Unsupervised vs. Supervised Learning

Marina Sedinkina

Ludwig Maximilian University of Munich
Center for Information and Language Processing

December 3, 2019

Overview

What Is Machine Learning?

- **Modeling: model** - specification of a mathematical (or probabilistic) relationship that exists between different variables.

What Is Machine Learning?

- **Modeling: model** - specification of a mathematical (or probabilistic) relationship that exists between different variables.
  - **business model**: number of users, profit per user, number of employees $\Rightarrow$ profit is income minus expenses

What Is Machine Learning?

- **Modeling: model** - specification of a mathematical (or probabilistic) relationship that exists between different variables.
  - **business model**: number of users, profit per user, number of employees $\Rightarrow$ profit is income minus expenses
  - **poker model**: the cards that have been revealed so far, the distribution of cards in the deck $\Rightarrow$ win probability

What Is Machine Learning?

- **Modeling: model** - specification of a mathematical (or probabilistic) relationship that exists between different variables.
  - **business model**: number of users, profit per user, number of employees $\Rightarrow$ profit is income minus expenses
  - **poker model**: the cards that have been revealed so far, the distribution of cards in the deck $\Rightarrow$ win probability
  - **language model in NLP**: a probability that a string is a member of a language (originally developed for the problem of speech recognition)

What Is Machine Learning?

- **Modeling: model** - specification of a mathematical (or probabilistic) relationship that exists between different variables.
  - **business model**: number of users, profit per user, number of employees $\Rightarrow$ profit is income minus expenses
  - **poker model**: the cards that have been revealed so far, the distribution of cards in the deck $\Rightarrow$ win probability
  - **language model in NLP**: a probability that a string is a member of a language (originally developed for the problem of speech recognition)
- **Machine Learning** - creating and using models that are learned from data (**predictive modeling** or **data mining**)

What Is Machine Learning?

- **Goal** - use existing data to develop models for predicting various outcomes for new data

What Is Machine Learning?

- **Goal** - use existing data to develop models for predicting various outcomes for new data
  - Predicting whether an email message is spam or not

What Is Machine Learning?

- **Goal** - use existing data to develop models for predicting various outcomes for new data
  - Predicting whether an email message is spam or not
  - Predicting which advertisement a shopper is most likely to click on

What Is Machine Learning?

- **Goal** - use existing data to develop models for predicting various outcomes for new data
  - Predicting whether an email message is spam or not
  - Predicting which advertisement a shopper is most likely to click on
  - Predicting which football team is going to win

What Is Machine Learning?

- **Goal** - use existing data to develop models for predicting various outcomes for new data
  - Predicting whether an email message is spam or not
  - Predicting which advertisement a shopper is most likely to click on
  - Predicting which football team is going to win

Examples in NLP:

???

What Is Machine Learning?

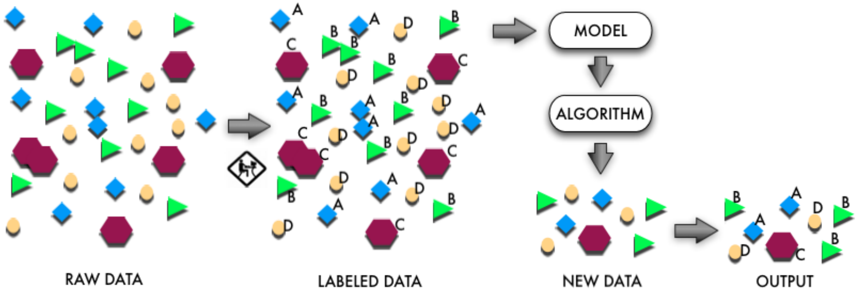
- **Goal** - use existing data to develop models for predicting various outcomes for new data
  - Predicting whether an email message is spam or not
  - Predicting which advertisement a shopper is most likely to click on
  - Predicting which football team is going to win

Examples in NLP:

- Speech Recognition
- Language Identification
- Machine Translation
- Document Summarization
- Question Answering
- Sentiment Detection
- Text Classification

