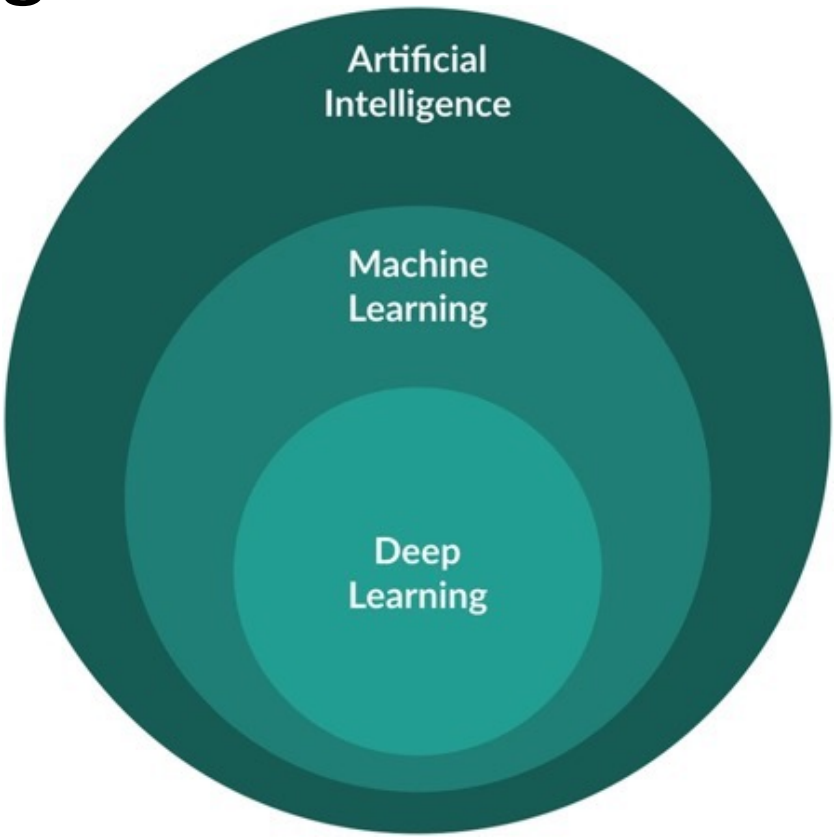
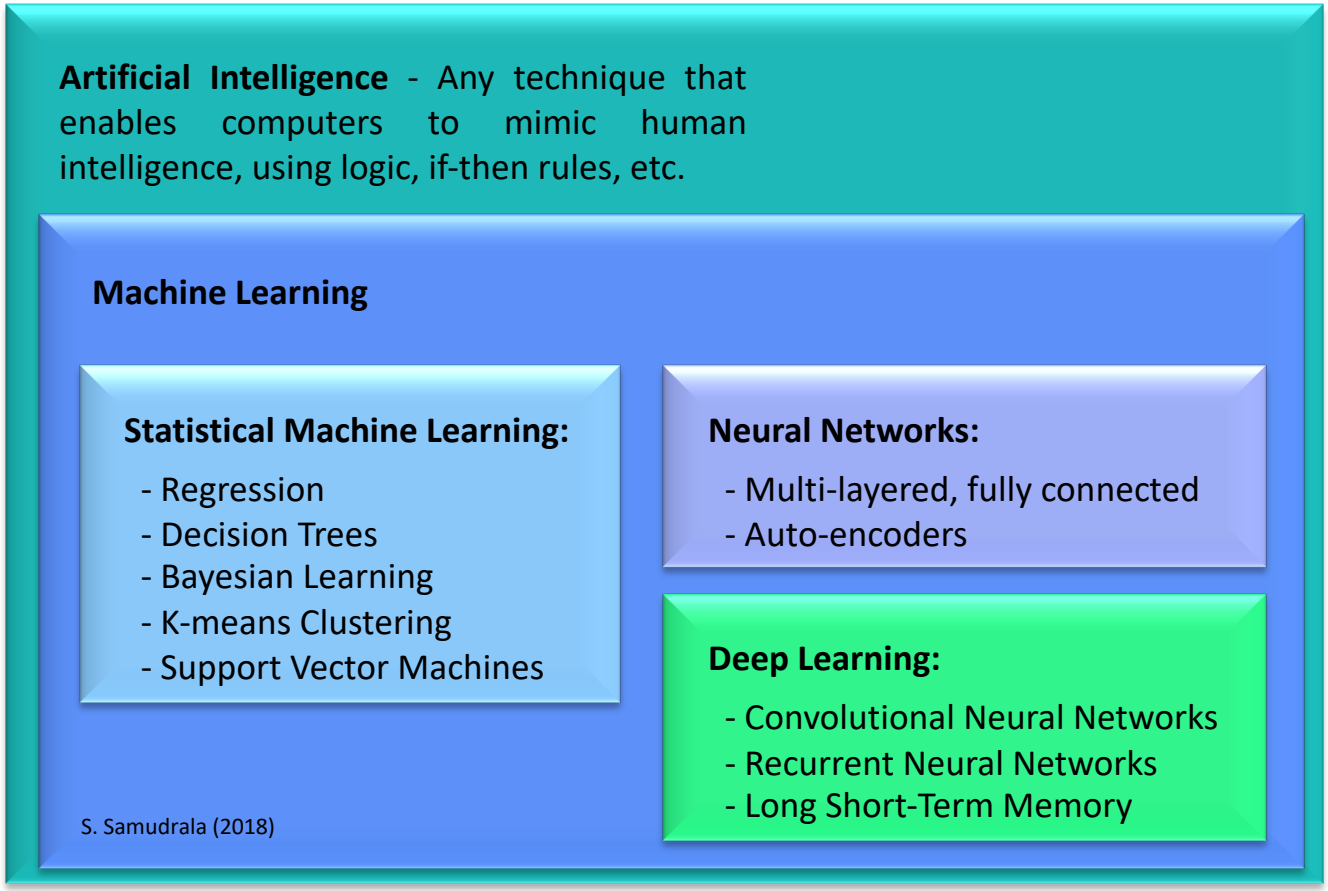
Machine Learning

08.10.21



Overview


Artificial Intelligence - Any technique that enables computers to mimic human intelligence, using logic, if-then rules, etc.
Machine Learning
Statistical Machine Learning:
- Regression
- Decision Trees
- Bayesian Learning
- K-means Clustering
- Support Vector Machines
Neural Networks:
- Multi-layered, fully connected
- Auto-encoders
Deep Learning:
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long Short-Term Memory
S. Samudrala (2018)

Overview

Artificial Intelligence - Any technique that enables computers to mimic human intelligence, using logic, if-then rules, etc.

Machine Learning

Statistical Machine Learning:

- Regression
- Decision Trees
- Bayesian Learning
- **K-means Clustering**
- **Support Vector Machines**

Neural Networks:

- **Multi-layered, fully connected**
- Auto-encoders

Deep Learning:

- **Convolutional Neural Networks**
- Recurrent Neural Networks
- Long Short-Term Memory

S. Samudrala (2018)

Some Terminology

Nature of the Problem:

- **Regression:**
  Variables are continuous numerical values (e.g. temperatures)
- **Classification:**
  Variables can adopt one of several possible values (e.g. grades)
- **Clustering:**
  Grouping of data into a specified number of cohesive units

Learning Methods:

- **Supervised Learning:**
  Requires in- and output variables (e.g. linear regression)
- **Unsupervised Learning:**
  Requires only input variables (e.g. k-means clustering)
- **Reinforced Learning:**
  Requires risk/reward inputs from environment (e.g. games)

Variables:

- **Feature Variables = Input Variables**
- **Label Variables = Output Variables**

Solving Methods:

- **Parametric Models:**
  Each input is given a weightage parameter based on their influence on the output (e.g. linear regression)
- **Non-Parametric Models:**
  Output is derived from the output of the training data, based on closest data record or group of records (e.g. decision trees)

Example Data – Distances between Large Cities

81 Cities with a population over 5 million people

Can the cities be clustered based on the distances between them?

Should I put the R scripts on Rspace?

Table with distances between cities in km:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		Tokyo	Delhi	Shanghai	Sao Paulo	Mexico City	Cairo	Mumbai	Beijing	Dhaka	Osaka	New York Cit	Karachi	Buenos Aires
2	Tokyo	0	5672	1765	16126	11322	9586	6738	2102	4903	403	10874	6920	18365
3	Delhi	5672	0	3993	13956	15124	4950	1073	3676	928	5293	12327	1424	16012
4	Shanghai	1765	3993	0	16498	12929	8369	5039	1068	3171	1364	11884	5323	19631
5	Sao Paulo	16126	13956	16498	0	6086	9006	13249	15432	14860	16297	5370	12565	3856
6	Mexico City	11322	15124	12929	6086	0	12393	15661	12478	15128	11701	3364	14883	7364
7	Cairo	9586	4950	8369	9006	12393	0	4361	7557	5856	9325	9040	3585	11812
8	Mumbai	6738	1073	5039	13249	15661	4361	0	4748	1887	6356	12555	873	14948
9	Beijing	2102	3676	1068	15432	12478	7557	4748	0	3031	1785	11014	4844	19266
10	Dhaka	4903	928	3171	14860	15128	5856	1887	3031	0	4512	12695	2352	16781
11	Osaka	403	5293	1364	16297	11701	9325	6356	1785	4512	0	11131	6562	18748
12	New York Cit	10874	12327	11884	5370	3364	9040	12555	11014	12695	11131	0	11707	8487
13	Karachi	6920	1424	5323	12565	14883	3585	873	4844	2352	6562	11707	0	14732
14	Buenos Aires	18365	16012	19631	3856	7364	11812	14948	19266	16781	18748	8487	14732	0
15	Chongqing	3173	2550	1446	15855	13934	7131	3593	1461	1731	2781	12226	3907	18504
16	Istanbul	8971	5139	8004	8974	11449	1233	4813	7074	5965	8766	8088	3953	12234
17	Kolkata	5140	760	3406	14716	15289	5710	1663	3263	237	4748	12766	2172	16544
18	Manila	2995	4289	1844	18078	14239	9188	5136	2843	3362	2660	13693	5712	17787
19	Lagos	13494	8501	12243	5721	11085	3915	7629	11470	9427	13239	8481	7079	7914
20	Rio de Janeir	18556	14447	18276	2430	7644	9909	13456	17346	15339	18698	7716	13059	1945
21	Tianjin	2028	3714	956	15543	12513	7653	4788	113	3047	1701	11096	4903	19379
22	Kinshasa	13367	7823	11801	6990	12742	4167	6815	11267	8700	13028	10261	6480	8255
23	Guangzhou	2909	3251	1212	16836	14141	8034	4209	1889	2337	2509	12902	4666	18505
24	Los Angeles	8828	13178	10452	8332	2494	12222	14005	10080	12933	9208	3949	13444	9828

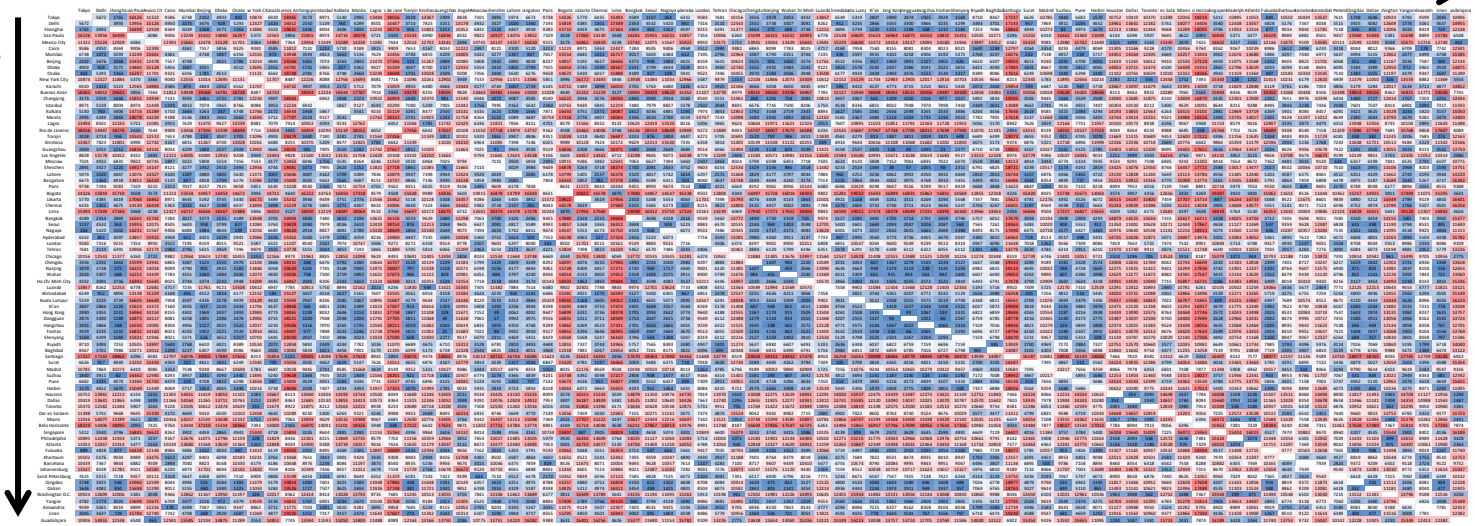
Data Pre-Processing

Models build on fewer feature variables usually generalize better, thus improving the quality of predictions

→ PCA can be used for dimensionality reduction

81 feature variables / dimensions

81 records



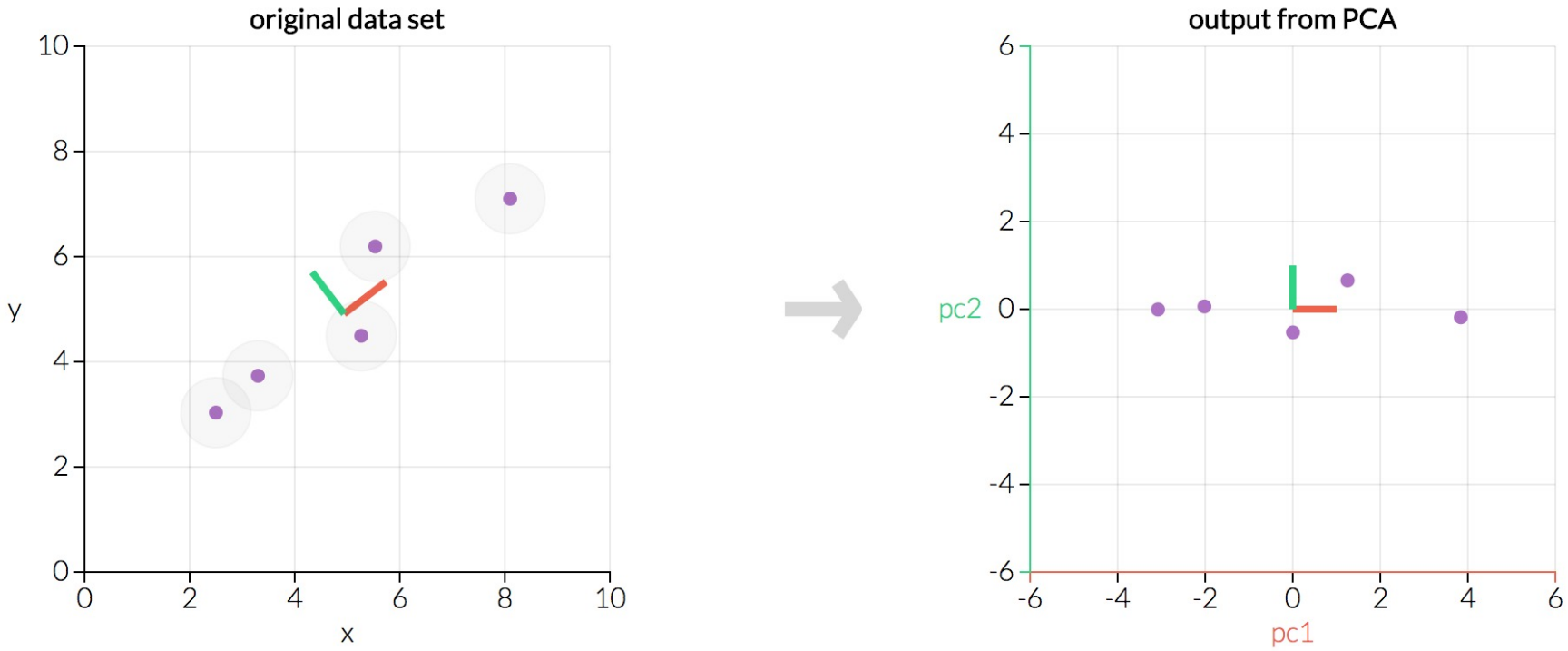
Blue indicates small and red large values

