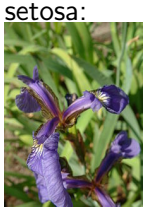
# Lecture 4: *k*-Nearest Neighbor

CS 167: Machine Learning

## Example: Iris Data Set

| sepal length | sepal width | petal length | petal width | species |
|---|---|---|---|---|
| 4.6 | 3.2 | 1.4 | 0.2 | Iris-setosa |
| 6.2 | 2.8 | 4.8 | 1.8 | Iris-virginica |
| 6.2 | 2.2 | 4.5 | 1.5 | Iris-versicolor |
| 6.3 | 2.7 | 4.9 | 1.8 | Iris-virginica |
| 6.9 | 3.1 | 5.1 | 2.3 | Iris-virginica |
| 6.6 | 2.9 | 4.6 | 1.3 | Iris-versicolor |
| 6.9 | 3.1 | 5.4 | 2.1 | Iris-virginica |
| 4.9 | 2.5 | 4.5 | 1.7 | Iris-virginica |
| 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |
| 6.7 | 3.1 | 5.6 | 2.4 | Iris-virginica |
| 6.3 | 2.3 | 4.4 | 1.3 | Iris-versicolor |
| 6.4 | 2.7 | 5.3 | 1.9 | Iris-virginica |
| 5 | 3.6 | 1.4 | 0.2 | Iris-setosa |

setosa:

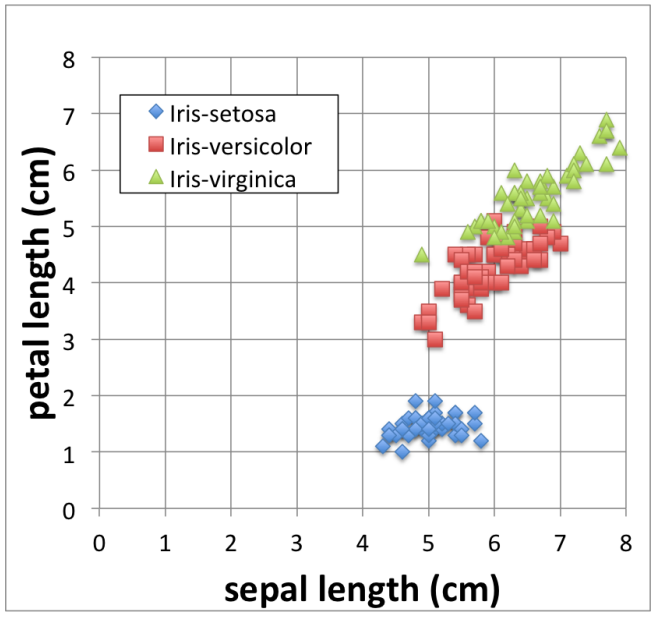


versicolor:

virginica:

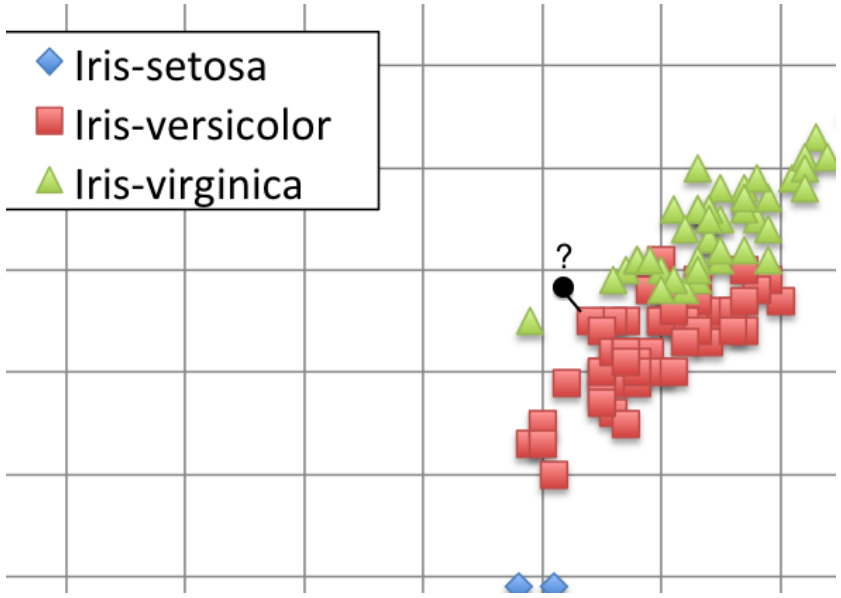


## Visualizing Petal vs. Sepal Lengths

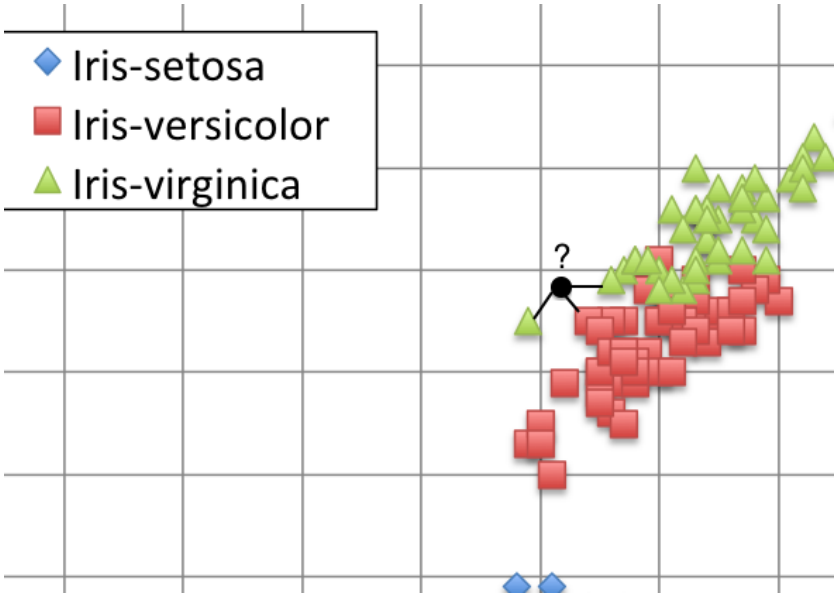





## Nearest-Neighbor Algorithm: Predict *versicolor*



## 3-Nearest-Neighbor Algorithm: Predict *verginica*






## $k$-Nearest-Neighbor Algorithm

**$k$-Nearest Neighbor Algorithm**: Predict the *most commonly appearing* class among the $k$ closest training examples.

Defining Nearness: you should come up with a way to determine how close any two train/test examples are.

Usually, you treat the $n$ attribute values as members of an $n$-dimensional space (in math, $\mathbb{R}^n$) and compute the *Euclidean Distance*.

Example in 2D space: distance between $(x_1, y_1)$ and $(x_2, y_2)$ is

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



## Distance in $n$-Dimenstional Space

Suppose attributes of example $x$ are denoted

$$\langle a_1(x), a_2(x), \ldots, a_n(x) \rangle$$

then the Euclidean distance is

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Example:

| sepal length | sepal width | petal length | petal width | species |
|---|---|---|---|---|
| 4.6 | 3.2 | 1.4 | 0.2 | Iris-setosa |
| 6.2 | 2.8 | 4.8 | 1.8 | Iris-virginica |



