# CS-660 Data Mining

# **Project - Mice Protein Expression**


B17096 Namrata Malkani

D19062 Geetanjali

B16065 Navneet Sharma

B17031 Aashima

# Problem Statement

For a mice, **80 features** are measured ( 77 proteins/protein modifications + genotype + behavior + treatment ). Given these features, **classify** it into one of the **8 classes**.

**Dataset source** : https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression

Clara Higuera Department of Software Engineering and Artificial Intelligence, Faculty of Informatics and the Department of Biochemistry and Molecular Biology, Faculty of Chemistry, University Complutense, Madrid, Spain.

Katheleen J. Gardiner, creator and owner of the protein expression data, is currently with the Linda Crnic Institute for Down Syndrome, Department of Pediatrics, Department of Biochemistry and Molecular Genetics, Human Medical Genetics and Genomics, and Neuroscience Programs, University of Colorado, School of Medicine, Aurora, Colorado, USA.

Krzysztof J. Cios is currently with the Department of Computer Science, Virginia Commonwealth University, Richmond, Virginia, USA, and IITiS Polish Academy of Sciences, Poland.

# Data Description

The data set consists of the expression levels of 77 proteins/protein modifications that produced detectable signals in the nuclear fraction of cortex**.** There are 38 control mice and 34 trisomic mice (Down syndrome), for a total of 72 mice. In the experiments, 15 measurements were registered of each protein per sample/mouse. Therefore, for control mice, there are 38x15, or 570 measurements, and for trisomic mice, there are 34x15, or 510 measurements. The dataset contains a total of 1080 measurements per protein. Each measurement can be considered as an independent sample/mouse.The eight classes of mice are described based on features such as genotype, behavior and treatment. According to genotype, mice can be control or trisomic. According to behavior, some mice have been stimulated to learn (context-shock) and others have not (shock-context) and in order to assess the effect of the drug memantine in recovering the ability to learn in trisomic mice, some mice have been injected with the drug and others have not.

| | |
|---|---|
| Instances | 1080 |
| Features | 80<br><br>77 proteins (Numerical)<br>78 Genotype (control, trisomy)<br>79 Treatment (memantine, saline)<br>80 Behavior (context-shock, shock-context) |
| Classes | 8<br><br>(c-CS-m, c-CS-m, c-SC-s, c-Sc-m, t-CS-s, t-CS-m, t-SC-s, t-SC-m) |
| Missing values | Yes |
| Associated Tasks | Classification<br>Clustering |

# Data Handling

**Data Split**

| | | |
|---|---|---|
| Training Data | 60 % | 640 |
| Validation Data | 10 % | 120 |
| Testing Data | 30 % | 320 |

**Shuffle Training Samples**

Model sees variety of samples even in a small batch

**Data Split**

| | | |
|---|---|---|
| Training Data | 60 % | 640 |
| Validation Data | 10 % | 120 |
| Testing Data | 30 % | 320 |

**Class Imbalance**

Take equal number of samples per class

80 samples / class

Equal weightage to each class

**Shuffle Training Samples**

Model sees variety of samples even in a small batch

Handle Missing Values in all - Training, Validation and Testing Data

**Missing Values**

Weka Explorer --> Preprocess --> Filter --> weka --> filters --> unsupervised --> attribute --> **ReplaceMissingValues**

ReplaceMissingValues filter replaces missing values with median or mean of the respective attribute values.

# Classification Algorithms

# Naive Bayes

- Machine learning algorithm based on Bayes theorem.
- **Applications** - Classification, filtering spam, sentiment prediction etc.

$$\underbrace{P(c \mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x \mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

# Evaluation metrics

- **Confusion matrix.**

1. Precision
2. Recall
3. F1-score
4. TP Rate
5. FP Rate
6. MCC
7. ROC Area
8. PRC Area

| | | Actual Values | |
|---|---|---|---|
| | | Positive (1) | Negative (0) |
| Predicted Values | Positive (1) | TP | FP |
| | Negative (0) | FN | TN |

# Evaluation on Training dataset

| Evaluation on Train Dataset | |
|---|---|
| Correctly Classified Instances | 585 i.e 91.4063 % |
| Incorrectly Classified Instances | 55 i.e 8.5938 % |
| Mean absolute error | 0.0213 |
| Root mean squared error | 0.1379 |
| Total Number of Instances | 640 |

# Evaluation on Test dataset

| Evaluation on test dataset | |
|---|---|
| Correctly Classified Instances | 158 i.e 49.375 % |
| Incorrectly Classified Instances | 162 i.e 50.625 % |
| Mean absolute error | 0.1291 |
| Root mean squared error | 0.3522 |
| Total Number of Instances | 320 |
| Testing time | 0.11 seconds |