Principal Component Analysis (PCA)

PCA transforms feature variables into principal components based on their “information content”

Principal components (PCs) are ranked according to their “information content”

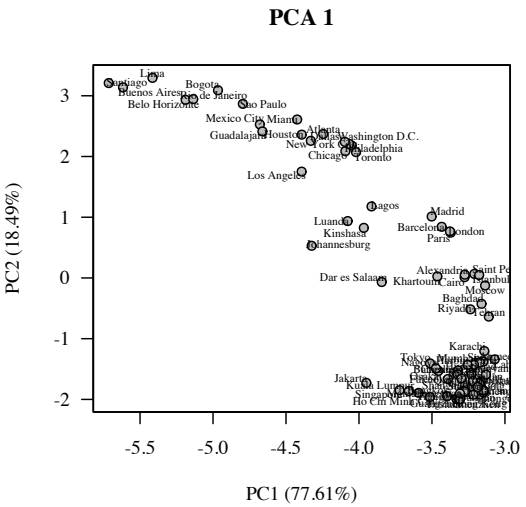
There are as many principal components as feature variables in the original data

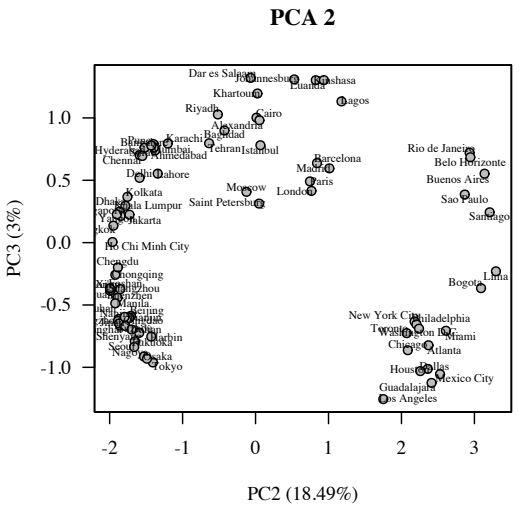


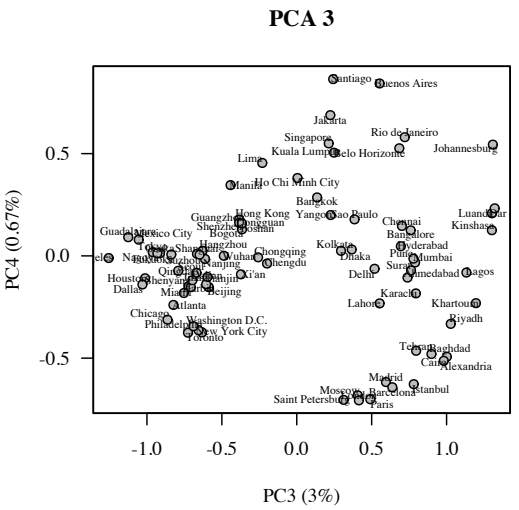
PCA – Score Plots

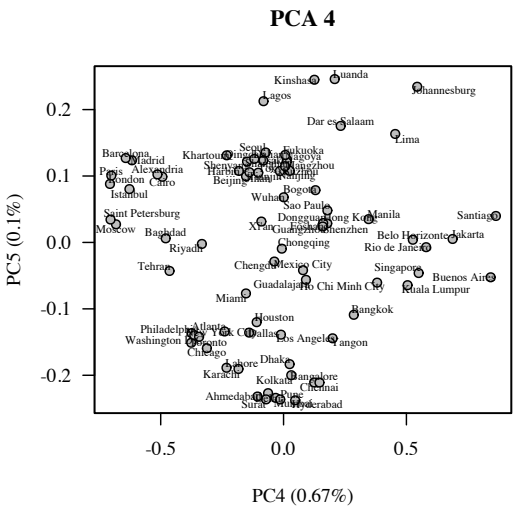
Plots show combinations of the first 5 PCs of the PCA

Data was normalized before the PCA: The distances were divided by half of earth's circumference to have "nicer" numbers

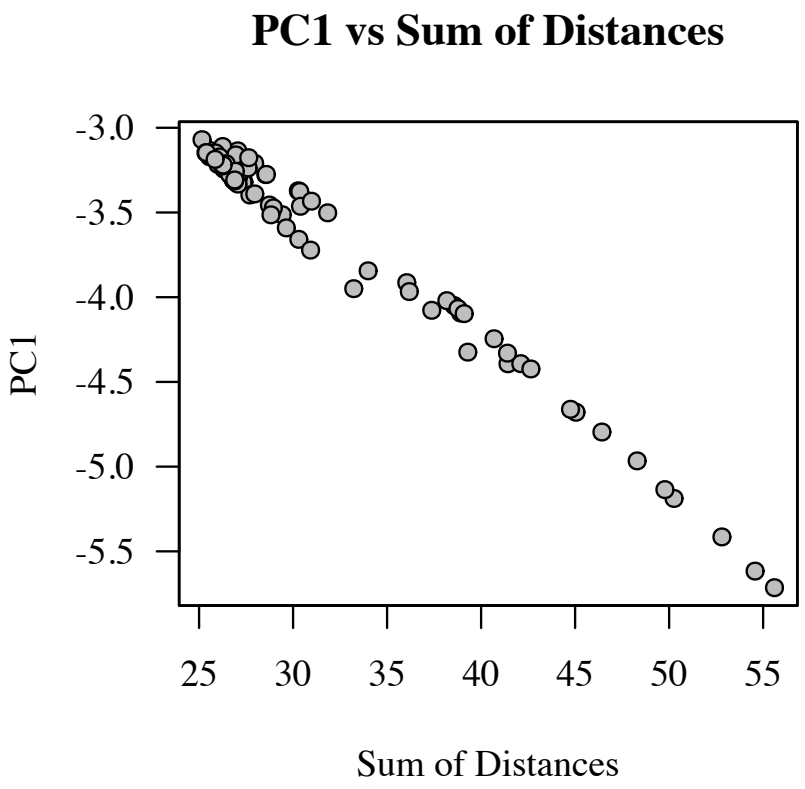








PC1 vs Sum of Distances

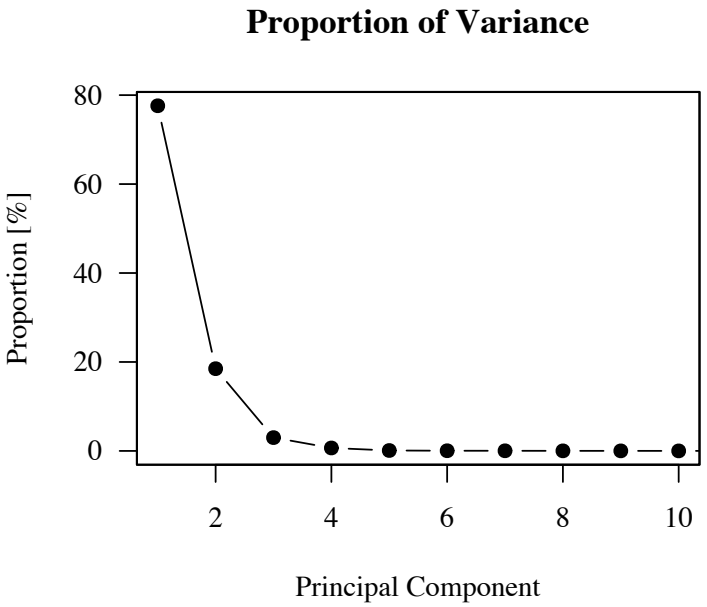


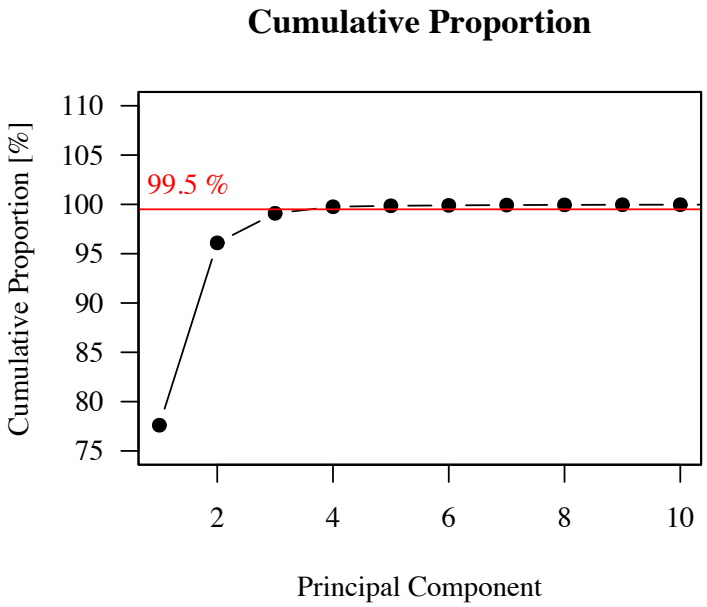
PC1 reflects mostly the "remoteness" of cities

Proportion of Variance

The first four principal components contain already 99.76 % of the variance found in the complete data set

81 feature variables can be reduced to just 4 without losing much information





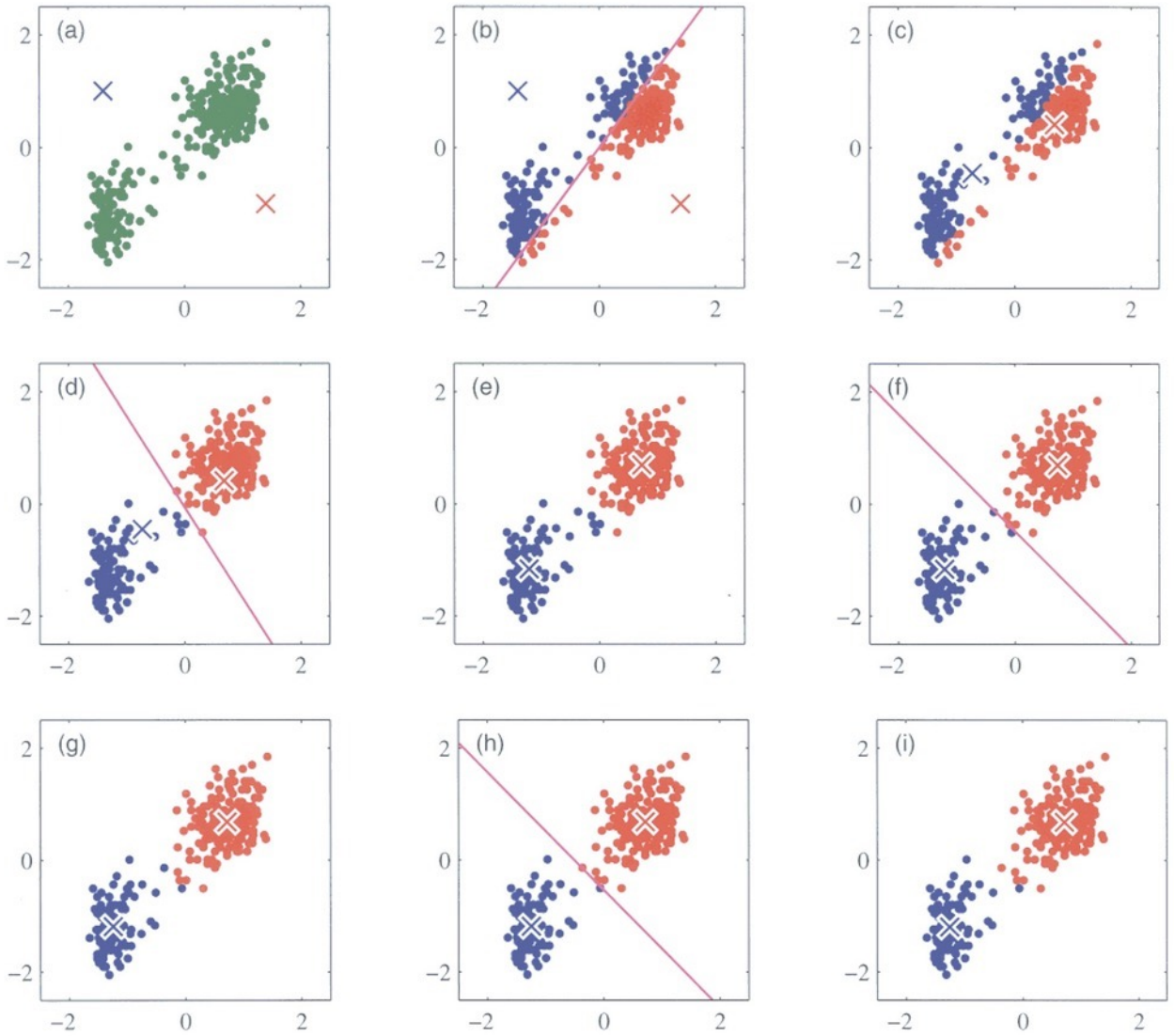
Pre-Processed Data

Each column is now a linear combination of the distances from before

This data can be used for further analysis using e.g. k-means clustering to group large cities

	A	B	C	D	E	F
1		PC1	PC2	PC3	PC4	
2	Tokyo	-3.5132573	-1.4065444	-0.9610935	0.01762064	
3	Delhi	-3.1302536	-1.590207	0.51942704	-0.0629664	
4	Shanghai	-3.2867935	-1.8580371	-0.6662294	0.01000054	
5	Sao Paulo	-4.7957519	2.86727848	0.38560758	0.17843665	
6	Mexico City	-4.6791094	2.52864189	-1.0547855	0.07925811	
7	Cairo	-3.276456	0.01300147	1.00188438	-0.493151	
8	Mumbai	-3.2043953	-1.4001769	0.78989779	-0.0322014	
9	Beijing	-3.1658421	-1.707514	-0.5875577	-0.1525491	
10	Dhaka	-3.1663136	-1.7850166	0.29540707	0.02467551	
11	Osaka	-3.4558816	-1.5304897	-0.9127806	0.01321126	
12	New York City	-4.0497117	2.18002834	-0.6325919	-0.3740346	
13	Karachi	-3.1413089	-1.2032136	0.79505517	-0.1833854	
14	Buenos Aires	-5.616636	3.13876499	0.55304565	0.84297111	
15	Chongqing	-3.1710076	-1.9222255	-0.2582592	-0.007073	
16	Istanbul	-3.2095743	0.06655967	0.78088486	-0.6272818	
17	Kolkata	-3.1743656	-1.7566679	0.36696909	0.03130614	
18	Manila	-3.5908738	-1.8918189	-0.4421297	0.34608063	
19	Lagos	-3.9141078	1.17757002	1.13229433	-0.0815739	
20	Rio de Janeiro	-5.1879471	2.93388618	0.72091375	0.58029891	
21	Tianjin	-3.1748138	-1.7289498	-0.6048015	-0.1387346	
22	Kinshasa	-3.9668242	0.82485771	1.30244522	0.12547932	
23	Guangzhou	-3.3078036	-1.9947158	-0.372274	0.15794958	
24	Los Angeles	-4.392923	1.75074907	-1.255036	-0.0106216	
25	Moscow	-3.1359644	-0.1245199	0.40707264	-0.6818615	
26	Shenzhen	-3.324617	-1.9934672	-0.3818544	0.17185379	
27	Lahore	-3.0702305	-1.3408782	0.55336648	-0.2322427	
28	Bangalore	-3.321047	-1.5224082	0.76074325	0.12487839	
29	Paris	-3.3717408	0.74546634	0.49096091	-0.7010445	
30	Bogota	-4.9659616	3.09070871	-0.3653326	0.12984391	
31	Jakarta	-3.9497271	-1.7278463	0.22399271	0.68806507	
32	Chennai	-3.3231616	-1.58858	0.70200733	0.14628188	
33	Lima	-5.4148923	3.29467677	-0.2300196	0.45439614	

