Text Analysis and Classification with NLTK and scikit-learn

Rachel Rakov

Greetings! What are we doing here?

- The goals of this workshop are as follows:
- Introduce text analysis with the Natural Language toolkit (NLTK)
- Use the ability to analyze text to build features to scaffold up to text classification using scikit-learn for machine learning
- Build a text classification system that can predict whether words belong in one category of text or another

What do you need for this workshop?

- Python 3
  - Download jupyter notebook file for this session
- The Natural Language Toolkit (NLTK)
- Scikit-learn (comes with Anaconda – can also be installed independently)
- Pandas (comes with Anaconda – can also be installed independently)

Text Analysis with NLTK

- What does this mean, exactly?
  - Let's break it down!
- There is a tremendous amount of textual data
  - Books, news articles, medical documentation, tweets, and everything in between
- More and more, this textual information is available online
  - In downloadable / scrapable / otherwise accessible formats
  - And more content is being created all the time (thank you, Internet!)
- Researchers are able to use tools to learn about textual data on a large scale

What tasks involve text analysis?

- Sentiment Analysis
- Information Extraction
- Automatic Summarization
- Authorship Identification
- Search
- Trend tracking
- To name simply a few!!!!

The Natural Language Toolkit

- A package for Python that provides users with tools for text analysis
- Premade tools and corpora
  - So you don't have to make them yourself!!
- Introduction to these tools, and see how they're used
- Let's get started!!

```
import nltk
import matplotlib
from nltk.book import *
```

Text Analysis: start with looking at what you've got

- Common contexts
  - Takes two words as an argument, returns contexts in which they appear similarly across the text (within one text)
- Word locations across text
  - Create a dispersion plot to see where particular words occur in your text
    - Takes a list of words as an argument

Basic analysis: Counting things

- Easy to do, and extremely helpful!
  - Total number of words in your corpus, specific words in your corpus
  - Percentage of word occurrence in your text
- Frequency Distributions
  - Allows us to find the frequency of each vocabulary token in our text
  - Built in method

Frequency Distributions methods

A happy chart from "Natural Language Processing with Python" (Bird, Klein & Loper)

# Example	**#Description**
fdist = FreqDist(samples)	create a frequency distribution containing the given samples
fdist[sample] += 1	increment the count for this sample
fdist['monstrous']	count of the number of times a given sample occurred
fdist.freq('monstrous')	frequency of a given sample
fdist.N()	total number of samples
fdist.most_common(n)	the n most common samples and their frequencies
for sample in fdist:	iterate over the samples
fdist.max()	sample with the greatest count
fdist.tabulate()	tabulate the frequency distribution
fdist.plot()	graphical plot of the frequency distribution
fdist.plot(cumulative=True)	cumulative plot of the frequency distribution
fdist1 |= fdist2	update fdist1 with counts from fdist2
fdist1 < fdist2	test if samples in fdist1 occur less frequently than in

Why do I keep getting punctuation as a word?

- NLTK makes use of *tokenization*, a method of separating every word from it's punctuation, making each their own separate tokens.
  - Punctuation often causes problems
- Word_tokenize returns tokenized text

Comparing texts with frequency distributions

- NLTK includes some of the Brown University Standard Corpus of Present-Day American English (Brown Corpus for short)
  - Contains one sample from each of the 15 categories

- What if we wanted to compare usage of modal verbs in one text category to the modal verbs in another text category?
  - A **modal verb** is a word like *can, could, will, must, might,* or *should.* Modal verbs are often used to distinguish between different registers of speech.

Analysis is fun! Let's tag parts of speech

- One way you might want to analyze your text is by looking at what parts of speech are contained within the text
- NLTK has built-in the UPenn part of speech tagset, which allows for relatively accurate part of speech tagging.
- Since some of the tags used aren't easily intuitive, use this piece of code to clarify them!

```
nltk.help.upenn_tagset("<tag>")
```

Representing your text as features

Text Analysis: Getting the most out of your data

- We've taken a look at some of the way we can analyze some of the basics of texts…but what can we do with this information?

- What if we wanted to compare sentences in the News and Romance categories to see how different they are?

Comparing News and Romance sentences

- For every sentence in the two different categories of the Brown corpus (news and romance), count how many of the words are nouns

  - A **noun**, in its most basic definition, is usually defined as a person, place, or thing. For example, a *movie*, a *book*, and a *burger* are all nouns. Counting nouns can help determine how many different topics are being discussed.[1]

1. Michelle Morales, 2017

Comparing News and Romance sentences

- This is a *feature representation* of the two categories.
  - It gives us a way to differentiate the categories by giving us an approximation of how many different topics are discussed within sentences in each category.

- The feature representation we are using is: How many nouns are in each sentence in the corpus?

What do we find from this?

```
print romance_counts[:20]
print news_counts[:20]
```

```
[2, 11, 2, 4, 8, 8, 3, 1, 3, 3, 4, 1, 4, 6, 3, 1, 3, 0, 5, 4]
[11, 13, 16, 9, 5, 5, 11, 1, 5, 9, 3, 6, 5, 11, 20, 5, 5, 5, 12, 1]
```

- The news corpus seems to have more nouns per sentence than the romance corpus (in a way that looks pretty obvious)
- One drawback: we are only looking at the first 20 sentences
  - Out of 4,623 sentences in the news corpus, and 4,431 in the romance corpus

Is this a good feature representation?

