4. ML project design with supervised learning (K-Nearest Neighbors)

M.A.Z. Chowdhury and M.A. Oehlschlaeger

Department of Mechanical, Aerospace and Nuclear Engineering
Rensselaer Polytechnic Institute
Troy, New York

chowdm@rpi.edu
oehlsm@rpi.edu

"Until you spread your wings, you'll have no idea how far you can fly."
- Napoleon

MANE 4962 and 6962

End-to-end ML Project Design

1. Frame the problem and look at the big picture.
2. Get the data.
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune your model.
7. Present your solution.
8. Launch, monitor, and maintain your system.

There are other ways to design ML projects but this is a good starting point. Every step is described as a checklist in Appendix B of reference Aurélien Géron's book.

Working with Real Data

1. Popular open data repositories.
   - UC Irvine Machine Learning Repository - Kaggle datasets
   - Amazon's AWS datasets
2. Meta portals (they list open data repositories) — Data Portals
   - OpenDataMonitor
   - Quandl
3. Other pages listing many popular open data repositories
   - Wikipedia's list of Machine Learning datasets
   - Quora.com
   - The datasets subreddit
   - Github, Zenodo
   - `https://www.uco.es/kdis/mllresources/`

Managing your project

Use version control to manage your code. I recommend GitHub. Following commands would come in handy.

```
git status
git add .
git commit -m "commit message"
git push
git pull
git clone
git init
git config
```

Data Wrangling with Pandas

Use the Pandas dataframe to work with your data.

```
import pandas as pd
df = pd.DataFrame()
df['column_name'] = a #a can be list/numpy array
df.head()
```

Help: `https://pandas.pydata.org/docs/reference/index.html`

Useful functions

head(), read_csv(), head(),
describe(), memory_usage(),
loc(), astype(), merge(), sort_values()

Sample project

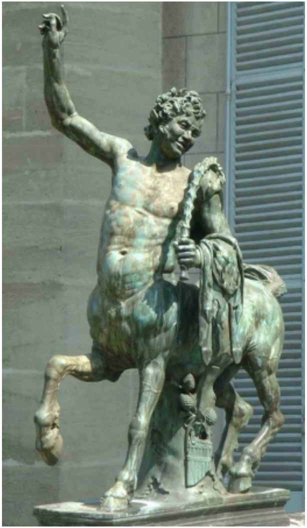
1. We want to make a autonomous drone which is going to identify iris species.
2. It is going to measure the length and width of the sepals and petals.
3. Use the Iris dataset to design a simple machine learning model.

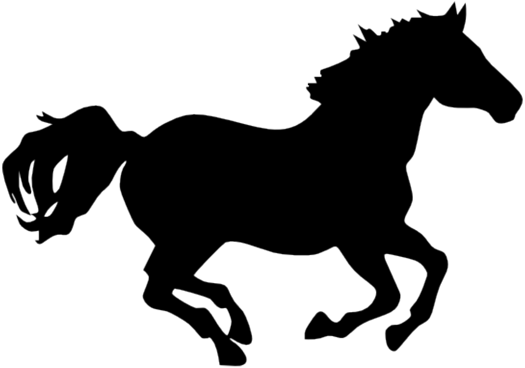
Similarity in nature

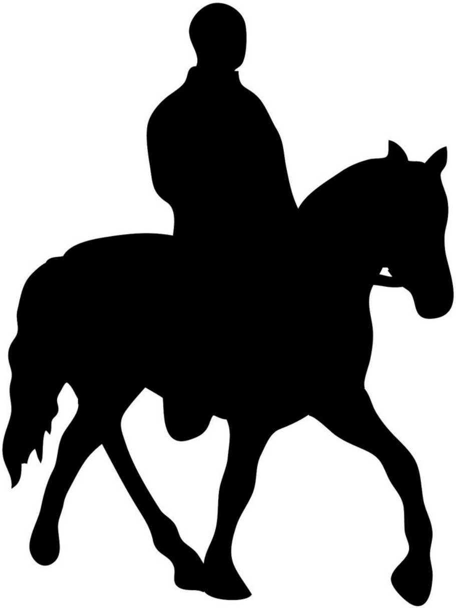
Birds of a feather flock together




A man or a horse or a horseman or a 'man-horse'?