K-Means Clustering - Unsupervised Learning

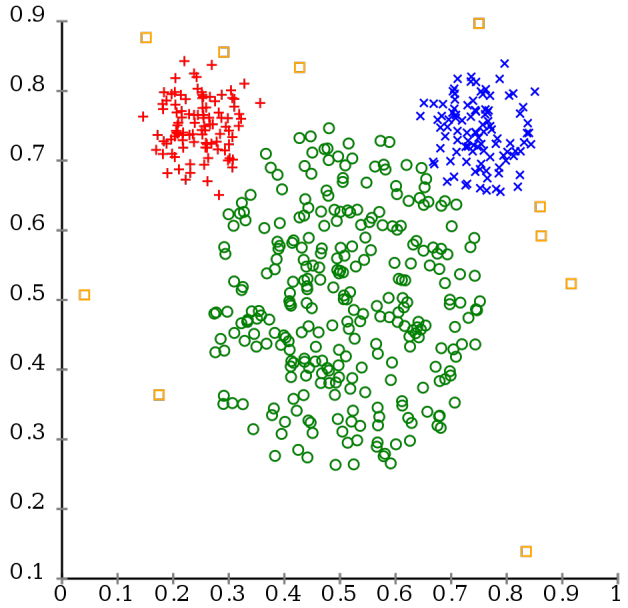


Bishop 2006

K-Means Clustering doesn’t work in every case

Different cluster analysis results on "mouse" data set:

Original Data

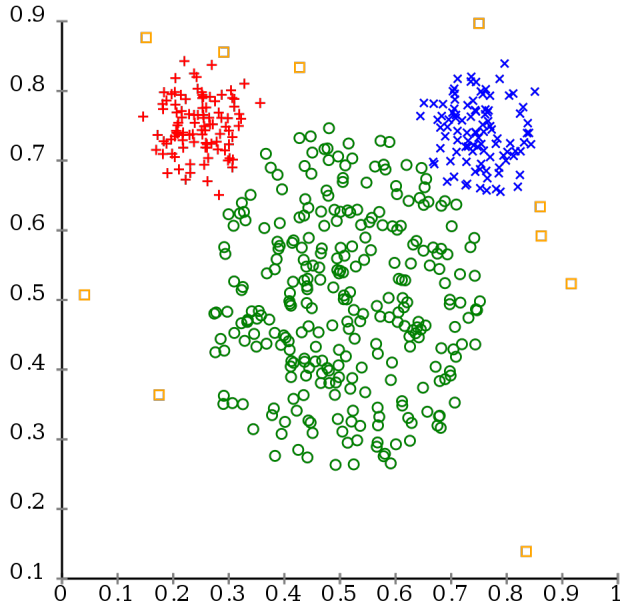


wikipedia

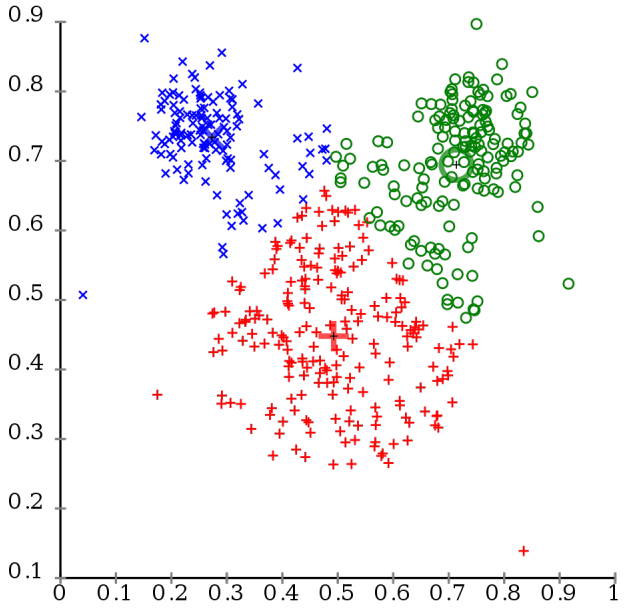
K-Means Clustering doesn’t work in every case

Different cluster analysis results on "mouse" data set:

Original Data



k-Means Clustering

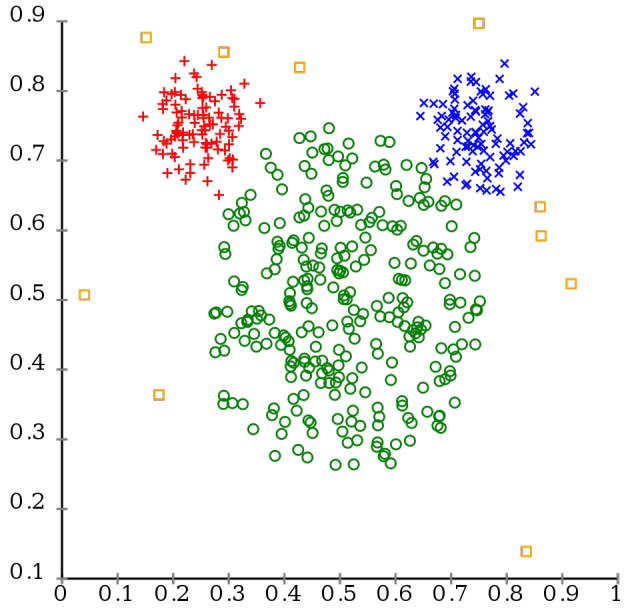


wikipedia

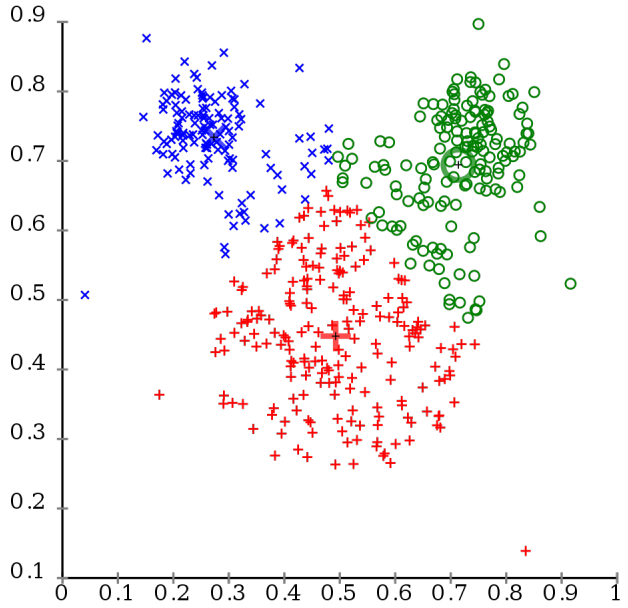
K-Means Clustering doesn’t work in every case

Different cluster analysis results on "mouse" data set:

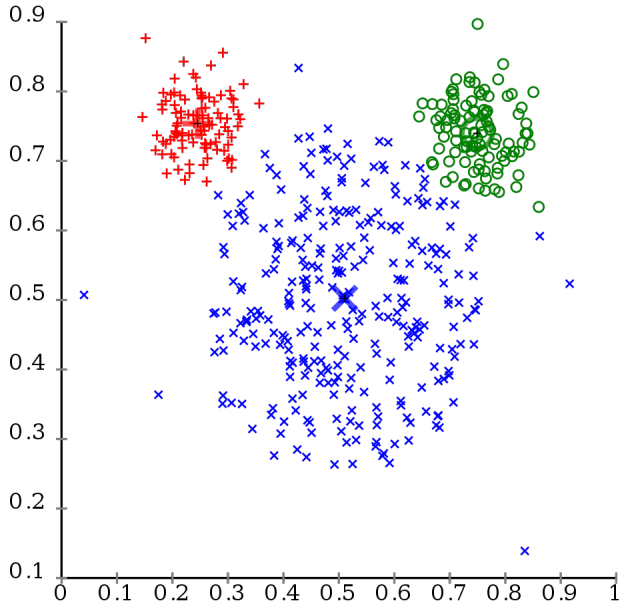
Original Data



k-Means Clustering



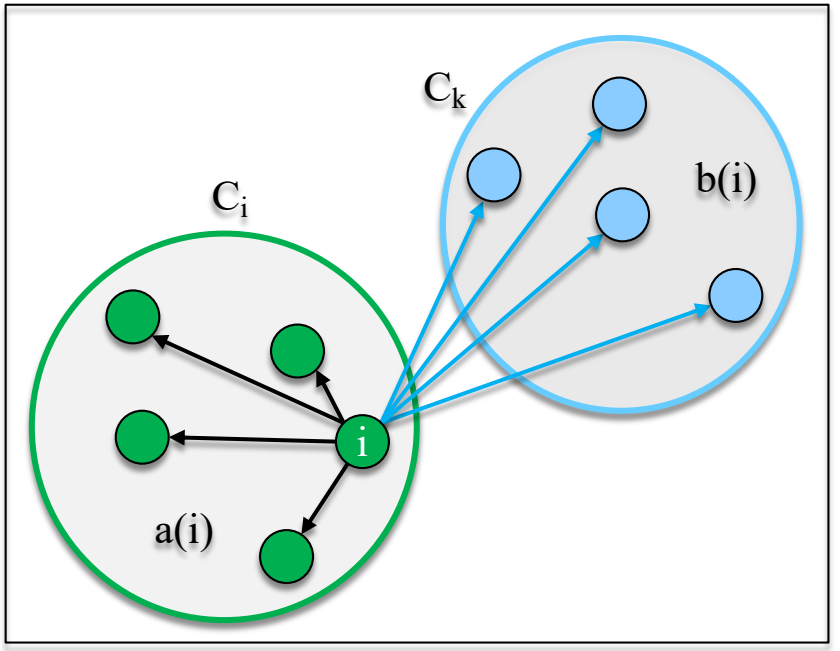
EM Clustering



wikipedia

Average Silhouette Method for Optimal Cluster Number

"The silhouette value is a measure of how similar an object is to its own cluster (cohesion, **a(i)**) compared to other clusters (separation, **b(i)**). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters." - Wikipedia



$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j)$$

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

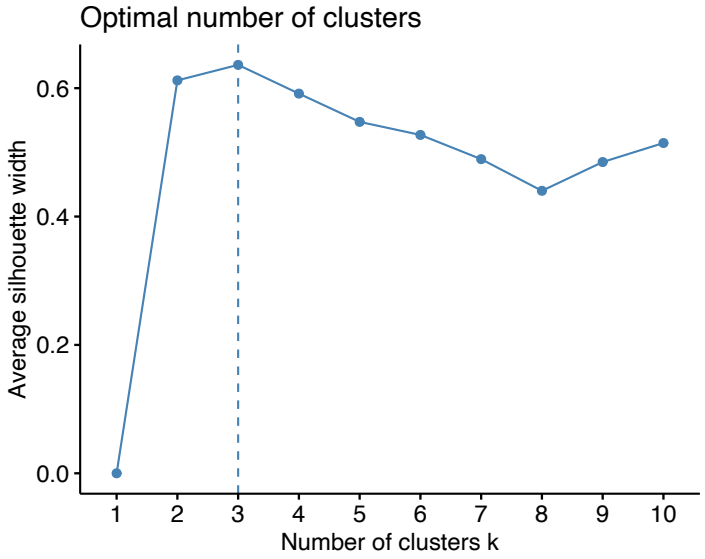
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$$-1 \leq s(i) \leq 1$$

The optimal number of clusters is based on the average silhouette of all data points, calculated for different cluster numbers. The optimal number of clusters should lead to the largest mean silhouette.

