Supervised Learning with Nearest Neighbors

CS 759/859 Natural Language Processing Lecture 4

Samuel Carton, University of New Hampshire

Last lecture

Concepts

- Bag-of-words representations (i.e. unigrams, 1-grams) of text
- Preprocessing text
  - Lower-casing
  - Tokenization
  - Stemming
- Vectorizing text
- Similarity metrics
  - Jaccard similarity
  - Cosine distance
  - Others
- TF-IDF

Toolkits

- NLTK
  - For tokenization & stemming
- Numpy
  - For manipulating vectors
- Scikit-Learn
  - For vectorizing lists of texts automatically
  - Also includes implementations of similarity metrics (Jaccard, cosine, etc)

Supervised learning for classification

Classification: given input text x, classify x by predicting label y

- "You are an ass!" → toxic
- "The movie was great." → positive
- "SALE! SALE! SALE!" → spam
- "You are a mensch!" → nontoxic
- "The movie was awful." → negative
- "I'm breaking up with you."→ not spam

Label, i.e. "**class**"

Supervised learning: given a training set X_{train} with labels Y_{train}, learn how to predict y for an unseen input x

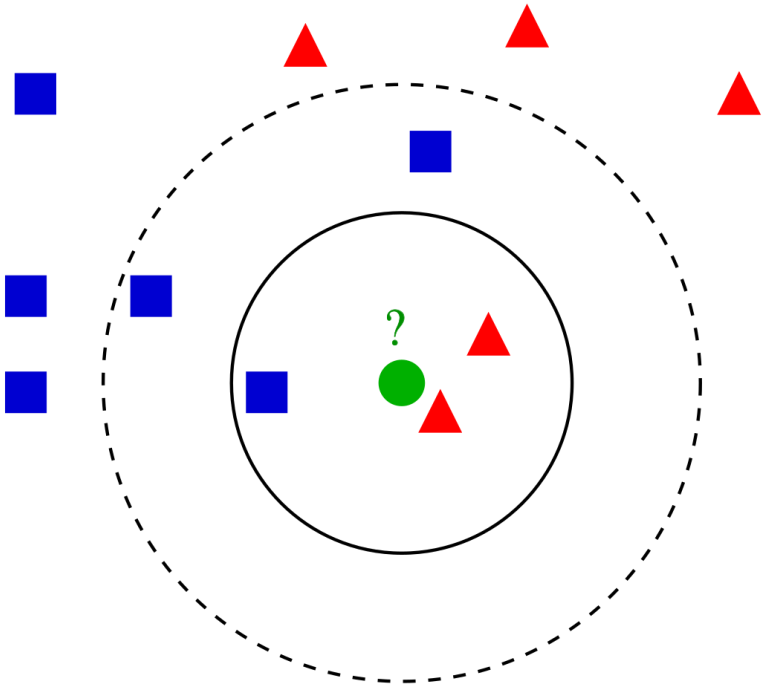
All we know how to do right now is text similarity. How to do supervised classification with just this tool?

K-nearest neighbors

Basic idea: when trying to classify x, find the K nearest neighbors of x within X_{train} and let $\hat{y}$ be the majority-vote true label y_{train} among those K neighbors

Why does it have to be K? Why not always K = 1?

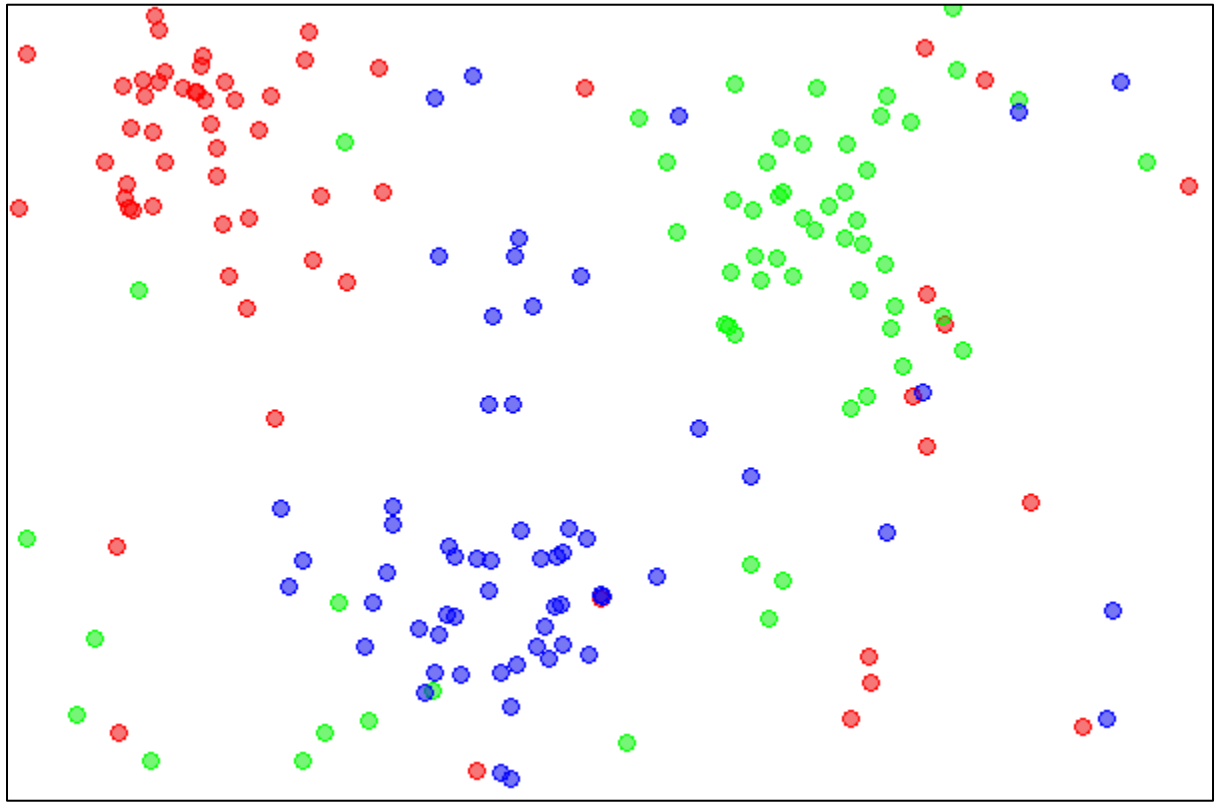
How would you implement this given what you already know?



