CPSC 340:
Machine Learning and Data Mining

Non-Parametric Models

Admin

- Course add/drop deadline tomorrow.
  - Waitlist down to 10.
- Assignment 1 is due Sunday.
  - Start the assignment ASAP, if you haven't already.
  - It is significantly longer and more challenging than Assignment 0
  - There were some typos/bugs, so please merge in the pull requests (PRs)
    - Info has been added to the general homework instructions
- Online feedback system now has a persistent home
  - http://skaha.cs.ubc.ca:11616/cpsc340/
  - Link also from the course website, using the "links" shortcut
  - I removed the "ask" option because it's hard for me to monitor.
- Tutorials today/tomorrow: Bita presents plotting, naive Bayes.

UBC Department of Computer Science
Undergraduate Events

More details @ https://my.cs.ubc.ca/students/development/events

Tech@RBC Info Session

Mon., Jan 16
5:45 pm
DMP 110

Resume Drop-in Editing

Tues., Jan 17, 12:30 – 3 pm
Fri., Jan 20, 2 – 4 pm
Rm 253 ICICS/CS

Vision Critical Tech Interview Practice Session (for co-op, interns)

Tues., Jan 17
5:30 pm
DMP 110

Garmin Cochrane Info Session (Navigation & Wearable Technology Company)

Wed., Jan 18
5:30 pm
Rm 2071 Brock Hall

AxiomZen Build An App Workshop

Thurs., Jan 19
5:30 pm
DMP 110

Parametric vs. Non-Parametric

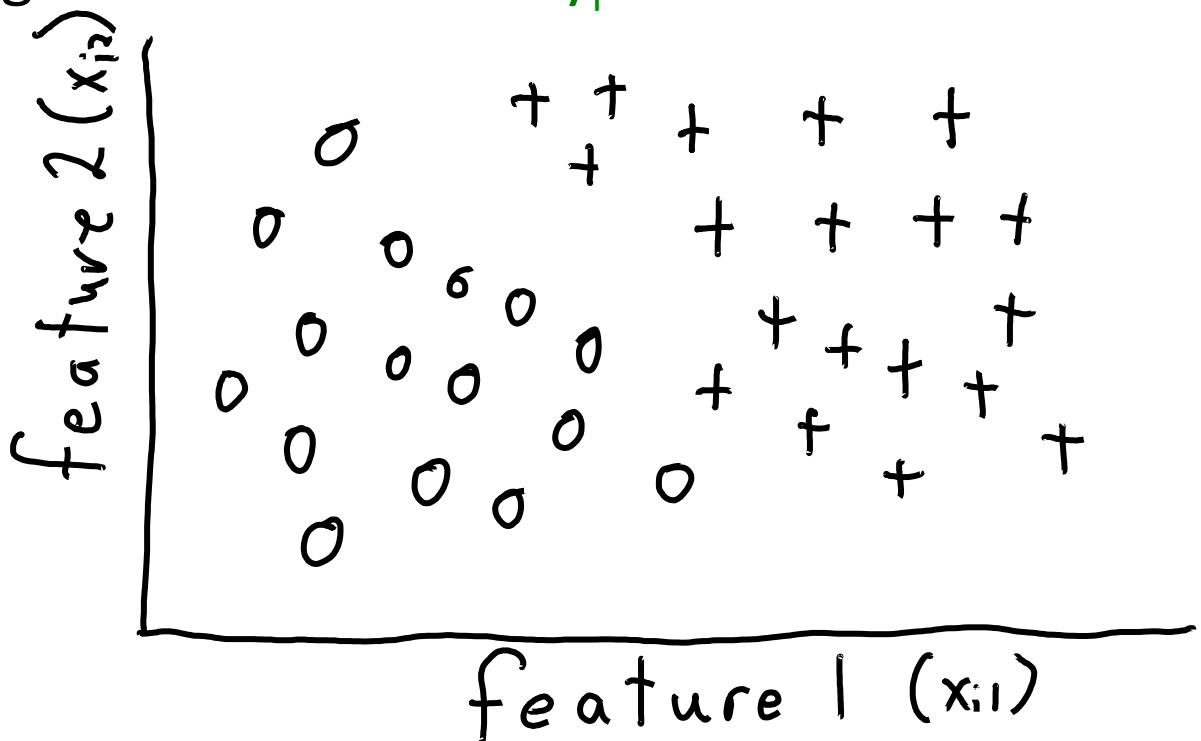
- Decision trees and naïve Bayes are often not very accurate.
  - Greedy rules or conditional independence might be bad assumptions.
  - They are also parametric models.

