Introduction to Machine Learning

Sarah Maddox

Cancer Systems Biology

4/9/2018

What is machine learning?

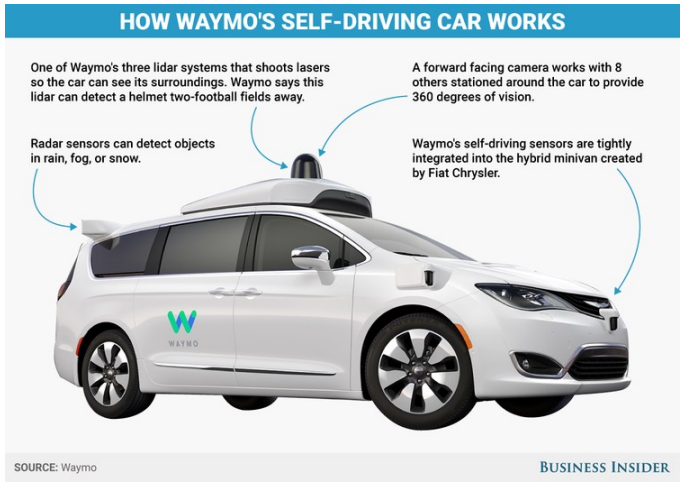
- A field of computer science that uses statistical techniques to give computer systems the ability to "learn”
- Can be broadly split into unsupervised and supervised techniques
- Can be used for dimensionality reduction
- Can be used to classify objects into coherent groups
- Can be used to predict how one variable depends on another random variable
- Uses statistics, data mining techniques, modeling, etc.

Examples?

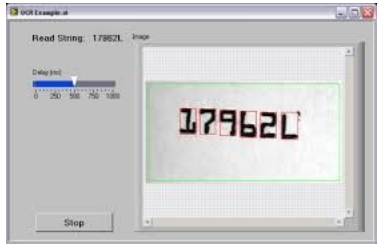
What Machine Learning Can Do

A simple way to think about supervised learning.

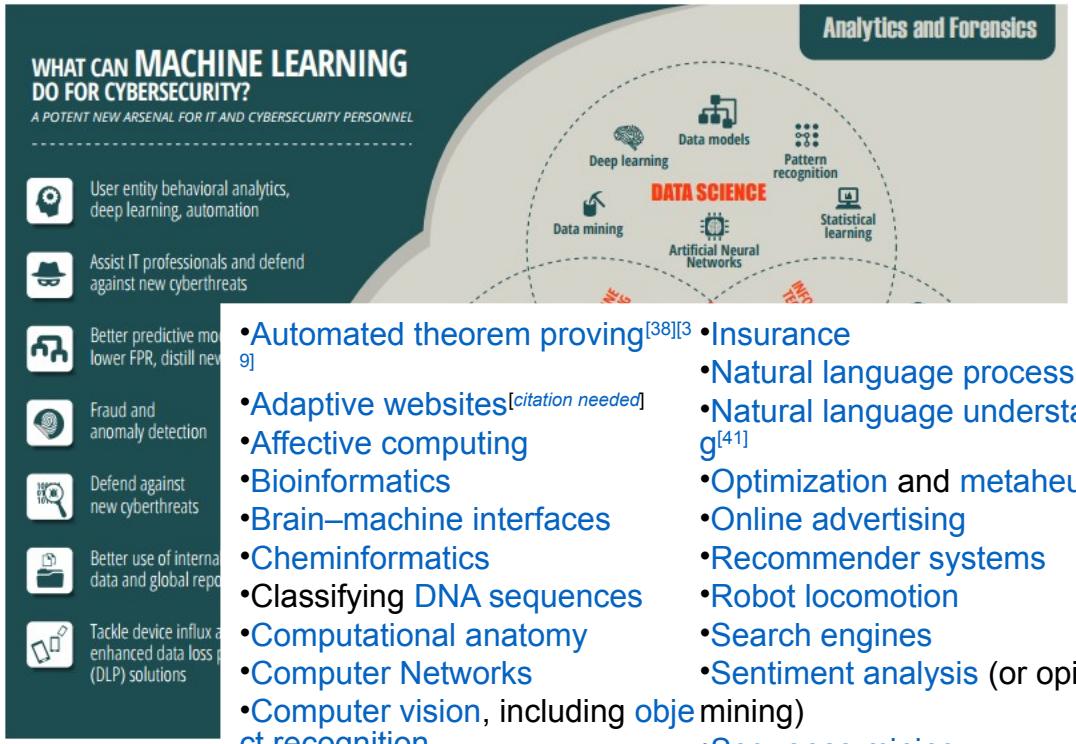
INPUT A	RESPONSE B	APPLICATION
Picture	Are there human faces? (0 or 1)	Photo tagging
Loan application	Will they repay the loan? (0 or 1)	Loan approvals
Ad plus user information	Will user click on ad? (0 or 1)	Targeted online ads
Audio clip	Transcript of audio clip	Speech recognition
English sentence	French sentence	Language translation
Sensors from hard disk, plane engine, etc.	Is it about to fail?	Preventive maintenance
Car camera and other sensors	Position of other cars	Self-driving cars