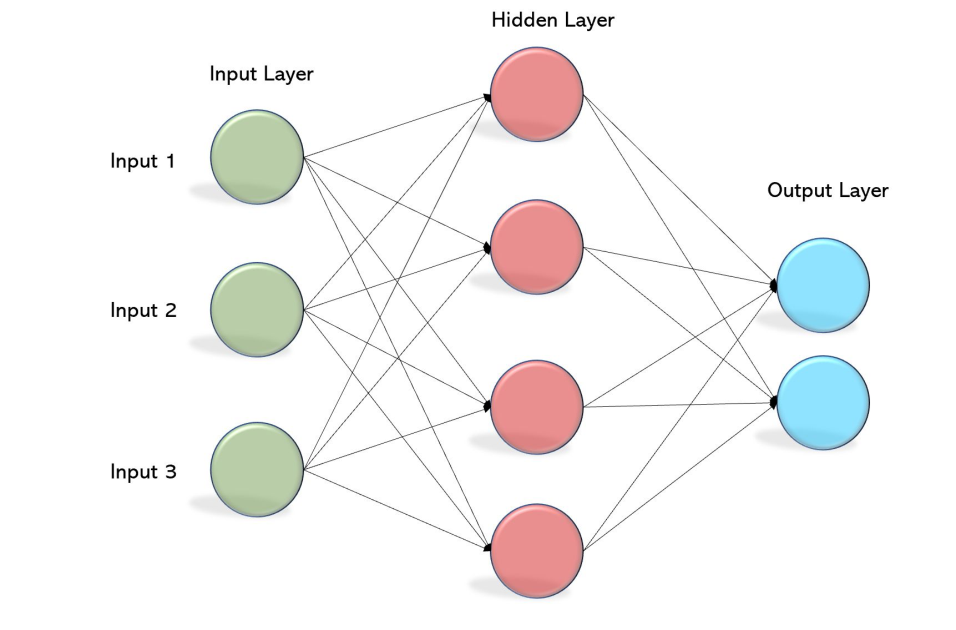
# Confusion Matrix

| Confusion Matrix | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Predicted Class | | | | | | | | |
| | | c-CS-m | c-CS-s | c-SC-m | c-SC-s | t-CS-m | t-CS-s | t-SC-m | t-SC-s |
| Actual Class | c-CS-m | **26** | 3 | 0 | 0 | 22 | 4 | 0 | 0 |
| | c-CS-s | 12 | **7** | 0 | 0 | 11 | 10 | 0 | 0 |
| | c-SC-m | 0 | 0 | **35** | 15 | 0 | 0 | 5 | 0 |
| | c-SC-s | 0 | 0 | 7 | **6** | 0 | 1 | 11 | 15 |
| | t-CS-m | 10 | 0 | 0 | 0 | **10** | 12 | 0 | 8 |
| | t-CS-s | 0 | 0 | 0 | 0 | 0 | **10** | 0 | 0 |
| | t-SC-m | 0 | 0 | 11 | 1 | 0 | 0 | **28** | 0 |
| | t-SC-s | 0 | 0 | 1 | 0 | 0 | 0 | 3 | **36** |
| | Accuracy = 26+7+35+6+10+10+28+36<br>= 158 (no of samples are correctly classified) / 320 (Total number of samples)<br>= 49.375 | | | | | | | | |

# Classification Results in Detail

| Detailed Accuracy by Class | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area | Class |
| 0.473 | 0. 083 | 0.542 | 0.473 | 0.505 | 0.412 | 0.857 | 0.614 | c-CS-m |
| 0.175 | 0. 011 | 0.700 | 0.175 | 0.280 | 0.312 | 0.933 | 0.612 | c-CS-s |
| 0.636 | 0. 072 | 0.648 | 0.636 | 0.642 | 0.569 | 0.926 | 0.636 | c-SC-m |
| 0. 150 | 0. 057 | 0.273 | 0.150 | 0.194 | 0.121 | 0.491 | 0.211 | c-SC-s |
| 0. 250 | 0. 118 | 0.233 | 0.250 | 0.241 | 0.128 | 0.800 | 0.251 | t-CS-m |
| 1. 000 | 0. 087 | 0.270 | 1.000 | 0.426 | 0.497 | 0.976 | 0.438 | t-CS-s |
| 0. 700 | 0. 068 | 0.596 | 0.700 | 0.644 | 0.591 | 0.923 | 0.488 | t-SC-m |
| 0. 900 | 0. 082 | 0.610 | 0.900 | 0.727 | 0.697 | 0.960 | 0.888 | t-SC-s |