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

K-nearest neighbors

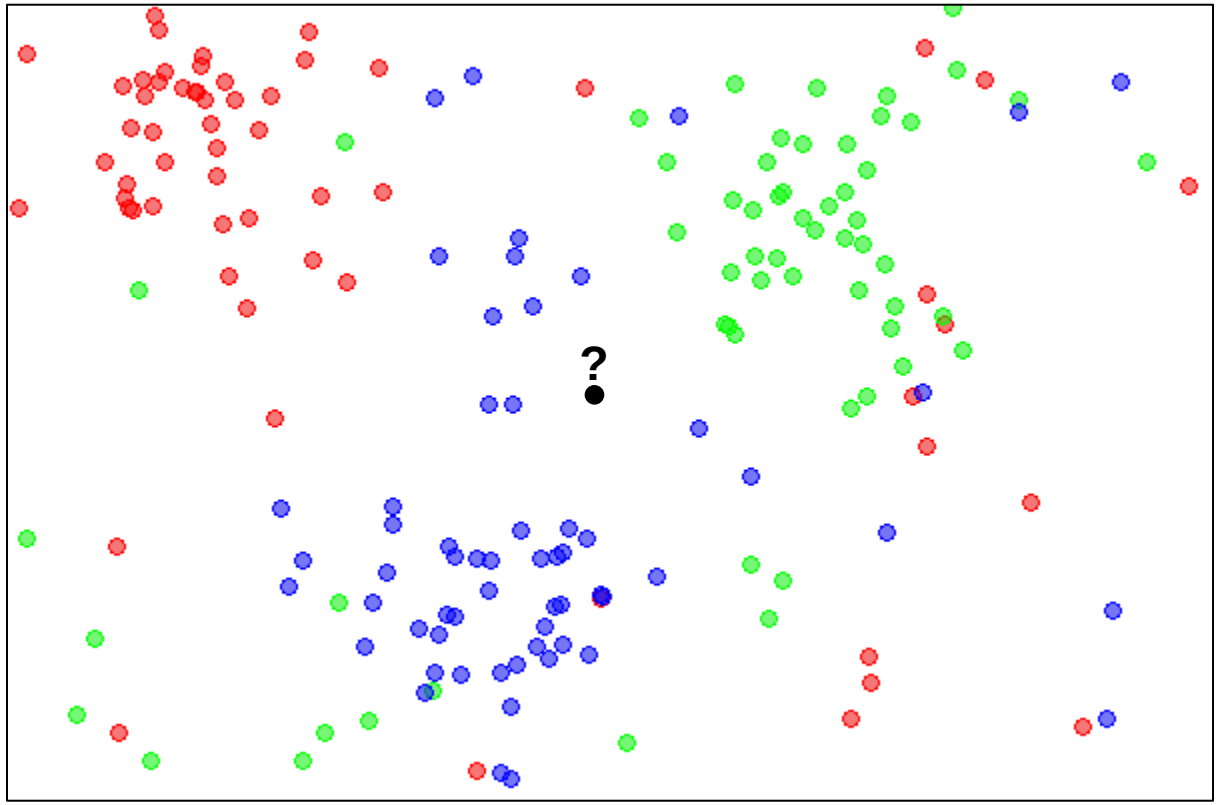
3-class dataset



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

K-nearest neighbors

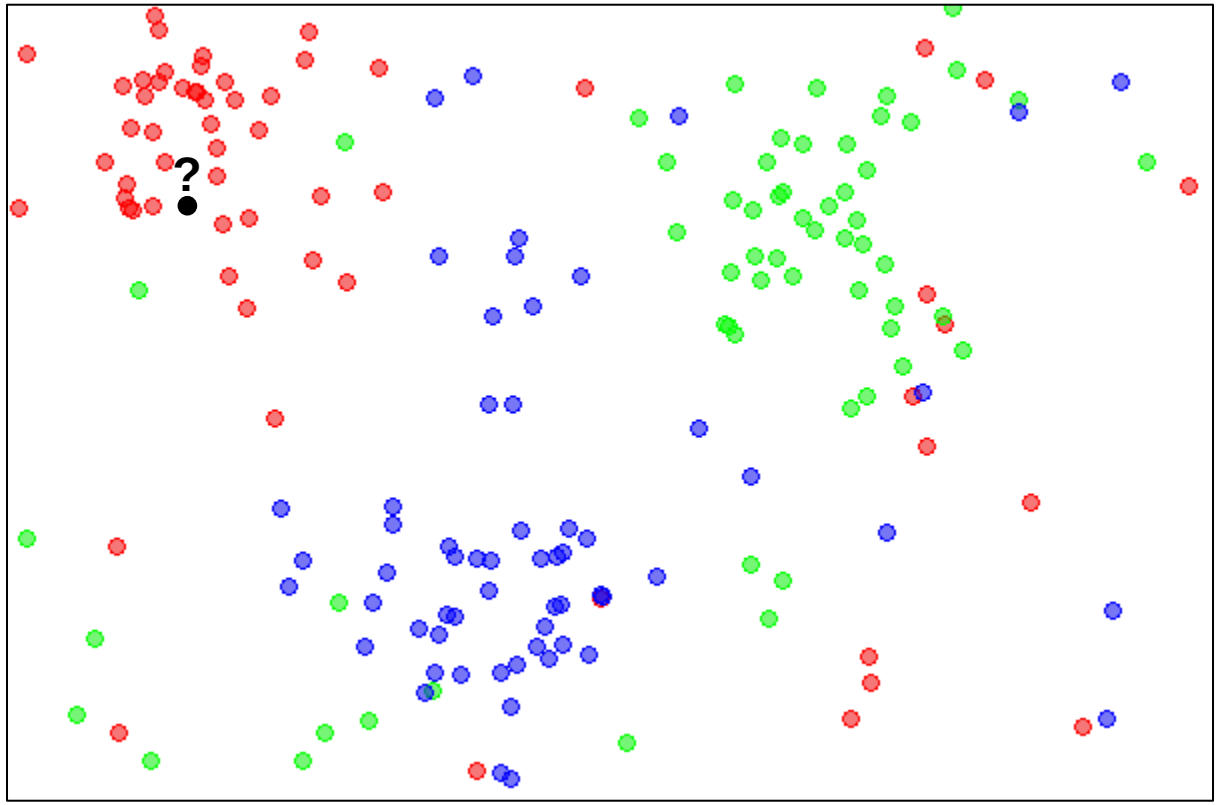