- Looks promising..
  - But since there are so many sentences, we can't really say this for sure just by looking at the numbers

- How can we test this theory on a larger scale?
  - We can build a machine learning model that predicts whether a sentence belongs in the news category or the romance category, based on how many nouns it has

Getting to meaningful feature representations

- We need to translate our language problem into something that a computer can understand.

- One way we can do this is by representing language as a set of features.

Example: Fruit

- How would you describe apples to a computer? How would they differ from oranges?
  - One way:

Object	Height	Width	Color	Mass	Round?
Apple	6cm	7cm	Red	330g	TRUE
Orange	6cm	7cm	Orange	330g	TRUE
Lemon	5cm	4cm	Yellow	150g	FALSE

What did we do?

```
print romance_counts[:20]
print news_counts[:20]
```

```
[2, 11, 2, 4, 8, 8, 3, 1, 3, 3, 4, 1, 4, 6, 3, 1, 3, 0, 5, 4]
[11, 13, 16, 9, 5, 5, 11, 1, 5, 9, 3, 6, 5, 11, 20, 5, 5, 5, 12, 1]
```

- We counted the number of nouns in each sentence, creating a *feature vector* (of a single feature per sentence)
  - A *feature vector* is a set of features that represent your text in such a way that they are machine-understandable
- We will use this feature vector as input for machine learning.

Feature representation for machine learning

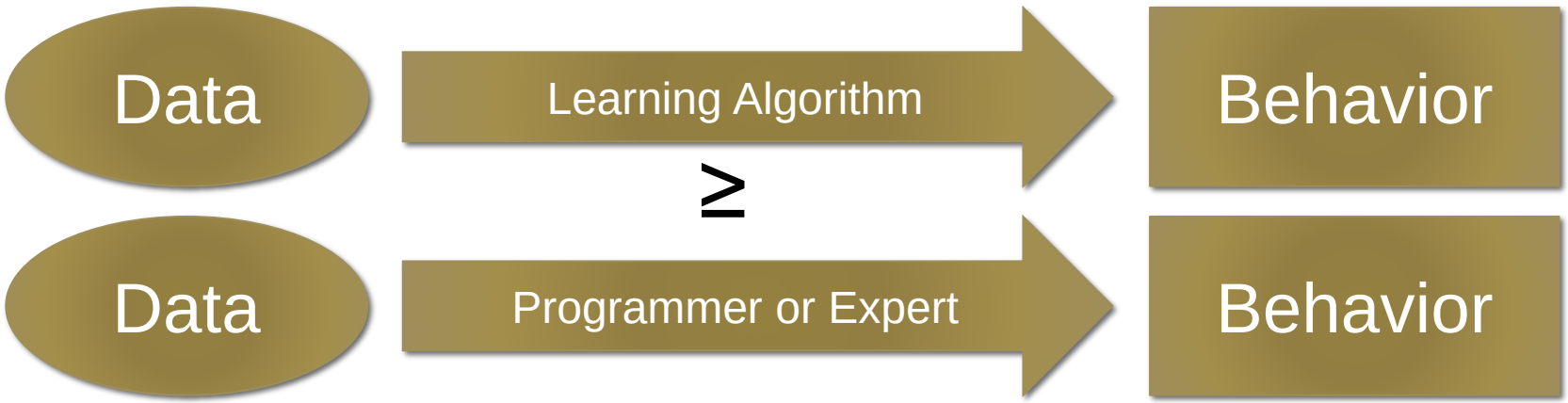
- Our approach: Use number of nouns in a sentence to predict whether the sentence belongs in the news or romance category
  - If the system performs well, we'll know this is a good feature representation
  - If the system performs poorly, we'll know this is not a good feature representation

Machine Learning: An Introduction to Classification

What is Machine Learning?

- Automatically identifying patterns in data
- Automatically making decisions based on these patterns

- Hypothesis:



Example: Fruit classification

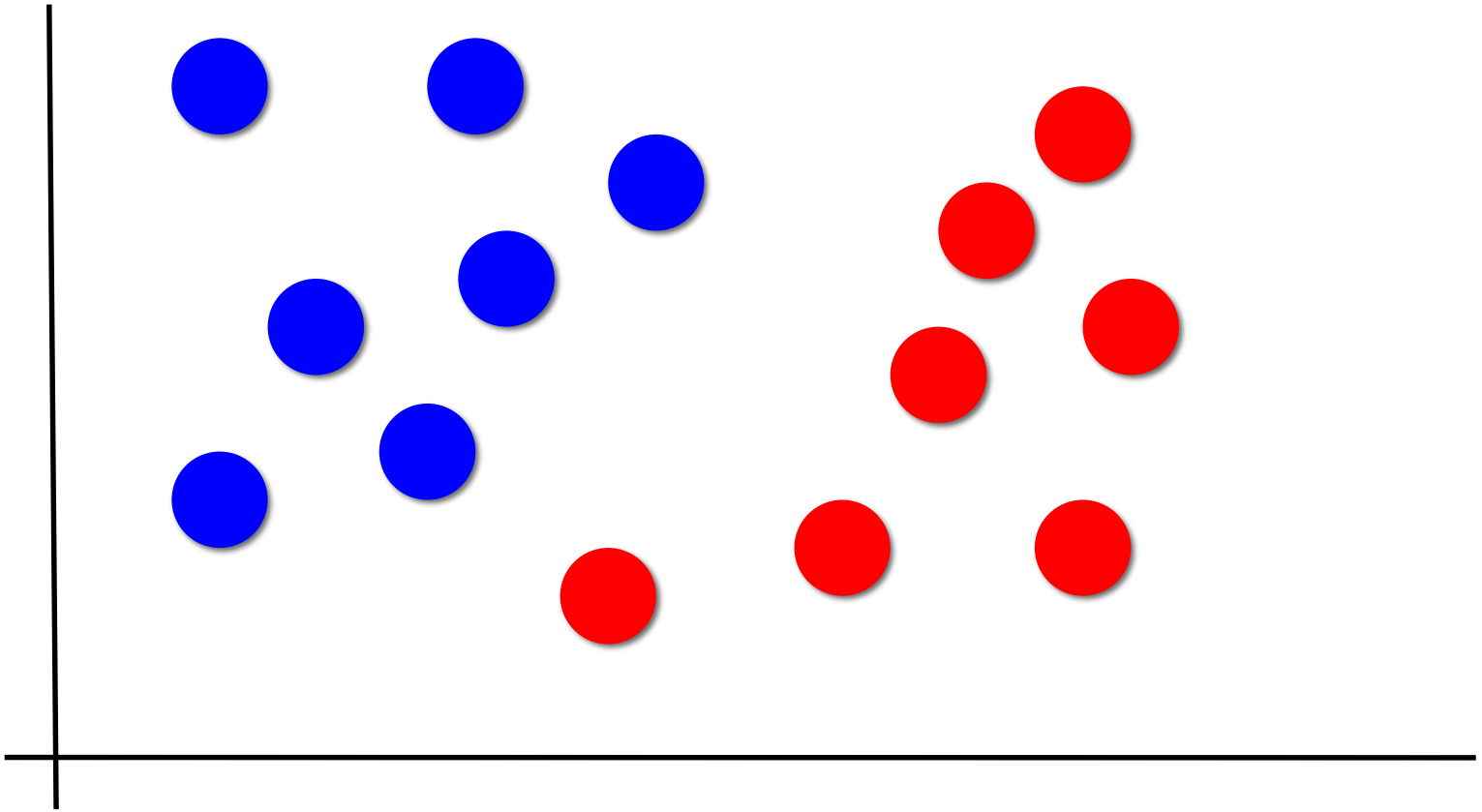
- What if we have a new, unknown fruit? How would you classify it?

Object	Height	Width	Color	Mass	Round?
Apple	6cm	7cm	Red	330g	TRUE
Orange	6cm	7cm	Orange	330g	TRUE
Lemon	5cm	4cm	Yellow	150g	FALSE
???	5cm	6cm	Orange	300g	TRUE