# weka.classifiers.functions.MultilayerPerceptron



## Feedforward

Signals from layer L flow to only layers > L and not to layers <= L

## Strictly Layered

Signals from layer L flow only to layer L+1

## Fully Connected

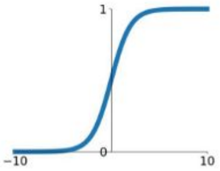
All nodes in layer L are connected to all nodes in layer L+1

## Hyperparameters

- 0 < Learning Rate < 1
- 0 < Momentum < 1
- Number of hidden layers
- Number of hidden nodes
- Number of epochs
- Batch size

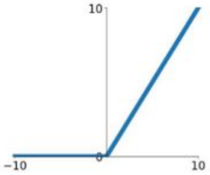
**Sigmoid**

$$\sigma(x) = \frac{1}{1+e^{-x}}$$




**ReLU**

$$\max(0, x)$$




## Backpropagation

Compute gradients of the loss function with respect to each weight using the chain rule of differentiation. Compute the gradient of one layer at a time, iterating it backward from the last layer.

## Loss Function

It is a mathematical way of measuring how wrong our predictions are.

weka.classifiers.functions.MultilayerPerceptron

Default

| | # Hidden Layers | Hidden Nodes | lr | m | Training Accuracy | Evaluation Time |
|---|---|---|---|---|---|---|
| | 1 | ( 52 ) | 0.3 | 0.2 | 100 % | 0.4 sec |
| A | 1 | ( 5 ) | 0.3 | 0.2 | 100 % | 0 sec |
| | 1 | ( 1 ) | 0.3 | 0.2 | 50 % | 0 sec |
| | 2 | ( 1 , 1 ) | 0.3 | 0.2 | 50 % | 0 sec |
| | 2 | ( 1 , 5 ) | 0.3 | 0.2 | 37.5 % | 0 sec |
| B | 1 | ( 2 ) | 0.3 | 0.2 | 100 % | 0 sec |

| Input Nodes | Output Nodes | Epochs | Batch Size |
|---|---|---|---|
| 80 | 8 | 500 | 100 |

weka.classifiers.functions.MultilayerPerceptron

| | # Hidden Layers | Hidden Nodes | lr | m | Training Accuracy | Evaluation Time |
|---|---|---|---|---|---|---|
| A | 1 | ( 5 ) | 0.3 | 0.2 | 100 % | 0 sec |
| B | 1 | ( 2 ) | 0.3 | 0.2 | 100 % | 0 sec |

| | # Hidden Layers | Hidden Nodes | lr | m | Validation Accuracy | Evaluation Time |
|---|---|---|---|---|---|---|
| A | 1 | ( 5 ) | 0.3 | 0.2 | 100 % | 0 sec |
| B | 1 | ( 2 ) | 0.3 | 0.2 | 100 % | 0 sec |

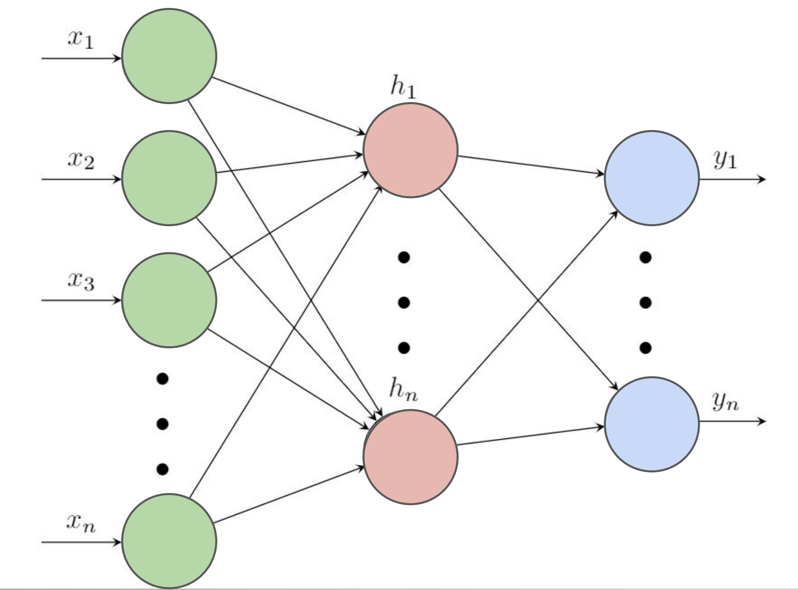
| Input Nodes | Output Nodes | Epochs | Batch Size |
|---|---|---|---|
| 80 | 8 | 500 | 100 |