Parametric vs. Non-Parametric

- Parametric models:
  - Have a fixed number of parameters: size of "model" is O(1) in terms 'n'.
    - E.g., decision tree just stores rules.
    - E.g., naïve Bayes just stores counts.
  - You can estimate the fixed parameters more accurately with more data.
  - But eventually more data doesn't help: model is too simple.
- Non-parametric models:
  - Number of parameters grows with 'n': size of "model" depends on 'n'.
  - Model gets more complicated as you get more data.
  - E.g., decision tree whose depth *grows with the number of examples*.

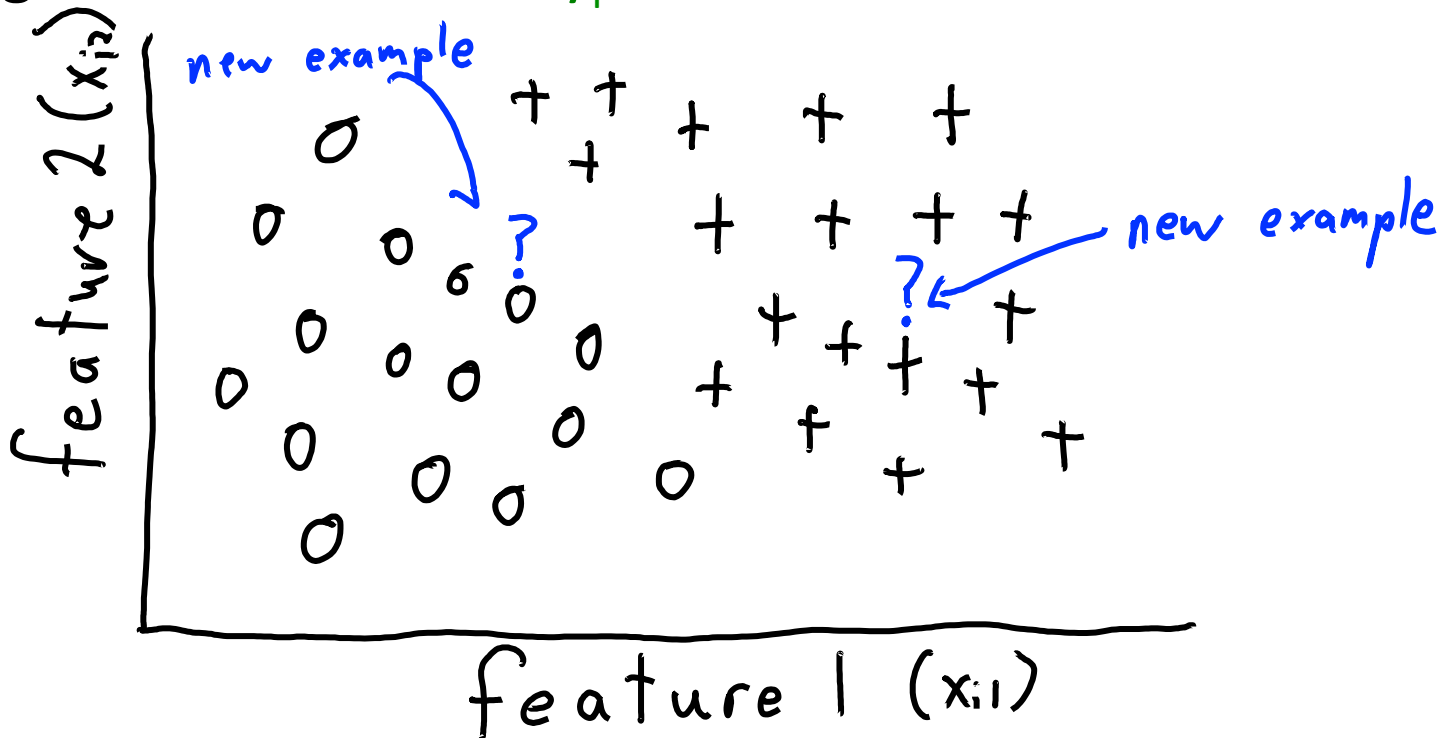
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object ‘x’:
  1. Find ‘k’ training examples x_i that are most “similar” to x.
  2. Classify using the mode of their y_i.