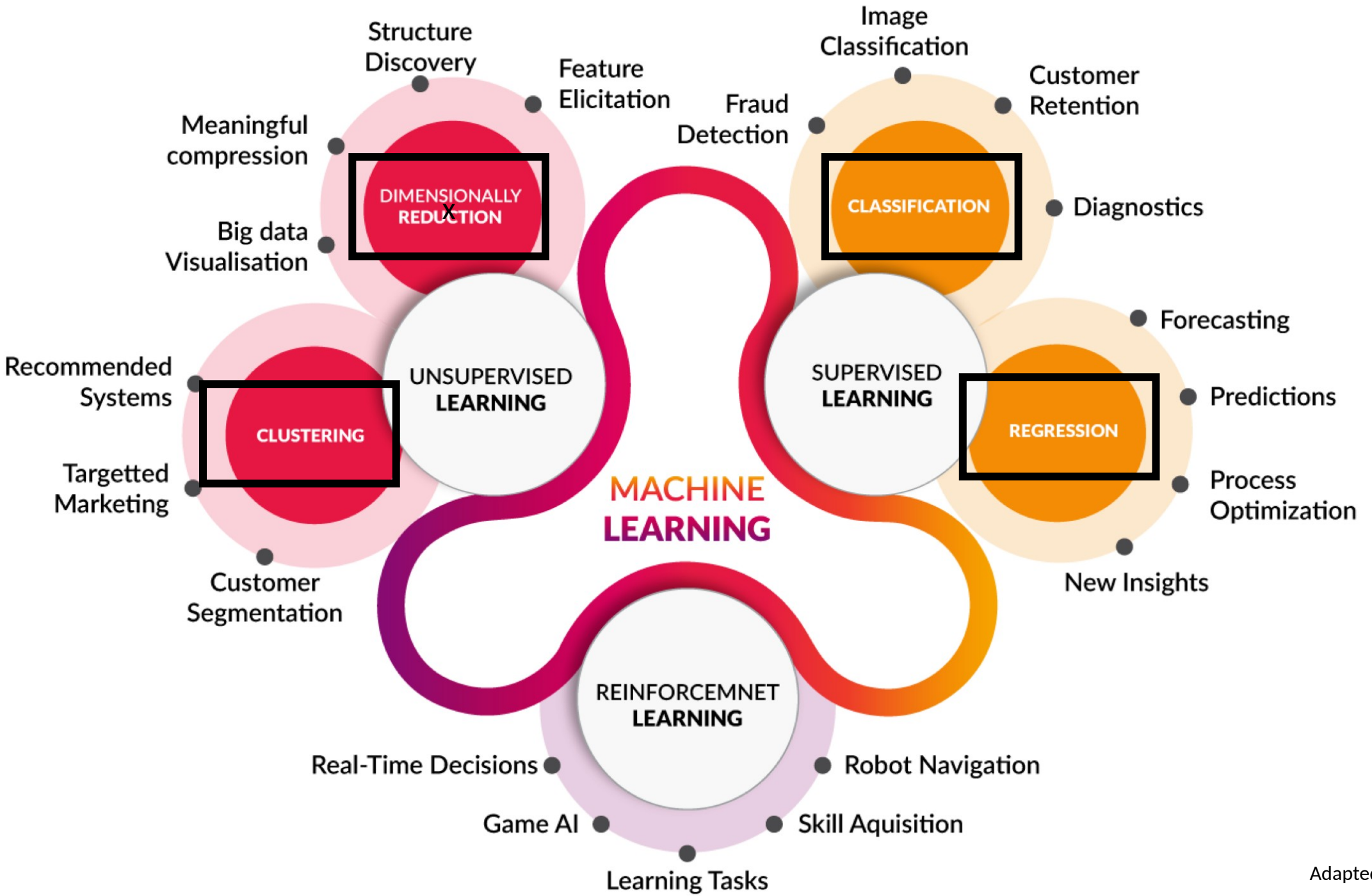








- Automated theorem proving[38][39]
- Adaptive websites[*citation needed*]
- Affective computing
- Bioinformatics
- Brain–machine interfaces
- Cheminformatics
- Classifying DNA sequences
- Computational anatomy
- Computer Networks
- Computer vision, including object recognition
- Detecting credit-card fraud
- General game playing[40]
- Information retrieval
- Internet fraud detection[27]
- Linguistics
- Marketing
- Machine learning control
- Machine perception
- Medical diagnosis
- Economics
- Insurance
- Natural language processing
- Natural language understanding[41]
- Optimization and metaheuristic
- Online advertising
- Recommender systems
- Robot locomotion
- Search engines
- Sentiment analysis (or opinion mining)
- Sequence mining
- Software engineering
- Speech and handwriting recognition
- Financial market analysis
- Structural health monitoring
- Syntactic pattern recognition
- Time series forecasting
- User behavior analytics
- Translation[42]



Adapted from Dr. Mesa


Machine Learning Algorithms
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression

Visualizing High-Dimensional Data

- Download the CCLE Microarray dataset (Legacy data → mRNA expression → **CCLE_Expression_Entrez_2012-09-29.gct**
- How big is this dataset? (hint: read into python using Pandas and use **df.shape**)
- What are the columns and what are the rows?
- How would you best visualize this dataset?
- Is there a good way to segment the dataset, or make it smaller? How would you find the important information?

Let's start with something smaller…

```
from sklearn import datasets
from sklearn.preprocessing import
StandardScaler

# import some data to play with
iris=datasets.load_iris()
x=iris.data
y=iris.target
# X_std = StandardScaler().fit_transform(X)
```

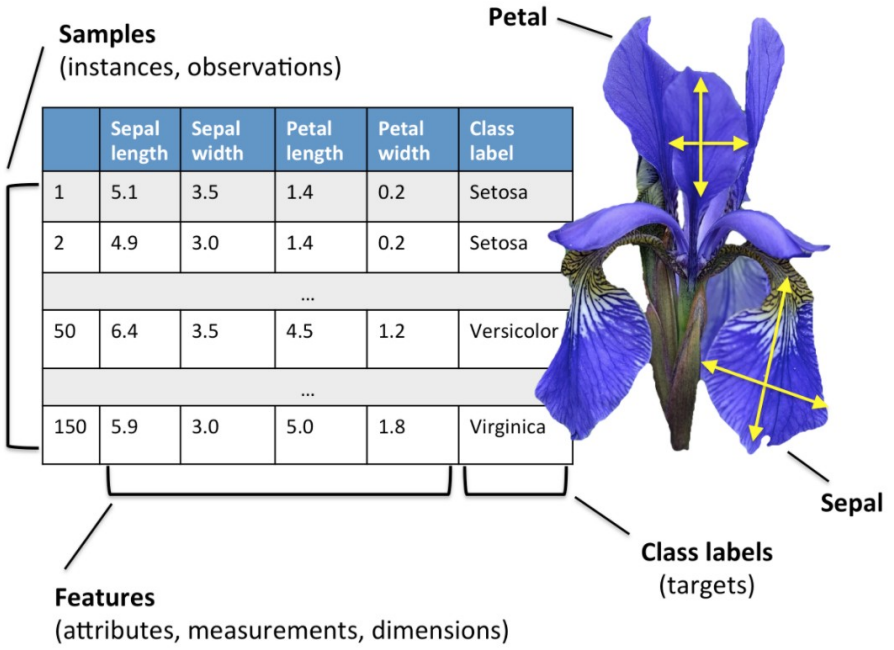
The iris dataset contains measurements for 150 iris flowers from three different species.

