Let’s do some KNNs

Logan and *Ishan*

01

Project Goal

KNNs

1. Take a test instance
2. Sort all train instances by distance to it
3. Take the K closest ones
4. Majority vote for class
5. Predict!

KNNs - the special sauce

1. Take a test instance
2. Sort all train instances by distance to it
3. Take the K closest ones
4. Majority vote for class
5. Predict!

We can do something else here!

02

Specifics, Techniques

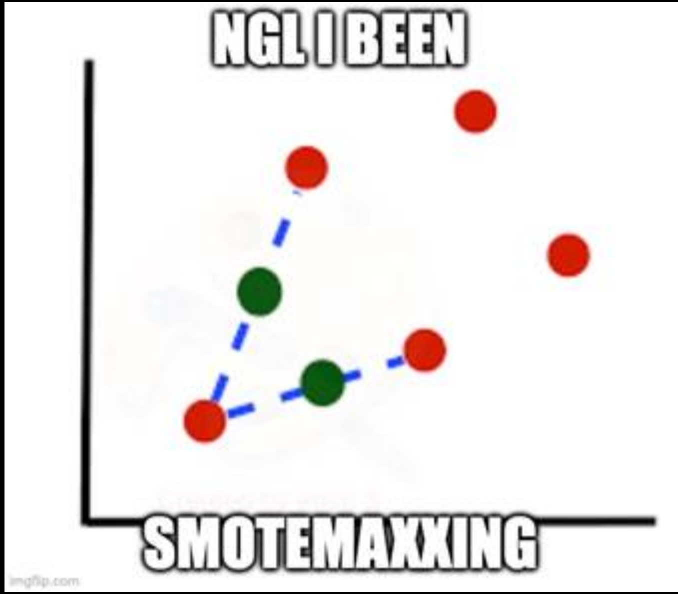
Our Versions:

1. Regular KNN (majority vote)
2. Weighted Averaging (vote but weighted by 1/frequency of class)
3. Weighted Distancing (weighted 1/distance)
4. Local Density Averaging
    - Ok, this one's a little complex:
        - For each neighbor:
            - Look at all other train points within radius R
            - Count em up, get local density
            - Normalize local density between all the neighbors
            - Weight by this normalized value

1. All of em combined

Preprocessing?

- SMOTE



What is SMOTE?

- Synthetic Minority Oversampling TEchnique
- Basically, take minority class, choose a point, grab the K-nearest neighbors, and place a new synthetic point somewhere randomly on the line in between the neighbor and the point. Repeat until all classes have the same number of instances.

So, our categories:

- Regular KNN
- Weighted Averaging KNN
- Weighted Distancing KNN
- Local Density Averaging KNN
- All Methods Combined KNN
- Regular KNN with SMOTE
- Weighted Averaging KNN with SMOTE
- Weighted Distancing KNN with SMOTE
- Local Density Averaging KNN with SMOTE
- All Methods Combined KNN with SMOTE

How did we find the optimal K and R values?

- Original plan was to search through them by trying a bunch
- Problem: code takes a really long time to run
- Solution: try a few, and hope we found the best one! (we're pretty confident)