k-Nearest Neighbors (kNN)

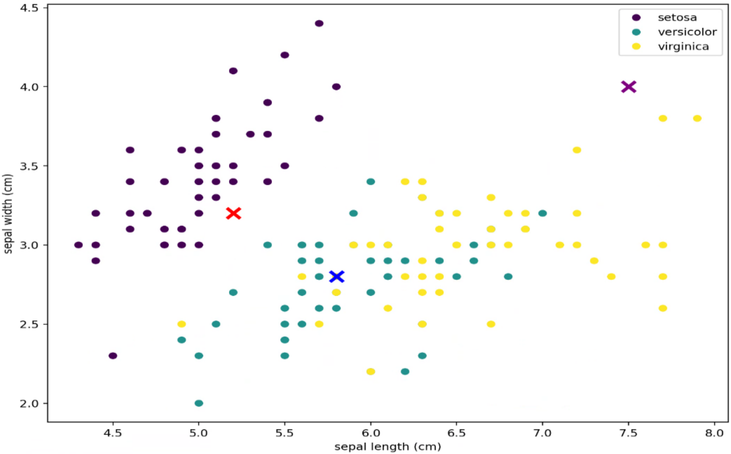
- Classify or regress data points based on similarity to nearby data points.
- Distance, $d(\underline{x}, \underline{x}^{'}) = ||\underline{x} - \underline{x}^{'}||$
- We can use L1 and L2 norms as distance metric to find nearby data points.
- These are commonly known as taxicab or manhattan distance and the euclidean distance.
- Scikit has kNNClassifier and kNNregressor classes for implementation.
- For classification, the class associated with the nearest neighbors or the mean class is returned.
- For regression, local interpolation of the targets associated with the nearest neighbors in the training set is returned.

k-NN algorithm (in simple words)

1. Find all the nearest neighbors
2. Show their average class as the class label of the unknown vector.

A visual example

- Say, we make three new measurements (×) and compare them to Fisher's Iris data.
- Predict the class labels (species types) for these measurements.

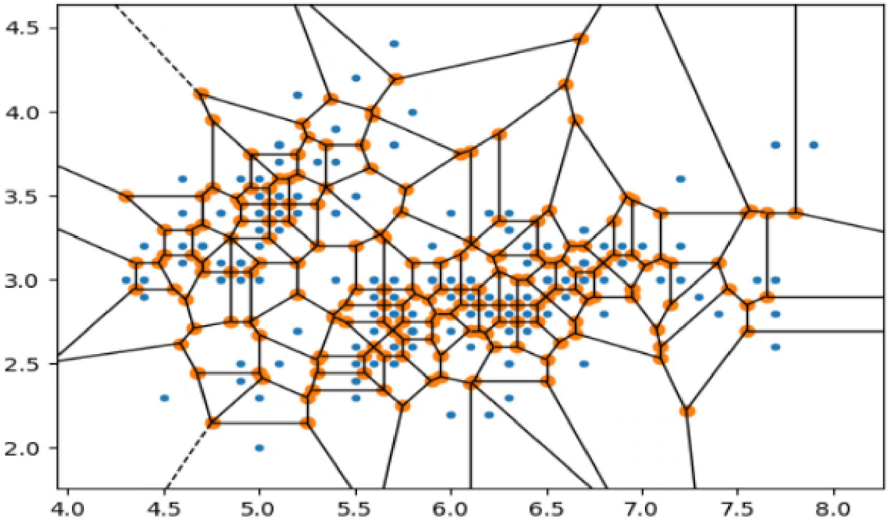


A simple implementation of K-NN

```
class My_KNNClassifier:
    def __init__(self, k=3):
        self.k = k
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
    def predict(self, X_test):
        predictions = []
        for i in range(X_test.shape[0]):
            predictions.append(self._knn_classifier(X_test[i]))
        return predictions
    def _knn_classifier(self, X_test):
        distances, targets = [], []
        for i in range(self.X_train.shape[0]):
            distance = np.linalg.norm(self.X_train[i]-X_test)
            distances.append([distance, i])
        distances = sorted(distances)
        for i in range(self.k):
            index = distances[i][1]
            targets.append(self.y_train[index])
        return max(targets, key=targets.count)
model = My_KNNClassifier()
model.fit(X_train, y_train)
preds = model.predict(X_test)
print(accuracy_score(y_test, preds))
```