The three classes in the Iris dataset are:

1. *Iris-setosa* (n=50)
2. *Iris-versicolor* (n=50)
3. *Iris-virginica* (n=50)

And the four features of in Iris dataset are:

1. *sepal length* in cm
2. *sepal width* in cm
3. *petal length* in cm
4. *petal width* in cm



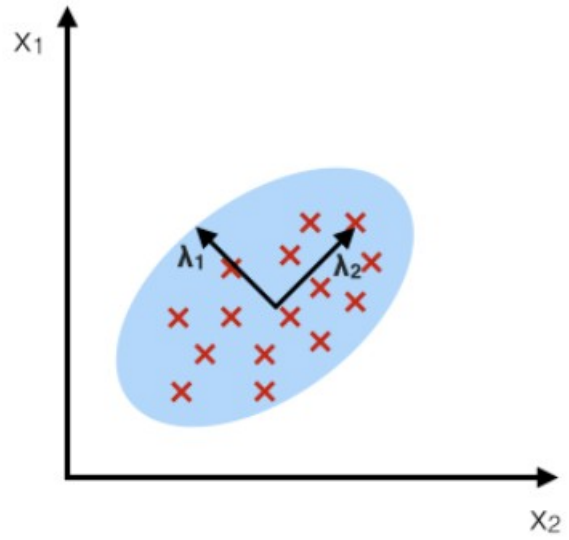
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
…					
50	6.4	3.5	4.5	1.2	Versicolor
…					
150	5.9	3.0	5.0	1.8	Virginica

PCA

- Here, our desired outcome of the principal component analysis is to **project a feature space** (our dataset consisting of n d-dimensional samples) **onto a smaller subspace** that represents our data "**well**". A possible application would be a pattern classification task, where we want to reduce the computational costs and the error of parameter estimation by **reducing the number of dimensions** of our feature space by extracting a subspace that describes our data "best".
- **What is a "good" subspace?**
  - Let's assume that our goal is to reduce the dimensions of a d-dimensional dataset by projecting it onto a (k)(k)-dimensional subspace (where k<d_k<d). So, how do we know what size we should choose for k, and how do we know if we have a feature space that represents our data "well"?
  - We can compute eigenvectors (the components) from our data set and collect them in a so-called scatter-matrix (or alternatively calculate them from the covariance matrix). Each of those eigenvectors is associated with an eigenvalue, which tell us about the "length" or "magnitude" of the eigenvectors. If we observe that all the eigenvalues are of very similar magnitude, this is a good indicator that our data is already in a "good" subspace. Or if some of the eigenvalues are much much higher than others, we might be interested in keeping only those eigenvectors with the much larger eigenvalues, since they contain more information about our data distribution. Vice versa, eigenvalues that are close to 0 are less informative and we might consider in dropping those when we construct the new feature subspace.
  - ***Or, roughly speaking in PCA we are trying to find the axes with maximum variances where the data is most spread (within a class, since PCA treats the whole data set as one class***)

PCA

PCA:
component axes that maximize the variance




```
pca = PCA(n_components = 2)

data = pca.fit_transform(x,y)

Import matplotlib .pyplot as plt

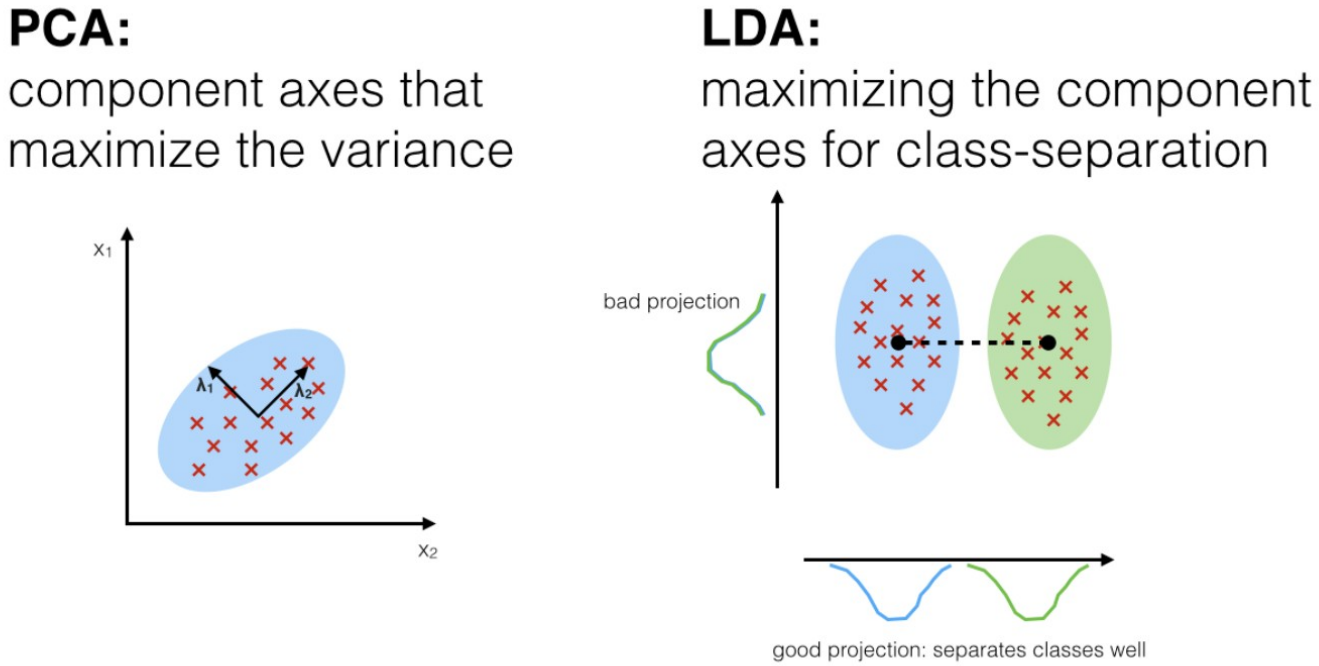
plt.scatter(*zip(*data))
plt.show()
```

