

# Mathematical techniques in data science

## Lecture 4: Nearest neighbors

# Reminders

- Office hours:
  - MW 3pm-4pm, Ewing Hall 312
  - By appointments

- Website:

`http://vucdinh.github.io/m637s22`

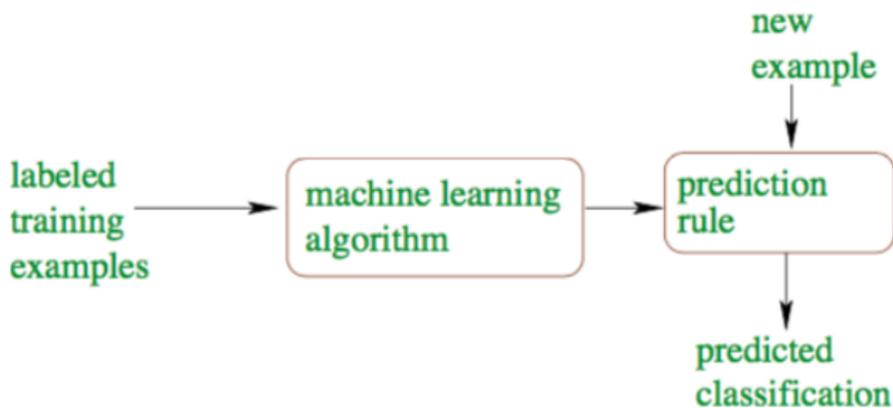
- Homework 0: due next Monday EOD
- Sign up for group projects by the end of Week 4

## ...from last lecture

- Review of probability
- Python and Numpy demos

## Nearest neighbors

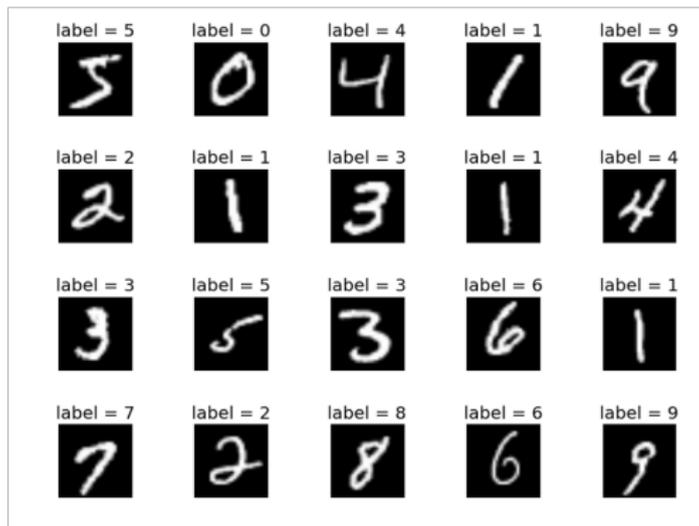
# Supervised learning



Learning a function that maps an input to an output based on example input-output pairs

