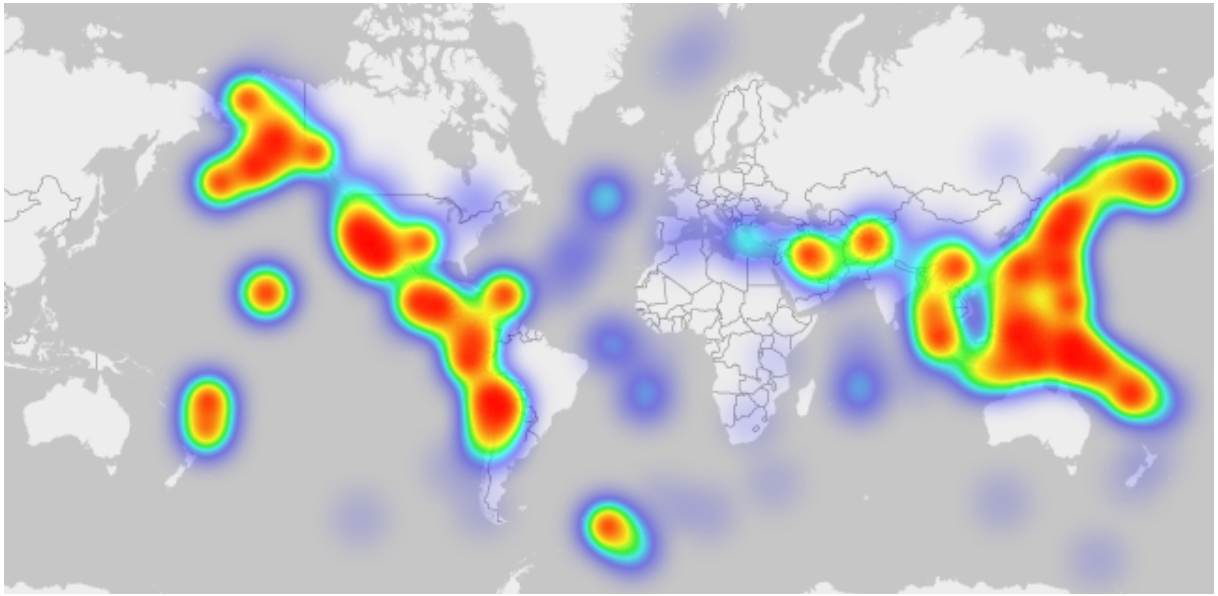


KernelDensity

The `KernelDensity` class in the `sklearn.neighbors` module performs one widely-used form of density estimation. KernelDensity is especially popular for use in creating heatmaps with geospatial data

```
kde = KernelDensity(kernel='gaussian', bandwidth=0.75).fit(X)
```

Kernel Density Example



Recent global earthquake activity (U.S. Geological Survey data)