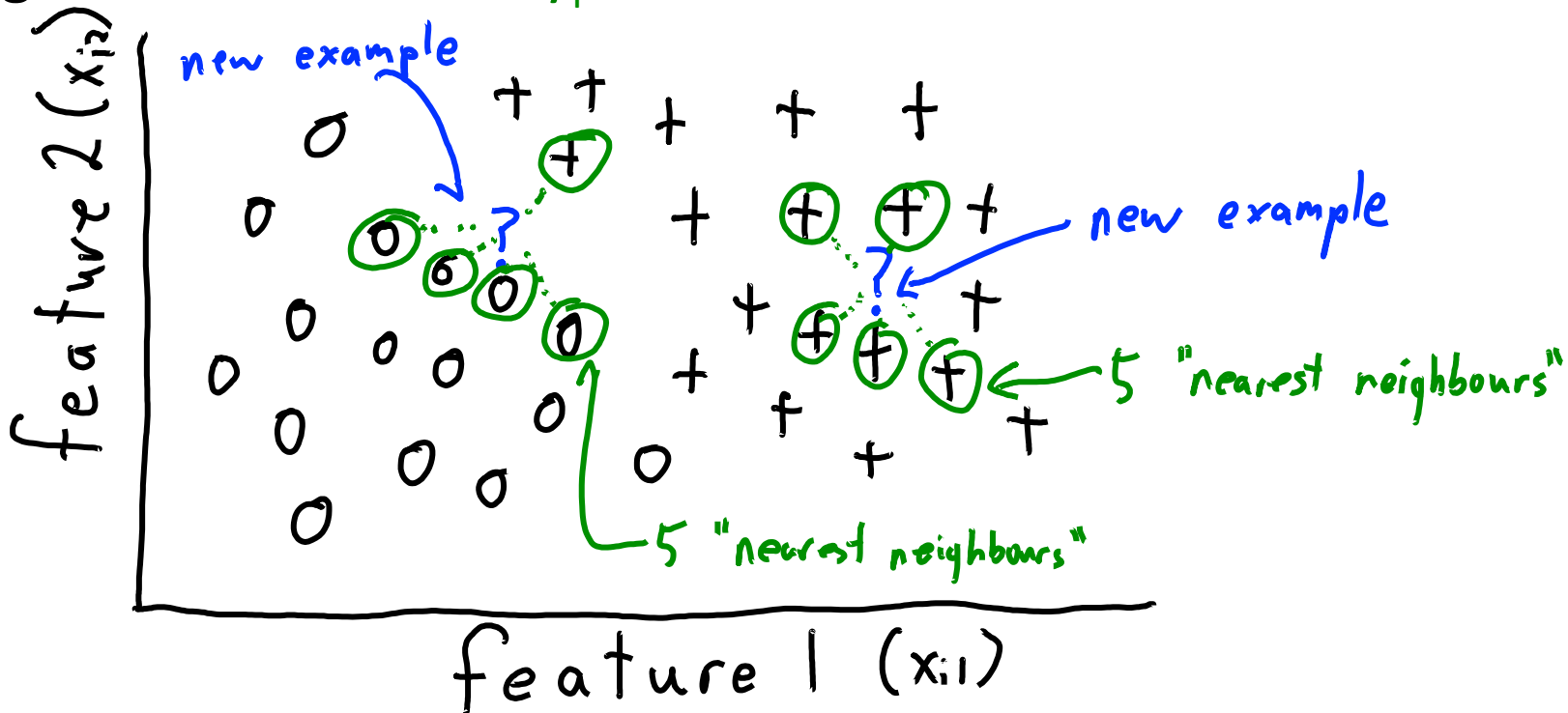
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object ‘x’:
  1. Find ‘k’ training examples x_i that are most “similar” to x.
  2. Classify using the mode of their y_i.