wikipedia

K-Means Clustering on City Data



Three clusters seem to be optimal to group the cities

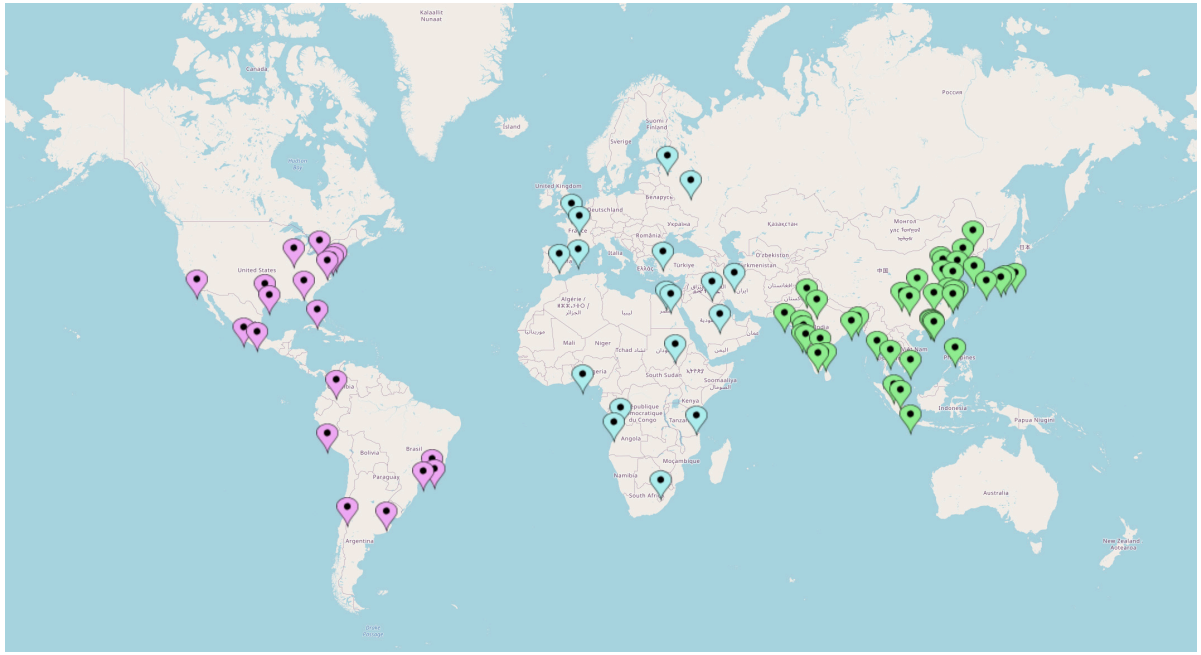
Feature Variables

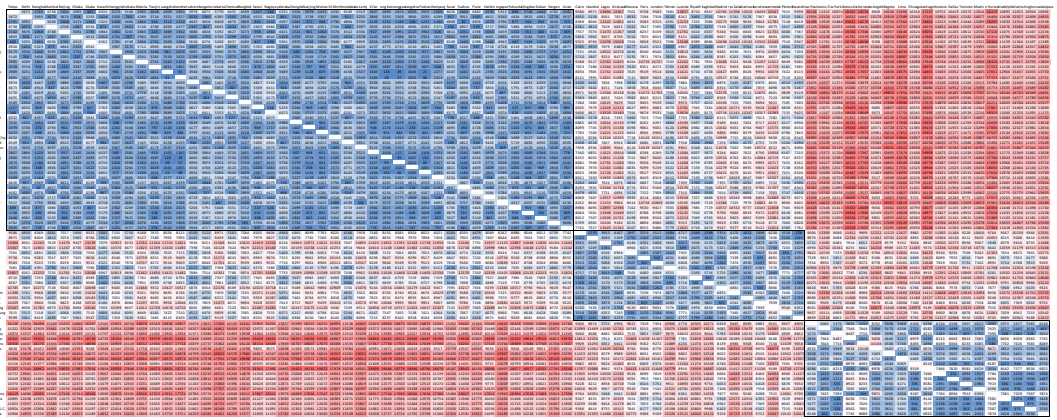
Label Variable

	A	B	C	D	E	F	G	H
1		PC1	PC2	PC3	PC4	Cluster		
2	Tokyo	-3.5132573	-1.4065444	-0.9610935	0.01762064	1		
3	Delhi	-3.1302536	-1.590207	0.51942704	-0.0629664	1		
4	Shanghai	-3.2867935	-1.8580371	-0.6662294	0.01000054	1		
5	Sao Paulo	-4.7957519	2.86727848	0.38560758	0.17843665	3		
6	Mexico City	-4.6791094	2.52864189	-1.0547855	0.07925811	3		
7	Cairo	-3.276456	0.01300147	1.00188438	-0.493151	2		
8	Mumbai	-3.2043953	-1.4001769	0.78989779	-0.0322014	1		
9	Beijing	-3.1658421	-1.707514	-0.5875577	-0.1525491	1		
10	Dhaka	-3.1663136	-1.7850166	0.29540707	0.02467551	1		
11	Osaka	-3.4558816	-1.5304897	-0.9127806	0.01321126	1		
12	New York City	-4.0497117	2.18002834	-0.6325919	-0.3740346	3		
13	Karachi	-3.1413089	-1.2032136	0.79505517	-0.1833854	1		
14	Buenos Aires	-5.616636	3.13876499	0.55304565	0.84297111	3		
15	Chongqing	-3.1710076	-1.9222255	-0.2582592	-0.007073	1		
16	Istanbul	-3.2095743	0.06655967	0.78088486	-0.6272818	2		
17	Kolkata	-3.1743656	-1.7566679	0.36696909	0.03130614	1		
18	Manila	-3.5908738	-1.8918189	-0.4421297	0.34608063	1		
19	Lagos	-3.9141078	1.17757002	1.13229433	-0.0815739	2		
20	Rio de Janeiro	-5.1879471	2.93388618	0.72091375	0.58029891	3		
21	Tianjin	-3.1748138	-1.7289498	-0.6048015	-0.1387346	1		
22	Kinshasa	-3.9668242	0.82485771	1.30244522	0.12547932	2		
23	Guangzhou	-3.3078036	-1.9947158	-0.372274	0.15794958	1		
24	Los Angeles	-4.392923	1.75074907	-1.255036	-0.0106216	3		
25	Moscow	-3.1359644	-0.1245199	0.40707264	-0.6818615	2		
26	Shenzhen	-3.324617	-1.9934672	-0.3818544	0.17185379	1		
27	Lahore	-3.0702305	-1.3408782	0.55336648	-0.2322427	1		
28	Bangalore	-3.321047	-1.5224082	0.76074325	0.12487839	1		
29	Paris	-3.3717408	0.74546634	0.49096091	-0.7010445	2		
30	Bogota	-4.9659616	3.09070871	-0.3653326	0.12984391	3		
31	Jakarta	-3.9497271	-1.7278463	0.22399271	0.68806507	1		
32	Chennai	-3.3231616	-1.58858	0.70200733	0.14628188	1		
33	Lima	-5.4148923	3.29467677	-0.2300196	0.45439614	3		

K-Means Clustering on City Data

Grouping in three clusters agrees with what could be intuitively grouped

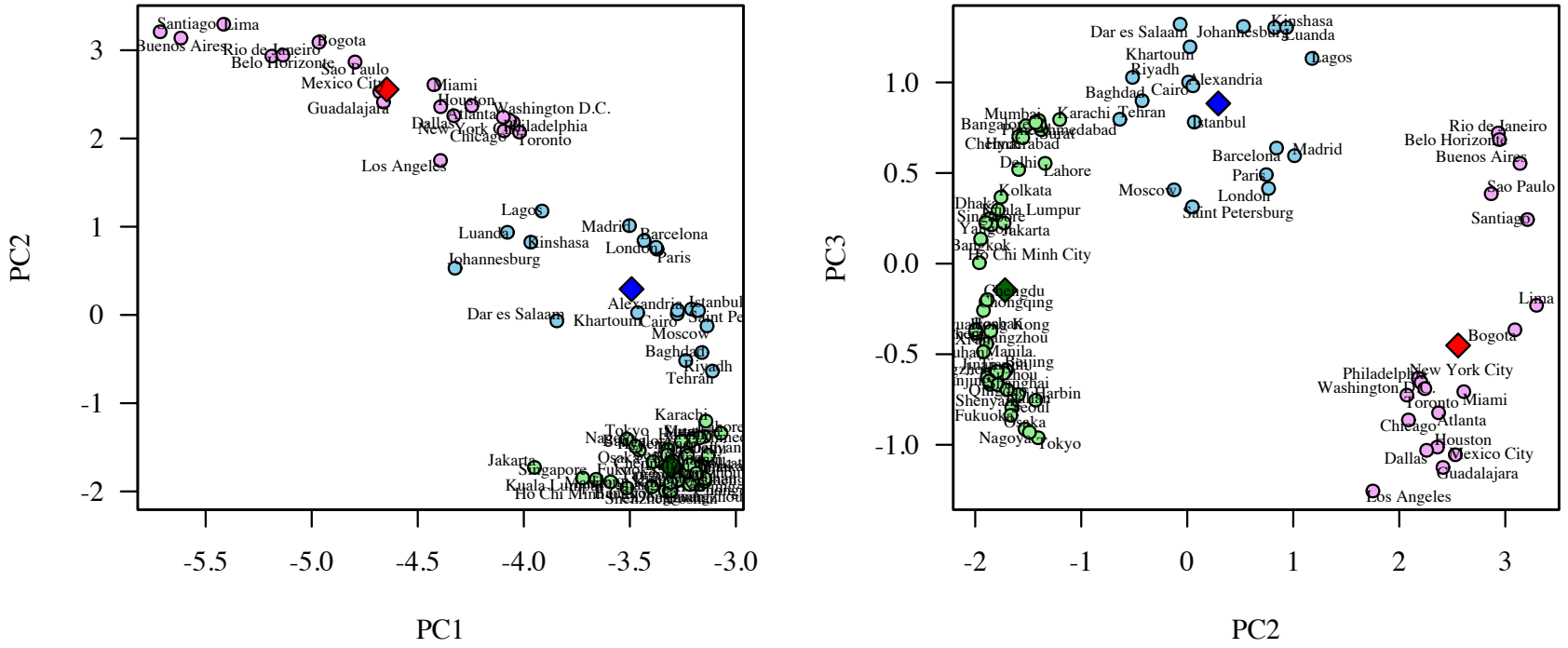




Original distance table sorted according to the three clusters

Blue indicates small and red large values

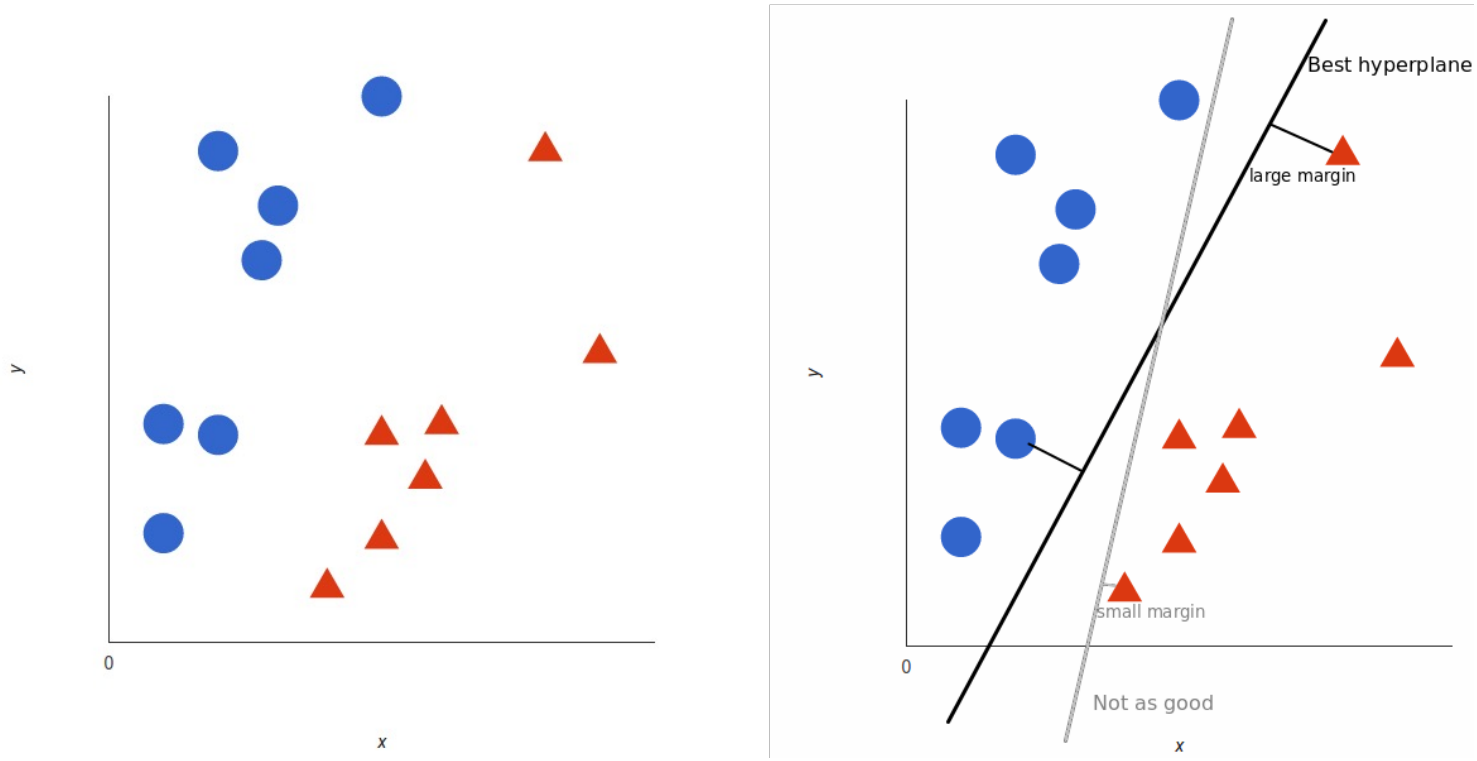
PCA Plots with Cluster Information



The clustering seems to be determined in PC2

The clusters can be used as label variables to train other machine learning algorithms to assign cities to any of the three clusters

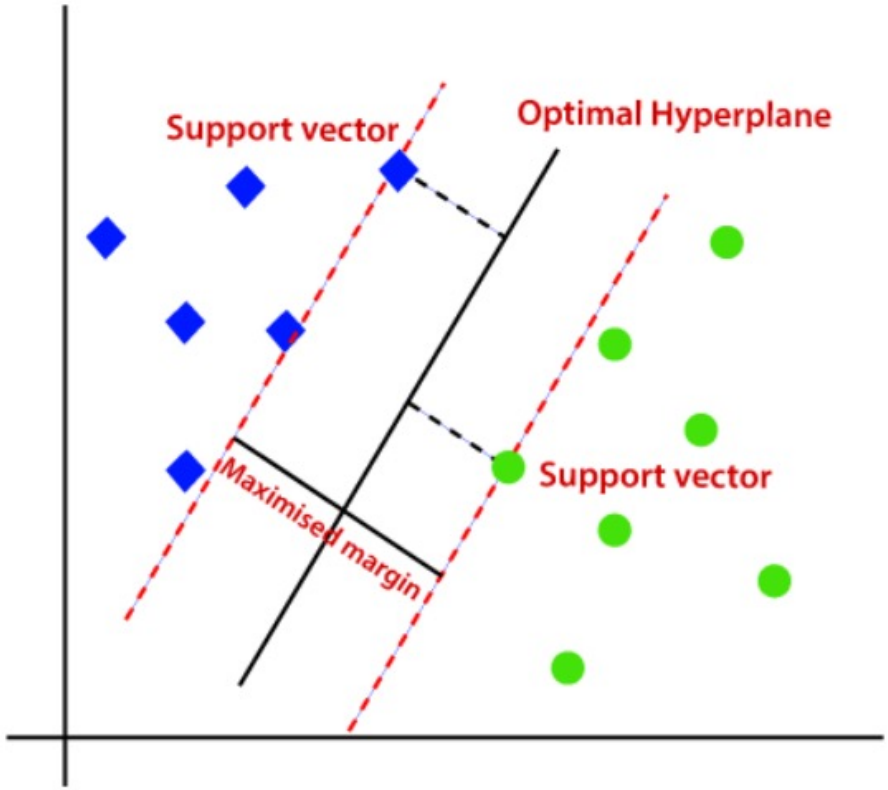
Support Vector Machine (SVM) for Classification



Hyperplane is introduced to separate two populations by the largest margin possible

Can be performed with any number of dimensions

Support Vector Machine (SVM) for Classification

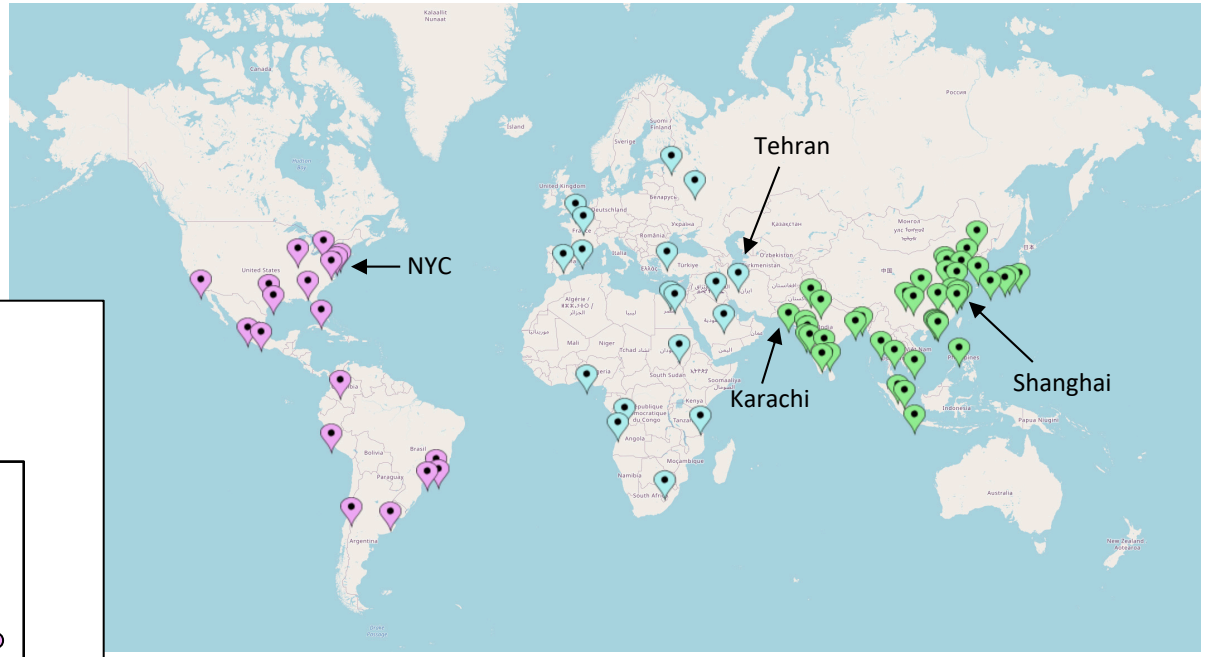


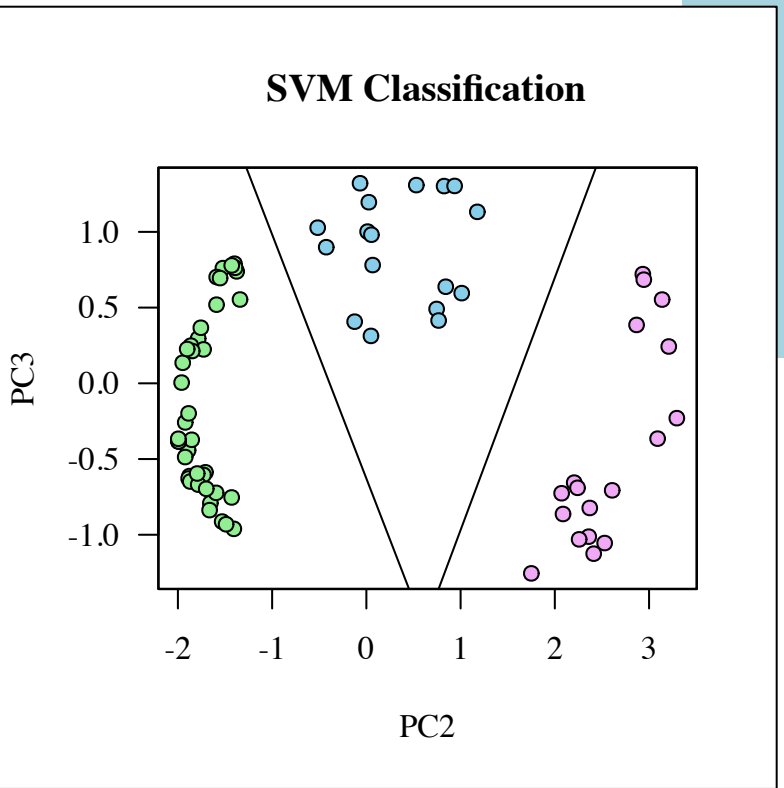
Hyperplane is defined by the nearest two points to either side (the support vectors)