Source: http://www.digital-geography.com/csv-heatmap-leaflet/

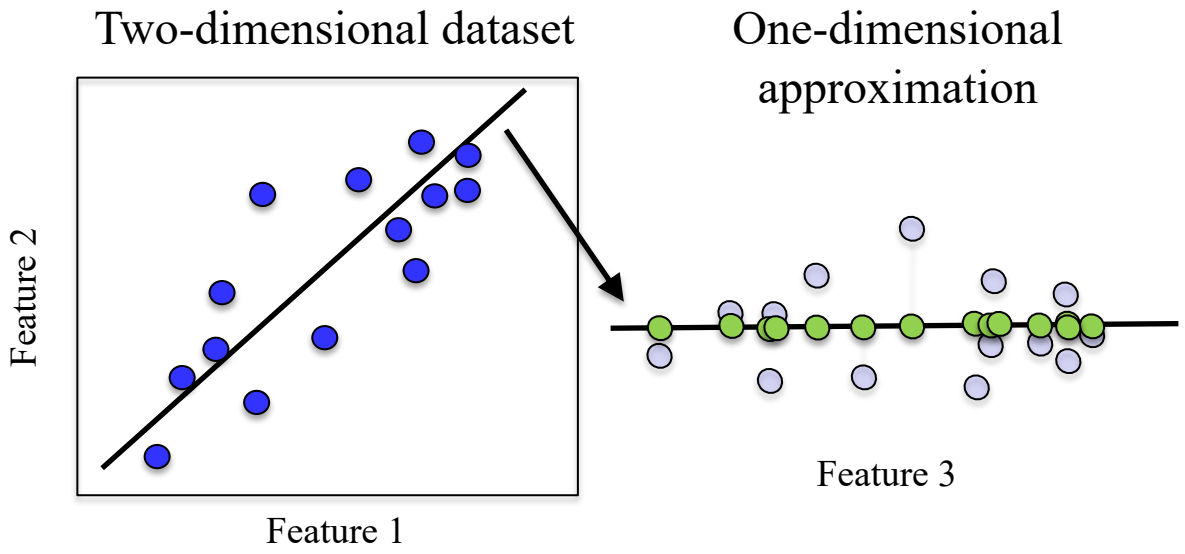
Other density estimation methods

- Gaussian mixture models (sklearn.mixture.GaussianMixture)

Break?

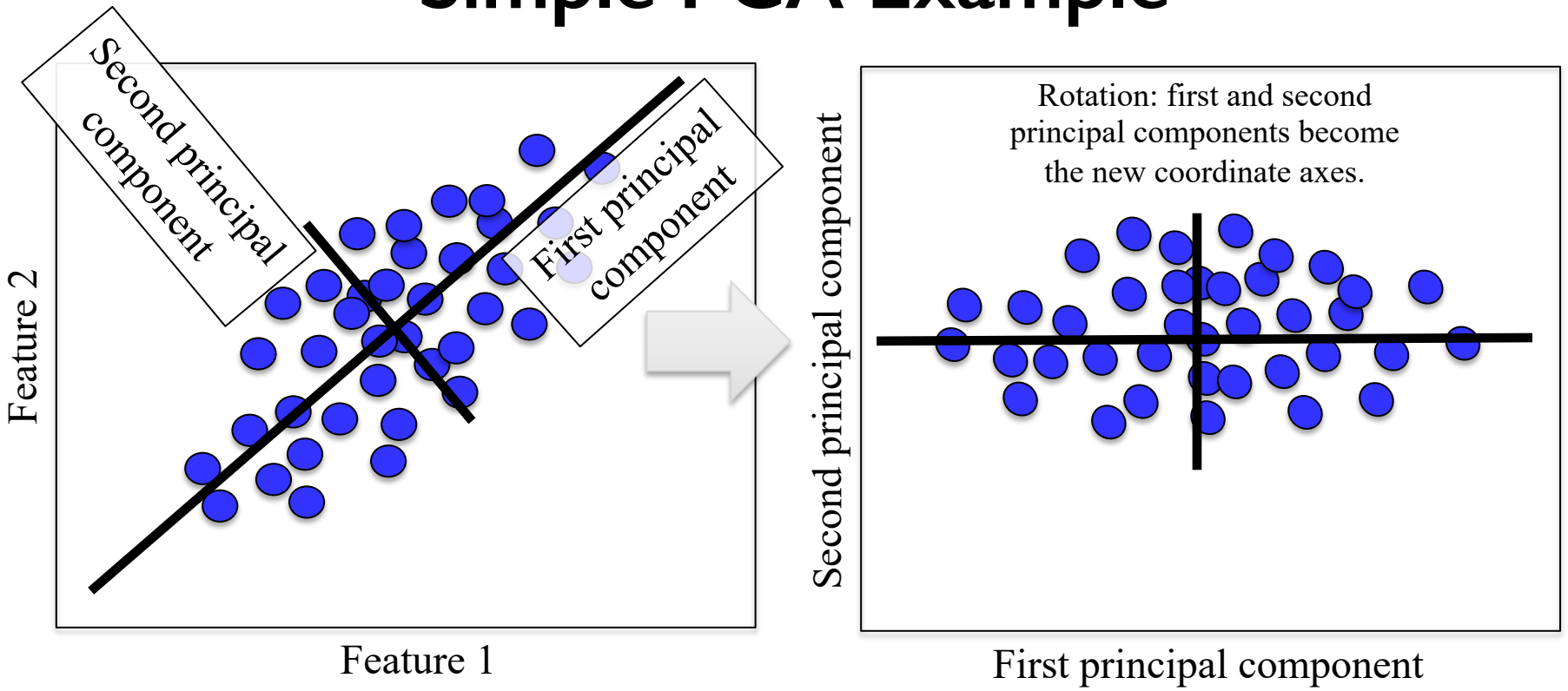
Dimensionality Reduction

- Finds an approximate version of your dataset using fewer features
- Used for exploring and visualizing a dataset to understand grouping or relationships
- Often visualized using a 2-dimensional scatterplot
- Also used for compression, finding features for supervised learning



The one-dimensional approximation is obtained by projecting the original points onto the diagonal line and using their position on that line as the new single feature.