3-class dataset



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

K-nearest neighbors

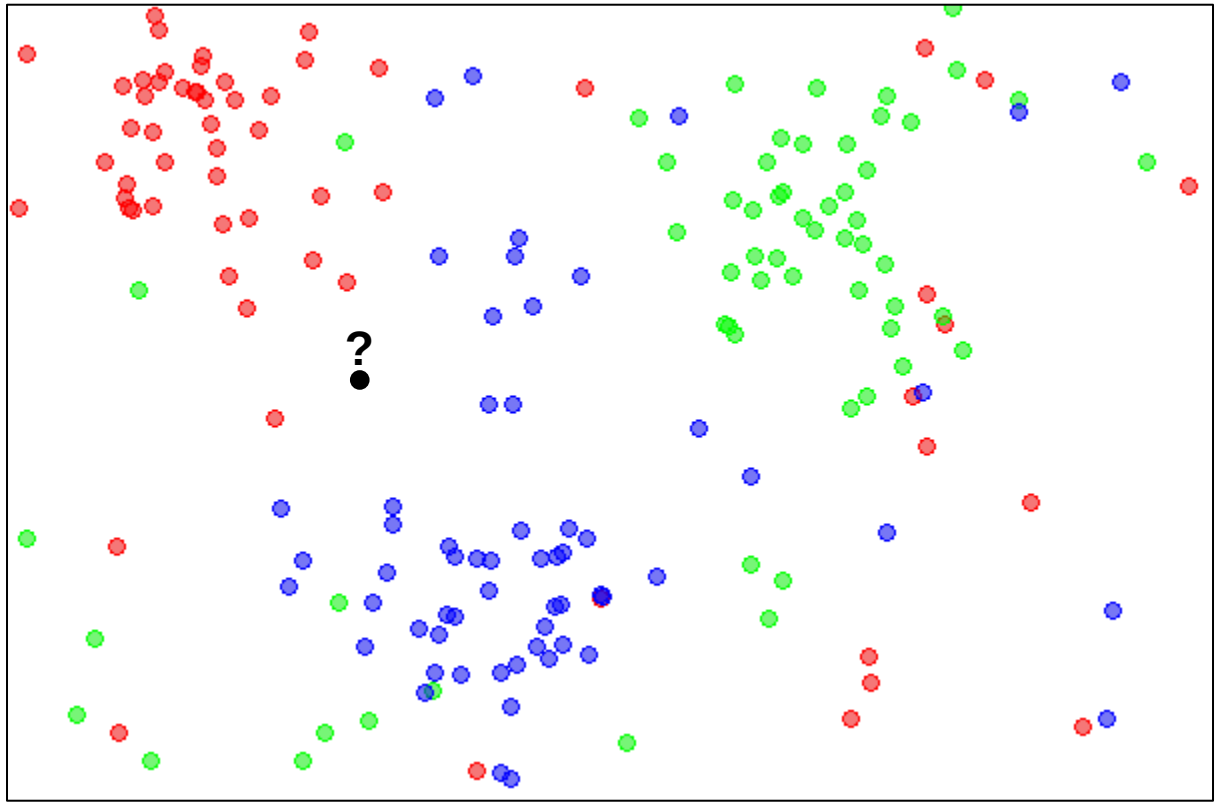
3-class dataset



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

K-nearest neighbors

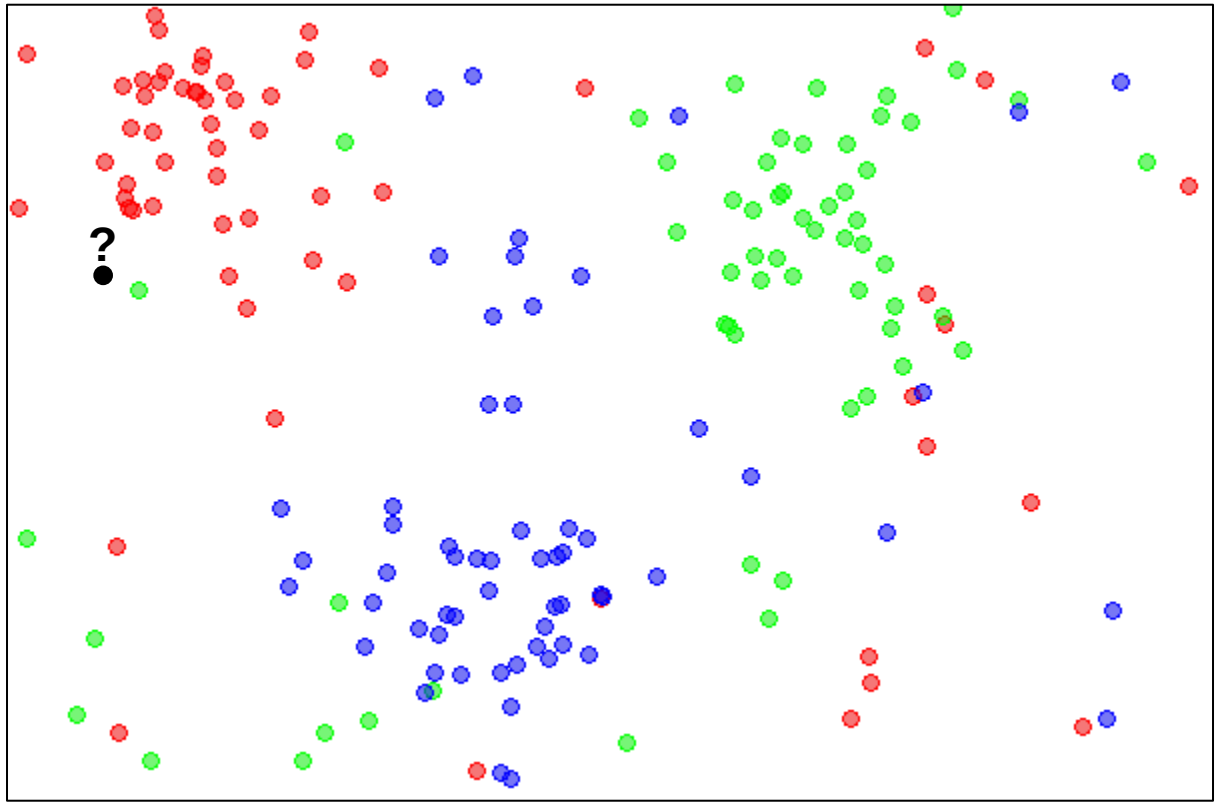