SVM can be used for “hard” classifications. Every datapoint can be assigned to one of the populations without any probabilities

SVM with City Data

Test data (NYC, Tehran, Karachi & Shanghai) was removed from training data



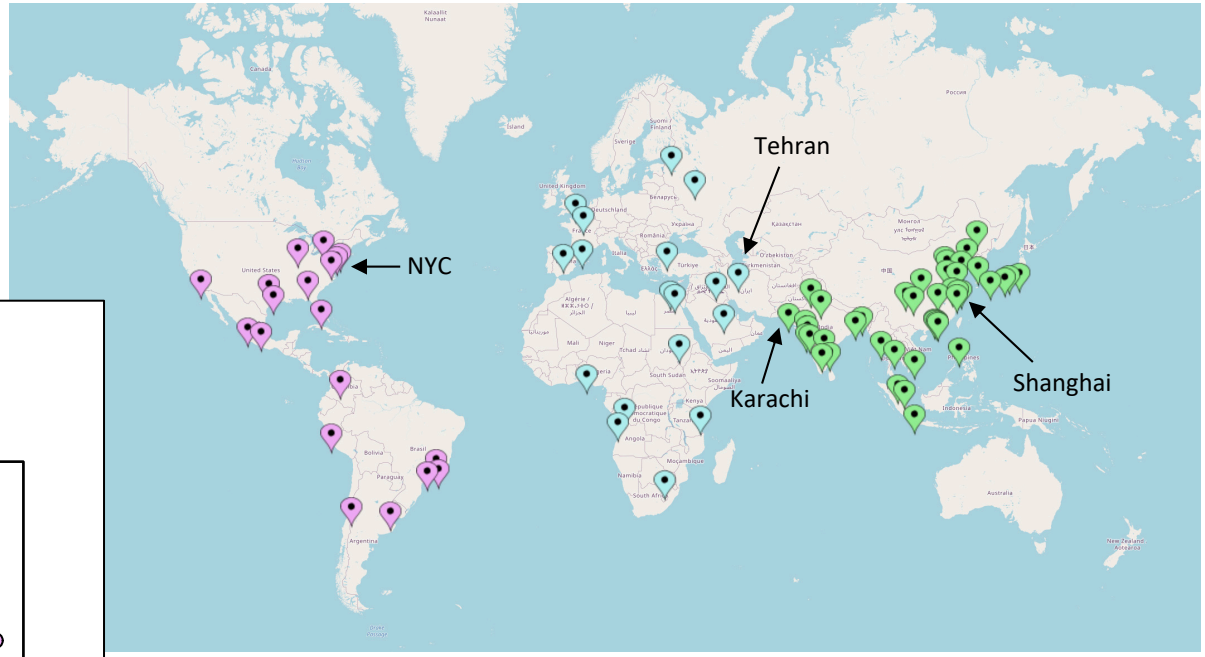


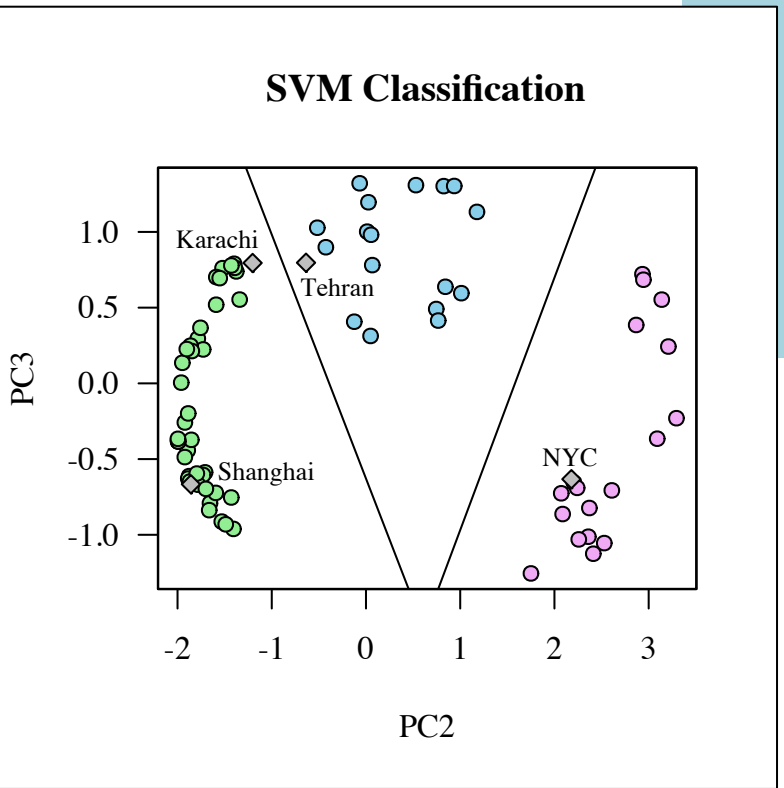
Hyperplanes between three groups were calculated in two rounds:

1. Group1 vs. non-Group1
2. Group3 vs non-Group3

SVM with City Data

Test data (NYC, Tehran, Karachi & Shanghai) was removed from training data

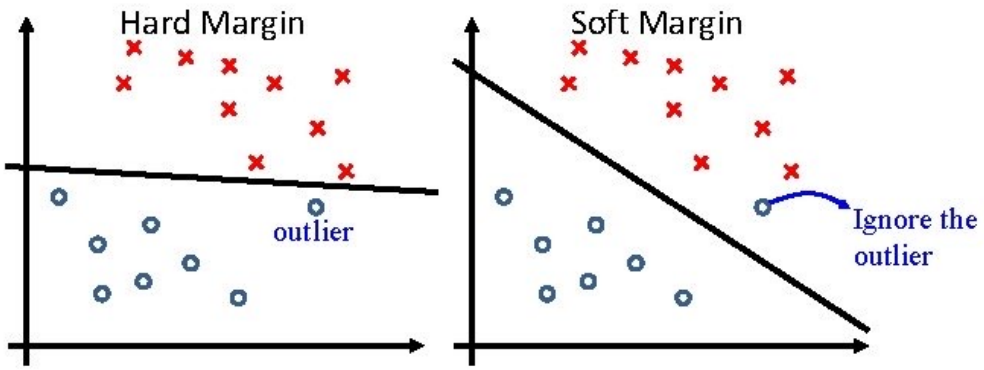




Test Cities can be assigned to corresponding clusters using SVM

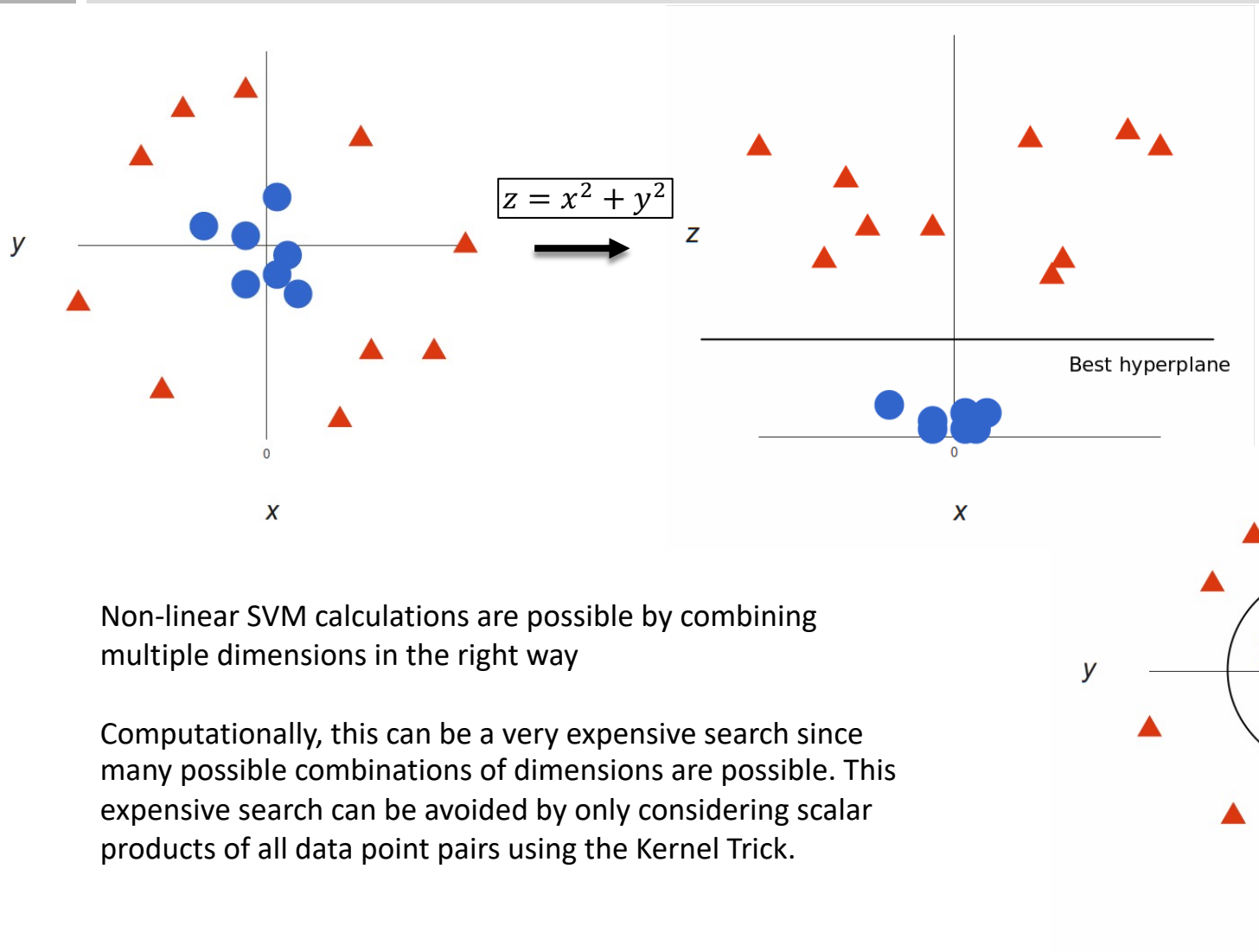
Hard vs Soft Margin SVM

SVM – Soft Margin



- **Hard Margin:**
  - If some training examples are outliers, separating all positive/negative examples may not be the best solution
- **Soft Margin:**
  - Tolerate some non-separable cases (outliers)

Non-linear SVM by Adding Dimensions



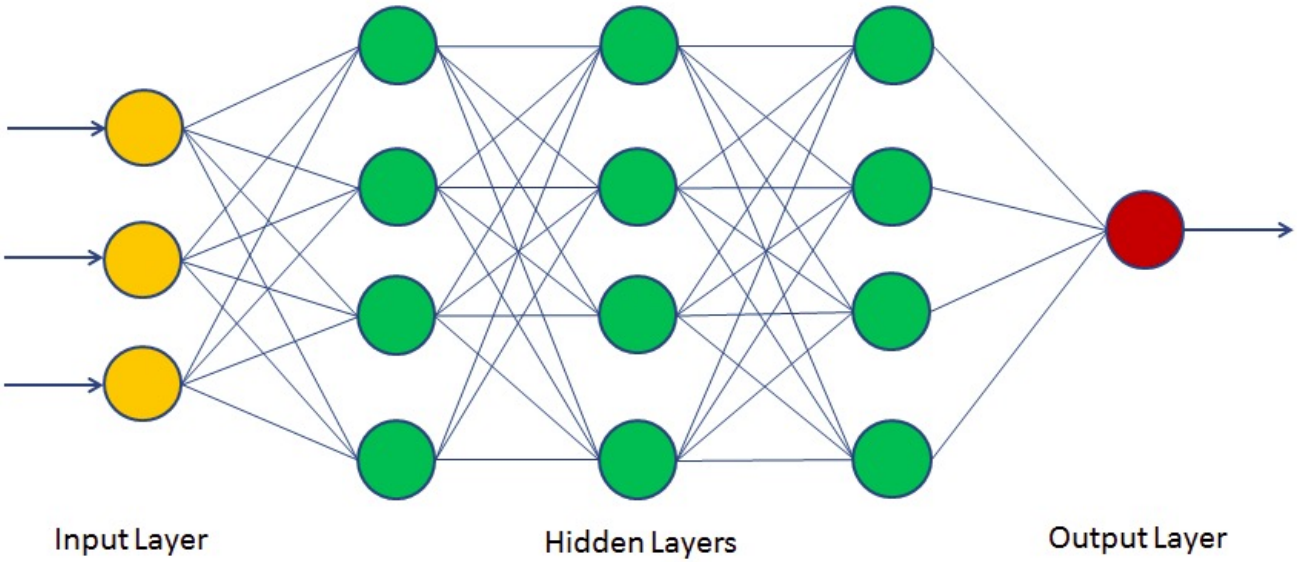
Non-linear SVM calculations are possible by combining multiple dimensions in the right way

Computationally, this can be a very expensive search since many possible combinations of dimensions are possible. This expensive search can be avoided by only considering scalar products of all data point pairs using the Kernel Trick.

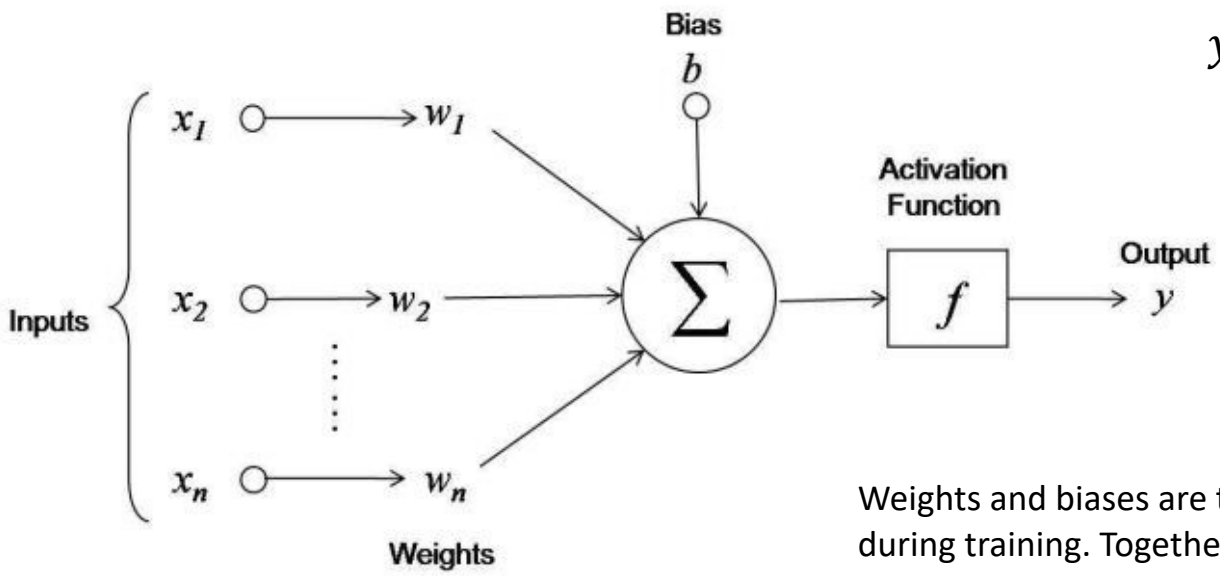
(Artificial) Neural Networks (NN)

The General Architecture of Neural Networks:

- Feedforward: Neurons in one layer only connect to neurons in the next layer
- Fully connected layers
- As many input neurons as feature variables
- Number of output neurons depend on the nature of the problem that needs to be solved (e.g. one for regression (if one label variable needs to be calculated), one or multiple for classification (corresponding to the number of classes))



Nodes of Neural Networks



$$y = f\left(b + \sum_{i=1}^{n} w_i x_i\right)$$

Weights and biases are the only modifiable parameters during training. Together, they form the weight matrix.