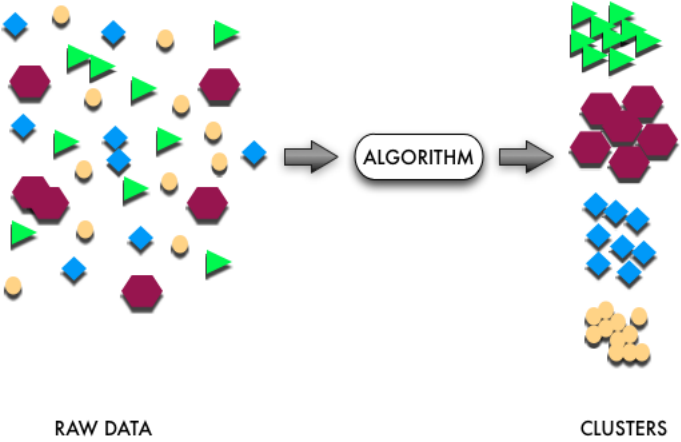
Approaches

supervised: data labeled with the correct answers to learn from



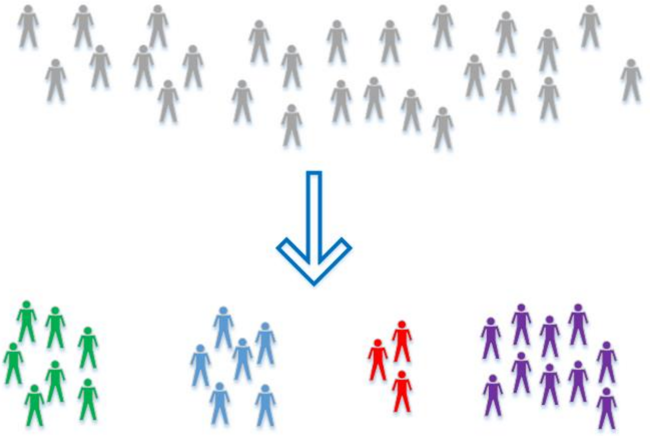
Approaches

unsupervised: no label given, purely based on the given raw data ⇒ find common structure in data

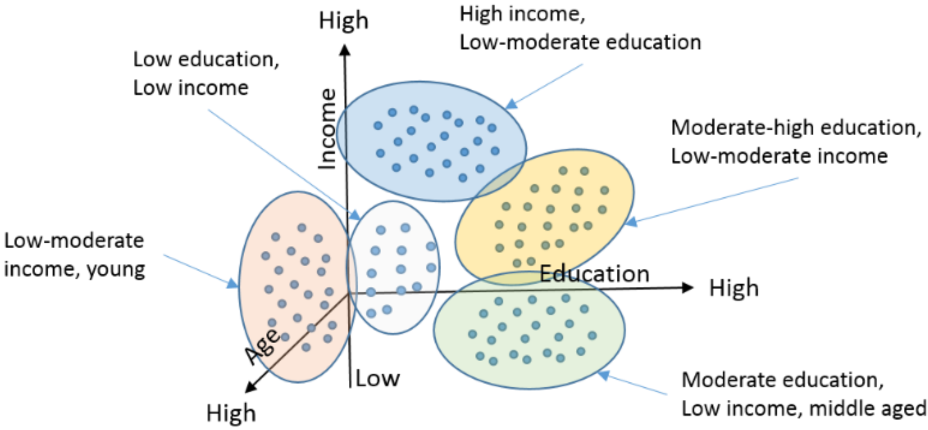


Unsupervised Learning: General Examples

- you see a group of people: divide them into groups



Unsupervised Learning: General Examples



Unsupervised Learning: General Examples

- cluster city names, trees

Unsupervised Learning: General Examples

- cluster city names, trees
- cluster similar blog posts: understand what the users are blogging about.

General Idea

- predict how I'm going to vote!