# weka.classifiers.functions.MultilayerPerceptron

80

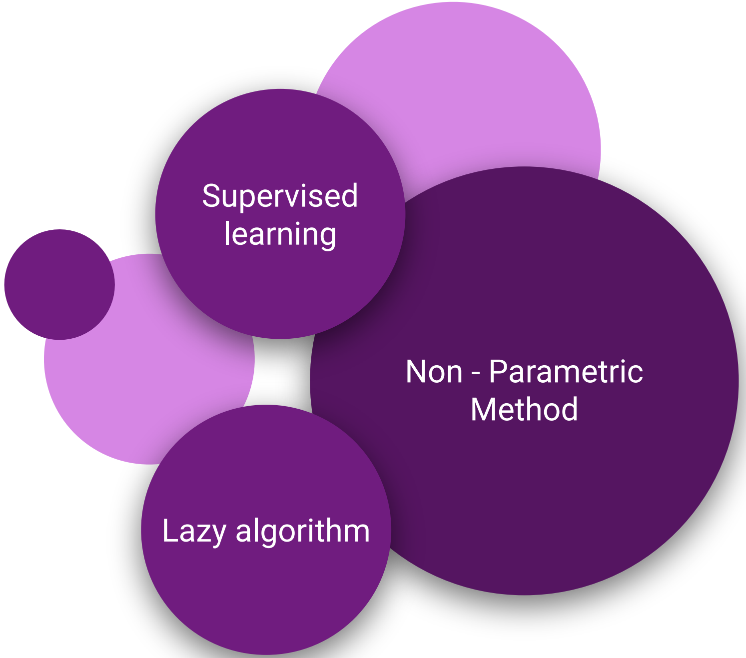
2

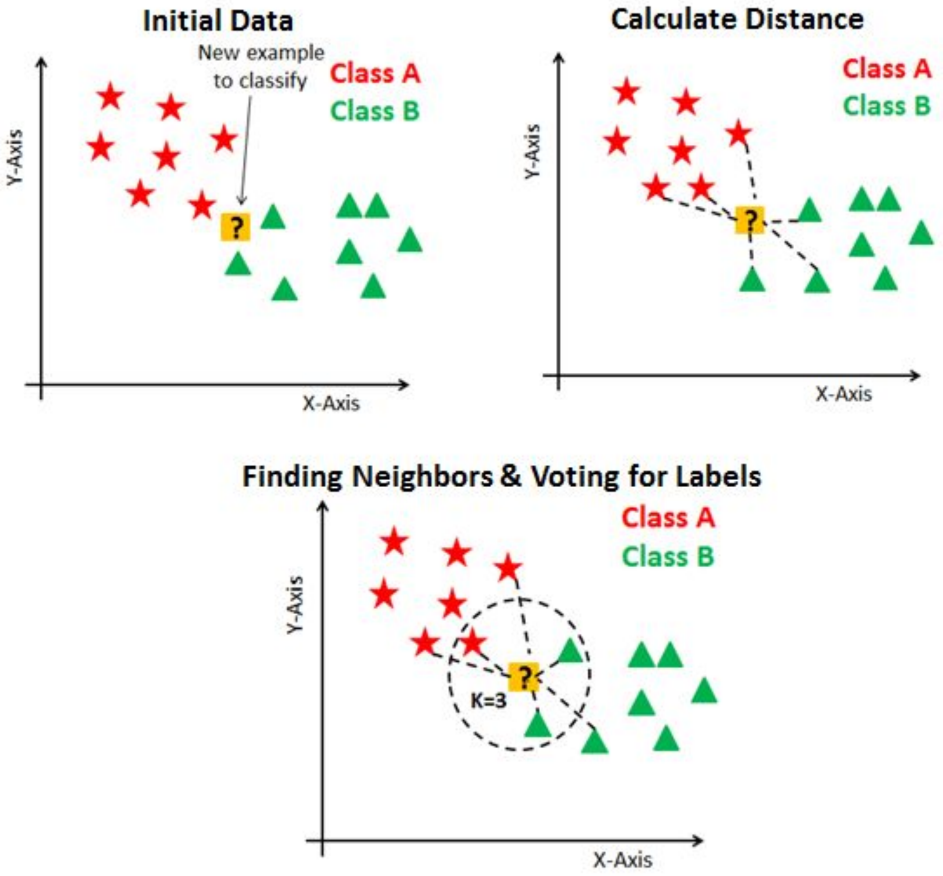
8




| | |
|---|---|
| **Hidden Layers** | 1 |
| **Hidden Nodes** | ( 2 ) |
| **Learning Rate** | 0.3 |
| **Momentum** | 0.2 |
| **Epochs** | 500 |
| **Batch size** | 100 |
| **Training Accuracy** | 100 % (0 sec) |
| **Validation Accuracy** | 100 % (0 sec) |
| **Testing Accuracy** | 100 % (0.01 sec) |