Can you write code to color the points by their classes?

http://setosa.io/ev/principal-component-analysis/

LDA

- Linear Discriminant Analysis
- **“The general LDA approach is very similar to a Principal Component Analysis, but in addition to finding the component axes that maximize the variance of our data (PCA), we are additionally interested in the axes that maximize the separation between multiple classes (LDA).”**



https://sebastianraschka.com/faq/docs/lda-vs-pca.html

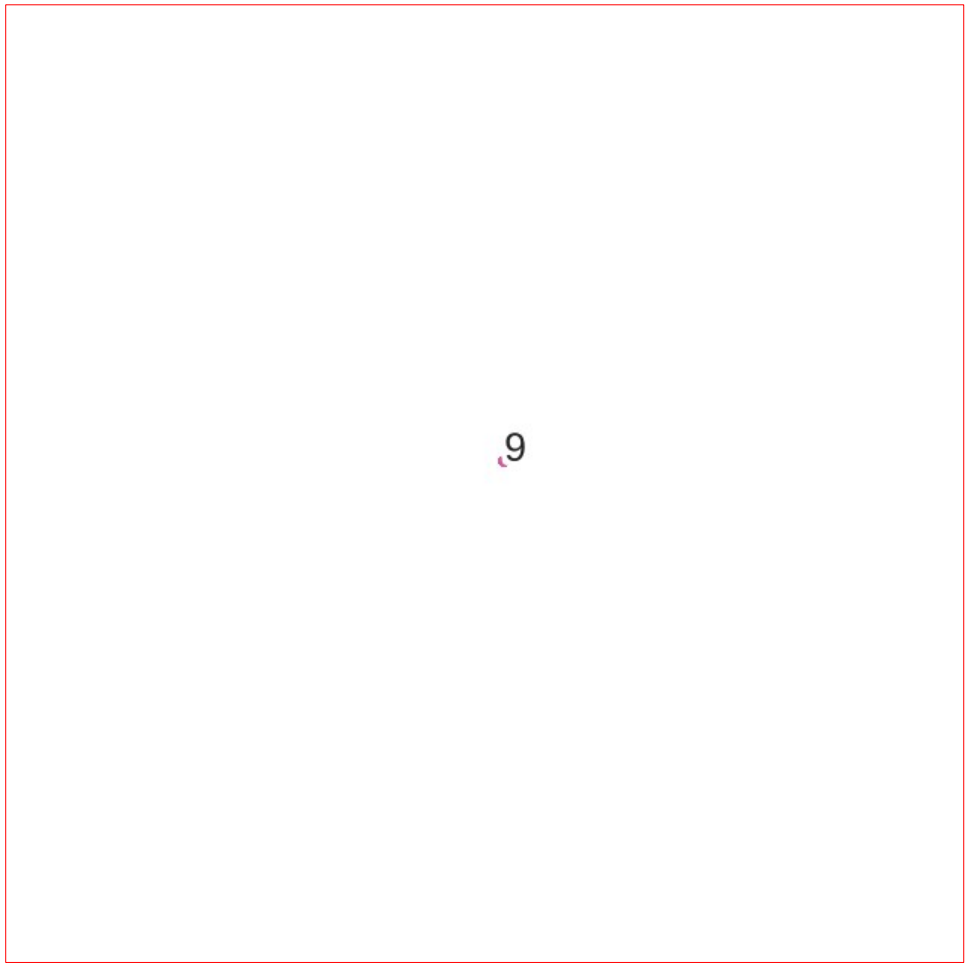
Change your PCA code to run LDA

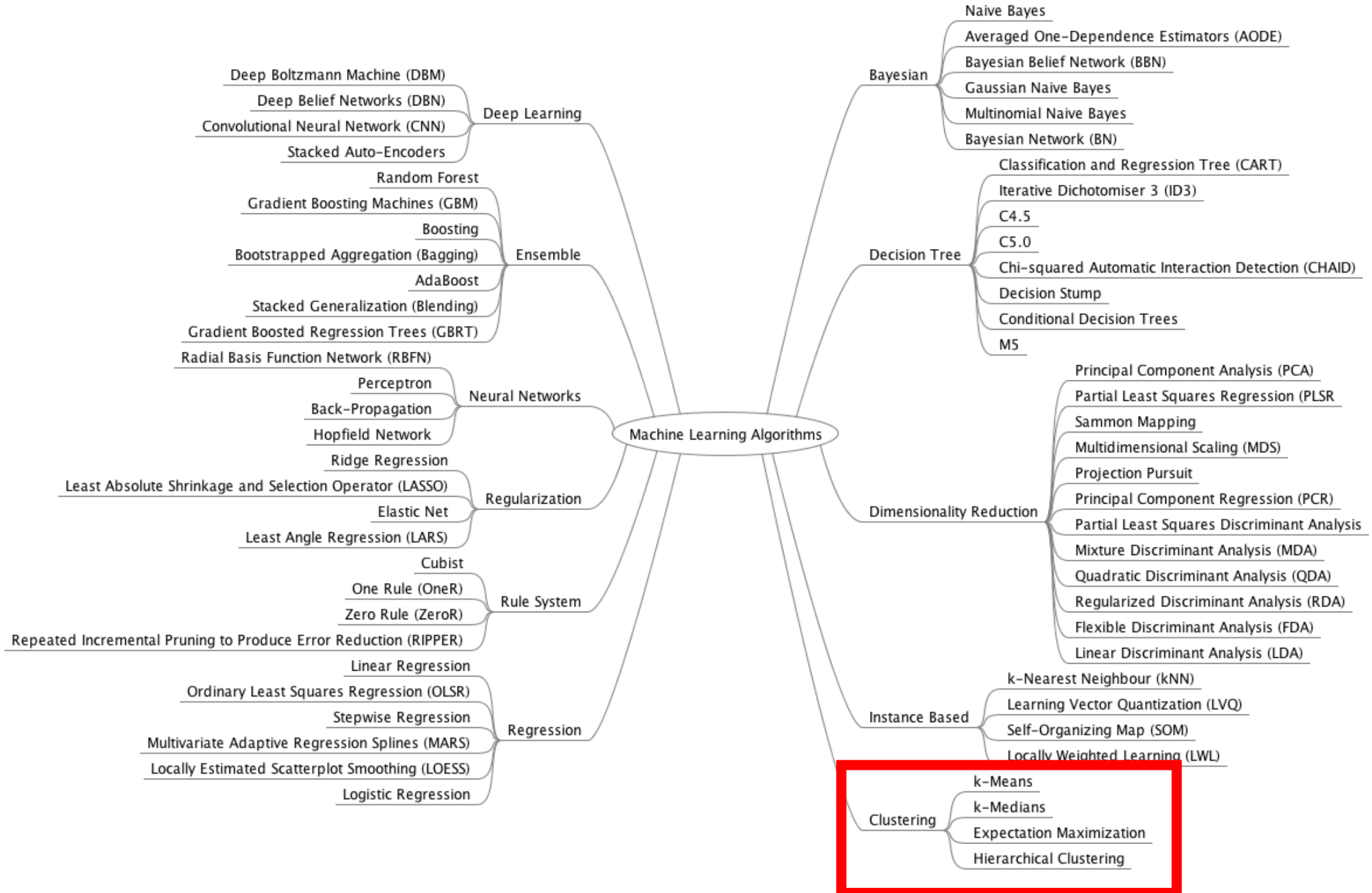
```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA #one line
lda = LDA(n_components = 2)
data = lda.fit_transform(x,y)
```