## Distance in $n$-Dimensional Space

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

Example:

| sepal length | sepal width | petal length | petal width | species |
|---|---|---|---|---|
| 4.6 | 3.2 | 1.4 | 0.2 | Iris-setosa |
| 6.2 | 2.8 | 4.8 | 1.8 | Iris-virginica |

$$\sqrt{(4.6 - 6.2)^2 + (3.2 - 2.8)^2 + (1.4 - 4.8)^2 + (0.2 - 1.8)^2}$$

$$\approx 4.1$$

## Discussion Question

**What do we do if the attributes aren't numbers?**

| Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Dodge, Mast | male | 4 | 0 | 2 | 33638 | 81.8583 | A34 | S |
| 0 | 3 | Williams, Mr | male | | 0 | 0 | A/5 2466 | 8.05 | | S |
| 1 | 1 | Peuchen, Ma | male | 52 | 0 | 0 | 113786 | 30.5 | C104 | S |
| 1 | 3 | Sandstrom, M | female | 4 | 1 | 1 | PP 9549 | 16.7 | G6 | S |
| 1 | 2 | Richards, Ma | male | 3 | 1 | 1 | 29106 | 18.75 | | S |
| 0 | 2 | Collander, M | male | 28 | 0 | 0 | 248740 | 13 | | S |
| 1 | 2 | Mellinger, M | female | 41 | 0 | 1 | 250644 | 19.5 | | S |
| 0 | 3 | Sage, Miss. C | female | | 8 | 2 | CA. 2343 | 69.55 | | S |
| 1 | 1 | Bjornstrom-S | male | 28 | 0 | 0 | 110564 | 26.55 | C52 | S |
| 0 | 2 | Gavey, Mr. L | male | 26 | 0 | 0 | 31028 | 10.5 | | S |
| 0 | 1 | White, Mr. P | male | 54 | 0 | 1 | 35281 | 77.2875 | D26 | S |
| 0 | 3 | Ford, Mrs. Ec | female | 48 | 1 | 3 | W./C. 6608 | 34.375 | | S |
| 0 | 3 | Kiernan, Mr. | male | | 1 | 0 | 367229 | 7.75 | | Q |
| 0 | 3 | Charters, Mr | male | 21 | 0 | 0 | A/5. 13032 | 7.7333 | | Q |
| 0 | 3 | Tomlin, Mr. E | male | 30.5 | 0 | 0 | 364499 | 8.05 | | S |



## Discussion Question

**What if the target function is continuous? Can we do regression? How?**

| sepal length | sepal width | petal length | petal width | Prettiness Factor |
|---|---|---|---|---|
| 5.8 | 2.7 | 5.1 | 1.9 | 68 |
| 6.7 | 3.1 | 4.4 | 1.4 | 0 |
| 6.3 | 3.3 | 6 | 2.5 | 38 |
| 6.2 | 2.2 | 4.5 | 1.5 | 46 |
| 5.8 | 2.7 | 5.1 | 1.9 | 0 |
| 7.2 | 3.2 | 6 | 1.8 | 16 |
| 5.4 | 3 | 4.5 | 1.5 | 22 |
| 5.5 | 2.4 | 3.7 | 1 | 40 |
| 6.9 | 3.1 | 5.1 | 2.3 | 26 |
| 5.7 | 2.8 | 4.5 | 1.3 | 76 |



## Variant: Weighted $k$-Nearest Neighbor

Instead of taking the average of the $k$ neighbors, you could weight the average based on how close they are:

$$w_{q,i} = \frac{1}{d(x_q, x_i)^2}$$

if $f(x)$ is the value of the target function on example $x$, then weighted $k$-NN predicts for $f(x_q)$:

$$\frac{\sum_{i=1}^{k} w_{q,i} f(x_i)}{\sum_{i=1}^{k} w_{q,i}}$$



## Discussion Questions

**Which problems are better for (weighted) $k$-NN?**

- data with lots of attributes?
- data with fewer attributes?

**What do we do if there are some irrelevant attributes like *ticket number*?**

**How can we automatically determine which attributes are relevant and which aren't?**

## Discussion Questions

**Which is better for (weighted) $k$-NN?**

- lots of training examples?
- fewer training examples?