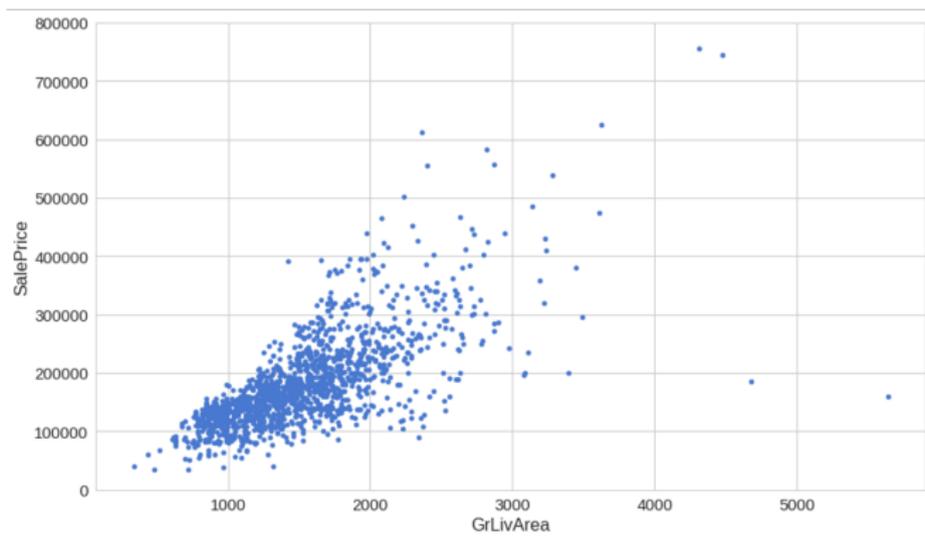
# Supervised learning: Classification

## Hand-written digit recognition



# Supervised learning: Regression

Predict house price by living area

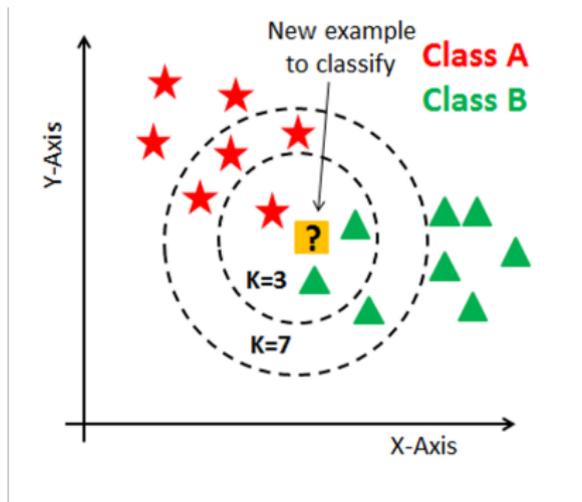


# Nearest Neighbors

- Very simple idea: Make predictions based on labels of the nearest training examples
- Applicable to both classification and regression

## K-nearest neighbor (K-NN) for classification

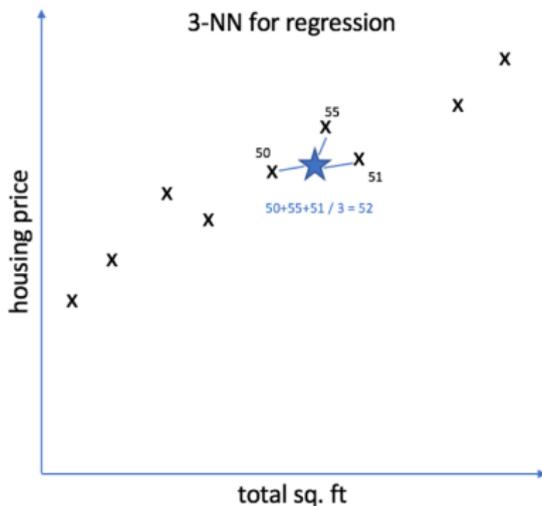
- Learning: Store all training examples
- Predict label of  $x$ :
  - Find the nearest  $K$  training examples to  $x$
  - Assign the most frequent label to  $x$



(Source: [kdnuggets.com](http://kdnuggets.com))

## K-nearest neighbor (K-NN) for regression

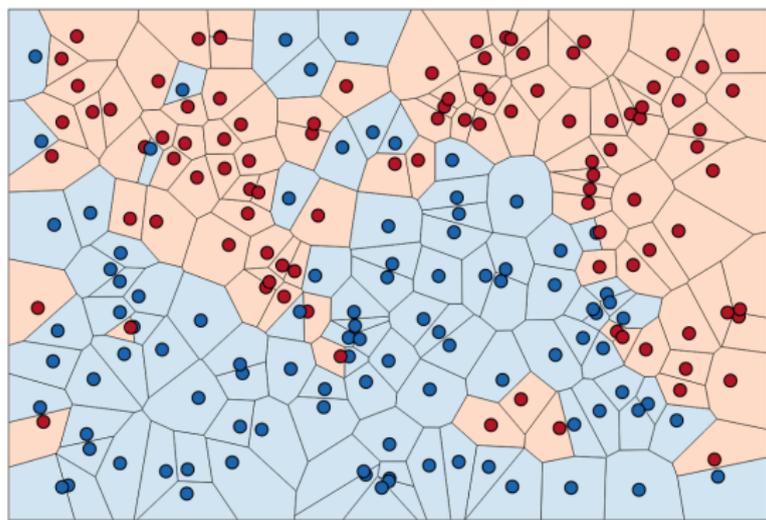
- Learning: Store all training examples
- Predict label of  $x$ :
  - Find the nearest  $K$  training examples to  $x$
  - Assign the average of the  $K$  nearest labels to  $x$



(Source: Jeremy Jordan)

## Decision boundary for K-NN (classification)

- Divide feature space into Voronoi cells
- Decision boundaries are linear



(Source: Kevin Zakka)

## How to define “near-ness”?

- Need to define a distance metric between data points
- Common metric: Euclidean distance (L2 distance)

$$d(x, x') = \sqrt{\sum_{i=1}^n (x_i - x'_i)^2}$$

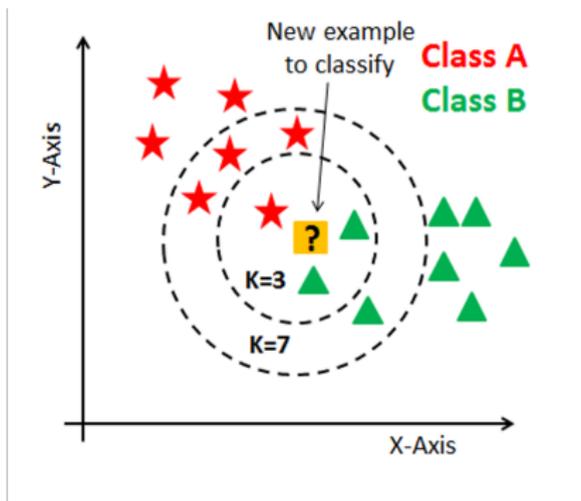
- Other metrics: Manhattan distance, Minkowski distance, etc.