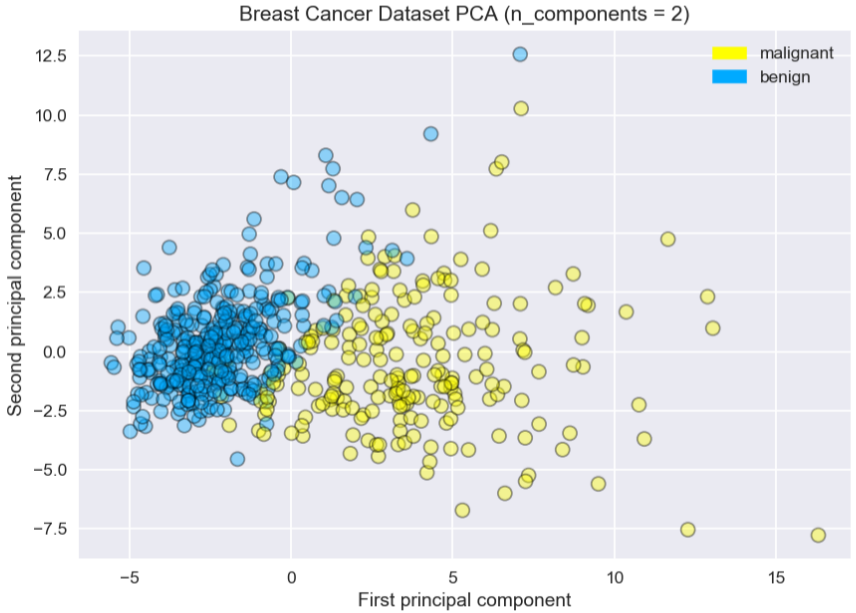
Simple PCA Example



Dimensionality Reduction with PCA in scikit-learn

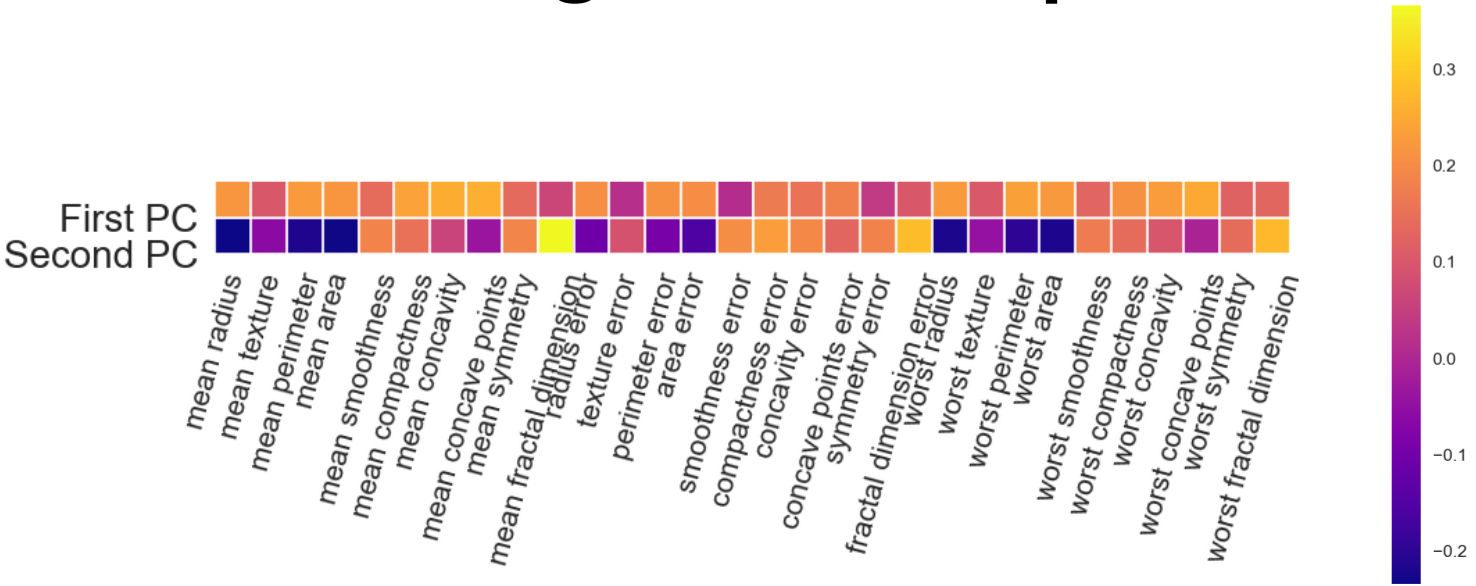
```
# PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)

# each feature should be centered (zero mean) and with unit variance
X_normalized = StandardScaler().fit(X_cancer).transform(X_cancer)

pca = PCA(n_components = 2).fit(X_normalized)

X_pca = pca.transform(X_normalized)
print(X_cancer.shape, X_pca.shape)
```

```
(569, 30) (569, 2)
```