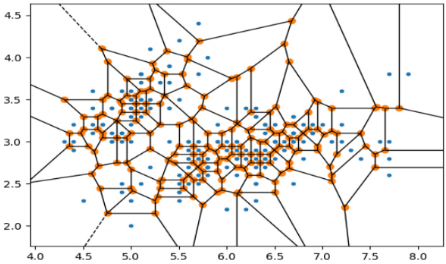
All the calculations are inside predict method. See notebook for comparison and tuning.

Voronoi Diagram



Voronoi Diagram

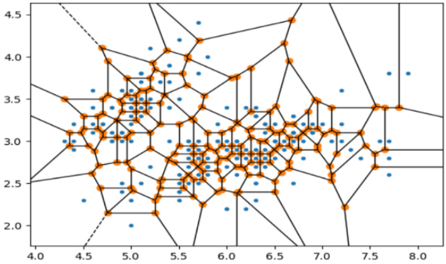
- A geometric representation of a dataset that partitions a space into regions based on the distance to a set of points.
- The points in the dataset are called the generators or seeds of the Voronoi diagram, and each region is defined as the set of points that are closest to a particular generator.
- Each region is represented by a polygon, which is the set of points that are closer to the generator of that region than to any other generator.
- Voronoi edges: Edges of the polygons are the lines that divide the space into the different regions.
- Voronoi vertices: The points where the Voronoi edges meet. These points are equidistant to two or more generators.



For two iris features.

Voronoi Diagram

- Interpreting a Voronoi diagram can provide insight into the structure of a dataset by highlighting patterns and relationships between the generators.
- If the generators are evenly spaced, the resulting Voronoi diagram will have regular, symmetric regions.
- If the generators are clustered together, the resulting diagram will have irregular regions with many small polygons.
- It is useful to visualize the k-NN decision boundary that separate the classes, where the decision boundary is the set of points that are equidistant to two or more class generators.
- This helps understand the shape and complexity of the decision boundary, and to identify any potential issues with overfitting or underfitting.

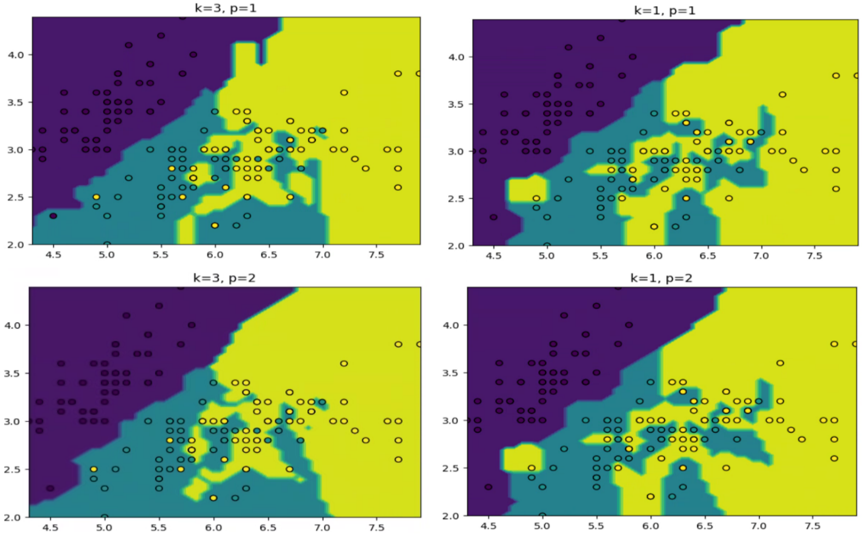


For two iris features.