3-class dataset



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

K-nearest neighbors

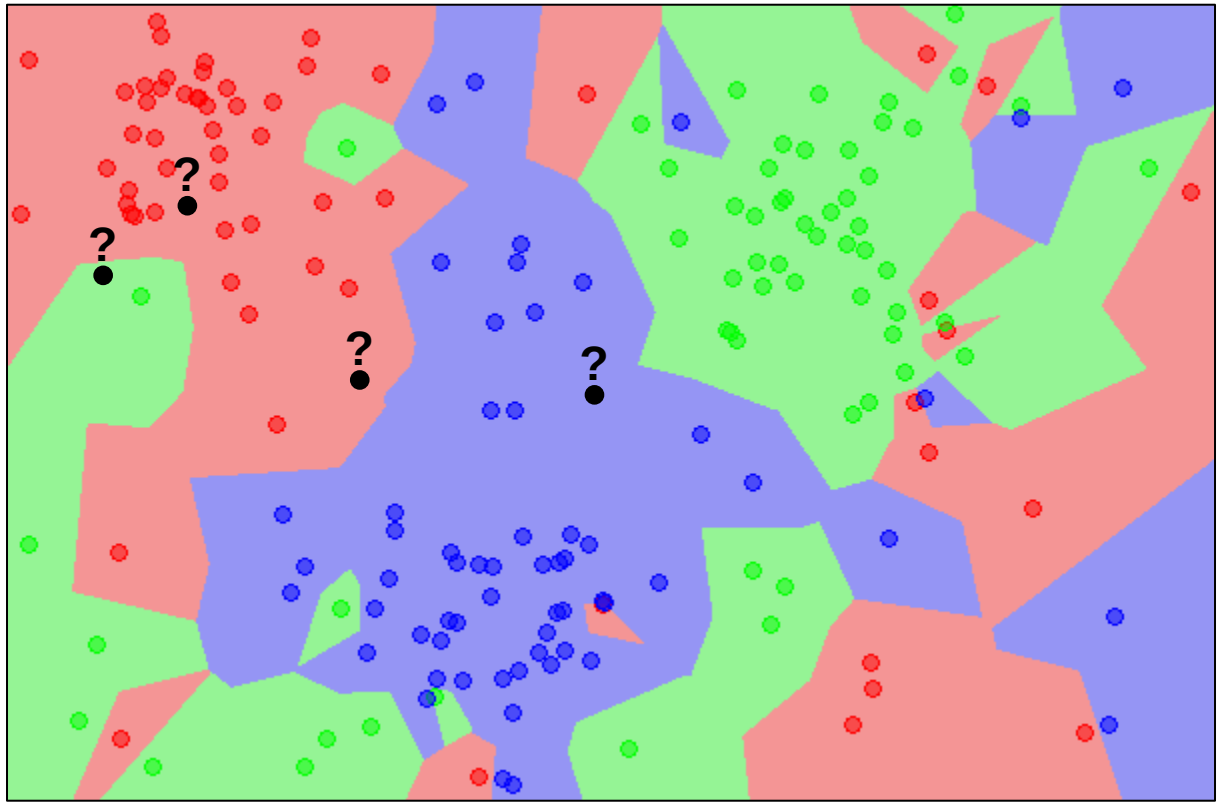
3-class dataset



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

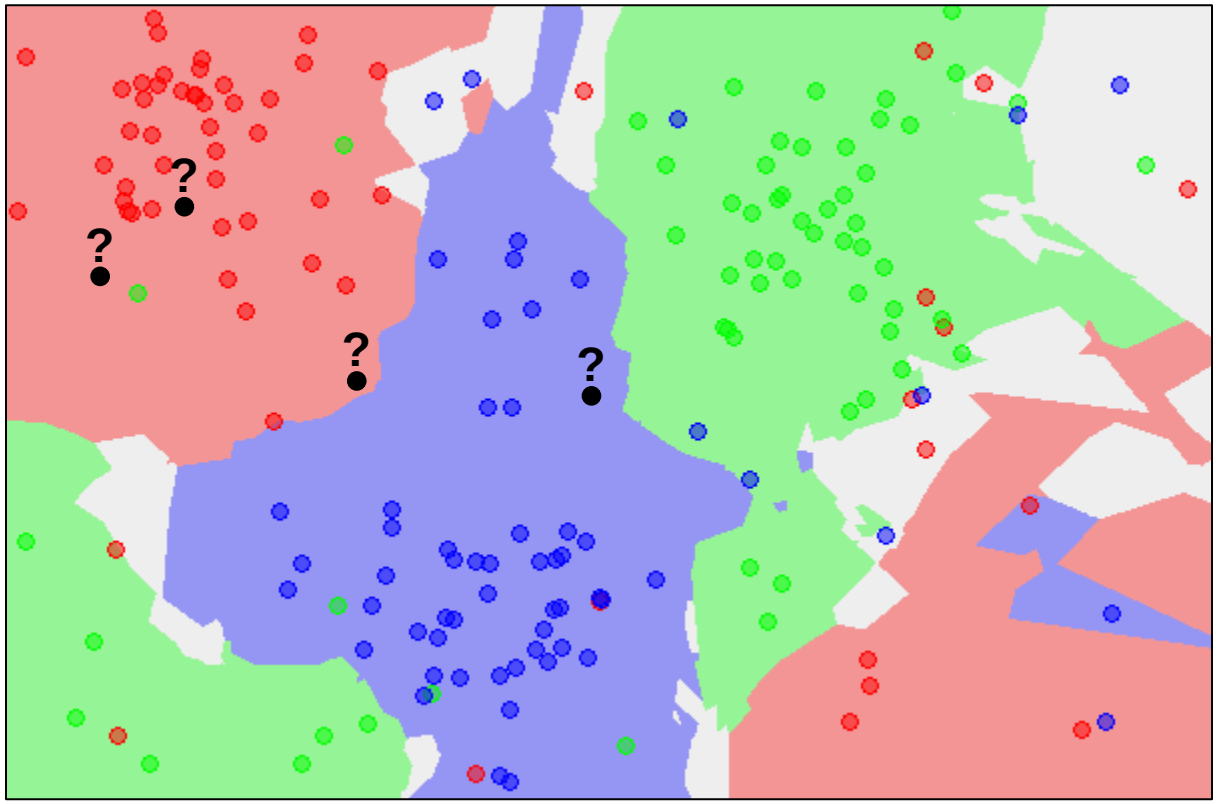