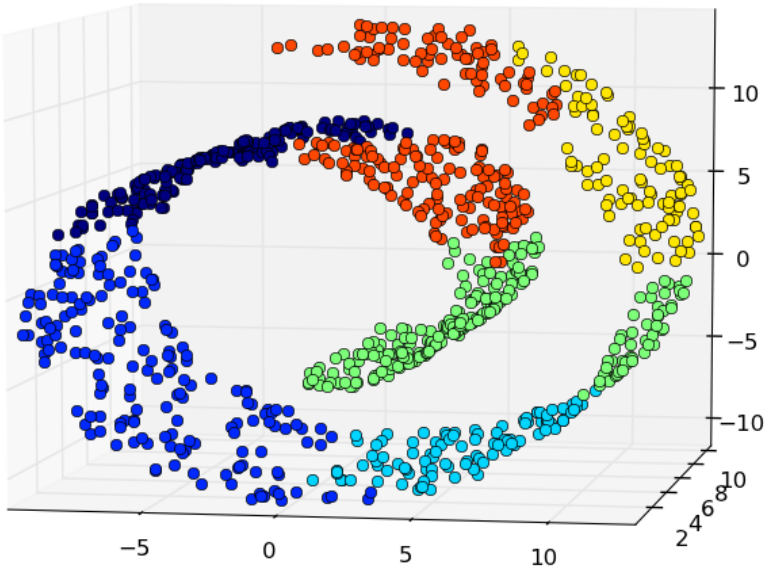



```
from adspy_shared_utilities import plot_labelled_scatter
plot_labelled_scatter(X_pca, y_cancer, ['malignant', 'benign'])

plt.xlabel('First principal component')
plt.ylabel('Second principal component')
plt.title("Breast Cancer Dataset PCA (n_components = 2)")
```