Supervised: K Nearest Neighbors Classification

General Idea

- predict how I'm going to vote!
- approach - look at my neighbors are planning to vote

General Idea

- predict how I'm going to vote!
- approach - look at my neighbors are planning to vote
- **better idea???**

General Idea

- predict how I'm going to vote!
- approach - look at my neighbors are planning to vote
- imagine you know:
  - my age

Supervised: K Nearest Neighbors Classification

General Idea

- predict how I'm going to vote!
- approach - look at my neighbors are planning to vote
- imagine you know:
  - my age
  - my income

Supervised: K Nearest Neighbors Classification

General Idea

- predict how I'm going to vote!
- approach - look at my neighbors are planning to vote
- imagine you know:
  - my age
  - my income
  - how many kids I have

Supervised: K Nearest Neighbors Classification

General Idea

- predict how I'm going to vote!
- approach - look at my neighbors are planning to vote
- imagine you know:
    - my age
    - my income
    - how many kids I have
- new approach - look at those neighbors with similar features $\rightarrow$ better prediction!

Nearest Neighbors: Classification rule

- classify a new object

Nearest Neighbors: Classification rule

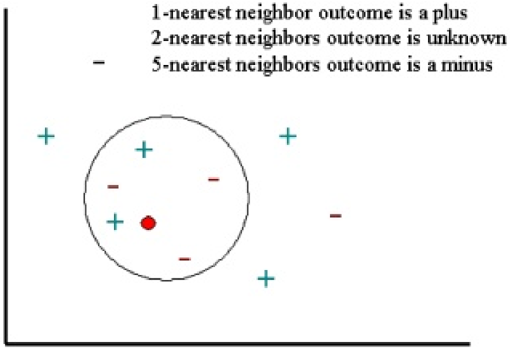
- classify a new object
- find the object in the training set that is most similar

Nearest Neighbors: Classification rule

- classify a new object
- find the object in the training set that is most similar
- assign the category of this nearest neighbor

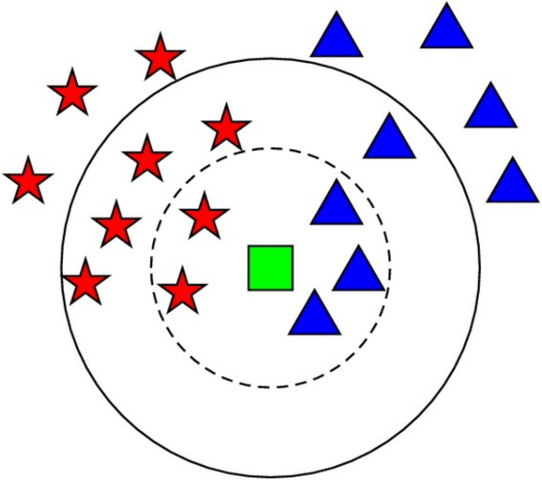
K Nearest Neighbor (KNN) Classification

Take k closest neighbors instead of one



K Nearest Neighbor (KNN) Classification

k = 5; 10

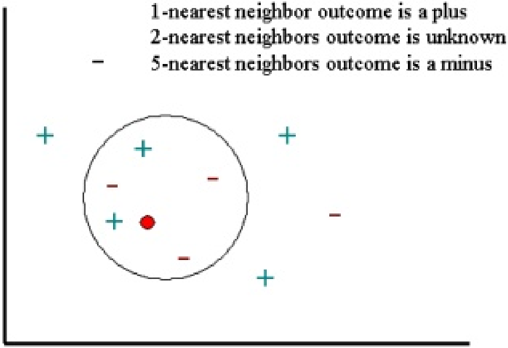


K Nearest Neighbor (KNN) Classification: Data points

- **Data points** are vectors in some finite-dimensional space.

K Nearest Neighbor (KNN) Classification: Data points

- **Data points** are vectors in some finite-dimensional space.
- **'+' and '-' objects** are 2-dimensional (2-d) vectors:



Data points

- if you have the **heights, weights**, and **ages** of a large number of people, treat your data as 3-dimensional vectors **(height, weight, age)**:

```
height_weight_age_point = [70, # kg
                             170, # cm,
                           40 ] # years
```

Data points: One-hot encoding

- **Task**: Represent each word from data as a vector (data point)

Data points: One-hot encoding

- **Task**: Represent each word from data as a vector (data point)
- Form **vocabulary (word types)** from data:

```
data: The quick quick brown fox
```