These variables are adjusted during training to reduce the cost function (representing the difference between computed and expected output)

Input neurons just transmit feature variables without any mathematical operations

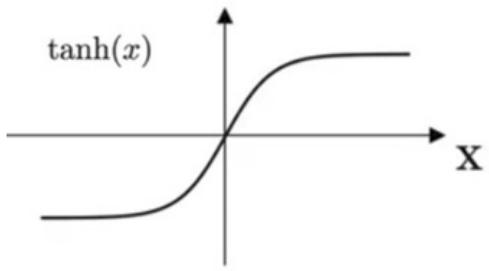
Activation Functions

Activation functions introduce non-linearity

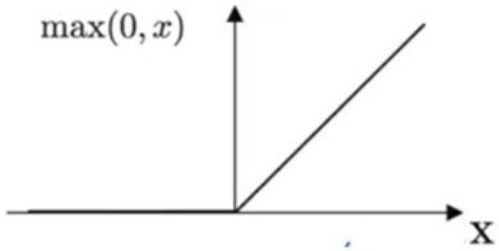
All neurons within same layer have the same activation function (layers are then usually named according to their activation function)

Data needs to be normalized to be within accessible range of the functions (e.g. from -3 to 3 for sigmoid activation function)

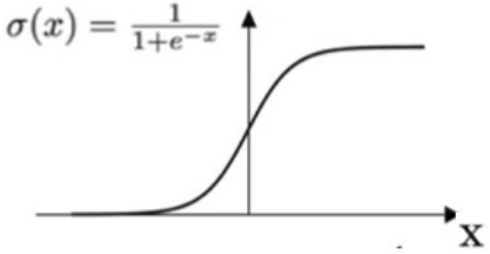
Hyper Tangent Function



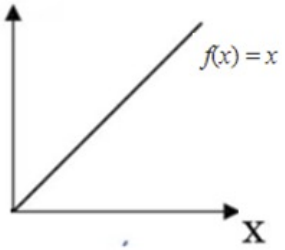
(Rectified Linear Unit)
ReLU Function



Sigmoid Function



Identity Function



Learning by Back-propagation and Gradient Descent

The cost function (e.g. RMSE, Cross Entropy) is the proxy for the error of the neural network. This function is continuous and smooth

Cost is minimized by optimizing the weight matrix during training

Cost gradients (differentiation of cost (and activation) function) for each variable in the weight matrix are first calculated in the output layer and then propagated back through the hidden layers

The cost gradient gives the change in the value of the cost when there is a small change in the weight

All weights are adjusted (in small steps) according to their cost gradients

Then cost gradients are recalculated and weights are readjusted iteratively, until the costs reach an acceptable level

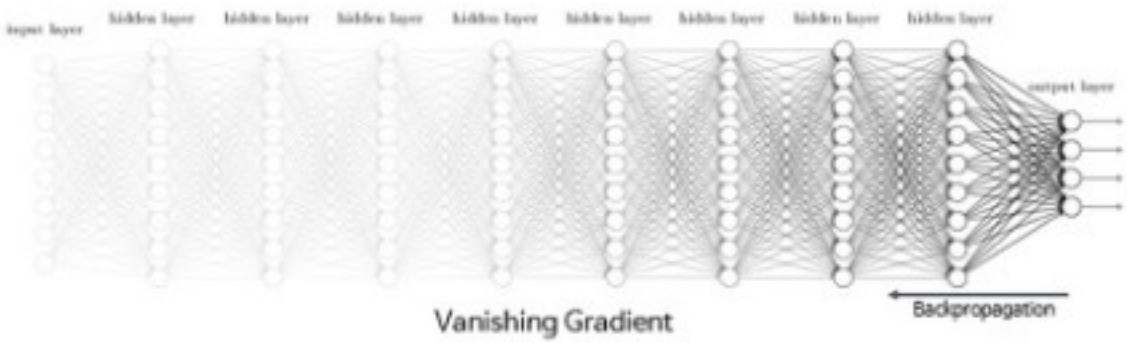
Neural Networks are nested mathematical functions. Weights closer to the output layer are less nested than weights far from the output layer. Gradients computed at the output layer are passed back through layers multiplying with local gradients (Can lead to vanishing gradients when many layers are present)

The learning rate hyperparameter determines the sizes of the weight adjustments. Too small and it will take too long to optimize, or algorithm might get stuck in local minimum, too large and minima might be missed

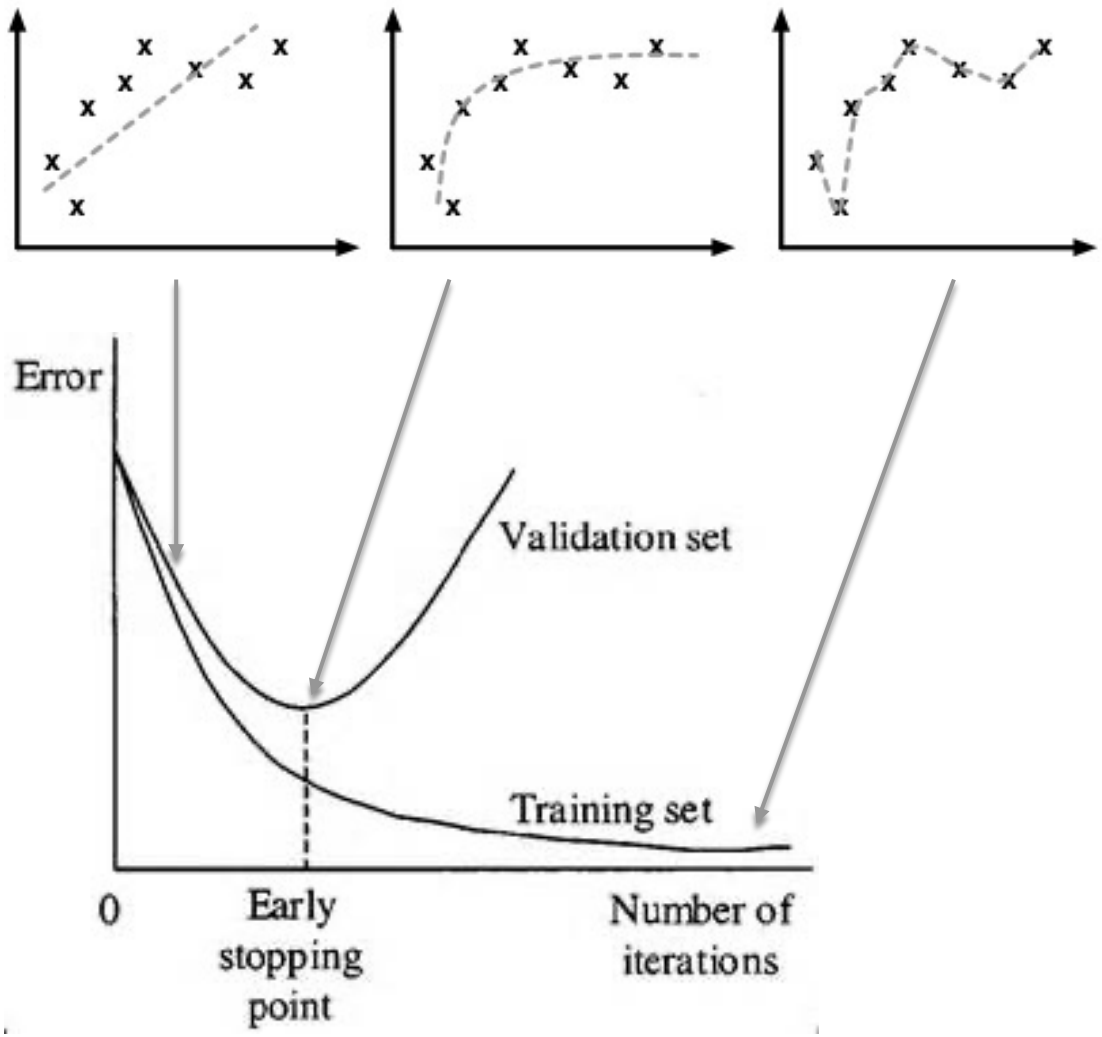
Vanishing Gradients

For many of the activation functions, cost gradients are less than 1

If multiplied together during back-propagation, the gradient disappears with too many hidden layers, and learning in layers close to the input layer becomes negligible -> It is better not to use too many hidden layers (Which also reduces the risks of overfitting due to too many variables)



Validation to Prevent Overfitting

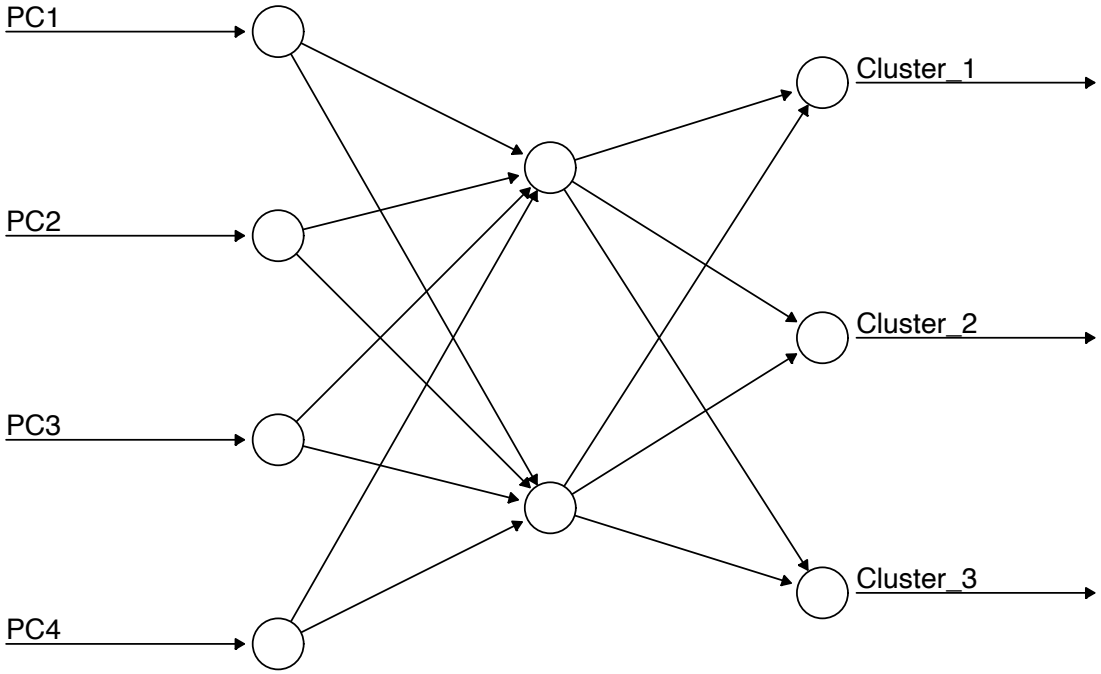


A part of the data is removed from the training data set to validate the training

Overfitting is present when the error of the validation set increases while the error of the training set further decreases (Marking the transition from learning to memorizing)

Overfitting can (for example) be reduced by adding additional restraints using regularization methods: The sum of (squares of) all weights are minimized together with the cost function

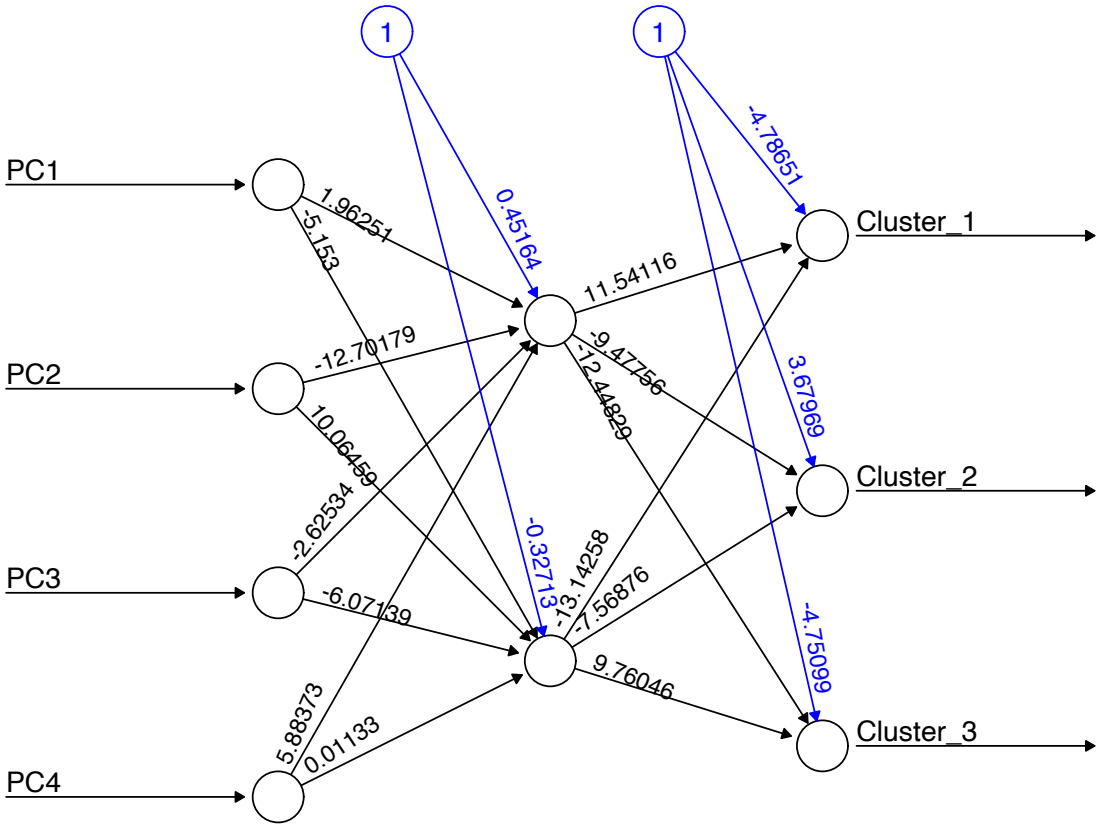
Neural Network for City Data Classification



Hidden and output layers are sigmoidal layers

Output neurons give values between 0 and 1 (Sigmoid Activation Function), which can be regarded as corresponding to the probability for a data point belonging to a certain cluster