Visualizing PCA Components



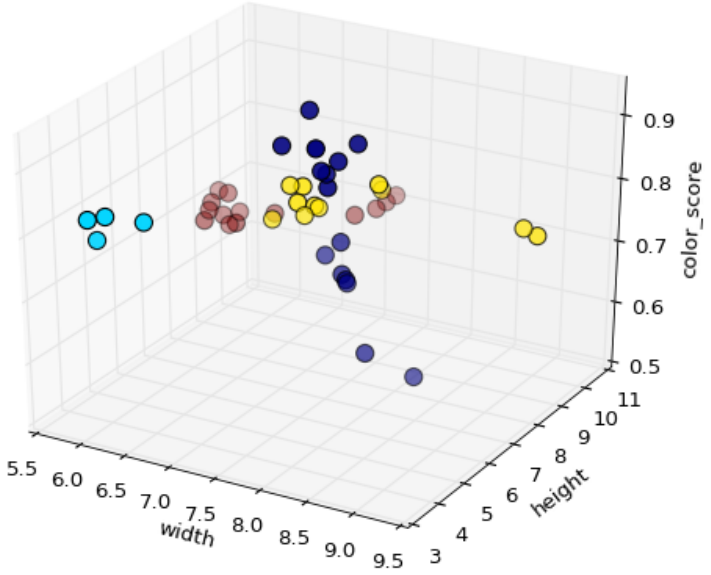
The "Swiss Roll" Dataset



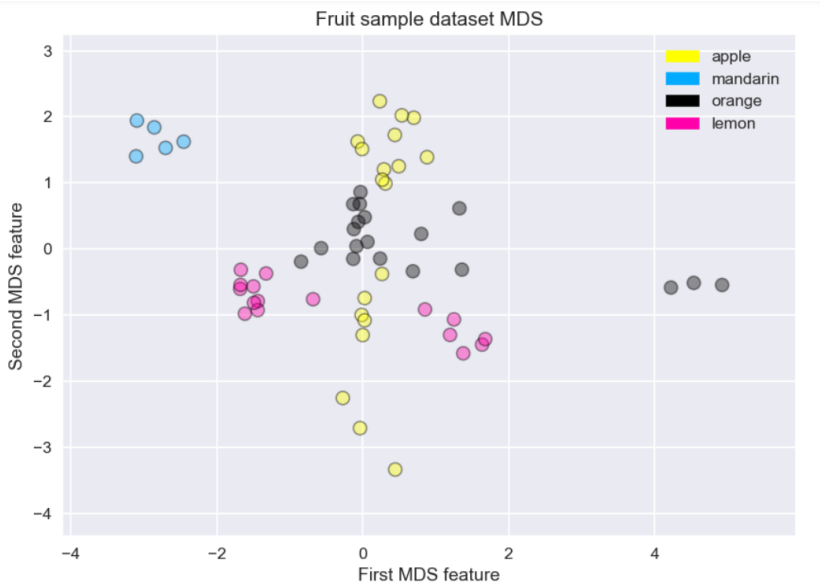

```
sklearn.datasets.make_swiss_roll(n_samples=1500, noise=0.05)
```

See: http://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering

Multidimensional scaling (MDS) attempts to find a distance-preserving low-dimensional projection