# How to find the nearest neighbors?

- Simplest algorithm: brute-force search
  - Compute distance between every training data point to the test point
  - Pick K points with the smallest distances
- Space partitioning algorithms: arrange data points into some data structure (such as a binary tree) for faster search
  - Ball tree ([https://en.wikipedia.org/wiki/Ball\\_tree](https://en.wikipedia.org/wiki/Ball_tree))
  - k-d tree ([https://en.wikipedia.org/wiki/K-d\\_tree](https://en.wikipedia.org/wiki/K-d_tree))
- Approximate algorithms

## Variant: Prediction with non-uniform weights

Nearer neighbors have more weights when making predictions



# Machine learning models from scikit-learn

Keys to read the documentations

- Parameters (to set up the model)
- Methods (what we can do with the model)
- Attributes (components we can extract from the model)

# Nearest Neighbors on scikit-learn

## sklearn.neighbors.KNeighborsClassifier

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30,
p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

[\[source\]](#)

Classifier implementing the k-nearest neighbors vote.

Read more in the [User Guide](#).

**Parameters:** **n\_neighbors** : *int*, *default=5*

Number of neighbors to use by default for `kneighbors` queries.

**weights** : *{'uniform', 'distance'}* or *callable*, *default='uniform'*

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

**algorithm** : *{'auto', 'ball\_tree', 'kd\_tree', 'brute'}*, *default='auto'*

Algorithm used to compute the nearest neighbors:

- 'ball\_tree' will use `BallTree`
- 'kd\_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

# Nearest Neighbors on scikit-learn

## sklearn.neighbors.KNeighborsRegressor

```
class sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30,
p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs) [source]
```

Regression based on k-nearest neighbors.

The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

Read more in the [User Guide](#).

New in version 0.9.

**Parameters:** **n\_neighbors** : *int*, **default=5**

Number of neighbors to use by default for `kneighbors` queries.

**weights** : {'uniform', 'distance'} or callable, **default='uniform'**

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Uniform weights are used by default.

**algorithm** : {'auto', 'ball\_tree', 'kd\_tree', 'brute'}, **default='auto'**

Algorithm used to compute the nearest neighbors:

- 'ball\_tree' will use `BallTree`
- 'kd\_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

# Nearest Neighbors on scikit-learn

## sklearn.neighbors: Nearest Neighbors

The `sklearn.neighbors` module implements the k-nearest neighbors algorithm.

**User guide:** See the [Nearest Neighbors](#) section for further details.

<code>neighbors.BallTree(X[, leaf_size, metric])</code>	BallTree for fast generalized N-point problems
<code>neighbors.DistanceMetric</code>	DistanceMetric class
<code>neighbors.KDTree(X[, leaf_size, metric])</code>	KDTree for fast generalized N-point problems
<code>neighbors.KernelDensity(*[, bandwidth, ...])</code>	Kernel Density Estimation.
<code>neighbors.KNeighborsClassifier([...])</code>	Classifier implementing the k-nearest neighbors vote.
<code>neighbors.KNeighborsRegressor([n_neighbors, ...])</code>	Regression based on k-nearest neighbors.
<code>neighbors.KNeighborsTransformer(*[, mode, ...])</code>	Transform X into a (weighted) graph of k nearest neighbors
<code>neighbors.LocalOutlierFactor([n_neighbors, ...])</code>	Unsupervised Outlier Detection using Local Outlier Factor (LOF)
<code>neighbors.RadiusNeighborsClassifier([...])</code>	Classifier implementing a vote among neighbors within a given radius
<code>neighbors.RadiusNeighborsRegressor([radius, ...])</code>	Regression based on neighbors within a fixed radius.
<code>neighbors.RadiusNeighborsTransformer(*[, ...])</code>	Transform X into a (weighted) graph of neighbors nearer than a radius
<code>neighbors.NearestCentroid([metric, ...])</code>	Nearest centroid classifier.
<code>neighbors.NearestNeighbors(*[, n_neighbors, ...])</code>	Unsupervised learner for implementing neighbor searches.
<code>neighbors.NeighborhoodComponentsAnalysis([...])</code>	Neighborhood Components Analysis
<hr/>	
<code>neighbors.kneighbors_graph(X, n_neighbors, *)</code>	Computes the (weighted) graph of k-Neighbors for points in X
<code>neighbors.radius_neighbors_graph(X, radius, *)</code>	Computes the (weighted) graph of Neighbors for points in X

# Nearest Neighbors Demo

## General steps to build ML models

- Get and pre-process data
- Visualize the data (optional)
- Create a model
- Train the model; i.e. call `model.fit()`
- Predict on test data
- Compute evaluation metrics (accuracy, mean squared error, etc.)
- Visualize the trained model (optional)

## Review: Evaluate a learned model

- How effective the model makes predictions on new (unseen) data
- Classification: accuracy or error rate
- Regression: average (squared) distance between predicted and true values (mean squared error)

## Review: Data splitting practices