K-nearest neighbors

K=1



K-nearest neighbors

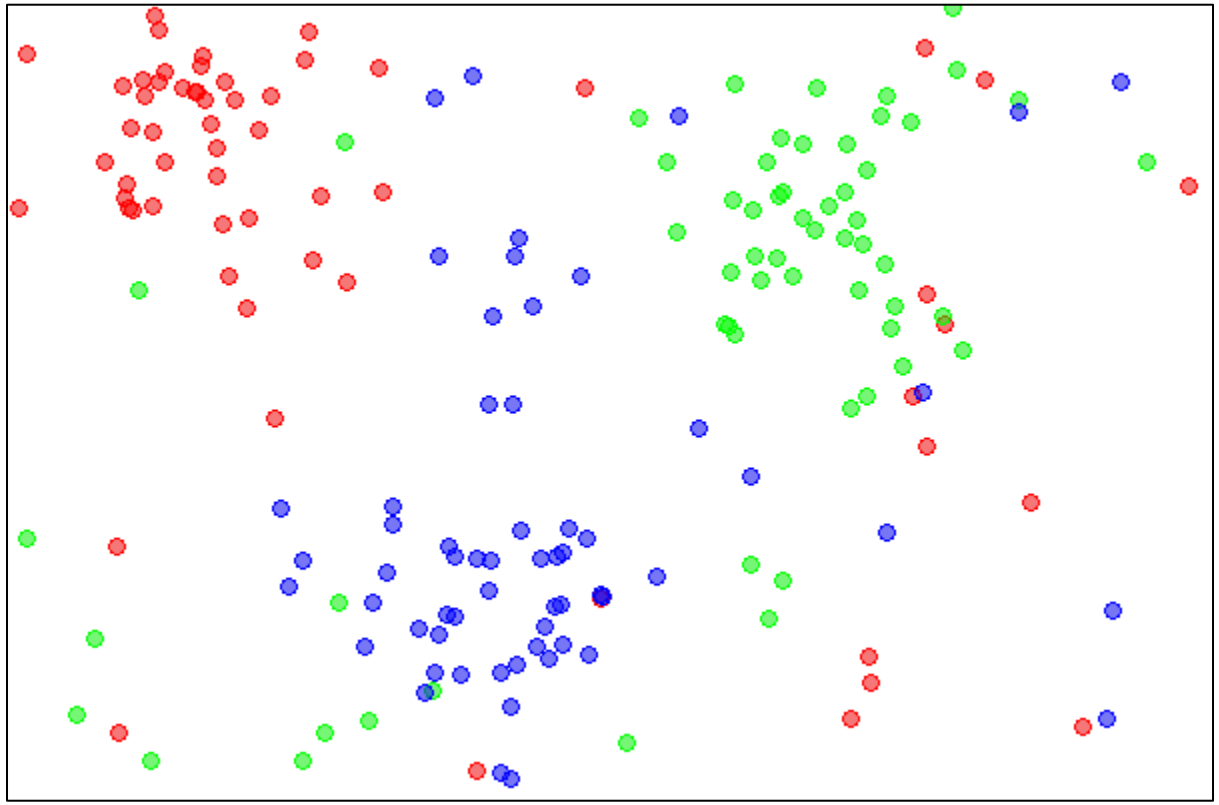
K=5



K-nearest neighbors

K=181?

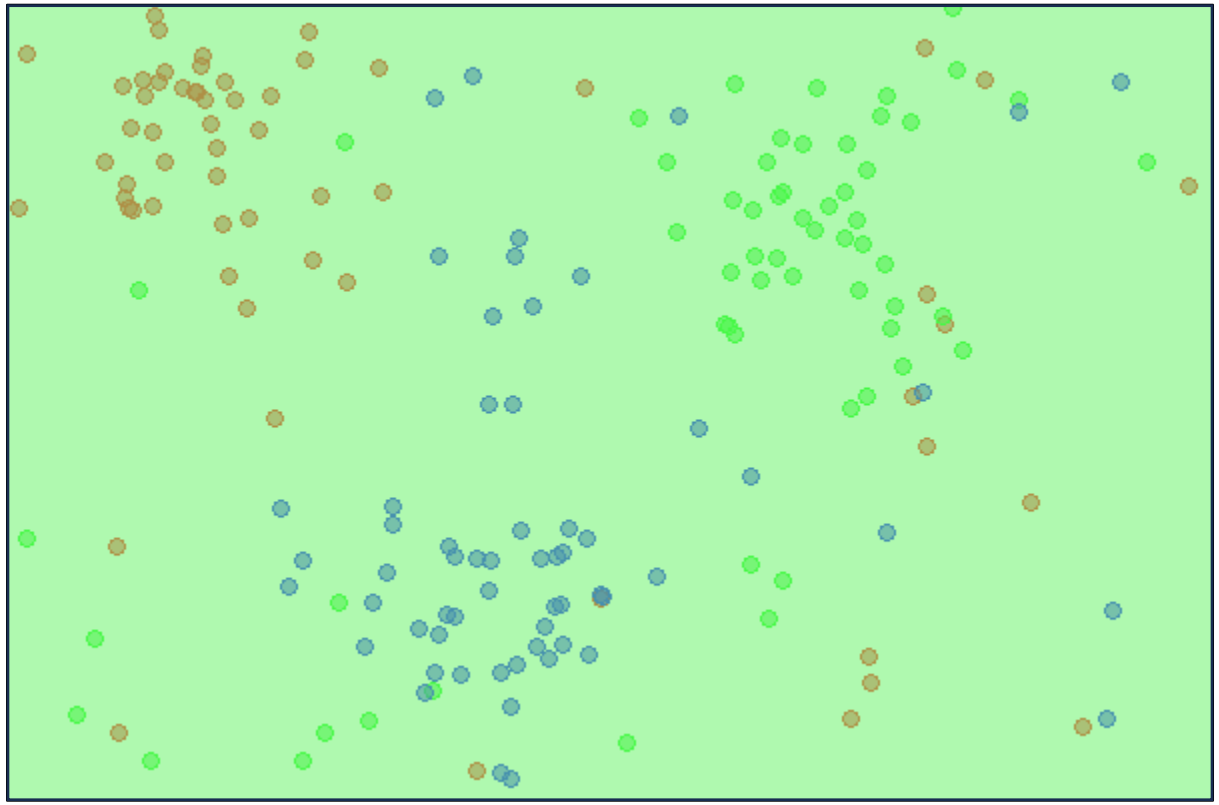
(If there were 60 red, 60 blue, and 61 green)



K-nearest neighbors

K=181?