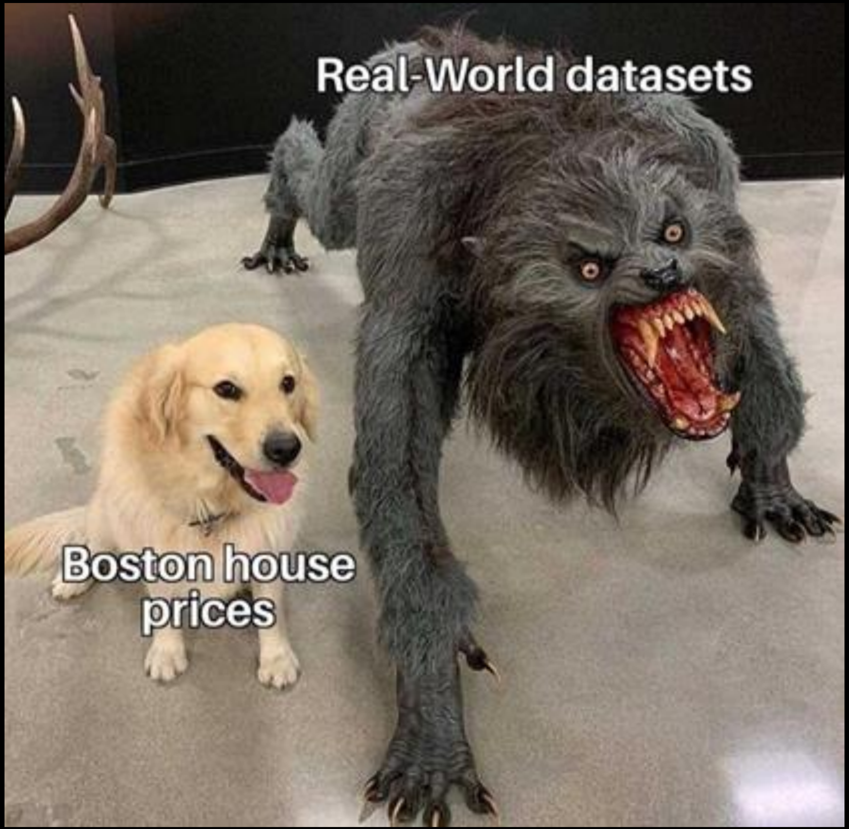
We selected K = 5, R = 30

03

The Dataset

Goals for Dataset

- Skewed class distribution
- Noise (bad data that slipped through preprocessing)
- Regular KNN can’t get above 75% accuracy



Synthetically-Generated Dataset

- 10 attributes
- 3 classes:
  - Majority class of 1105 instances, centered at (0,0...0) with a standard deviation of 3
  - Minority class of 409 instances, centered at (3,3...3) with a standard deviation of 7
  - Minority class of 386 instances, centered at (-3,-3...-3) with a standard deviation of 7

Started with blobs of 1000, 300, 300 instances, then added 300 instances of random noise (105 to majority class, 109 to first minority, 86 to second minority)

Train-Test Split, SMOTE

80% to train set, 20% to test set

1520 train instances
380 test instances

2652 train instances after SMOTE

04

The Results

(% accuracy)	Non-SMOTE	SMOTE
Regular KNN	73.42%	77.11%
Weighted Averaging	**76.58%**	77.11%
Weighted Distancing	73.68%	**77.63%**
Local Density Weighting	74.47%	77.37%
All Methods Combined	75.26%	76.58%

How about the minorities?

- Not great, but improved

(% accuracy)	Class 1	Class 2
Regular KNN	38.96%	56.10%
Weighted Averaging	53.25%	58.54%
Regular KNN (with SMOTE)	54.55%	64.63%
Weighted Distancing (SMOTE)	55.84%	64.63%
	(16.88% increase!)	(8.53% increase!)

Results, Explained

- Surprisingly, all the methods increased the accuracy!
- SMOTE is really good!
- After using SMOTE, the other methods aren't as useful as there's no class imbalance
  (for example, weighted averaging now does nothing at all)
- However, weighted distancing is still effective

Why is the accuracy still not 100%?

- These techniques aren’t perfect
- But…
  - There is random noise in the data
  - The clusters overlap so it’s impossible to tell some apart

- Given this, the improvement from ~73% to ~78% is pretty good!

05

Conclusion

To Conclude:

- KNNs do struggle with skewed and noisy data
- Different weighting methods can help
- SMOTE is very cool
- If you combine all the methods, it doesn't actually improve

Looking back…

- Probably put too much noise in the dataset
- Minority classes were 30%+ noise, so they really can’t get above 70% accuracy

Looking forward…

- With more computing resources:
  - More exhaustive search of K and R values
  - Multiple datasets
  - More advanced weighting/preprocessing methods

Any questions?