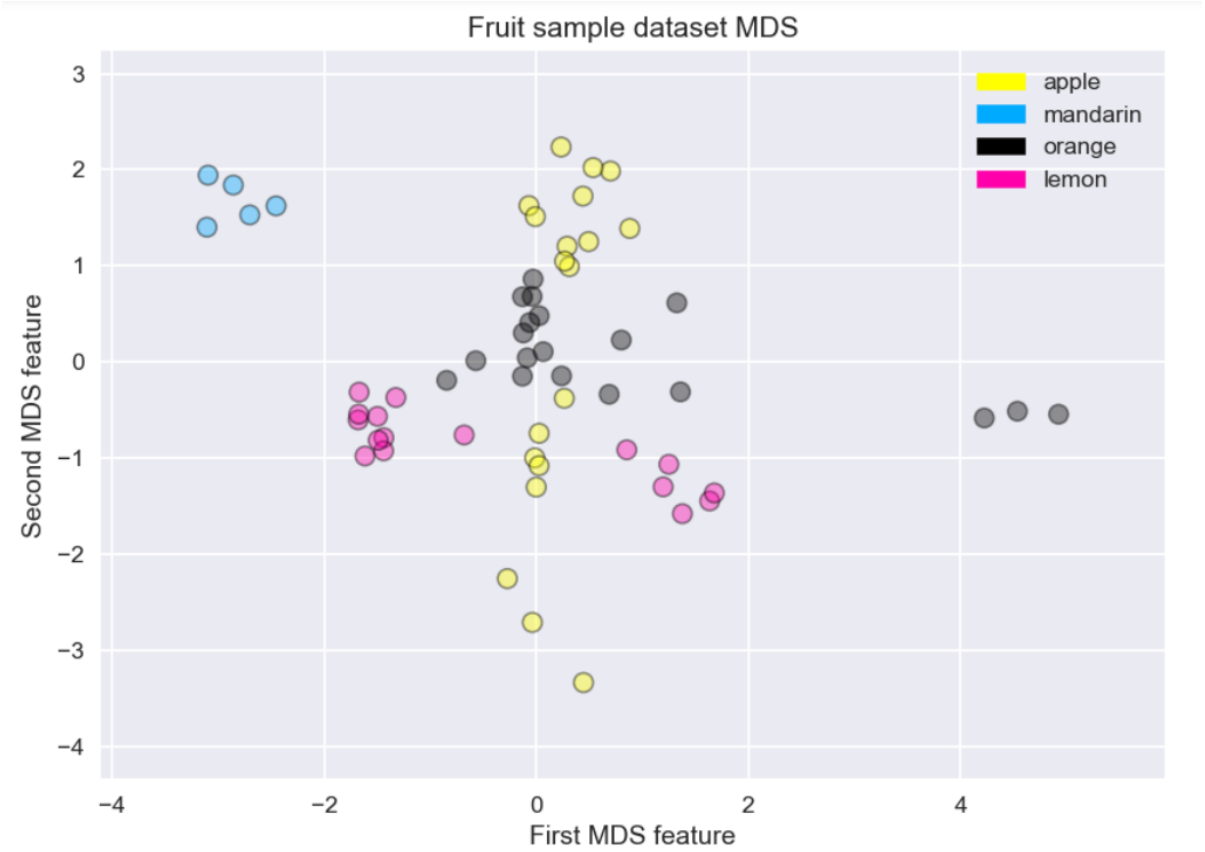
High-dimensional dataset



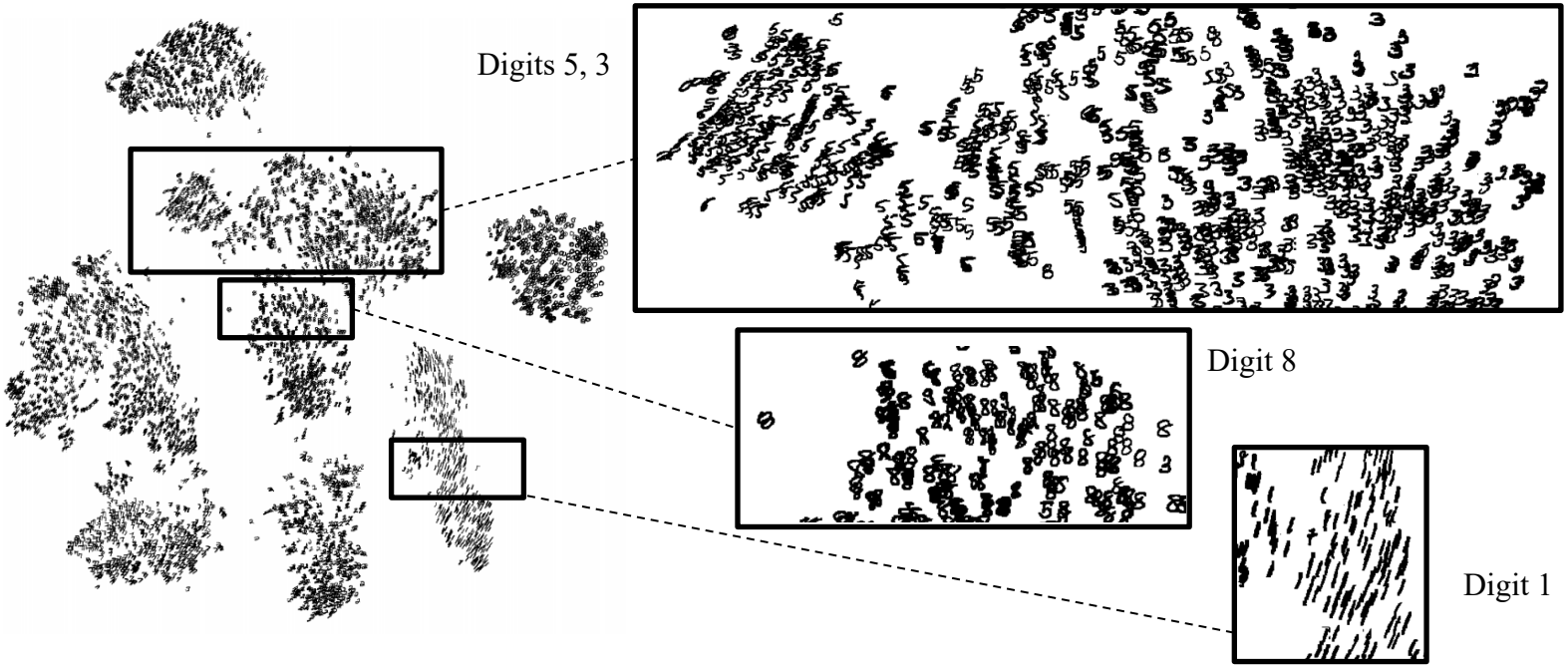
Two-dimensional MDS projection

Notebook: MDS on the Fruit Dataset

```
# Multidimensional scaling
from adspy_shared_utilities import plot_labelled_scatter
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import MDS

# each feature should be centered (zero mean) and with unit variance
X_fruits_normalized = StandardScaler().fit(X_fruits).transform(X_fruits)

mds = MDS(n_components = 2)

X_fruits_mds = mds.fit_transform(X_fruits_normalized)

plot_labelled_scatter(X_fruits_mds, y_fruits, ['apple', 'mandarin', 'orange', 'lemon'])
plt.xlabel('First MDS feature')
plt.ylabel('Second MDS feature')
plt.title("Fruit sample dataset MDS")
```


Fruit sample dataset MDS
apple
mandarin
orange
lemon
Second MDS feature
