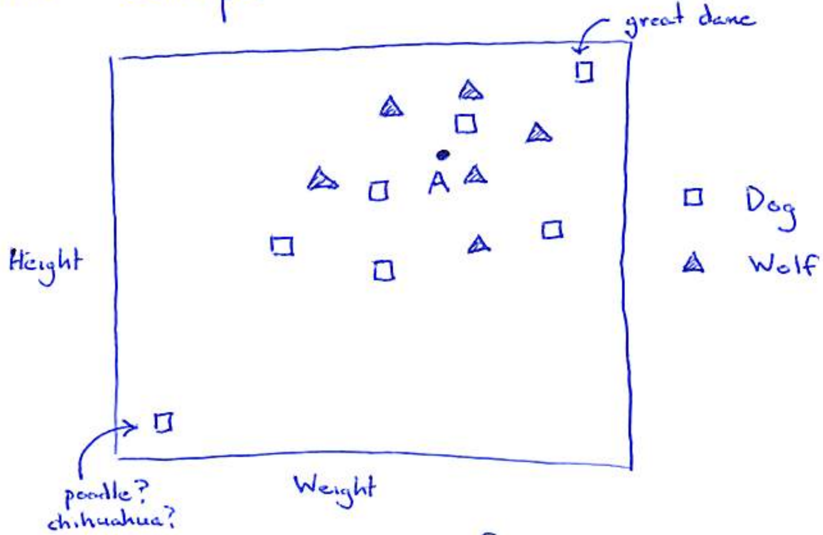
1

k - Nearest Neighbors

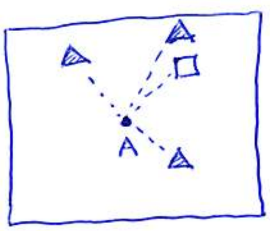
Office Hours			
Nick	M 4-6pm		EPS 1316
Duncan	TR	?	MSD 1147
Michael	F 9-11am		MSD 1147

The setup:



How can we classify point A as a dog or a wolf?

Look at points closest to A:



Most are wolves, so A is probably a wolf.

Two questions:

1. How many neighbors should we use?

   This is a bias-variance tradeoff. Adding more neighbors biases the prediction, but lowers the variance (imagine moving the test point around the space — prediction changes less frequently if we use lots of neighbors).

   However, the best way to choose k is with cross validation.

   It's also nice to have k be odd, so there are no ties to break (but odd k might not be optimal).

Break ties randomly, or have votes weighted by inverse distance (closer points get more weight). For a random tie-breaker, put more probability on closer points.

② How should we measure distance?

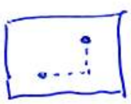
It depends on the problem! Again, use cross validation.

You have a lot of options:

Minkowski Distances

- Euclidean distance "straight-line" — $p = 2$

$$\left[(x_1-y_1)^2 + (x_2-y_2)^2 + \dots + (x_m-y_m)^2\right]^{1/2}$$

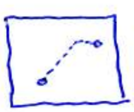


- Manhattan distance "taxicab" — $p = 1$

$$\left[|x_1-y_1|^1 + |x_2-y_2|^1 + \dots + |x_m-y_m|^1\right]^{1/1}$$

- Chessboard distance — $p = \infty$

Movement on an integer lattice / approx warehouse crane (only moves on angles that are multiples of 45°)



$$\max_{1 \le i \le m} |x_i - y_i|$$

use m for dimension of space

- Minkowski distance

$$\left[|x_1-y_1|^p + \dots + |x_m-y_m|^p\right]^{1/p}$$

Varies between taxicab and straight-line for $1 \le p \le 2$

- Problem-specific distances are also possible

See Elements of Statistical Learning for more details on kNN.

[3]

Implementing k-Nearest Neighbors

What do we start with?

Test points (data frame), training points (data frame), labels, k

What's the output?

Predictions

So write a function:

```
knn = function (test, train, labels, k) {
  # ...
  return (predictions)
}
```

Suppose there's just one test point. How do we make a prediction?

① Compute distance to all training points. dist()

With dist() this step is easy to generalize to many test points (and doesn't depend on k).

② Order the training points by distance. order()

③ Get labels for top k. [ ]

④ Vote, breaking ties if necessary. table(), max()

The order() function gives back indexes — use them to subset labels.

Suggestion: write a function for steps 2-3 on a single test point. Then use apply.

```
vote = function (distances, labels) {
  # ...
  return (prediction)
}
```

Keeping the distance calculation separate will make cross validation much more efficient! We'll discuss this next week.