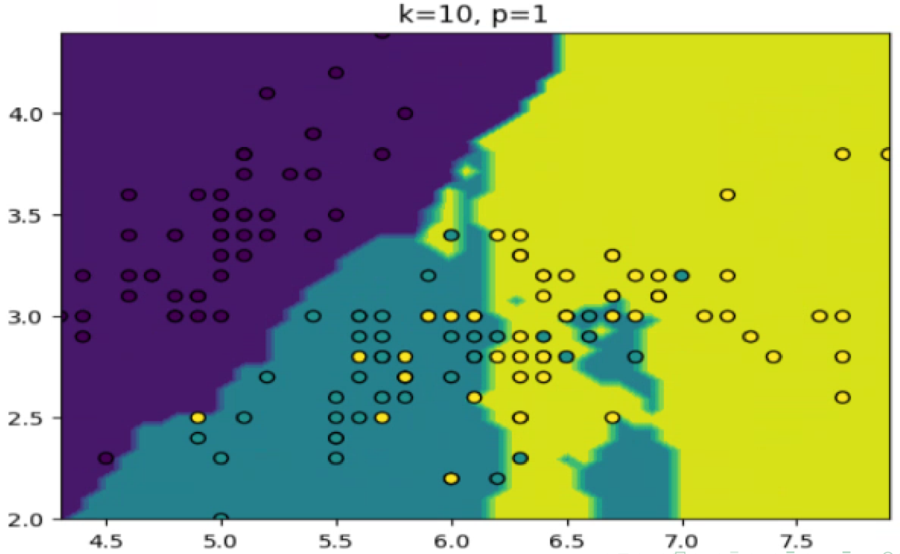
Choosing k

- k=3 [Good starting point for many problems]
- $k = \sqrt{N}$, N is the size of the train set.
- Cross-validation
  - small k
    - good for capturing fine-grained patterns
    - but it may overfit since it will be sensitive to random peculiarities present in the training data
  - large k
    - makes stable predictions by averaging over lots of examples
    - but it may underfit by failing to capture important regularities

Varying k



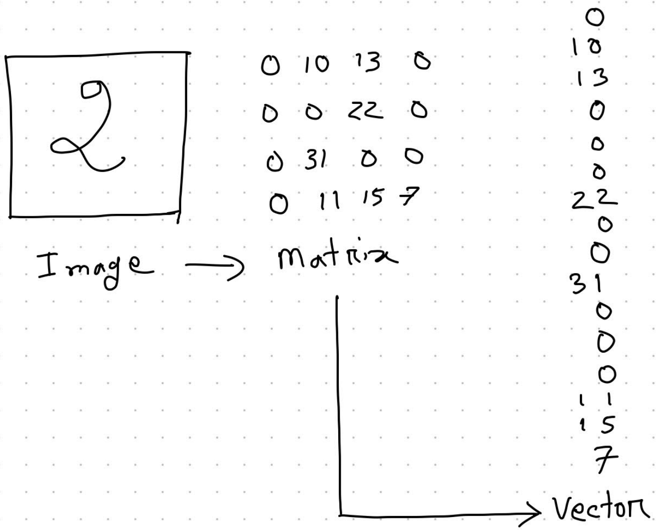
Varying k



Remarks

- **Advantages:**
  - Fast learning, no explicit training
  - Simple, easy to explain the method and results to users and customers.
  - Can control the complexity by varying k
  - Number of computations at training time is zero!
  - Can easily do multi-class.
  - Can easily adapt to regression.
- **Disadvantages:**
  - Memory intensive since it needs to store the entire dataset in memory!
  - Predictions can take a long time
  - No model to shed light on process that generated data
  - Thousands of work hours have gone into algorithms and data structures for efficient nearest neighbors with high dimensions and/or large datasets.
  - Suffers from the Curse of Dimensionality

Applicable to high dimensional data



Advantages of using feature vector or input vector

- ML algorithms handles lots of different types of data: signals, images, text, video etc.
- Represent the input as an input vector $\in \mathbb{R}^{dim}$
- **Representation** = mapping to another space that's easy to manipulate.
- Vectors provide a wonderful mechanism to work with the input features of the data because we can use linear algebra!