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

Data points: One-hot encoding

- **Task**: Represent each word from data as a vector (data point)
- Form **vocabulary (word types)** from data:

```
data: The quick quick brown fox
```

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

- **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

   ```
   data: The quick quick brown fox
   ```

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size = 4, one-hot 4-d vector of word "The" at the position 0 is $v_{\vec{The}} = (1000)$:

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

```
data: The quick quick brown fox
```

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size = 4, one-hot 4-d vector of word "The" at the position 0 is $v_{\vec{The}} = (1000)$:

One-hot representation

$v_{\vec{The}} = (1000)$

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

```
data: The quick quick brown fox
```

$$\mathrm{Vocab}(s) = \begin{cases} \text{"The"} \\ \text{"quick"} \\ \text{"brown"} \\ \text{"fox"} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size = 4, one-hot 4-d vector of word "The" at the position 0 is $v_{\vec{The}} = (1000)$:

One-hot representation

$v_{\vec{The}} = (1000)$ $v_{\vec{quick}} = (????)$

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

   `data: The quick quick brown fox`

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size $= 4$, one-hot 4-d vector of word "The" at the position 0 is $v_{\vec{The}} = (1000)$:

One-hot representation

$v_{\vec{The}} = (1000)$ $v_{\vec{quick}} = (0100)$

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

   ```
   data: The quick quick brown fox
   ```

   $$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size = 4, one-hot 4-d vector of word "The" at the position 0 is $v_{\vec{The}} = (1000)$:

One-hot representation

$v_{\vec{The}} = (1000)\ v_{\vec{quick}} = (0100)\ v_{\vec{brown}} = (????)$

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

```
data: The quick quick brown fox
```

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size = 4, one-hot 4-d vector of word "The" at the position 0 is $\vec{v_{The}} = (1000)$:

One-hot representation

$\vec{v_{The}} = (1000)$ $\vec{v_{quick}} = (0100)$ $\vec{v_{brown}} = (0010)$

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

```
data: The quick quick brown fox
```

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size = 4, one-hot 4-d vector of word "The" at the position 0 is $v_{\vec{The}} = (1000)$:

One-hot representation

$v_{\vec{The}} = (1000)$ $v_{\vec{quick}} = (0100)$ $v_{\vec{brown}} = (0010)$ $v_{\vec{fox}} = (????)$

Data points: One-hot encoding

1. **Task**: Represent each word from data as a vector (data point)
2. Form **vocabulary (word types)** from data:

```
data: The quick quick brown fox
```

$$\mathrm{Vocab}(s) = \begin{cases} \text{“The”} \\ \text{“quick”} \\ \text{“brown”} \\ \text{“fox”} \end{cases}$$

3. **One-hot vector** is a vector filled with 0s, except for a 1 at the position associated with word
4. Vocabulary size = 4, one-hot 4-d vector of word "The" at the position 0 is $v_{\vec{The}} = (1000)$:

One-hot representation

$v_{\vec{The}} = (1000)$ $v_{\vec{quick}} = (0100)$ $v_{\vec{brown}} = (0010)$ $v_{\vec{fox}} = (0001)$

Data points: Document representation

How we can represent a document???

Document representation

- fixed set of elements (e.g., documents): $D = \{d_1, ...d_n\}$

Document representation

- fixed set of elements (e.g., documents): $D = \{d_1, ...d_n\}$
- document d (data point) is represented by a vector of features: $d \in \mathbb{N}^k \rightarrow d = [x_1 x_2 ... x_k]$

Document representation

- fixed set of elements (e.g., documents): $D = \{d_1, ...d_n\}$
- document d (data point) is represented by a vector of features: $d \in \mathbb{N}^k \rightarrow d = [x_1 x_2 ... x_k]$
- feature weights are numerical statistics (TF-IDF)

Document Representation: binary

Vectorize a text corpus, by turning each text into a vector where the coefficient for each token could be **binary**:

```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
X_train = ["first␣text:␣first␣sentence","second␣text",
                          "third␣text"]

tokenizer.fit_on_texts(X_train)
tokenizer.word_index
>>>{'first': 2, 'second': 4, 'sentence': 3,
    'text': 1, 'third': 5}

tokenizer.texts_to_matrix(X_train,  mode='binary')
>>>array([[ 1., 1., 1., 0., 0.],
        [1., 0., 0., 1., 0.],
        [1., 0., 0., 0., 1.]])
```