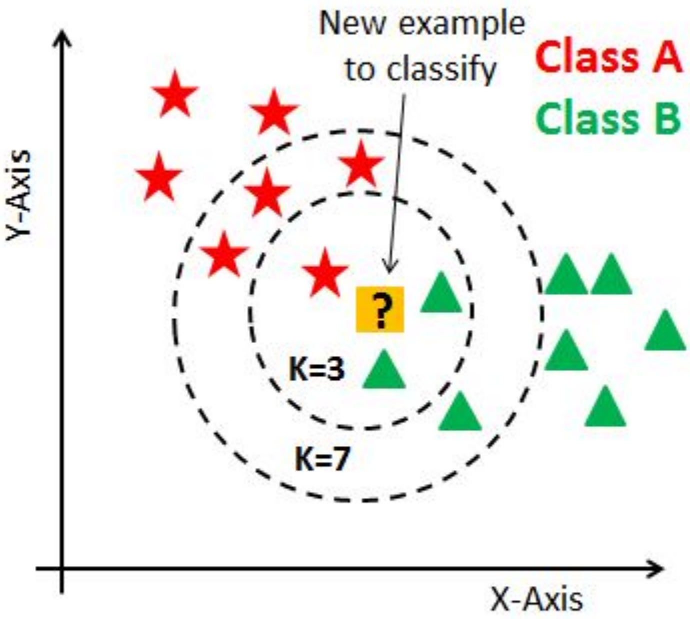
# K - Nearest Neighbours



# Parameters

## K in KNN



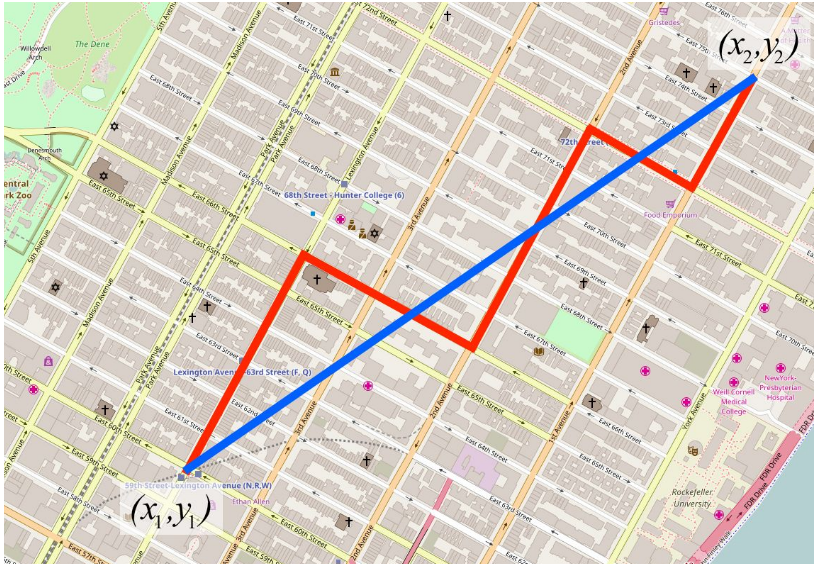

## K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



# Results with Testing Dataset

| K ⇨ / Distance Function ⇩ | **K = 1** | **K = 2** | **K = 3** | **K = 5** | **K = 10** |
|---|---|---|---|---|---|
| **Euclidian** | 99.375 % | 99.6875 % | 99.0625 % | 98.4375 % | 97.5 % |
| **Manhattan** | 76.875 % | 77.1875 % | 75.3125 % | 73.75 % | 73.125 % |

# Performance Discussion - Classification

| Classification Algorithm | Test Accuracy | Evaluation Time |
|---|---|---|
| Naive Bayes | 49.375 % | 0.18 sec |
| K - Nearest Neighbours (K = 2) | 99.6875 % | 0.11 sec |
| Multilayer Perceptron | 100 % | 0 sec |

| | | |
|---|---|---|
| Naive Bayes | 49.375 % | 0.18 sec |
| Multilayer Perceptron | 100 % | 0 sec |

Why **Naive Bayes** performs **poor**?

Naive Bayes makes a very strong assumptions about the data i.e **all attributes are mutually independent**. Hence, a “naive” classifier.