Resulting Neural Network

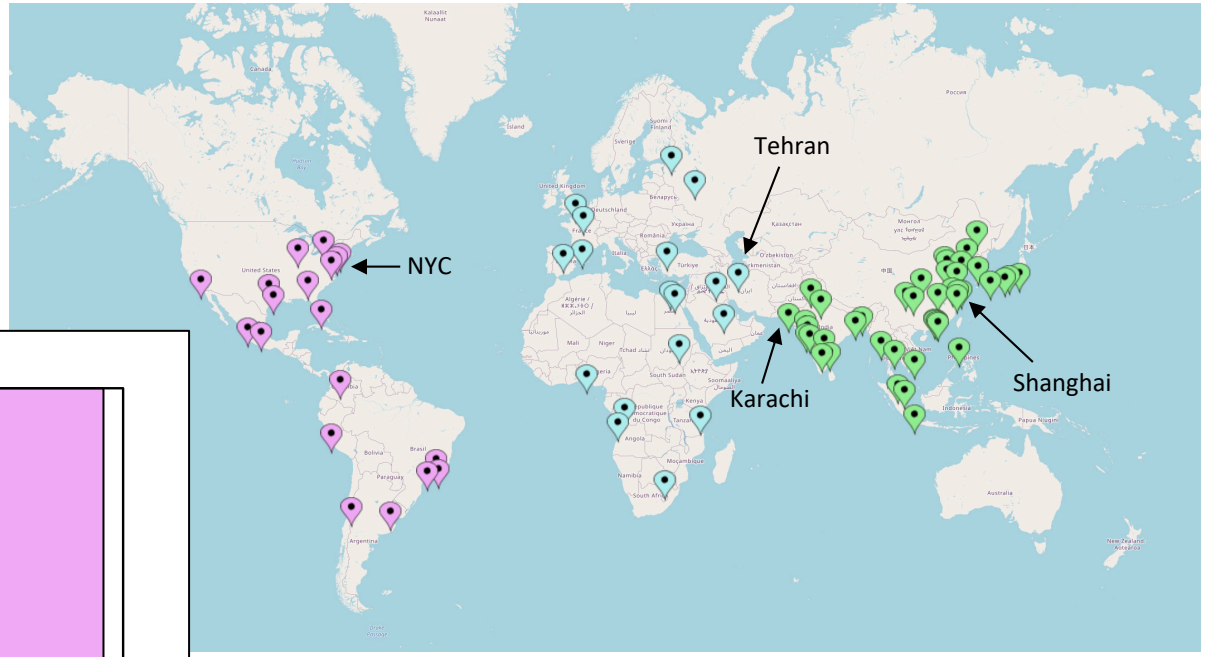


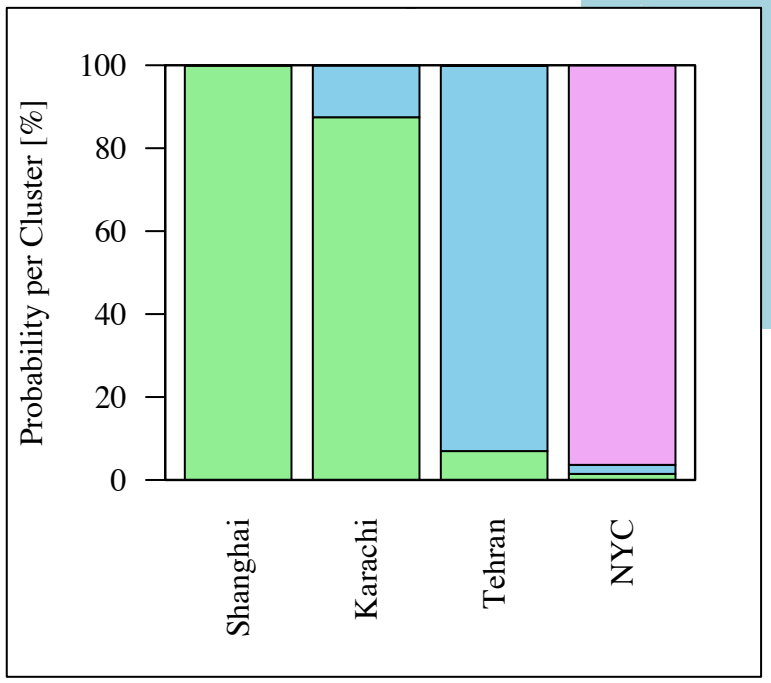
Error: 0.025973 Steps: 119

PC2 is weighted most for hidden layer

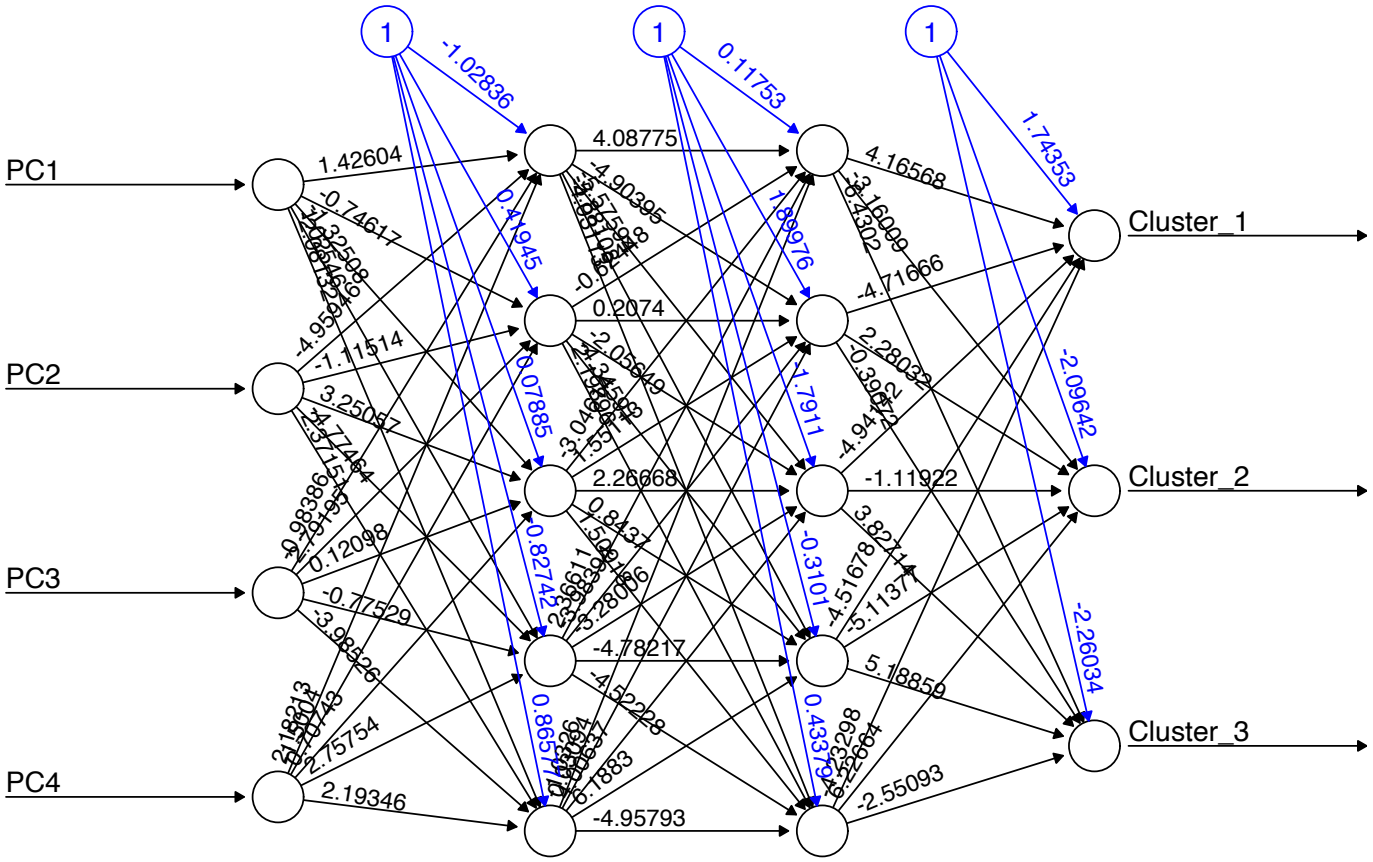
Neural Networks for City Data Classification

Test data (NYC, Tehran, Karachi & Shanghai) was removed from training data



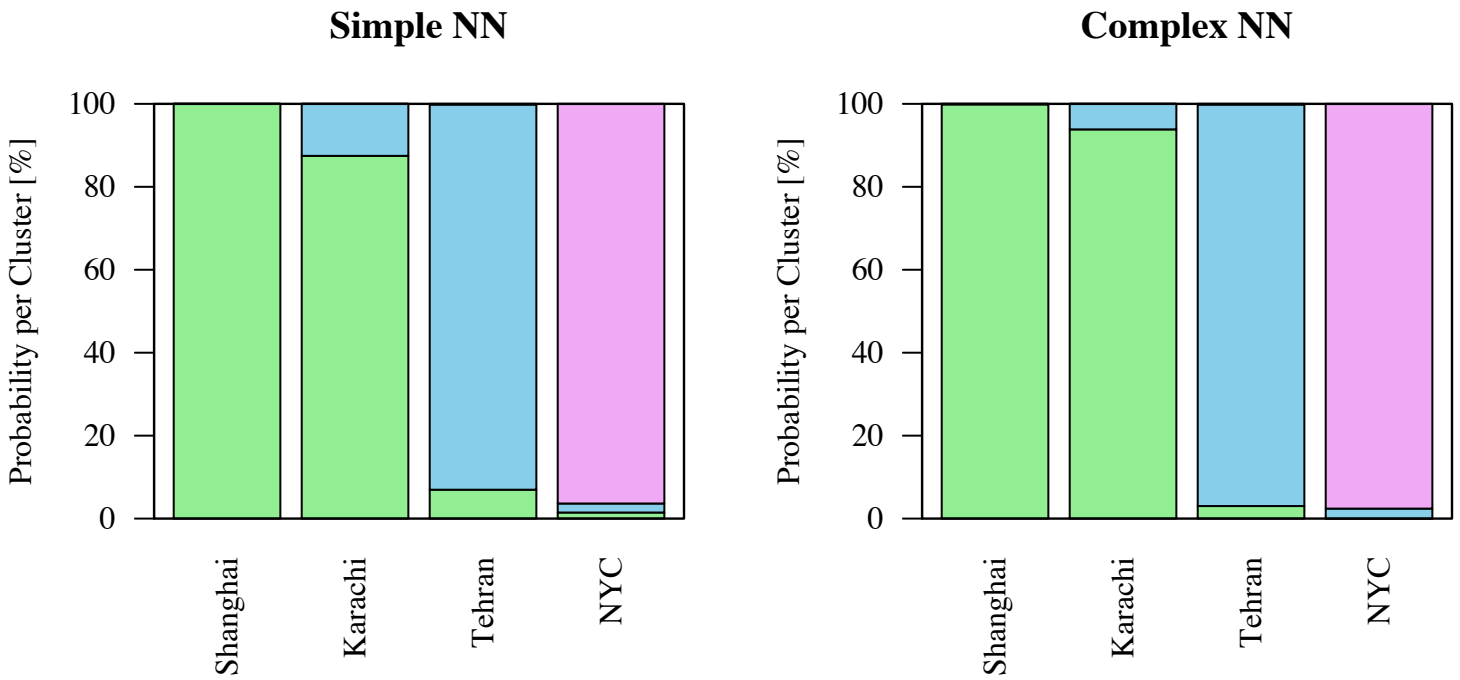


A Neural Network with more Neurons



Error: 0.010465 Steps: 54

Neural Networks for City Data



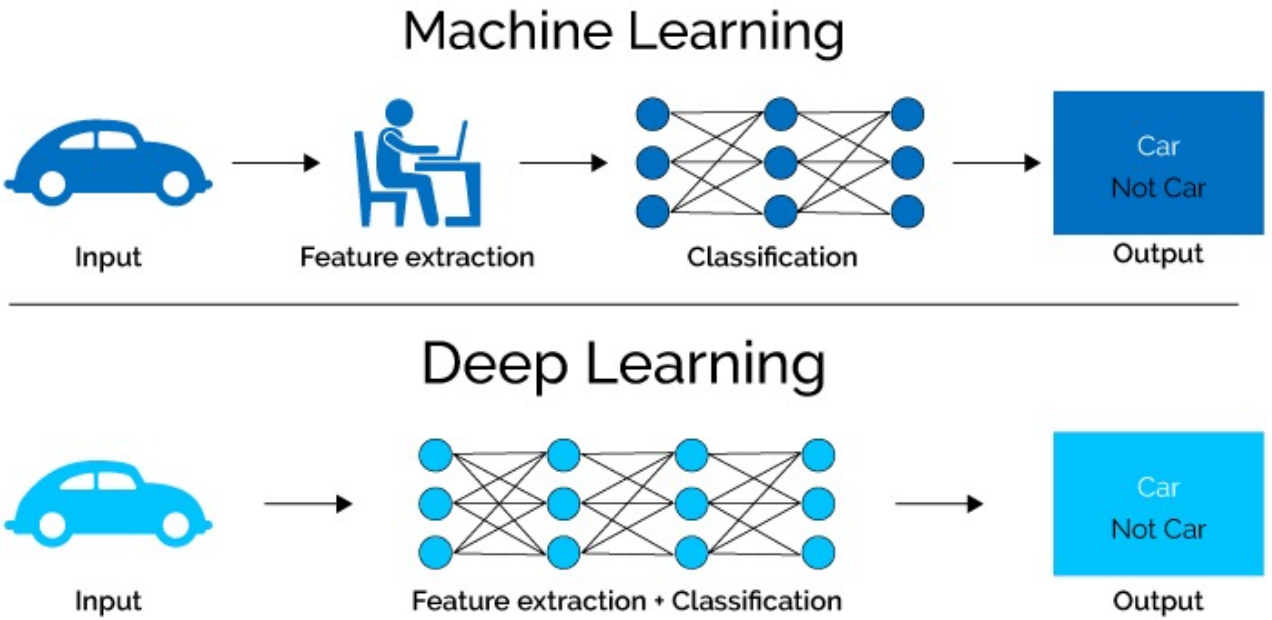
Complexity does not necessarily increase the power of the prediction by a lot

(Prediction success also varied between runs)

Deep Learning

Deep learning algorithms are based on neural networks, but have a more complex architecture

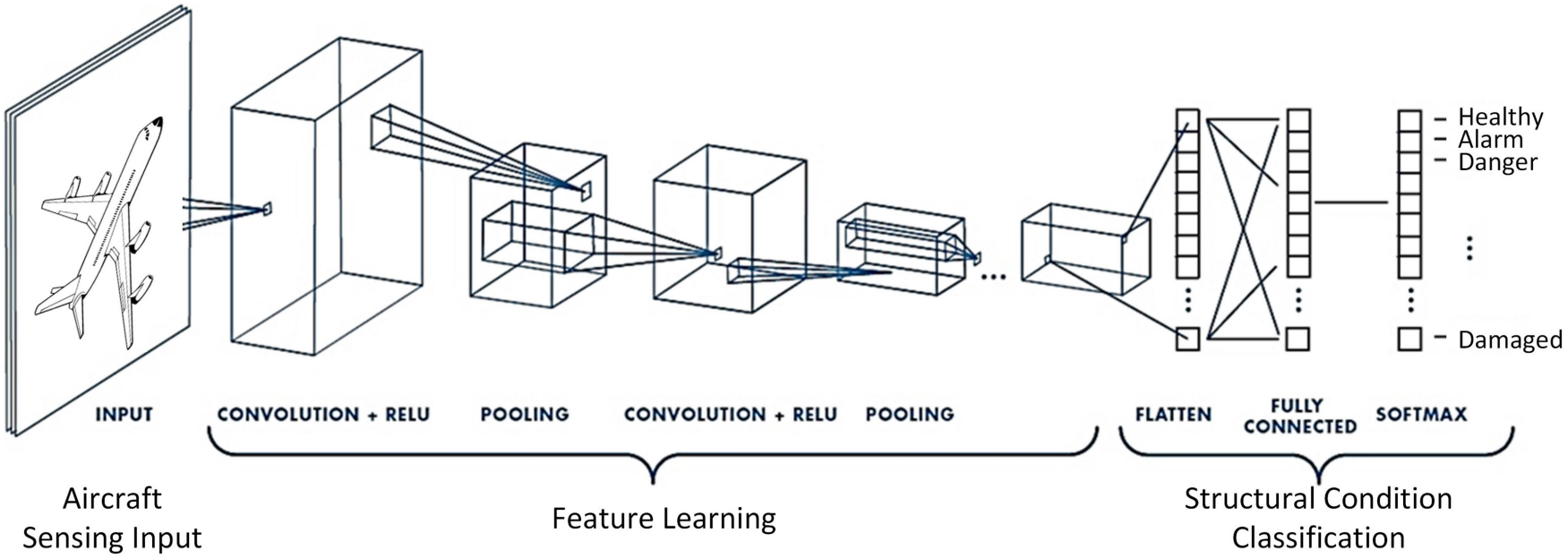
Deep learning is much better suited to handle unstructured data, temporal/sequential relationships and spatial relationships