Document Representation: count

Vectorize a text corpus, by turning each text into a vector where the coefficient for each token could based on **word count**:

```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
X_train = ["first text: first sentence","second text",
                              "third text"]

tokenizer.fit_on_texts(X_train)
tokenizer.word_index
>>>{'first': 2, 'second': 4, 'sentence': 3,
    'text': 1, 'third': 5}

tokenizer.texts_to_matrix(X_train,  mode='count')
>>array([[0., 1., 2., 1., 0., 0.],
        [0., 1., 0., 0., 1., 0.],
        [0., 1., 0., 0., 0., 1.]])
```

Document Representation: tf-idf

Vectorize a text corpus, by turning each text into a vector where the coefficient for each token could based on **tf-idf**:

```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
X_train = ["first text: first sentence","second text",
                        "third text"]

tokenizer.fit_on_texts(X_train)
tokenizer.word_index
>>>{'first': 2, 'second': 4, 'sentence': 3,
    'text': 1, 'third': 5}

tokenizer.texts_to_matrix(X_train, mode='tfidf')
>>[[0 0.55961579 1.55141507 0.91629073 0          0]
   [0 0.55961579 0          0          0.91629073 0]
   [0 0.55961579 0          0          0 0.91629073]]
```

K Nearest Neighbor (KNN) Classification

```
def knn_classify(k, labeled_points, new_point):
    """each labeled point is a pair (point, label)"""

    # order points descending
    similarities = sorted(labeled_points,
                   key=lambda x:
                   -cosin_sim(x[0], new_point))

    # find the labels for the k closest
    k_nearest_labels = [label for _,label
                        in similarities[:k]]

    # and choose one
    return choose_one(k_nearest_labels)
```

Recall: Sort List of Tuples

```
>>> students = [
        ('john', 22),
        ('jane', 20),
        ('dave', 25)]

>>> sorted(students)
[('dave', 25), ('jane', 20), ('john',22)]

>>> sorted(students, key=lambda x: x[1])
[('jane', 20), ('john', 22), ('dave', 25)]

>>> sorted(students, key=lambda x: x[1],reverse=True)
[('dave', 25), ('john', 22), ('jane', 20)]

>>> sorted(students, key=lambda x: -x[1])
[('dave', 25), ('john', 22), ('jane', 20)]
```

Requirements. Metric for distance computation

```
import math
def dot_product(v1, v2):
    return sum([value1*value2 for value1, value2
                in zip(v1,v2)])


def cosin_sim(v1, v2):
    #compute cosine similarity
    prod = dot_product(v1, v2)
    len1 = math.sqrt(dot_product(v1, v1))
    len2 = math.sqrt(dot_product(v2, v2))
    return prod / (len1 * len2)

cosin_sim([1,2],[3,4])
>>> 0.9838699100999074
```

Cosine Similarity

- dot product expresses how much the two vectors are pointing in the same direction

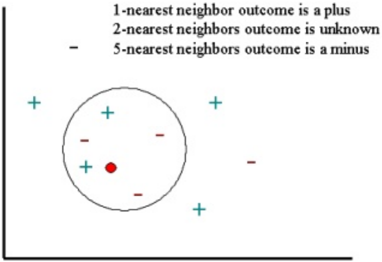
Cosine Similarity

- dot product expresses how much the two vectors are pointing in the same direction
- if two documents share a lot of common terms, their tf-idf vectors will point in a similar direction

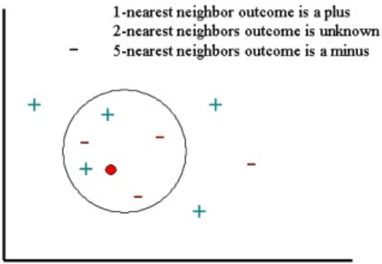
Cosine Similarity

- dot product expresses how much the two vectors are pointing in the same direction
- if two documents share a lot of common terms, their tf-idf vectors will point in a similar direction
- cosine similarity = an indicator how close the documents are in the semantics of their content

What if we have two winners ($k = 2$)?