Other Common DR Techniques

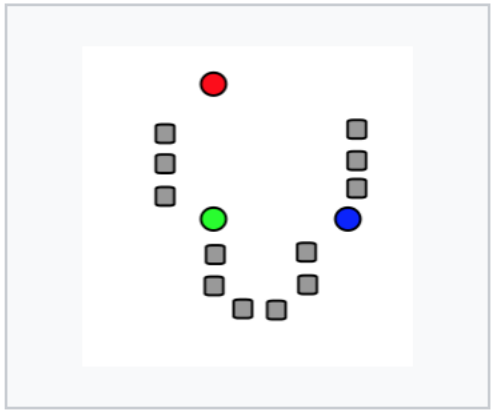
- Kernel PCA (nonlinear PCA)
- T-SNE
- Filters:
  - High correlation filter
  - Low variance filter

n_components : int, optional (default: 2)	Dimension of the embedded space.
perplexity : float, optional (default: 30)	The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selcting a value between 5 and 50. The choice is notn extremely critical since t-SNE is quite insensitive to this parameter.
early_exaggeration : float, optional (default: 4.0)	Controls how tight natural clusters in the original space are in the embedded space and how much space will be between them. For larger values, the space between natural clusters will be larger in the embedded space. Again, the choice of this parameter is not very critical. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high.
learning_rate : float, optional (default: 1000)	The learning rate can be a critical parameter. It should be between 100 and 1000. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high. If the cost function gets stuck in a bad local minimum increasing the learning rate helps sometimes.
n_iter : int, optional (default: 1000)	Maximum number of iterations for the optimization. Should be at least 200.
metric : string or callable, (default: “euclidean”)	The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter, or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS If metric is “precomputed”, X is assumed to be a distance matrix. Alternatively, if

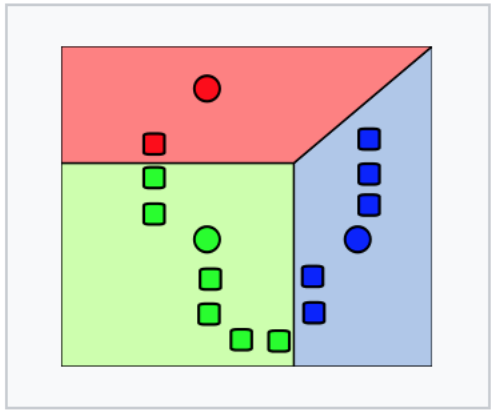



Machine Learning Algorithms
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression

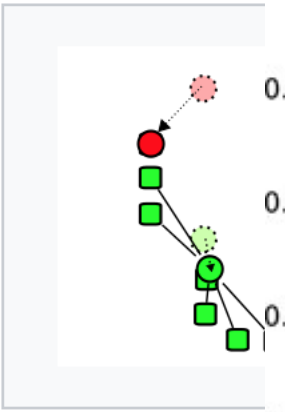
K-Means Clustering



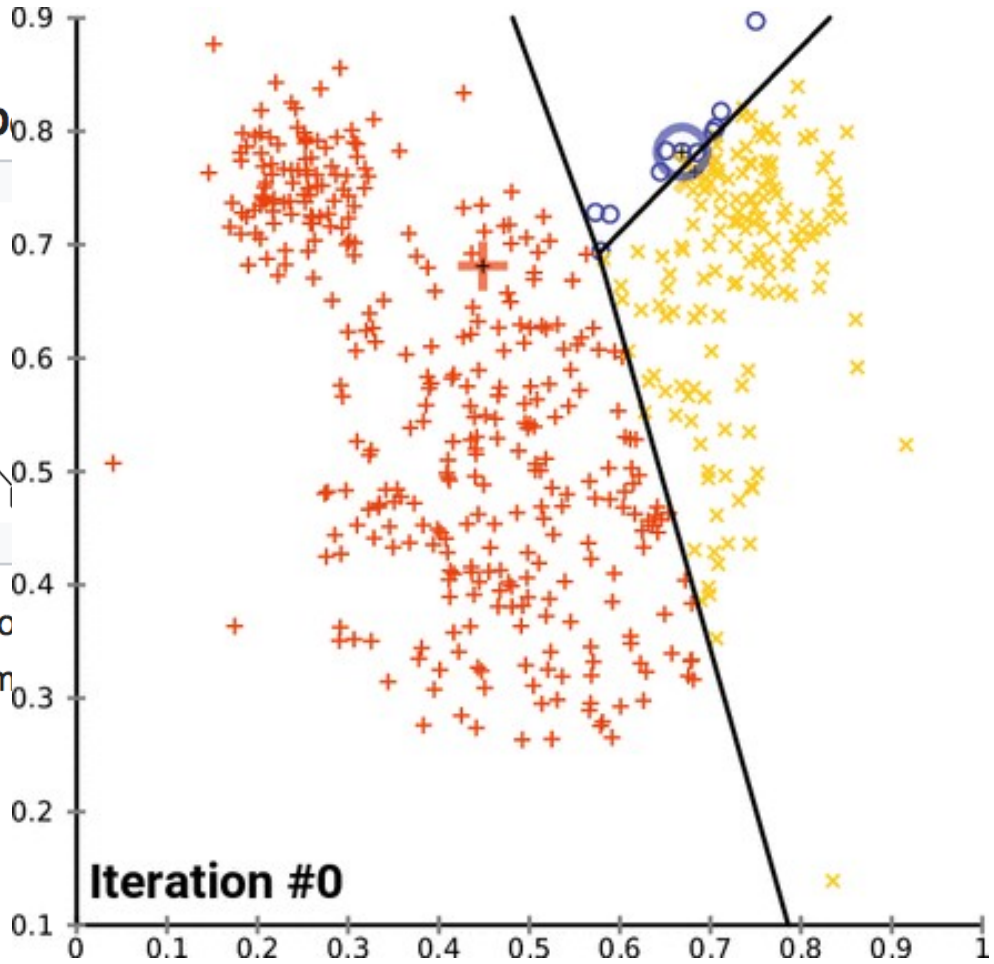
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3. The centroid o
k clusters becom
mean.



http://stamfordresearch.com/k-means-clustering-in-python/

Let’s practice…

- http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html

K-Means Clustering

```
from sklearn.cluster import KMeans
import sklearn.metrics as sm
```

- Can you use the sklearn documentation to build a k-means classifier on the iris dataset?
- Can you plot this classification?
- How well does the classification do? (sklearn.metrics)

Buil

This is th lines.

First we c the

Visuali

Lets plot the act

```
# View the
# Set the s
plt.figure(

# Create a
colormap =

# Plot the
plt.subplot
plt.scatter
plt.title('

# Plot the
plt.subplot
plt.scatter
plt.title('
```

Python

0, 0,

2, 2,

0, 2,

Accuracy

```
# Performance Metrics
sm.accuracy_score(y, predY)
```

Python

My result was 0.8933333333333331, so we can say that the model has an accuracy of 89.3%. Not base considering the model was unsupervised.

Confusion Matrix

```
# Confusion Matrix
sm.confusion_matrix(y, predY)
```

My results are…

```
array([[50, 0, 0],
[ 0, 48, 2],
[ 0, 14, 36]])
```

Hopefully the table below will render correctly, but we can summaries a the confusion matrix as shown below:

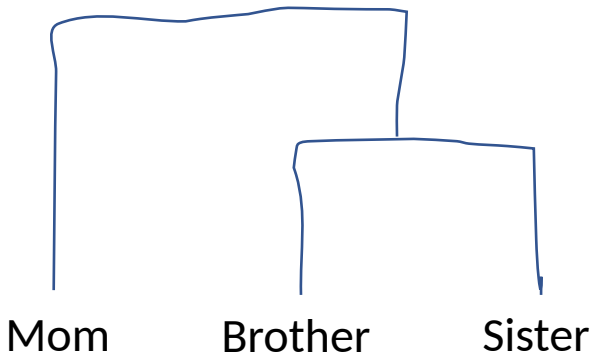
- correctly identifed all 0 classes as 0's
- correctly classified 48 class 1's but miss-classified 2 class 1's as class 2
- correctly classified 36 class 2's but miss-classified 14 class 2's as class 1

		Real Class		
		0	**1**	**2**
Predicted Class	**0**	50	0	0
	1	0	48	2
	2	0	14	36

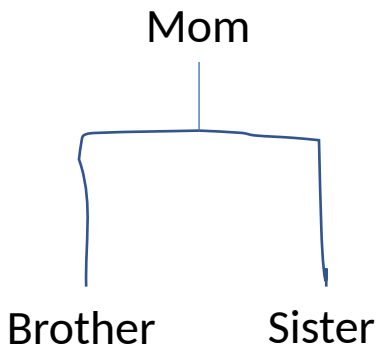
Hierarchical Clustering

Write all of these words in a horizontal line so you can put the words into a “tree” structure, where things farther down the same branch are more similar to each other. Each word should be the end of a branch. i.e. No one has a “parent.”