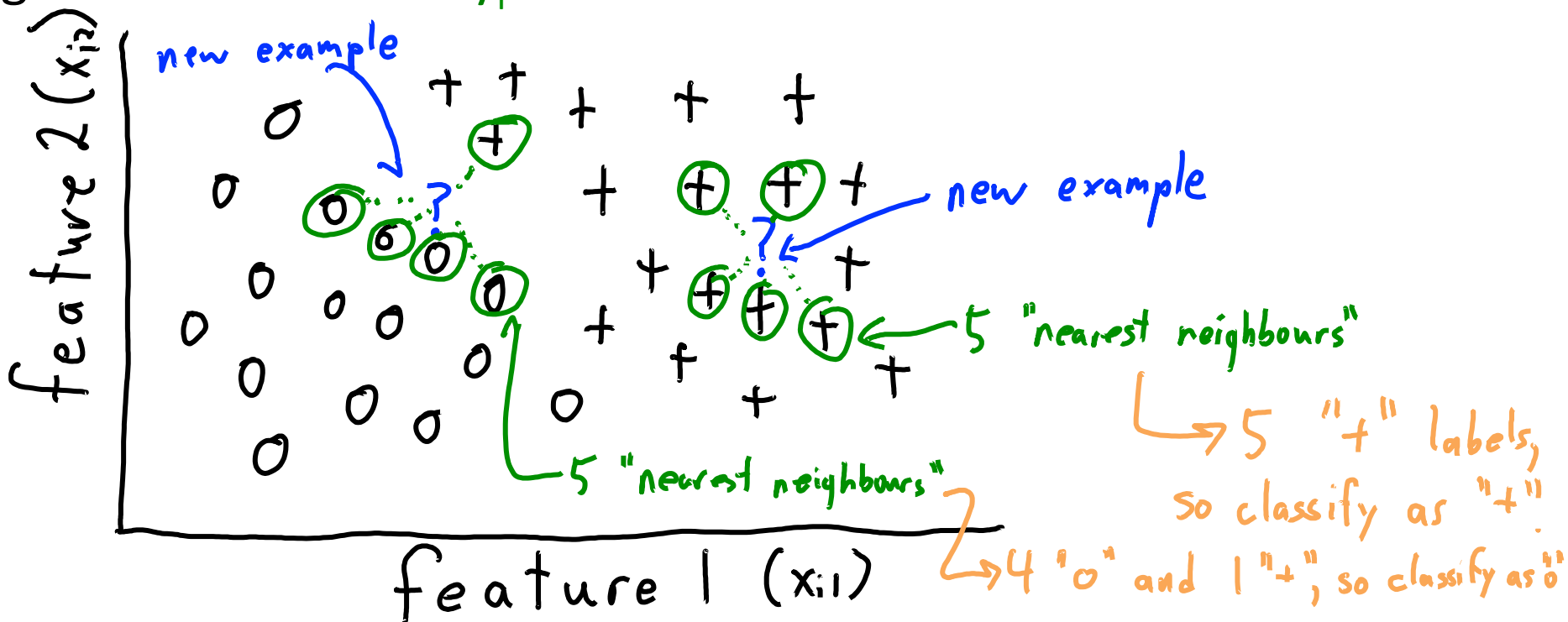
K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object 'x':
  1. Find 'k' training examples x_i that are most "similar" to x.
  2. Classify using the mode of their y_i.



K-Nearest Neighbours (KNN)

- Classical non-parametric classifier is k-nearest neighbours (KNN).
- KNN algorithm for classifying an object 'x':
  1. Find 'k' training examples x_i that are most "similar" to x.
  2. Classify using the mode of their y_i.



K-Nearest Neighbours (KNN)

- Assumption:
  - Objects with similar features likely have similar labels.

- There is no training phase ("lazy" learning).
  - You just store the training data.
  - Non-parametric because the size of the model is O(nd).

- But predictions are expensive: O(nd) to classify 1 test object.
  - Tons of work on reducing this cost (we'll discuss these later).

How to Define 'Nearest'?

- There are many ways to define similarity between x_i and x_j.
- Most common is Euclidean distance:

$$D(x_1, x_2) = \sqrt{\sum_{j=1}^{d} (x_{1j} - x_{2j})^2}$$

- Other possibilities:
  - L_1 distance:

$$D(x_1, x_2) = \sum_{j=1}^{d} |x_{1j} - x_{2j}|$$

  - Jaccard similarity (binary):

$$D(x_1, x_2) = \frac{x_1 \cap x_2}{x_1 \cup x_2}$$

$x_1 \cap x_2$ → # times both are '1'

$x_1 \cup x_2$ → # times either is '1'

  - Cosine similarity.
  - Distance after dimensionality reduction (later in course).
  - Metric learning (*learn* the best distance function).