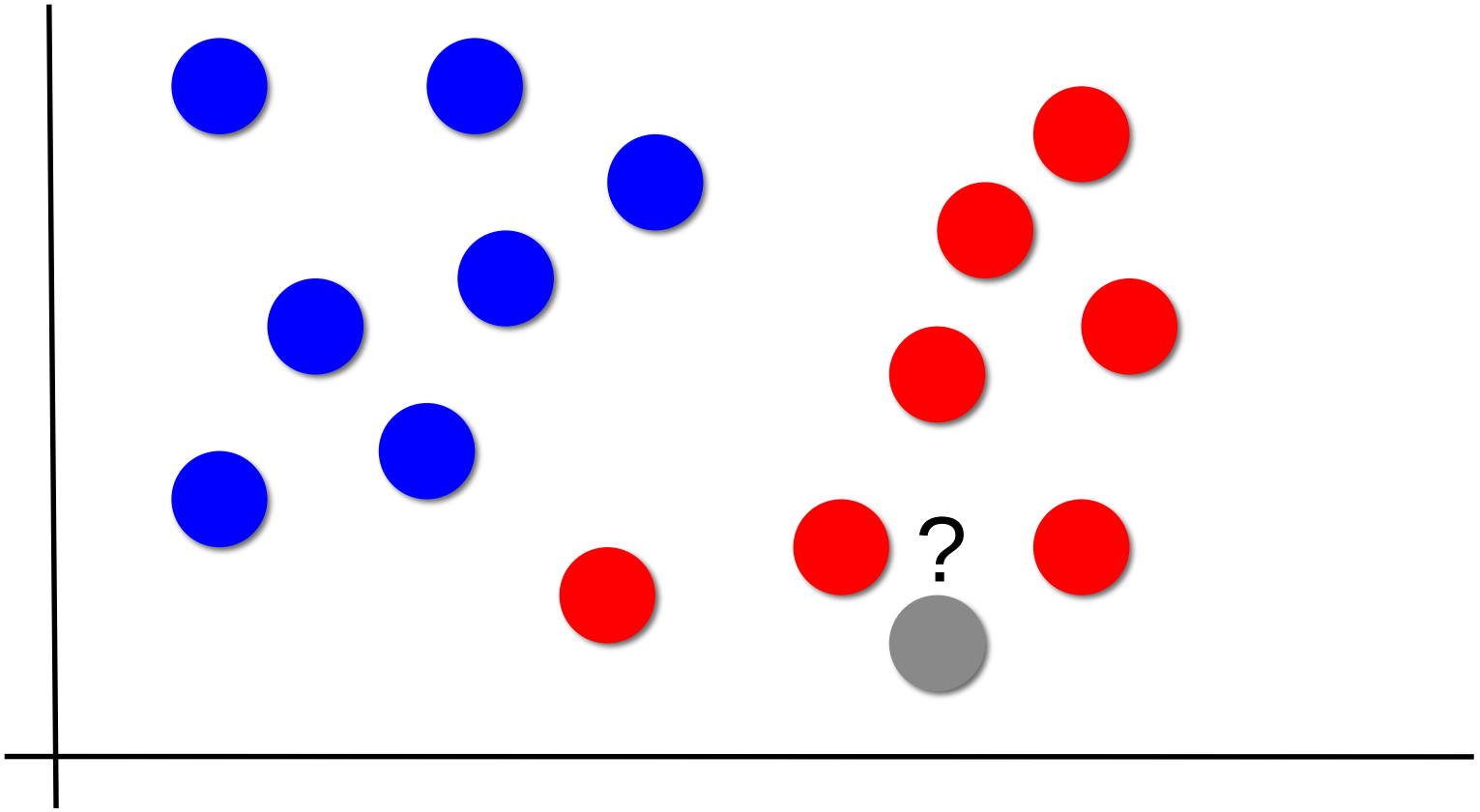
Machine Learning tasks

- The fruit example is an example of a *classification* task
  - We predict a **categorical** value
- Another kind of machine learning task is a *regression* task
  - This task predicts a **continuous** value
- Classification and regression are both examples of *supervised machine learning*
- Another type of machine learning task is a *clustering* task
  - This is an *unsupervised machine learning* task
  - Used to **identify groups** of similar entities
  - Groups unlabeled data points by proximity