If categorical variable has a category in test data set, which was not observed in training dataset, then the model will assign 0 probability and will be unable to make a prediction. This is often known as **zero frequency**. To solve this, a smoothing technique called **laplace estimation** is used.

**Assumption of likelihood function** can cause errors if the data varies significantly from the assumption.

Why **Multilayer Perceptron** performs **excellent**?

NN make **no assumption about the data**. They don't hand engineered features; they **learn** features.

They have the ability to learn and model **non-linear complex relationships**.

A NN with 2 hidden layers and sufficient hidden nodes has proved to a a universal estimator.

Their ability to **generalise** allows them to infer unseen relationships in unseen data.

# Clustering Algorithms

# Clustering: Dividing data into natural groups or clusters

## K-Means Clustering

- Specify *k,* desired number of clusters.
- Choose k points at random, as cluster centers
- Assign all instances to there closest clusters
- Calculate the centroid of instances in each cluster
- These centroid instances are the new clusters
- Continue untill cluster centers don’t change.

## Hierarchical Clustering

- Specify *k,* desired number of clusters.
- Each of the *r* patterns is a separate cluster.
- If k clusters are found, return from the process
- Find pairwise distance between the clusters, merge two closes ones into one. If more than one pair of clusters are closest, randomly choose two to merge. Go to previous step.

# K-Means Clustering: Classes to clusters evaluation

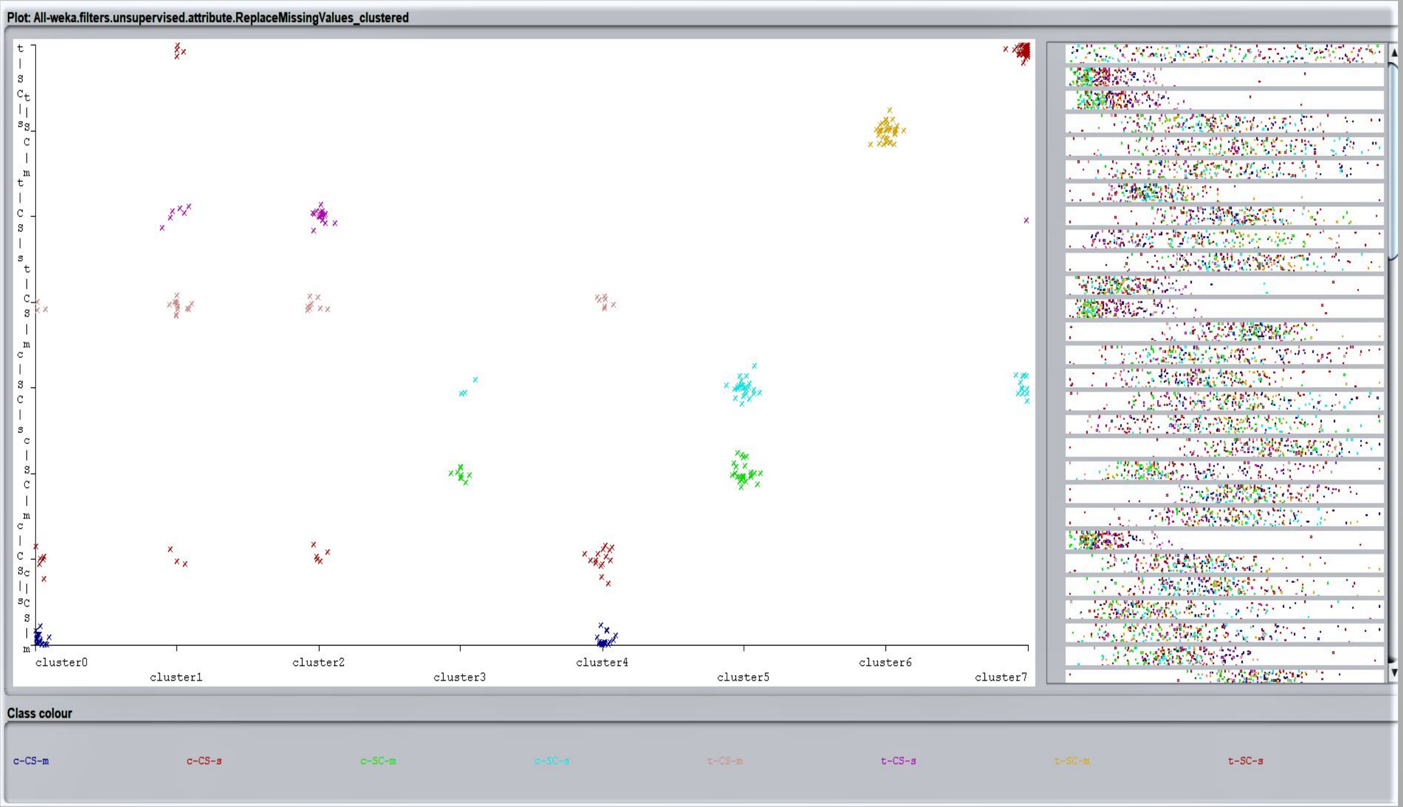
| | Clustered Instances |
|---|---|
| 0 | 135 ( 13%) |
| 1 | 135 ( 13%) |
| 2 | 136 ( 13%) |
| 3 | 149 ( 14%) |
| 4 | 105 ( 10%) |
| 5 | 135 ( 13%) |
| 6 | 135 ( 13%) |
| 7 | 150 ( 14%) |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ← Clusters |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 12 | 138 | 0 | 0 | 0 | 0 | c-CS-m |
| 0 | 0 | 0 | 0 | 0 | 0 | 135 | 0 | c-CS-s |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 150 | c-SC-m |
| 0 | 135 | 0 | 0 | 0 | 0 | 0 | 0 | c-SC-s |
| 0 | 0 | 124 | 11 | 0 | 0 | 0 | 0 | t-CS-m |
| 0 | 0 | 0 | 0 | 105 | 0 | 0 | 0 | t-CS-s |
| 135 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | t-SC-m |
| 0 | 0 | 0 | 0 | 0 | 135 | 0 | 0 | t-SC-s |