Consistency of KNN

- With a small dataset, KNN model will be very simple.
- With more data, model gets more complicated:
  - Starts to detect subtle differences between examples.
- With a fixed ‘k’, it has appealing consistency properties:
  - With binary labels and under mild assumptions:
    - As ‘n’ goes to infinity, KNN test error is less than twice irreducible error.
- A “best” machine learning model as ‘n’ goes to ∞.

Summary

1. Non-parametric models grow with number of training examples.
2. K-Nearest neighbours:
   - A simple non-parametric classifier.
   - Appealing consistency properties.

- Next Time:
  - Learning behind Microsoft Kinect.

- All the remaining slides are “bonus”.
- We may go through them briefly, if time permits.

Consistency of KNN (continued)

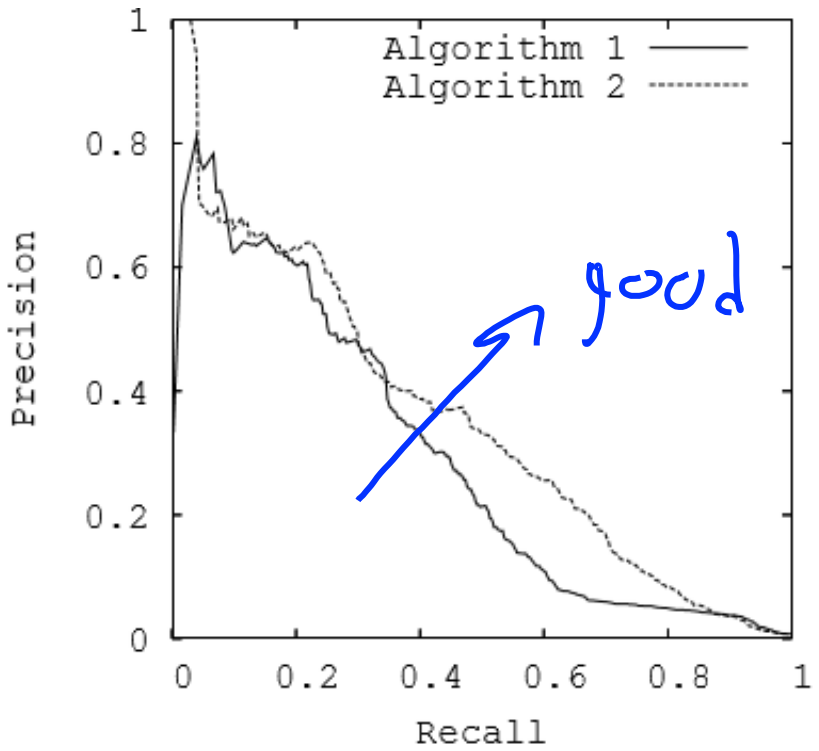
- Stone's Theorem:
  - If k/n goes to zero and 'k' goes to infinity:
    - KNN is 'universally consistent': test error converges to the irreducible error.
    - First algorithm shown to have this property.
- Does Stone's Theorem violate the no free lunch theorem?
  - No, requires assumptions on data and says nothing about finite training sets.

Bonus Slide: Other Performance Measures

- Classification error might be wrong measure:
  - Use weighted classification error if have different costs.
  - Might want to use things like Jaccard measure: TP/(TP + FP + FN).
- Often, we report precision and recall (want both to be high):
  - Precision: "if I classify as spam, what is the probability it actually is spam?"
    - Precision = TP/(TP + FP).
    - High precision means the filtered messages are likely to really be spam.
  - Recall: "if a message is spam, what is probability it is classified as spam?"
    - Recall = TP/(TP + FN)
    - High recall means that most spam messages are filtered.