**What are the positives/negatives to more training examples?**



## Python Function Example

```
def findMin(some_list):
    smallest = float("inf")
    for n in some_list:
        if n < smallest:
            smallest = n
    return smallest


print( findMin([6,1,3,8,-4,0,-1,10]) )
```

```
-4
```



## More Titanic Exercises

Continuing with the *titanic* data:
The rows in your 2D list probably look something like this:
`['1', '1', '"Dodge', ' Master.  Washington"', 'male', '4', '0', '2', '33638', '81.8583', 'A34', 'S']`

You can use the `float()` function to convert strings to floats like this:

```
float("38") #returns the numerical value 38.0
```

Exercise 7: Write a function that can take the 2D list and a column number as arguments and then convert all values in that column to floats. Call the function for each numerical column in the data (i.e., survived, pclass, age, sibsp, parch, & fare).

Exercise 8: Figure out a way to convert the `Sex` and `Embarked` columns to numbers as well.



## Computing Distance

Exercise 9: Write a function that can take two examples (e.g., two rows from the 2D list) as arguments and compute the distance between them (look up how to do square roots). Do *one* of the following to deal with irrelevant columns:

- Write a function that can remove a column entirely from the 2D list.
- Write the distance function to take in a list of columns to use in computing the distance (and ignore the rest).

**Make sure to test this well!** Pick two rows from the data and pass them to your function. Print the result, and verify that it is correct by calculating the distance by hand.

## Split into training and test sets

Exercise 10: Set aside some examples (say 10% of the examples) from the training set to use for testing. I suggest you use the `shuffle` function from the `random` library to shuffle the rows and then use slicing to grab part for testing (call this `test_data`) and part for training (call this `train_data`) - make sure that they don't overlap!

```
import random
list_to_shuffle = [1,2,3,4,5,6,7]
random.shuffle(list_to_shuffle)
print(list_to_shuffle)
```

```
[1, 5, 3, 2, 6, 4, 7]
```



## Implementing 1NN

Exercise 11: Write a function that can take a single test example (one of the rows in `test_data`) and find its nearest neighbor from the training set (`train_data`).

- one test example gets compared to *every* training example
- keep track of the training example that has the lowest distance compared to the one test example.
- Test it out. Print out both the test example and the closest training example that you found. They should be similar (though they shouldn't be exactly the same person). Does it seem to give the right answers?

Now that you have the nearest neighbor, you will use the *Survived* column of the neighbor in the training set as your prediction for the example in the test set.

- change it so that the function returns this prediction instead of the actual neighbor example



## Computing Accuracy

Exercise 12: Write a function that can return a list of predictions for a whole test set.

- Loop through every test example, for each one
  - call the function from Exercise 11, passing in this one testing example and the *whole training set*
  - it should return the prediction for this test example
  - put the returned prediction into a running list

Exercise 13: Write a function that compares the predictions with the actual `Survived` values in the test set and then returns the *accuracy*: (correct predictions)/(number of test examples)



## Tips for Extending to (weighted) $k$NN

You computed the prediction for 1NN, to make it into $k$NN

- Instead of finding the single nearest neighbor, you need to keep a list of size $k$ (a parameter) of the $k$ closest neighbors from the training set you've seen so far. I suggest keeping it sorted by distance - so your list of size $k$ might have to include both the neighbor and that neighbor's distance to the test example.
- When you look at the next training example, you will compute its distance to the test example, and then compare to the first thing in the $k$-closest-neighbors list. If it is closer, insert it into the list right there. If not, compare to the second neighbor, and so on.
- You will have to include put the first $k$ examples in no-matter what (but still keep them sorted!), after that, if you ever put a new one in, remove the one at the end.
- After you've gone through them all, predict the most common `Survived` value among those in the $k$-sized neighbor list.

# Tips for Extending to (weighted) $k$NN

When extending for the weighted version, instead of predicting the most common `Survived` value, take a weighted average, where the weight is

$$\frac{1}{\text{distance}^2}$$

If this weighted average is above 0.5, predict 1. If it is below, predict 0.