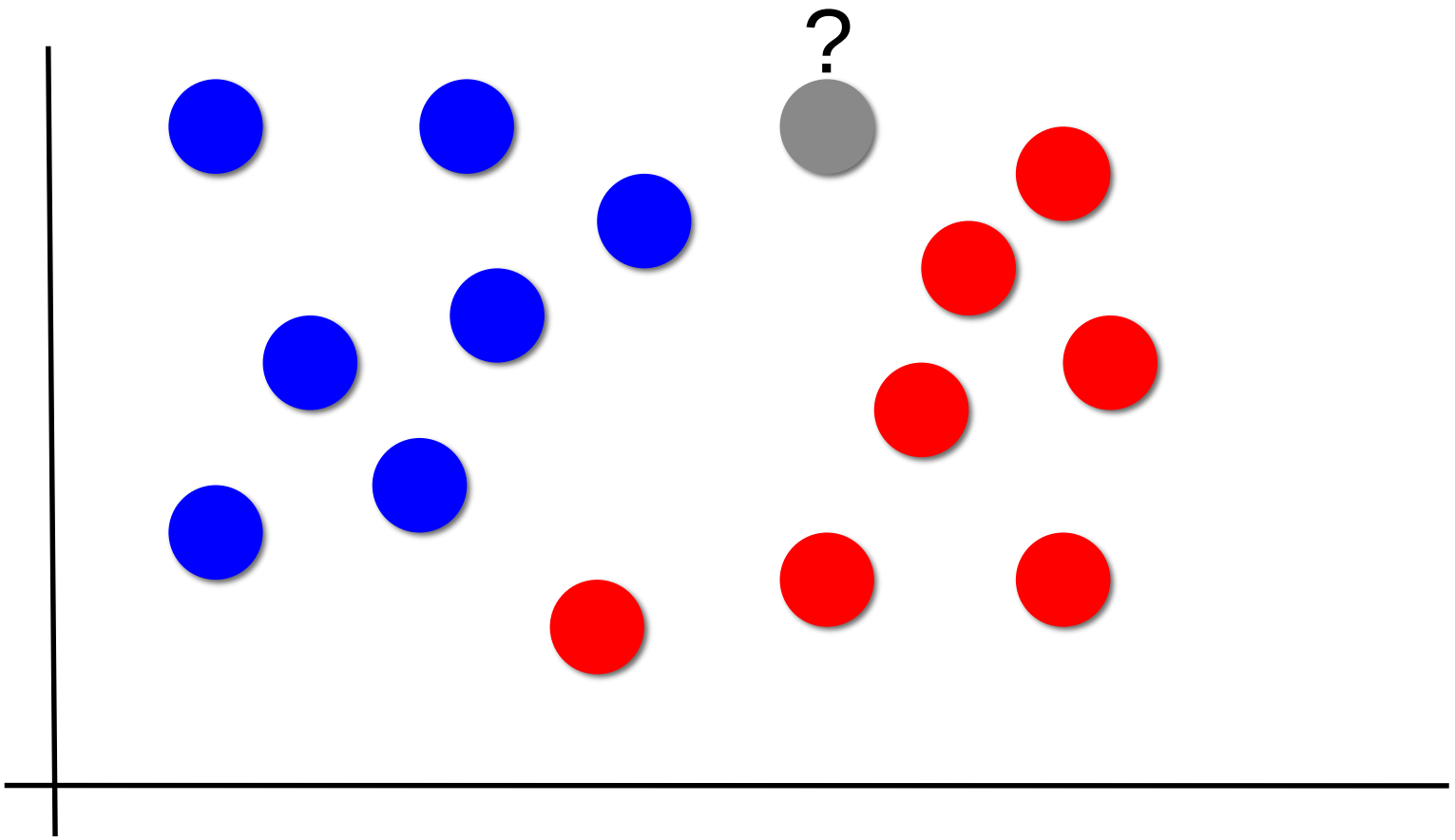
Graphical Example of Classification



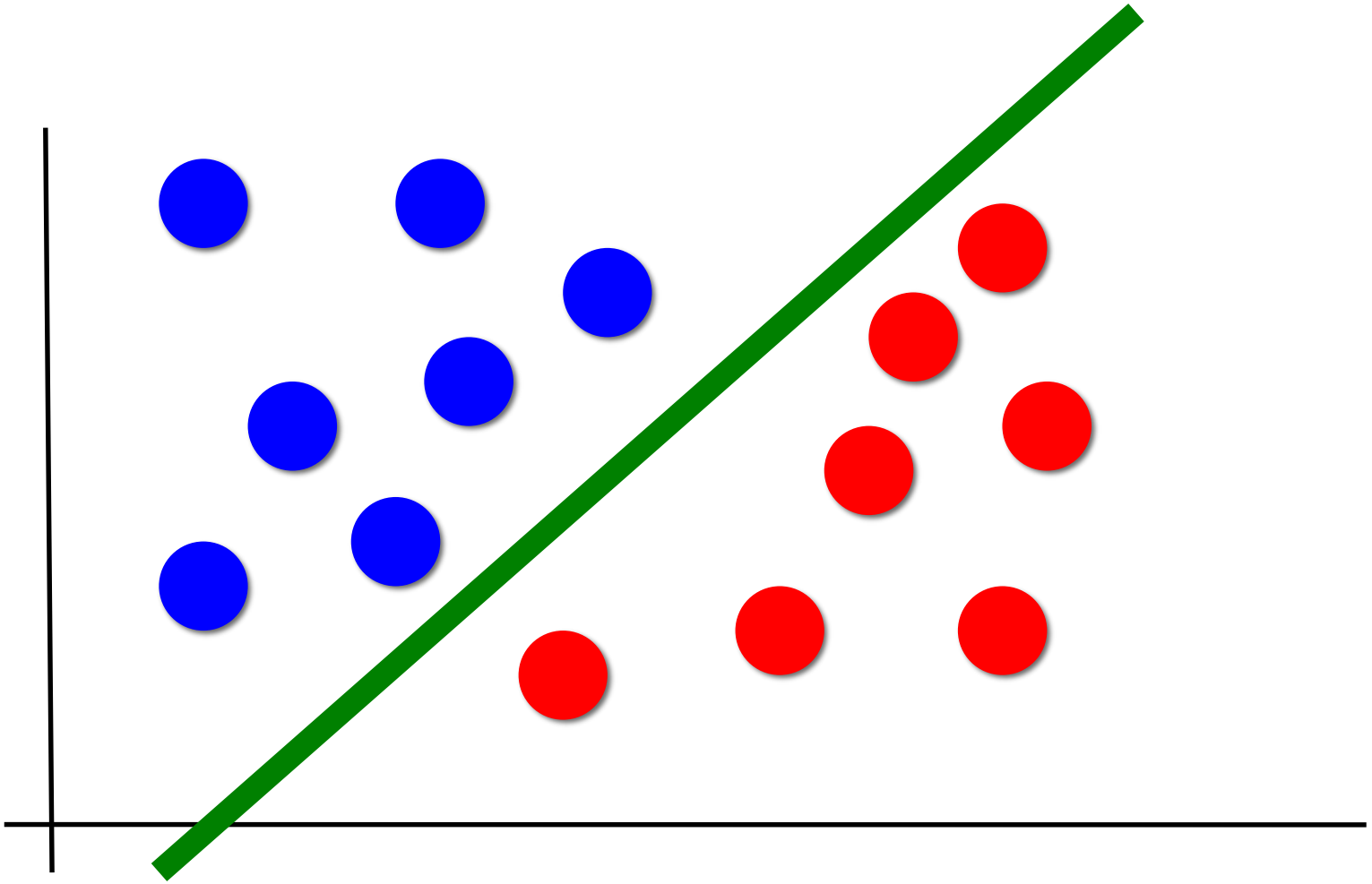
Graphical Example of Classification



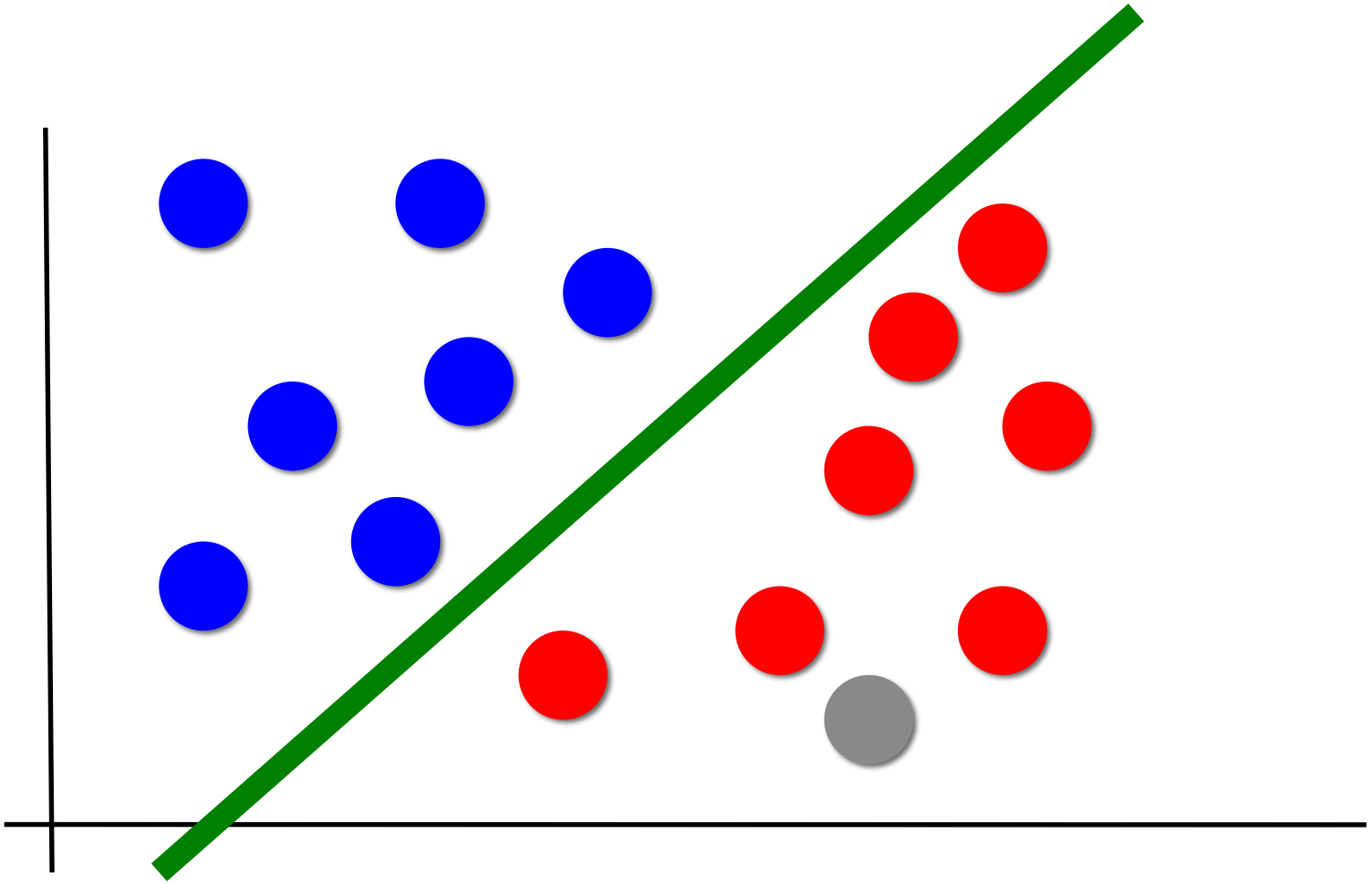
Graphical Example of Classification



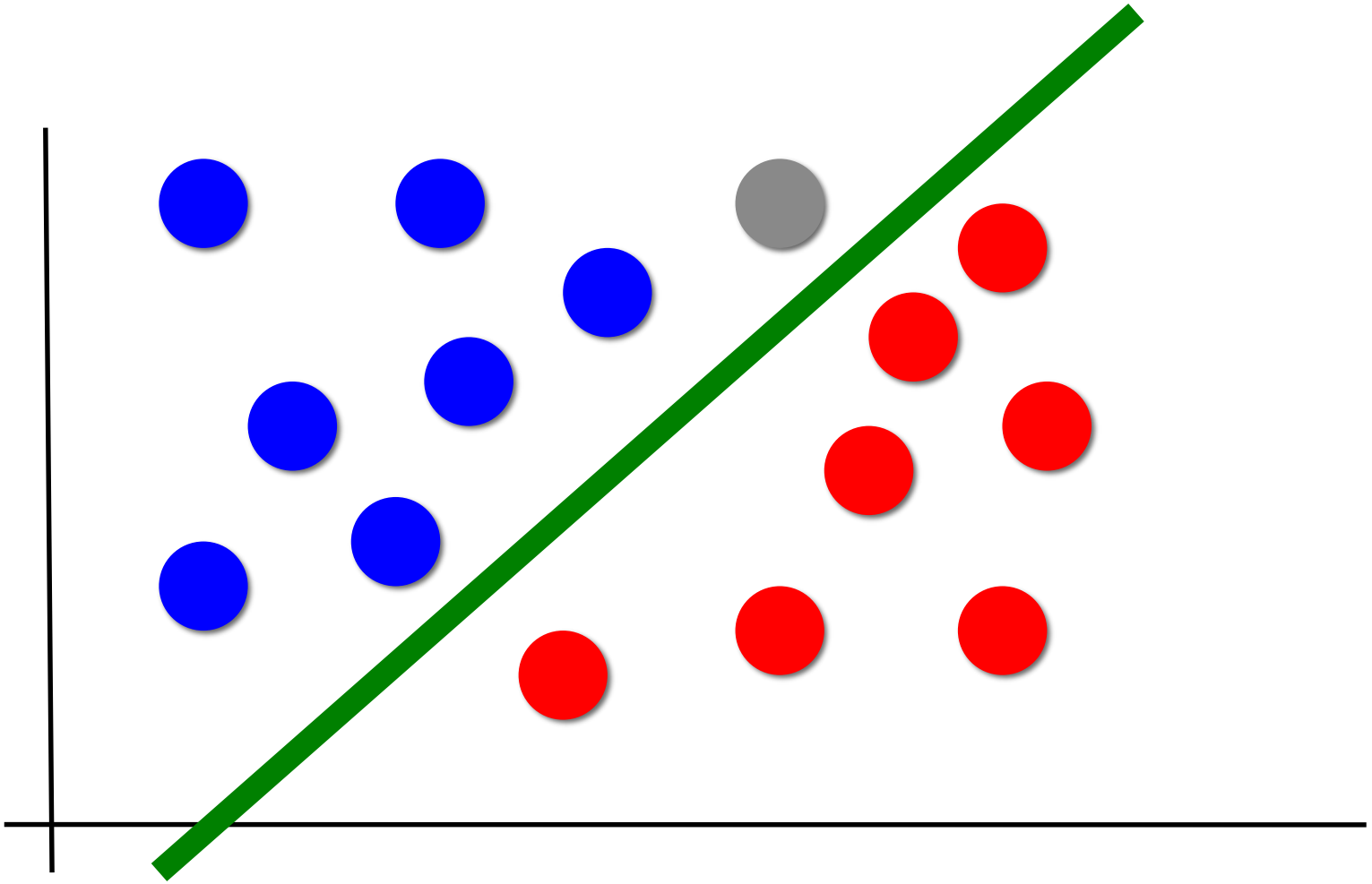
Graphical Example of Classification



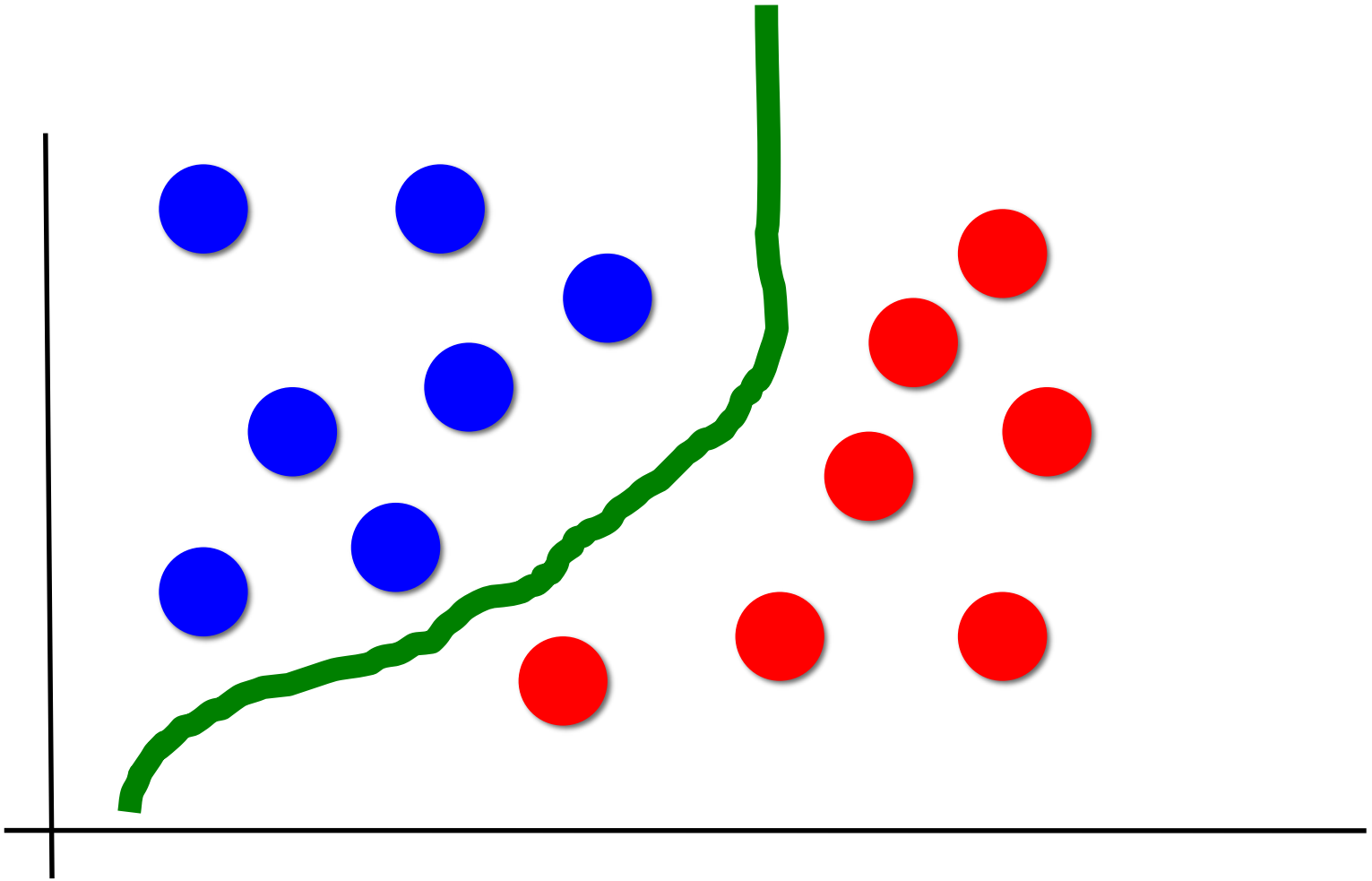
Graphical Example of Classification



Graphical Example of Classification



Decision Boundaries



So back to our Fruit example!

- What if we have a new, unknown fruit? What kind of machine learning task would we need to predict the type of the unknown fruit?

Object	Height	Width	Color	Mass	Round?
Apple	6cm	7cm	Red	330g	TRUE
Orange	6cm	7cm	Orange	330g	TRUE
Lemon	5cm	4cm	Yellow	150g	FALSE
???	5cm	6cm	Orange	300g	TRUE

So back to our nouns!

- What kind of machine learning task would suit our task and our data?

- Remember: We would like use our features (number of nouns per sentence) to predict whether a sentence belongs in the news category or the romance category.

- Classification
  - Used to predict categorical values

How does machine learning work?

- Step 1: Training

- Step 2: Testing

Machine Learning: In Practice

Using scikit-learn