| Cluster | Class |
|---|---|
| 0 | c-CS-m |
| 1 | c-CS-s |
| 2 | c-SC-m |
| 3 | c-SC-s |
| 4 | t-CS-m |
| 5 | t-CS-s |
| 6 | t-SC-m |
| 7 | t-SC-s |

# K-Means Clustering: Percentage Split (75%)

| Clustered Instances | |
|---|---|
| 0 | 27 ( 10%) |
| 1 | 25 ( 9%) |
| 2 | 30 ( 11%) |
| 3 | 12 ( 4%) |
| 4 | 40 ( 15%) |
| 5 | 53 ( 20%) |
| 6 | 36 ( 13%) |
| 7 | 47 ( 17%) |



# Hierarchical Clustering: Classes to clusters evaluation

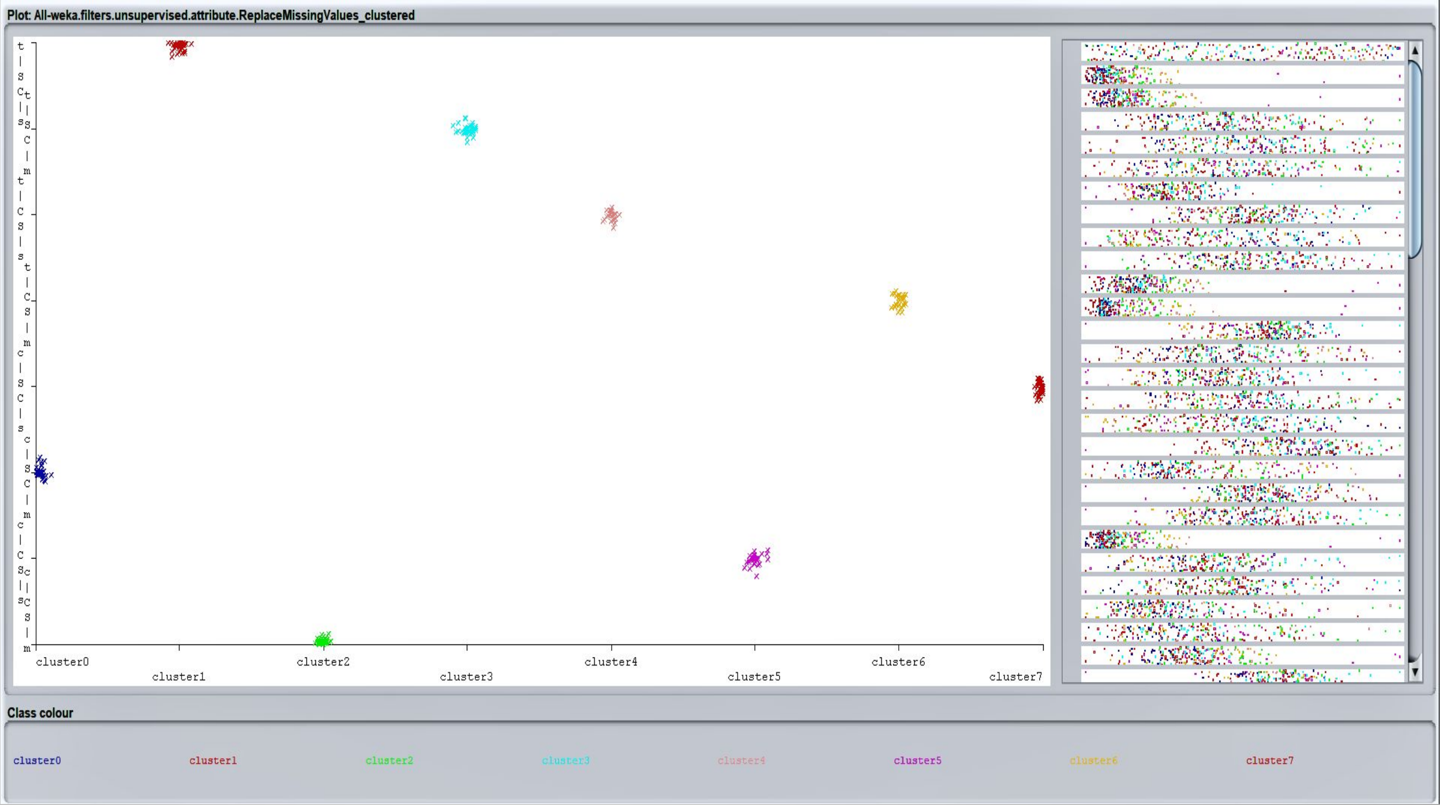
| Clustered Instances | |
|---|---|
| 0 | 150( 14%) |
| 1 | 135 ( 13%) |
| 2 | 150 ( 14%) |
| 3 | 135 ( 13%) |
| 4 | 135 ( 13%) |
| 5 | 105 ( 10%) |
| 6 | 135 ( 13%) |
| 7 | 135 ( 13%) |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ← Clusters |
|---|---|---|---|---|---|---|---|---|
| 150 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | c-CS-m |
| 0 | 135 | 0 | 0 | 0 | 0 | 0 | 0 | c-CS-s |
| 0 | 0 | 150 | 0 | 0 | 0 | 0 | 0 | c-SC-m |
| 0 | 0 | 0 | 135 | 0 | 0 | 0 | 0 | c-SC-s |
| 0 | 0 | 0 | 0 | 135 | 0 | 0 | 0 | t-CS-m |
| 0 | 0 | 0 | 0 | 0 | 105 | 0 | 0 | t-CS-s |
| 0 | 0 | 0 | 0 | 0 | 0 | 135 | 0 | t-SC-m |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 135 | t-SC-s |

| Cluster | Class |
|---|---|
| 0 | c-CS-m |
| 1 | c-CS-s |
| 2 | c-SC-m |
| 3 | c-SC-s |
| 4 | t-CS-m |