(If there were 60 red, 60 blue, and 61 green)



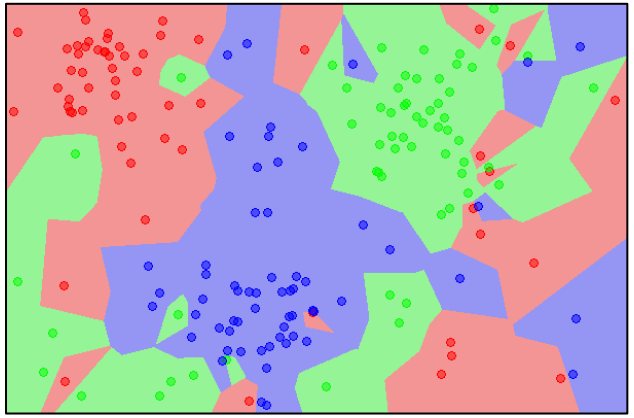
Underfitting vs. overfitting

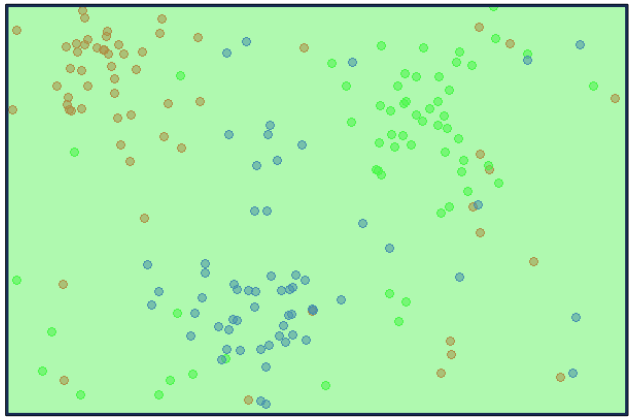
Underfitting: the model imposes too many assumptions and fails to learn the shape of the data