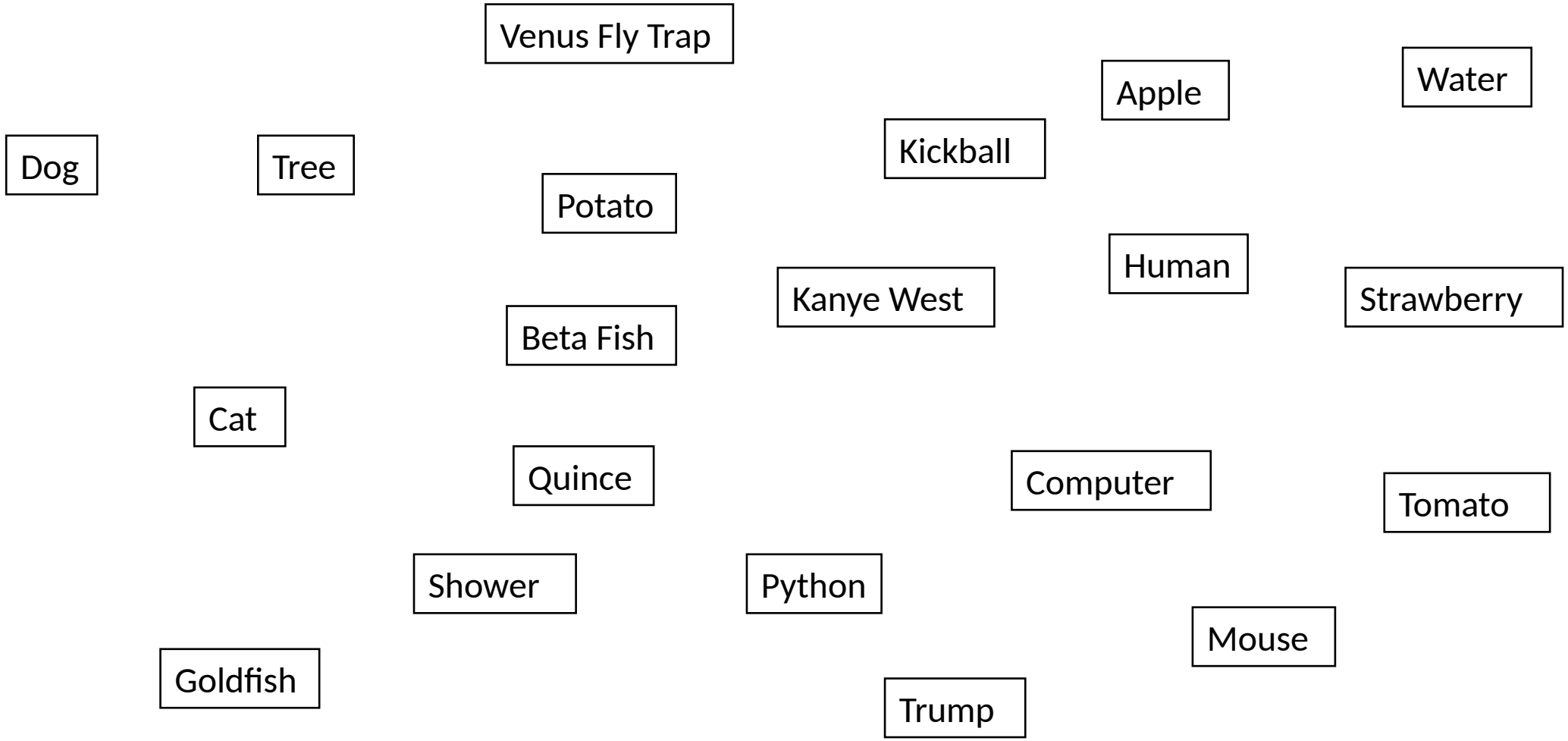
Correct:



Incorrect:

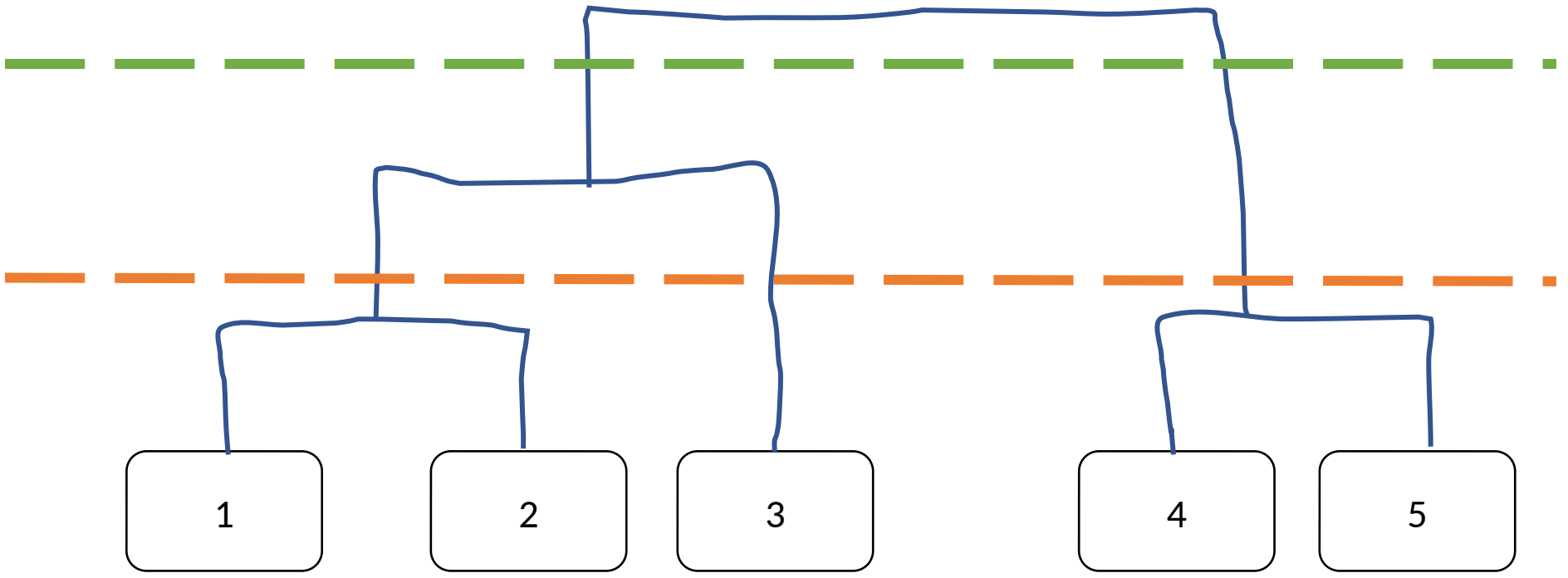


Hierarchical Clustering



Hierarchical Clustering

Now pick somewhere to draw a “cut” across your tree, such that the words below the line get grouped together, and any words above the line remain by themselves. Pick a cut that makes sense to your group!