Convolutional Neural Networks (CNNs)

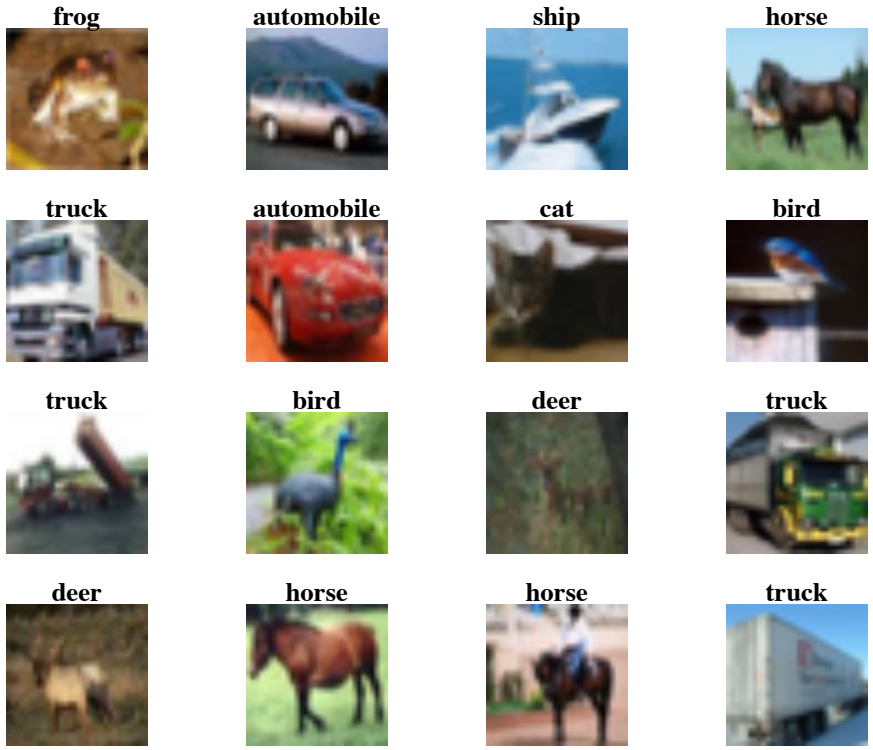
CNNs are able to process spatial data using four main types of layers:

- **Convolutional Layer**: Pixel data is transmitted in the form of small and overlapping subframes to single neurons
- **Pooling Layer**: Compressing several neurons (without overlap) from convolutional layer into single neuron
- **Fully Connected Layer**: Usually a neural network using also the ReLU activation function
- **Softmax Layer**: Produces output for classification problems with probabilities (Sum of Softmax Layer equals 1)



Tabian, Fu & Khodaei (2019)

Convolutional Neural Networks (CNN) - CIFAR

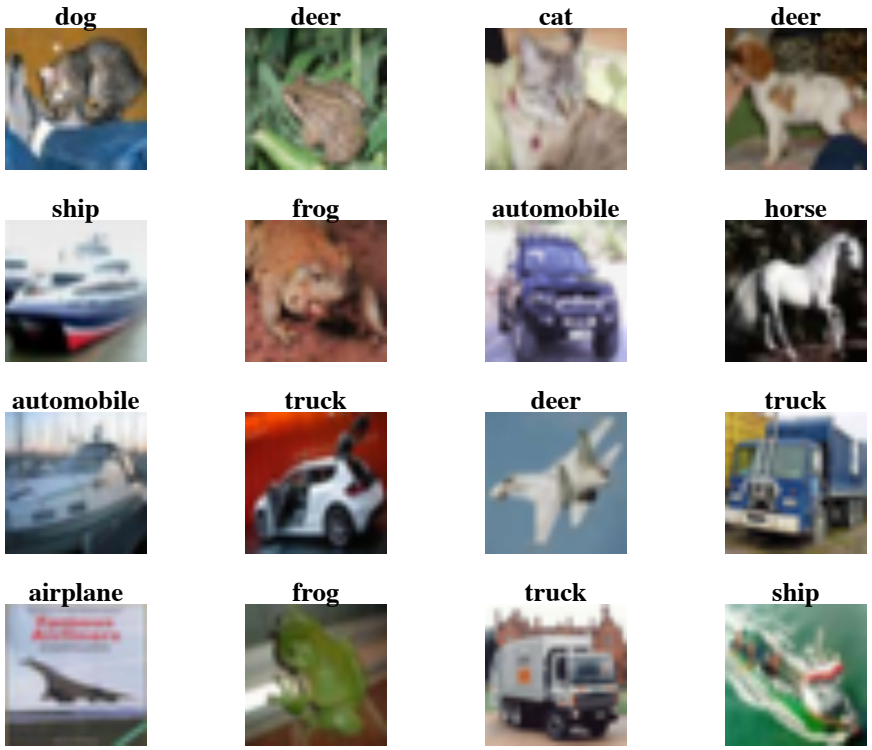


CIFAR10 dataset contains 60’000 images from 10 different classes. 50’000 of the images are used for training and 10’000 for validation. Each picture has 32x32 pixels.

Example is based on the packages tensorflow and keras. They can be used with R, but they run calculations in python

There were problems with the M1 chip and I had to use windows for this example

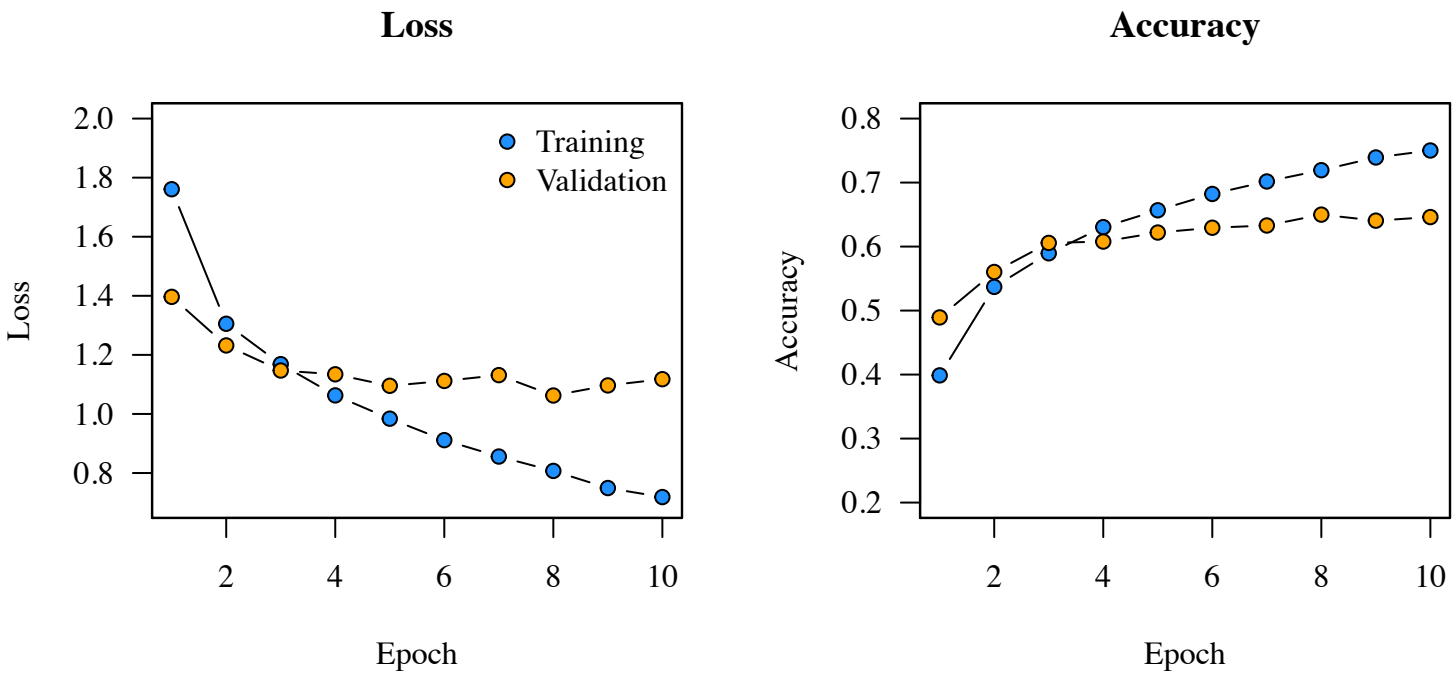
CNN - 1



The Architecture of this CNN:
Conv. (3x3) -> Max Pooling (2x2) -> Conv. (3x3) ->
Max Pooling (2x2) -> Conv. (3x3) ->
Flat Layer -> ReLU Layer (64 neurons) ->
Softmax Layer (10 neurons)

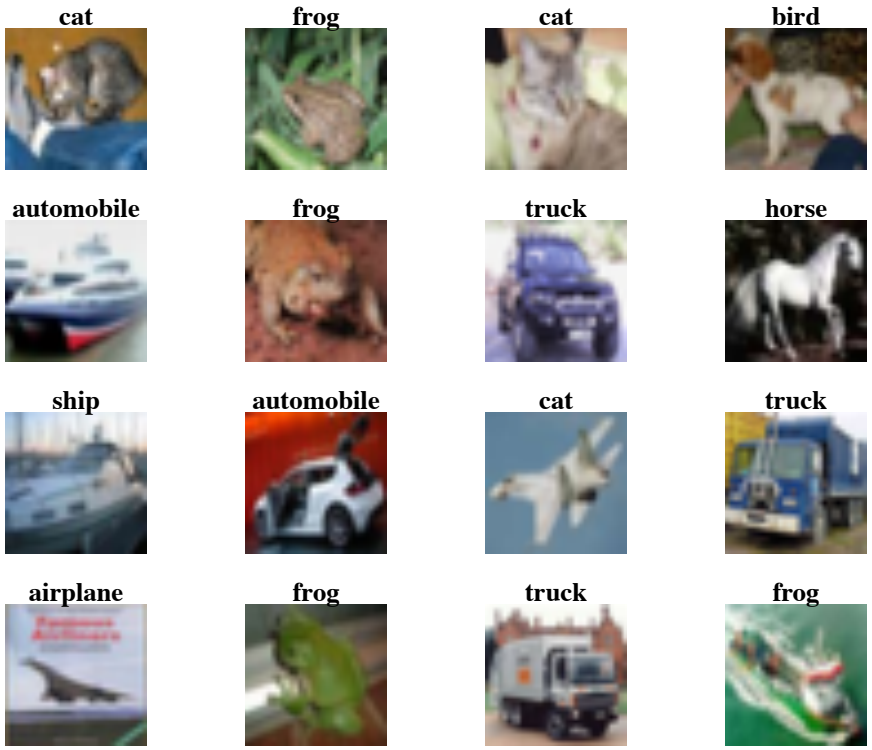
```
34
35 model <- keras_model_sequential() %>%
36   layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
37                 input_shape = c(32,32,3)) %>%
38   layer_max_pooling_2d(pool_size = c(2,2)) %>%
39   layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
40   layer_max_pooling_2d(pool_size = c(2,2)) %>%
41   layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu")
42
43 model %>%
44   layer_flatten() %>%
45   layer_dense(units = 64, activation = "relu") %>%
46   layer_dense(units = 10, activation = "softmax")
47
48 model %>% compile(
49   optimizer = "adam",
50   loss = "sparse_categorical_crossentropy",
51   metrics = "accuracy"
52 )
```

CNN - 1



The classification of the validation data doesn't improve anymore after a few steps, whereas the training data still does, indicating memorizing instead of learning

CNN - 2

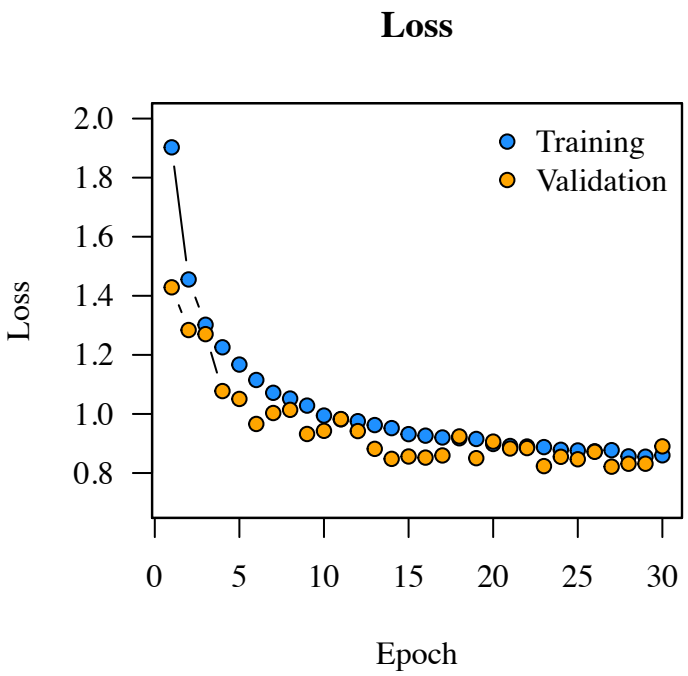


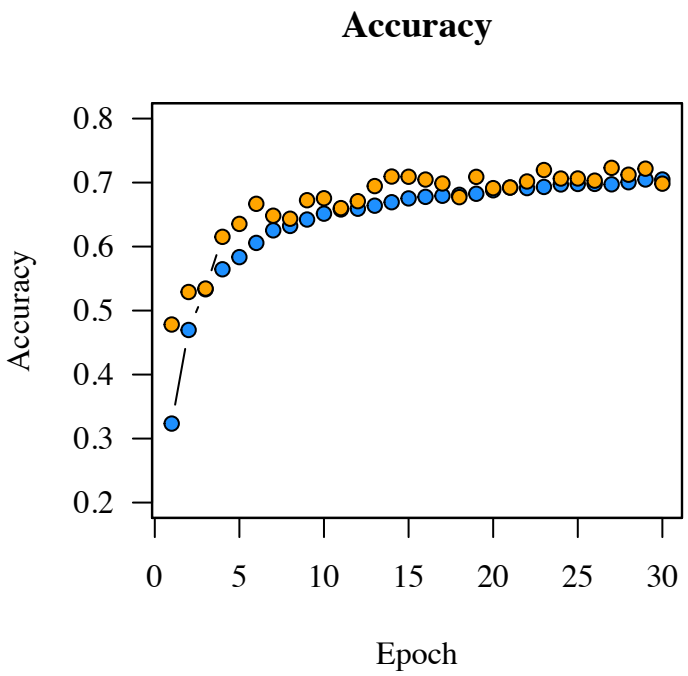
This CNN was slightly more complex:
Conv. (3x3) -> Conv. (3x3) -> Max Pooling (2x2) -> Conv. (3x3) -> Conv. (3x3) -> Max Pooling (2x2) -> Flat Layer -> ReLU Layer (512 neurons) -> Softmax Layer (10 neurons)

With Dropouts to improve learning -> Some nodes in hidden layers are randomly dropped in every learning iteration

```
35
36  model <- keras_model_sequential() %>%
37    layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
38                  input_shape = c(32,32,3), padding = "same") %>%
39    layer_conv_2d(filters  = 32, kernel_size = c(3,3), activation = "relu") %>%
40    layer_max_pooling_2d(pool_size = c(2,2)) %>%
41    layer_dropout(0.25) %>%
42    layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
43                  padding = "same") %>%
44    layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu") %>%
45    layer_max_pooling_2d(pool_size = c(2,2)) %>%
46    layer_dropout(0.25)
47
48  model %>%
49    layer_flatten() %>%
50    layer_dense(units = 512, activation = "relu") %>%
51    layer_dropout(0.5) %>%
52    layer_dense(units = 10, activation = "softmax")
53
54  model %>% compile(
55    optimizer = "adam",
56    loss = "sparse_categorical_crossentropy",
57    metrics = "accuracy"
58  )
```

CNN - 2





The more complex CNN seems to work much better when compared to the validation data. The overall accuracy is around 70%, which seems good for just a few lines of code.

Thank you for your attention!