- K=181 is the most underfit you could possibly be

Overfitting: the model doesn't impose enough assumptions about the data and ends up overly sensitive to outliers

- K=1 is going to tend to overfit





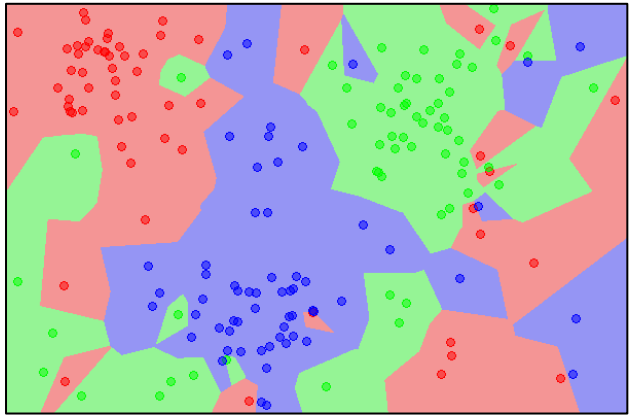
Bias-variance tradeoff

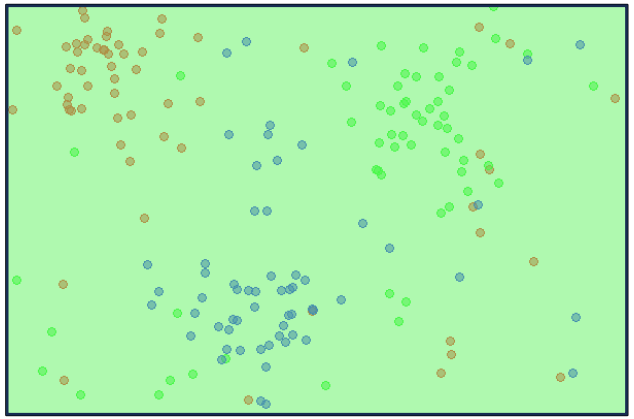
Bias: the tendency of the model to impose overly strong structural/statistical assumptions on the data and miss the real structure of the data

- Leads to underfitting

Variance: tendency of the model to accidentally capture random noise in the data and thus be vulnerable to outliers and random variation

- Leads to overfitting





Hyperparameters

For any model, bias and variance are balanced using **hyperparameters**

- How many neighbors to use
- What distance metric to use
- Whether to set binary=True or False in the vectorizer

A big part of model building is finding the best (or adequately okay) set of hyperparameters

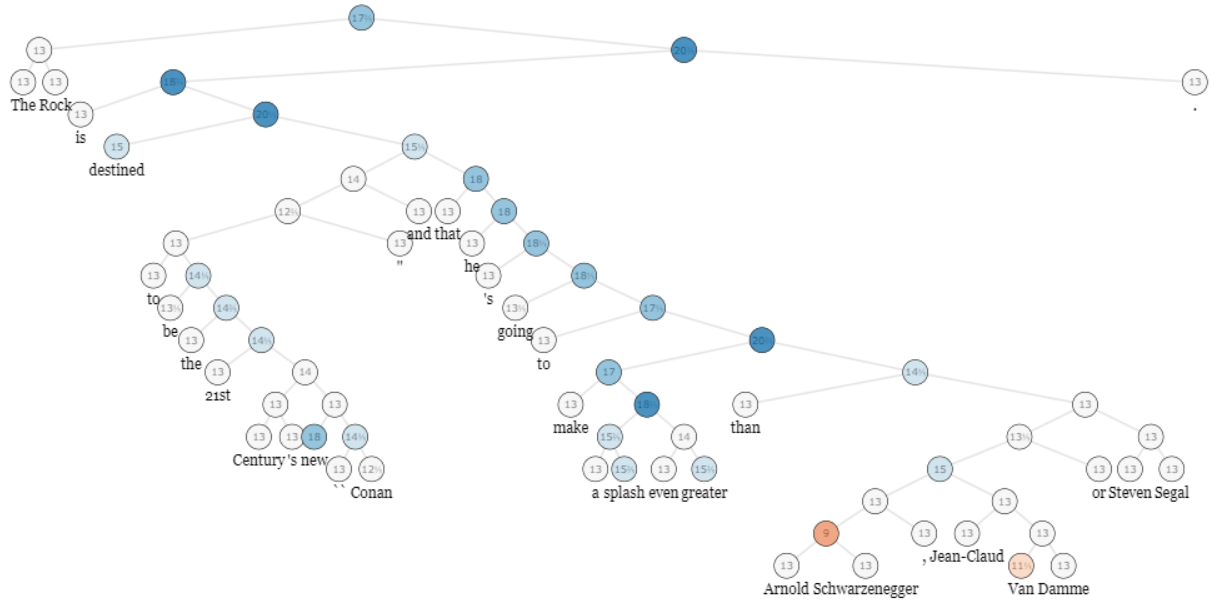
Simplest and most common approach is to just search exhaustively over space of possible values—called **grid search**

Contrast with **parameters**, which are learned from the data

- K-NN is a **non-parametric** model, but we'll get to parametric models soon

Case study: SST-2

- Stanford Sentiment Treebank (2-class version)
- Short movie reviews, tagged as positive or negative in sentiment
- Created by Socher et al. (2013)
- https://nlp.stanford.edu/sentiment/treebank.html
- Included as part of GLUE benchmark
  - https://gluebenchmark.com/tasks
- Size:
  - 67,349 training examples
  - 872 dev examples
  - 1821 test examples




```
the rock is destined to be the 21st century 's new `` conan '' and that he 's going to make a splash even greater than arnold schwarzenegger , jean-claud van damme or steven segal .
```

⬇

positive (1)

Pandas: read and manipulate data

Pandas is a useful Python library for reading and manipulating datasets of various kinds (text included)

https://pandas.pydata.org/

Largely consists of an implementation of “DataFrame” from the R statistical analysis language

- Swiss army knife data structure

Likely to be covered more thoroughly in a “data science” course.