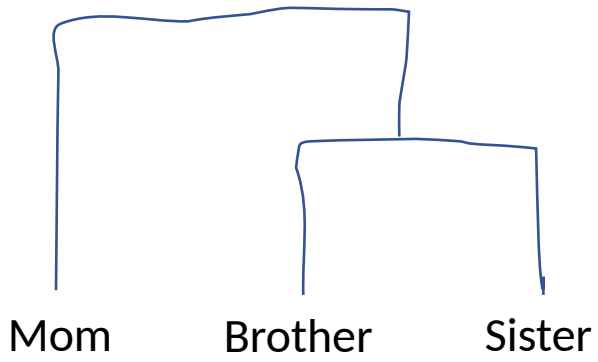
Clusters from green cut:
1 & 2 & 3
4 & 5

Clusters from orange cut:
1 & 2
3
4 & 5

Hierarchical Clustering

Make a heat map using a “distance” metric. Remember the diagonals should be 1!

Example:



	Mom	Brother	Sister
Mom	1	.5	.5
Brother	.5	1	1
Sister	.5	1	1

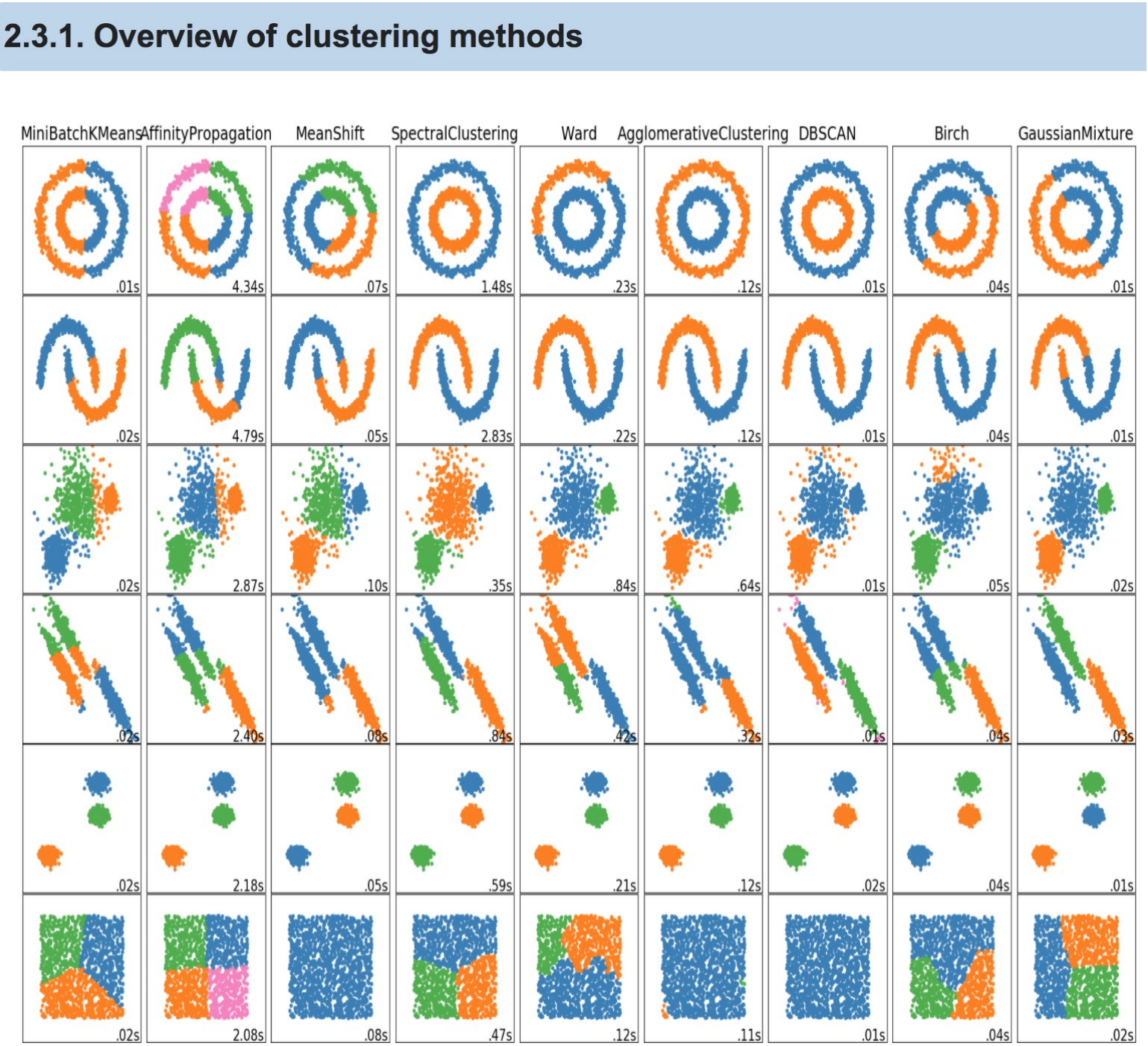
Hierarchical Clustering + Heatmaps

- Can you cluster the CCLE data with hierarchical clustering?
- Can you make a heatmap out of this?
- Remember, there is no “correct answer,” but there are some things you can expect to see
  - For example, I would expect to see different tissue types clustered together, and genes clustered by function
- Hints:
  - Look at **sklearn.cluster.AgglomerativeClustering**
  - Look at **seaborn.clustermap** : this goes directly from the data → clustering → heatmap
  - Play around with **Clustergrammer** (you’ll have to make the dataset smaller first) http://amp.pharm.mssm.edu/clustergrammer/
  - Try picking out only the genes in the dataset from your Cell Collective network (or another network)

Other types of unsupervised clustering

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

“Sklearn choosing the right estimator”
“sklearn clustering”



Problems with unsupervised learning?

- IRL, no way to compare your model to the “real” answer because you don’t know the real best way to reduce dimensions or to cluster the data
- You need to really know what you’re doing, especially when playing with parameters
  - https://distill.pub/2016/misread-tsne/
- Need to clean the data in a way that you find what you are interested in
  - E.g. if you want to know about genes that change over the cell cycle, you probably want to look at the same/similar cell types (or else you’ll introduce a lot of variance not related to cell cycle, and confuse your results)

Some more references

- https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/
- https://medium.com/@luckylwk/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b
- http://lvdmaaten.github.io/tsne/
- https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/
- https://www.naftaliharris.com/blog/visualizing-k-means-clustering/