| 5 | t-CS-s |
| 6 | t-SC-m |
| 7 | t-SC-s |

# Hierarchical Clustering: Percentage Split (75%)

| Clustered Instances | |
|---|---|
| 0 | 35( 13%) |
| 1 | 38 ( 14%) |
| 2 | 36 ( 13%) |
| 3 | 36 ( 13%) |
| 4 | 24 ( 9%) |
| 5 | 29 ( 11%) |
| 6 | 30 ( 11%) |
| 7 | 42 ( 16%) |



# Test Results (Classes to Clusters)

- K-Means Clustering (Euclidean Distance, k=8)

| Seed | -1 | 2 | 8 | 10 | 100 | 1000 | |
|---|---|---|---|---|---|---|---|
| Accuracy | 66.85% | 79.35% | 86.39% | 97.87% | 71.5% | 56.20% | |

- Hierarchical Clustering (Chebyshev Distance, k=8)

| | |
|---|---|
| Accuracy | 100% |

# Performance Discussion

## K-Means vs Hierarchical Clustering

- Accuracy w.r.t Distance Metric (Classes to Clusters, k=8):

| | Euclidean Distance | Manhattan Distance | Chebyshev Distance |
|---|---|---|---|
| K-Means | 97.87% | 56.12% | - |
| Hierarchical Clustering | 73.33% | 17.5% | 100% |

- Time Metric: K-Means is faster then Hierarchical clustering for large datasets modelling, because of the algorithmic time complexity. O(n) < O(n^2)

# Performance Discussion

## Classes to Clusters vs Percentage Split

- Classes to Clusters → Complete Dataset == Training Dataset
- The issue of Overfitting
- Pseudo high performance
- Better idea of overall accuracy

# Thank You