Bonus Slide: Precision-Recall Curve

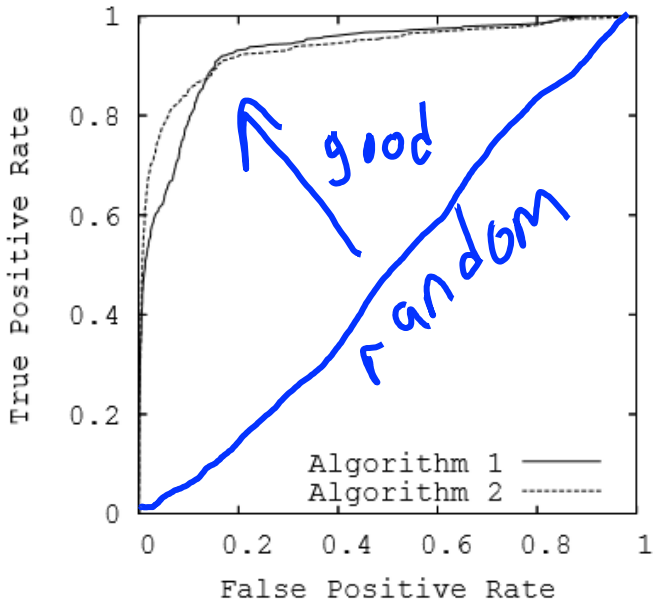
- Consider the rule $p(y_i = \text{'spam'} \mid x_i) > t$, for threshold 't'.
- Precision-recall (PR) curve plots precision vs. recall as 't' varies.



http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf

Bonus Slide: ROC Curve

- Receiver operating characteristic (ROC) curve:
  - Plot true positive rate (recall) vs. false positive rate (FP/FP+TN).
    (negative examples classified as positive)



  - Diagonal is random, perfect classifier would be in upper left.
  - Sometimes papers report area under curve (AUC).

http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf

Bonus Slide: Computing all distances in Matlab

Note: Matlab can be slow at executing operations in 'for' loops, but allows extremely-fast hardware-dependent vector and matrix operations. By taking advantage of SIMD registers and multiple cores (and faster matrix-multiplication algorithms), vector and matrix operations in Matlab will often be several times faster than if you implemented them yourself in a fast language like C. If you find that calculating the Euclidean distances between all pairs of points takes too long, the following code will form a matrix containing the squared Euclidean distances between all training and test points:

```
[n,d] = size(X);
[t,d] = size(Xtest);
D = X.^2*ones(d,t) + ones(n,d)*(Xtest').^2 - 2*X*Xtest';
```

Element $D(i, j)$ gives the squared Euclidean distance between training point i and testing point j.

Bonus Slide: Computing all distances in Matlab

Note: Matlab can be slow at executing operations in 'for' loops, but allows extremely-fast hardware-dependent vector and matrix operations. By taking advantage of SIMD registers and multiple cores (and faster matrix-multiplication algorithms), vector and matrix operations in Matlab will often be several times faster than if you implemented them yourself in a fast language like C. If you find that calculating the Euclidean distances between all pairs of points takes too long, the following code will form a matrix containing the squared Euclidean distances between all training and test points:

```
[n,d] = size(X);
[t,d] = size(Xtest);
D = X.^2*ones(d,t) + ones(n,d)*(Xtest').^2 - 2*X*Xtest';
```

Element $D(i,j)$ gives the squared Euclidean distance between train

The trick to figuring out what matrix multiplication operations like this do is usually to figure what an individual element of the result looks like by writing it as an inner product or writing it in summation notation (as you'll do in the tutorials this week).

In this case we have that:

- Element (i,j) of "X.^2" is given by $X_{i,j}^2$.

- Element (i,j) of "X.^2*ones(d,t)" is given by $\sum_{j=1}^{d} X_{i,j}^2 \cdot 1 = \|x_i\|_2^2$ where x_i is training example i.

- By the same logic, element (i,j) of "ones(n,d)*(Xtest').^2" gives $\|\hat{x}_j\|_2^2$ where $\hat{x}_j$ is test example j.

- Finally, element (i,j) of "2*X*Xtest'" gives $2x_i^T\hat{x}_j$.

Putting everything together, each element (i,j) of the result gives

$$\|x_i\|_2^2 - 2x_i^T\hat{x}_j + \|\hat{x}\|_2^2.$$

Let's re-write this as

$$x_i^T x_i - 2x_i^T\hat{x}_j + \hat{x}_j^T\hat{x}_j,$$

and if you now "complete the square" you get

$$(x_i - \hat{x}_j)^T(x_i - \hat{x}_j),$$

which is equal to $\|x_i - \hat{x}_j\|_2^2$.

(You could take the square root if you want the Euclidean distance, but since that won't change the ordering of neighbours it isn't necessary.)