Pandas, SST-2 and K-NN

Code description

- Basic Pandas usage,
- Using Pandas to import the SST-2 dataset into a DataFrame
- Vectorizing the imported data
- Training a K-NN model on it

Headings

Pandas: a useful way to manipulate data

Pandas basics

.apply() method

DataFrame filtering

Reading the dataset

Preprocessing the text

Nearest-neighbors classification

Vectorizing the text

Training the model

Evaluating classifiers

Given a set of predictions $\hat{Y}$ and the true labels Y, there are a few different ways to evaluate how well we did.

One way to divide up predictions is into errors ($\hat{y} \neq y$) and non-errors ($\hat{y} = y$)

In a binary classification setting (like SST-2), we can also think about different kinds of errors and non-errors:

- True positives (TPs): $\hat{y} = 1;\ y = 1$
- True negatives (TNs): $\hat{y} = 0;\ y = 0$
- False positives (FPs): $\hat{y} = 1;\ y = 0$
- False negatives (FNs): $\hat{y} = 0;\ y = 1$

https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers

Things get more complicated with 3+ classes, but don't worry about it for now

Accuracy

What percentage of my guesses were correct?

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} = \frac{COR}{COR + ERR}$$

Problematic when the true labels are highly **unbalanced** (e.g. 90% positive, 10% negative)

- 91% accuracy looks good by itself, but not so great if you could get 90% by just guessing the most common class.

Recall

Of all the positives examples, what percentage of them did I correctly guess were positive?

AKA sensitivity, true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

Particularly important when we *really* don't want to miss any positives

- I.e. we want to avoid false negatives
- What are tasks for which this is this the case?

Precision

When I guessed positive, how likely was I to be correct?

AKA positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

Particularly important when we really don't want to falsely predict any example as positive

- What are tasks where this is the case?

F1 score

Defined as the harmonic mean of precision and recall

Balances precision and recall.

$$\mathrm{F}_1 = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Does a better job of handling unbalanced data

- Although you still probably want to calculate F1 for both possible definitions of "positive", then take the mean of *that* value

Classification metrics—summary

Accuracy: most common metric, has the most intuitive interpretation, can be misleading on unbalanced data

F1: Less common, but gives a better performance estimation for unbalanced data

Precision: Used when you care about false positives

Recall: Used when you care about false negatives

Other notes:

- All but accuracy require definition of which class represents “positive”
- So common together that often abbreviated as A/P/R/F1

Evaluating and debugging the model

Code description

- Evaluating the model performance
- Inspecting its predictions on an individual example

Headings

Evaluating the predictions

Understanding individual predictions

Debugging the model

Finding what we think should be the nearest neighbor(s)

Understanding why it isn't

Understanding "was" versus "delight"

Examining the two distances

The problem

3 main things going on here:

1. It turns out that “was” is less common in the corpus (only 608 instances) than we might expect compared to delight (325 instances)
2. “was” occurs twice in "the film was a delight -- i was riveted .”, so it gets a higher tf-idf weight for that vector
3. Because cosine distance is normalized by vector magnitude, the tf-idf values in shorter texts get a higher value than the same ones in longer texts

We can’t do anything about 1 without changing the corpus, or about 3 without using a different distance metric

But what about 2?

Training a new model

Code description

- Training a new model to address the issues we just identified

Headings

Training a new model

Why did I go through that with you?

1. I had to deal with it when I was writing the code, so now you get to deal with it too.

2. These kinds of issues come up all the time. Model debugging is part of the life of an NLP or data science practitioner.

Model confidence

Sometimes you want not just a prediction, but a **confidence estimate** of how certain the classifier is in its prediction.

What are some cases where you might want this?

How to calculate confidence varies from model to model, and doing it robustly is a whole research topic in and of itself.

For K-nearest-neighbors, you can just look at the votes of the K neighbors.

Model confidence & hyperparameters

Code description

- Showing how to assess model confidence for K-NN
- Showing how to set hyperparameters for K-NN

Headings

Model confidence

Hyperparameters

Other things to know

How to use each set:

- Train on the training set
- Experiment on the dev set
- Leave the test set alone until the very end (notice we didn't even use it)

When dealing with temporal data (which SST-2 is not, really)

- Never, ever, train on future data and test on past data
- Super common mistake in the wild

Concluding thoughts

Pretty cool that we can already build models with what little we've learned so far. Non-parametric models so far, but we're getting there.

When doing nearest-neighbor classification (and classification generally for 1 and 2):

1. How you choose to vectorize your text matters a lot
2. The distance metric you use matters a lot
3. Sometimes more sensible individual predictions don't translate to better performance

Conclusion

New toolkit: Pandas

Concepts:

- Supervised learning for classification
- K-nearest neighbors model
- Underfitting/overfitting
- Bias-variance trade-off
- Hyperparameters
- Evaluation metrics
- Model confidence