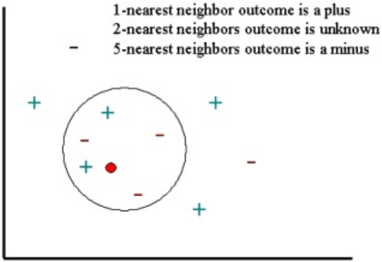
What if we have two winners ($k = 2$)?



Strategies:

1. Pick one of the winners at random

What if we have two winners ($k = 2$)?



Strategies:

1. Pick one of the winners at random
2. Reduce k until we find a unique winner

K Nearest Neighbor (KNN) Classification

```
#labels sorted from nearest to farthest
labels = ['sport', 'cars', 'religion'
                'religion','sport']
```

K Nearest Neighbor (KNN) Classification

```
#labels sorted from nearest to farthest
labels = ['sport', 'cars', 'religion'
                'religion','sport']
```

2 winners: 'sport' and 'religion'

K Nearest Neighbor (KNN) Classification

```
#labels sorted from nearest to farthest
labels = ['sport', 'cars', 'religion'
          'religion','sport']
```

2 winners: 'sport' and 'religion'

Reduce k until we find a unique winner:

reduced_labels = ???

K Nearest Neighbor (KNN) Classification

```
#labels sorted from nearest to farthest
labels = ['sport', 'cars', 'religion'
                'religion','sport']
```

2 winners: 'sport' and 'religion'

Reduce k until we find a unique winner

reduced_labels = labels[:-1]

```
print(reduced_labels)

>>> ['sport', 'cars', 'religion',  'religion']
```

K Nearest Neighbor (KNN) Classification

```
#labels sorted from nearest to farthest
labels = ['sport', 'cars', 'religion'
                    'religion','sport']
```

2 winners: 'sport' and 'religion'

Reduce k until we find a unique winner

reduced_labels = labels[:-1]

```
print(reduced_labels)

>>> ['sport', 'cars', 'religion',  'religion']
```

now 1 winner: 'religion'

K Nearest Neighbor (KNN) Classification

```
#labels sorted from nearest to farthest
labels = ['sport', 'cars', 'religion',  'politics']
```

Winner???

K Nearest Neighbor (KNN) Classification

```
labels = ['sport', 'cars', 'religion',  'politics']
```

Winner:

'sport'

K Nearest Neighbor (KNN) Classification

```
labels = ['sport', 'cars', 'cars', 'sport']
```

Winner???

K Nearest Neighbor (KNN) Classification

```
labels = ['sport', 'cars', 'cars',  'sport']
```

Winner:

'cars'

K Nearest Neighbor (KNN) Classification

```
def choose_one(labels):
    """labels are ordered from nearest to farthest"""

    counts = Counter(labels)
    winner, winner_count = counts.most_common(1)[0]

    # count number of winners in a list,
    # i.e. how many words with equal winner_count?
    ...

    #if unique winner, so return it
    ...

    #else: reduce the list and try again,
    # i.e call choose_one again but with reduced list
    ...
```

Counter

```
from collections import Counter
colors = ['red', 'blue', 'red', 'green',
                'blue', 'blue', 'red']
cnt = Counter(colors)
print(cnt)
>>> Counter({'red': 3, 'blue': 3, 'green': 1})

most_common_tuple = cnt.most_common(1)
print(most_common_tuple)
>>>[('red', 3)]

winner, winner_count = most_common_tuple[0]
print(winner, winner_count)
>>> red 3
```

Document Classification with KNN

- fixed set of elements (e.g., documents): $D = \{d_1, ...d_n\}$
- document d (data point) is represented by a vector of features: $d \in \mathbb{N}^k \rightarrow d = [x_1 x_2 ... x_k]$
- feature weights are numerical statistics (like TF-IDF)
- weights are not re-weighted during learning $\rightarrow$ KNN is **"non-parametric" classifier**

