

# KNN Classifier Implementation

Alyssa Wilcox  
Department of Computer Science and Engineering  
California State University of San Bernardino  
CSE 5160 Machine Learning  
Dr. Yan Zhang  
Fall 2020



# The Project: Implement KNN - Data

# Programming Language Used

- For this project, C++ was used
  - Familiarity
  - Speed



# Data Set Used

- Training data set corresponds to seven measurements taken from three different varieties of wheat seeds:
  - Kama (Classified as 1)
  - Rosa (Classified as 2)
  - Canadian (Classified as 3)



# Attributes

1. Area A
2. Perimeter P
3. Compactness C
4. Length of Kernel L
5. Width of Kernel W
6. Asymmetry Coefficient AC
7. Length of Kernel Groove LG



# Training Set

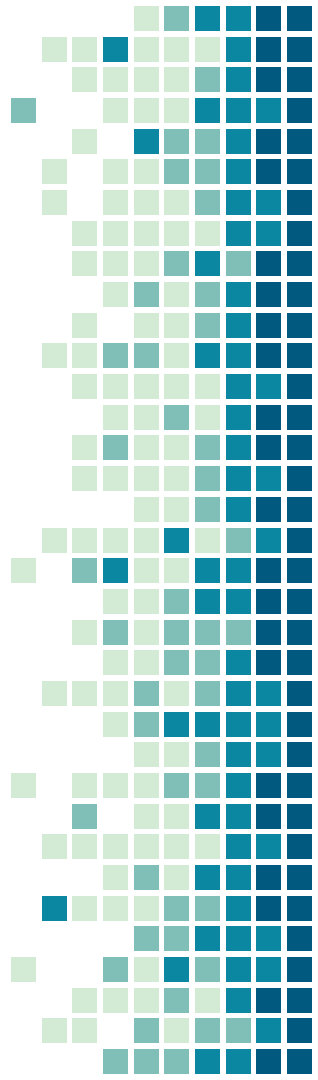
- 180 training instances
  - 60 corresponding to Kama wheat seeds
  - 60 corresponding to Rosa wheat seeds
  - 60 corresponding to Canadian wheat seeds



# The Project: Implement KNN – How it Works

# Step 1: Get User Input

- Get the test instance from the user  
**test\_instance = {feature<sub>1</sub>, feature<sub>2</sub>, ..., feature<sub>7</sub>}**
- Get the k number of neighbors to use when classifying





## Step 2: Get Training Data

- Read from a text file that contains training set
- Save this data to a 2D vector

```
training_set = { training_instance1,  
                  training_instance2,  
                  training_instance3,  
                  ....  
                  training_instance4 }
```

# Step 3: Feature Scaling – Min-Max Normalization

$$\frac{\text{feature } X \text{ value} - \min(X)}{\max(x) - \min(x)}$$

- For a single feature X:
  - Find the minimum and maximum values from all feature X values in training set and test instance
  - min-max normalize every feature X in the training set and test instance
- Repeat for each feature



## Step 4: Find Euclidean Distances

- Find Euclidean distance between the test instance and a training instance
  - $\vec{p}$  = test instance
  - $\vec{q}$  = training instance
- Add distance onto the back of the training instance
- Repeat for all training instances

$$d(\vec{p}, \vec{q}) = \sqrt{\sum_{i=0}^n (p_i - q_i)^2}$$

# Step 5: Sort the Training Set

- Sort the training set based on euclidean distances
  - Ascending order



## Step 6: Get the Output if k Nearest Neighbors

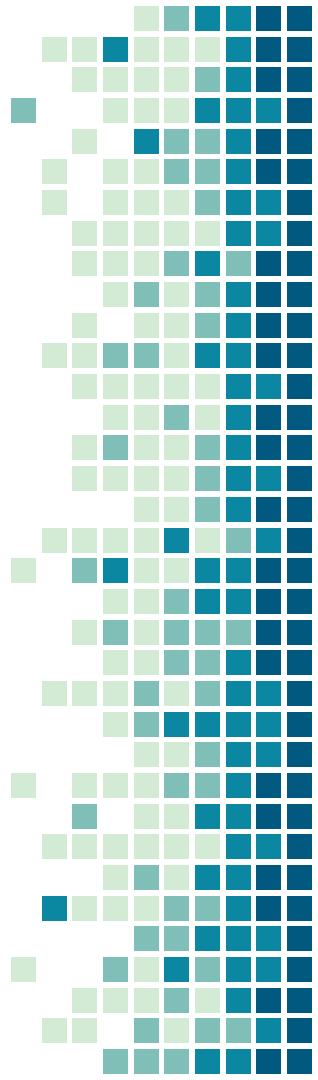
- Using the sorted training set, store the outputs of the first k training instances into an **output** vector

**output = {1, 1, 1, 2, 1}**



## Step 7: Classify the Test Instance

- Count the number of each output type found in the **output** vector
- Find which output type has the majority
  - Classify the test instance based on that



## Step 8: Display the Results

- Output the classification of the test instance to the user



# The Project: Implement KNN – Source Code and Demonstration



# The Project:

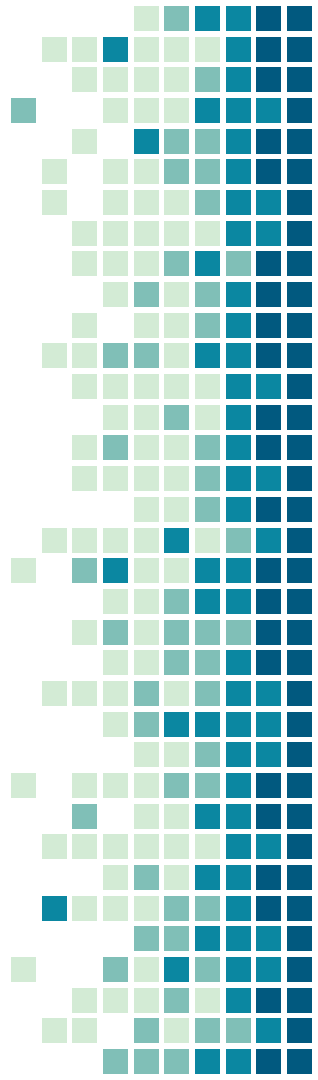
## Implement KNN -

## Evaluate the Classifier

# KNN Accuracy

- 210 original training instances
  - 30 taken to be the testing set
  - 180 remaining training instances
- Tested with different values of k:

□ k = 5:	27/30 correct	90% accuracy
□ k = 10:	26/30 correct	87% accuracy
□ k = 15:	27/30 correct	90% accuracy
□ k = 20:	26/30 correct	87% accuracy



# Limitations of the Program

- Designed to read off of the wheat seed data set
- Cannot classify if there is no majority
- Inefficient:
  - Traversing a one dimensional vector performs in  $\Theta(n)$
  - Traversing a two dimensional vector performs in  $\Theta(n^2)$



# Limitations of KNN

- Lazy learner: does not build a model explicitly
- Classification of test instances is expensive
  - Min-max normalize every value in training set and test instance
  - Find distances between test instance and every training instance
- Prediction accuracy reliant on k input



Thank you for  
watching!