First MDS feature
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

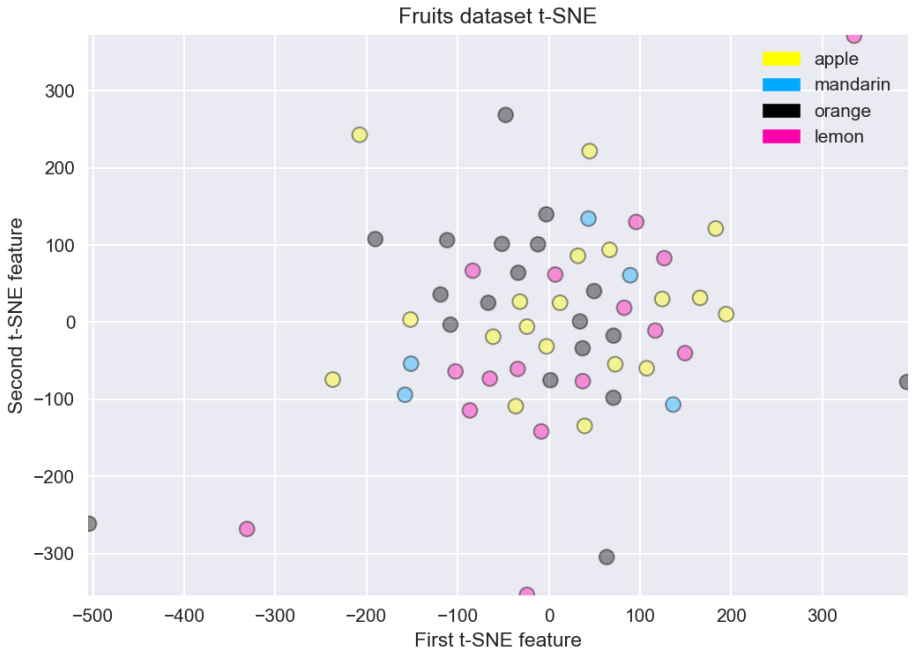
t-SNE: a powerful manifold learning method that finds a 2D projection preserving information about neighbors



Source: van der Maaten & Hinton https://lvdmaaten.github.io/tsne/

Notebook: t-SNE on the Fruit Dataset

```
from sklearn.manifold import TSNE

tsne = TSNE(random_state = 0)

X_tsne = tsne.fit_transform(X_fruits_normalized)

plot_labelled_scatter(X_tsne, y_fruits,
    ['apple', 'mandarin', 'orange', 'lemon'])
plt.xlabel('First t-SNE feature')
plt.ylabel('Second t-SNE feature')
plt.title("Fruits dataset t-SNE")
```



Lab

- Can sit at tables
- Limited number of seats
- Only those at table can communicate together.
- Please seek us (Grant, Teng, Zhuofeng) out for help.
  - *Can search and locate us.*