Document Classification with KNN

- fixed set of elements (e.g., documents): $D = \{d_1, ...d_n\}$
- document d (data point) is represented by a vector of features: $d \in \mathbb{N}^k \rightarrow d = [x_1 x_2 ... x_k]$
- feature weights are numerical statistics (like TF-IDF)
- weights are not re-weighted during learning $\rightarrow$ KNN is **"non-parametric" classifier**
- **Goal** - find the most similar document for a given document d and assign the same category (1NN classification)

Unsupervised: K-Means

- clustering algorithm

Unsupervised: K-Means

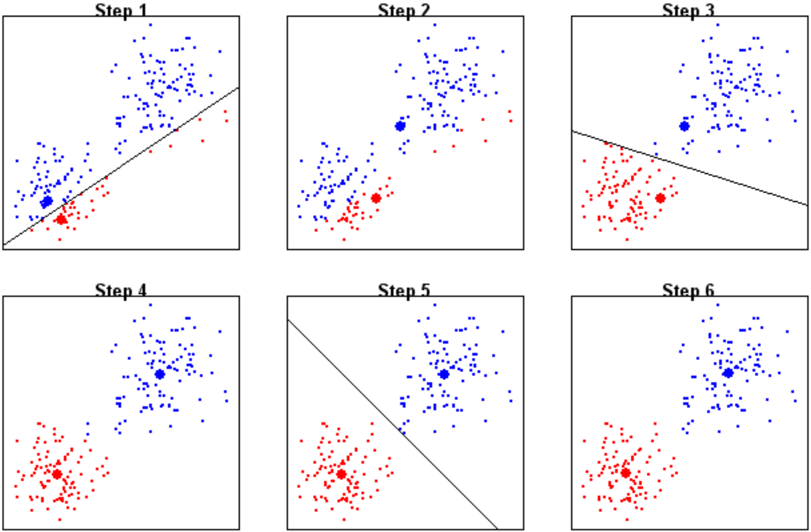
- clustering algorithm
- the number of clusters k is chosen in advance

Unsupervised: K-Means

- clustering algorithm
- the number of clusters k is chosen in advance
- partition the inputs into sets $S_1, ..., S_k$ using cluster centroids

K-Means

K-means clustering technique



K-Means

k-means clustering technique

1. randomly initialize cluster centroids
2. assign each point to the centroid to which it is closest:
   - use Euclidean distance to measure the distance

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{1}$$

3. recompute cluster centroids
4. go back to 2 until nothing changes (or it takes too long)

```
class KMeans:
  """performs k-means clustering"""

  def __init__(self, k):
    self.k = k # number of clusters
    self.means = None  # means of clusters

  def classify(self, input):
    """return the index of the cluster
    closest to the input (step 2)"""
    return min(range(self.k),
           key=lambda i:
           distance(input, self.means[i]))
```

Python min() Function

```
>>> a = [(0.2222, 1),(0.1111, 2),(0.6666, 3)]
>>> min(a, key= lambda x: x[0])
>>>(0.1111, 2)

>>> min(a, key= lambda x: x[1])
(0.2222, 1)

>>> k_clusters = 3
>>> input_vec = [1,2,3]
>>> means =[[1.5,2.5,3.5],[4.5,5.5,6.5],[7.5,8.5,9.5]]

>>> range(k_clusters)
[0,1,2]

>>> min(range(num_clusters), key=lambda x:
                        distance(input_vec,means[x]))
```

K-Means

```
def train(self, inputs):
    # choose k random points as the initial means
    self.means = random.sample(inputs, self.k)#step 1
    assignments = None
    while True:
        # Find new assignments
        new_assignments = map(self.classify, inputs)
        if assignments == new_assignments:
            return # If nothing changed, we're done.

        assignments = new_assignments
        for i in range(self.k): #compute new means
            i_points = [p for p, a in zip(inputs,
                        assignments) if a == i]
            if i_points:
                self.means[i] = mean(i_points)
```

Map

```
r = map(func, seq)

import functools
def fahrenheit(T):
    return ((9.0/5)*T + 32)
temp = [36.5, 37, 37.5, 39]
F = map(fahrenheit, temp)

print(list(F))
>>> [97.7, 98.60000000000001, 99.5, 102.2]
```

K-Means: Real Example

- organize meetup for users

K-Means: Real Example

- organize meetup for users
- goal - choose 3 meetup locations convenient for all users

```
clusterer = KMeans(3)
clusterer.train(inputs)
print(clusterer.means)
```

K-Means: Real Example

- organize meetup for users
- goal - choose 3 meetup locations convenient for all users

```
clusterer = KMeans(3)
clusterer.train(inputs)
print(clusterer.means)
```

- you find three clusters and you look for meetup venues near those locations

Kmeans with NLTK

```
from nltk import cluster
from nltk.cluster import euclidean_distance
from numpy import array
vectors = [array(f) for f in [[3, 3], [1, 2], [4, 2],
                              [4, 0], [2, 3], [3, 1]]]
clusterer = cluster.KMeansClusterer(2,
                    euclidean_distance)
clusters = clusterer.cluster(vectors)
print('Clustered:', vectors)
print('As:', clusters)
print('Means:', clusterer.means())

>>> Clustered:[array([3,3]), array([1,2]),
array([4,2]),array([4,0]),array([2,3]),array([3,1])]
>>> As: [0, 0, 0, 1, 0, 1]
>>> Means: [array([ 2.5,  2.5]), array([ 3.5,  0.5])]
```

Kmeans with NLTK

```
...
# classify a new vector
vector = array([3, 3])
print('classify(%s):' % vector)
print(clusterer.classify(vector))

>>> classify([3 3]):
>>> 0
```

Conclusion

- K-means is a **clustering** or **classification** algorithm?

Conclusion

- K-means is a **clustering** or **classification** algorithm?
  - → **clustering** algorithm

Conclusion

- K-means is a **clustering** or **classification** algorithm?
    - $\rightarrow$ **clustering** algorithm
    - partitions points into K clusters: points in each cluster tend to be near each other

Conclusion

- K-means is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **clustering** algorithm
  - partitions points into K clusters: points in each cluster tend to be near each other
  - **supervised** or **unsupervised**?

Conclusion

- K-means is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **clustering** algorithm
  - partitions points into K clusters: points in each cluster tend to be near each other
  - $\rightarrow$ **unsupervised**: points have no external classification
- K-nearest neighbors is a **clustering** or **classification** algorithm?

Conclusion

- K-means is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **clustering** algorithm
  - partitions points into K clusters: points in each cluster tend to be near each other
  - $\rightarrow$ **unsupervised**: points have no external classification
- K-nearest neighbors is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **classification** algorithm

Conclusion

- K-means is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **clustering** algorithm
  - partitions points into K clusters: points in each cluster tend to be near each other
  - $\rightarrow$ **unsupervised**: points have no external classification
- K-nearest neighbors is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **classification** algorithm
  - determines the classification of a new point

Conclusion

- K-means is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **clustering** algorithm
  - partitions points into K clusters: points in each cluster tend to be near each other
  - $\rightarrow$ **unsupervised**: points have no external classification
- K-nearest neighbors is a **clustering** or **classification** algorithm?
  - $\rightarrow$ **classification** algorithm
  - determines the classification of a new point
  - **supervised** or **unsupervised**?

Conclusion

- K-means is a **clustering** or **classification** algorithm?
  - → **clustering** algorithm
  - partitions points into K clusters: points in each cluster tend to be near each other
  - → **unsupervised**: points have no external classification
- K-nearest neighbors is a **clustering** or **classification** algorithm?
  - → **classification** algorithm
  - determines the classification of a new point
  - **supervised** or **unsupervised**?
  - **supervised:** classifies a point based on the known classification of other points.

References

Joel Grus (2015).
Data Science from Scratch.
OReilly.
http://choonsiong.com/public/books/Big%20Data/Data%20Science%20from%20Scratch.pdf

Christopher D. Manning, Hinrich Schtze 2000).
Foundations of Statistical Natural Language Processing
The MIT Press Cambridge, Massachusetts London, England.
http://ics.upjs.sk/~pero/web/documents/pillar/Manning_Schuetze_StatisticalNLP.pdf