scikit-learn (sklearn)

- Open source software for machine learning in Python
- User friendly
- Well documented
- Quickly becoming standard in industry for machine learning
  - Used by Spotify, OKCupid, Evernote (just to name a few)

- We will be using scikit-learn for the machine learning section of this workshop

Data management & wrangling: pandas

- Often it can be useful to put your data together into a format that makes it easy to visualize and manage

- Pandas is a library for Python that allows for data wrangling and analysis
  - Makes use of a data structure called a DataFrame, which looks similar to an Excel spreadsheet
  - scikit-learn often prefers for data to be in a DataFrame format

Training a machine learning algorithm

- Goal: Build a machine learning model that can predict whether a sentence belongs in the news category or the romance category of the Brown corpus

- We cannot use the same data that we trained the model on to evaluate how well the model does
  - The model will remember the data, and therefore always make correct predictions
  - This will not tell us how well our model will perform on new data
  - We will therefore partition the data into two parts

Partitioning data

- We partition our data into two sets: *training data* (train set) and *testing data (*test set)
- We use the training data to build our machine learning model
- We use the testing data to evaluate how well our model performs on previously unseen data
- A good starting place?
  - 75% data for train
  - 25% data for test

Creating a training and testing set

- For both the training and testing sets of data, we need to create:
  - A feature vector that represents our data
  - A label for each sentence (does it come from news or romance?)

- Why do we need labels for the testing set?
  - So we can evaluate our model and see how well it predicts by comparing it's predictions to the ground truth labels

Create labels

- Decide how much data we want to hold out for testing
  - 500 sentences from news, 500 sentences from romance
- Create a list of labels for as many sentences as we have in the corpus, minus 500 from each category
  - These are our training labels
- Create another list of 500 news labels and 500 romance labels
  - These are our testing labels

Create data

- Remember our features?
  - Number of nouns in each sentence in the news and romance categories
- Remove the first 500 features from news and romance and concatenate them into a list (preserving order so the labels align)
  - This is our testing set
- Concatenate the remaining features from news and romance into a single list (again preserving order)
  - This is our training set

Final steps in data prep

- Once we have created our data, we need to make sure it's the correct format for scikit-learn
  - Split the full DataFrame into training and testing sets

- In sklearn, it is convention to use X as a variable name for data, and y as a variable name for labels
  - This will leave us with a training set called `X_train, y_train` (training data and training labels)
  - We will also have a testing set called `X_test, y_test` (testing data and testing labels)

One additional step…

- Because we are only using one feature (most machine learning tasks use many features), sklearn asks us to reshape our data using the following syntax

```
X_train = X_train.reshape(-1,1)
X_test = X_test.reshape(-1,1)
```

- sklearn is excellent at throwing useful warnings and errors, letting you know what needs to be changed in order for it to run!

We're ready for machine learning!

- …What machine learning algorithm do we use?

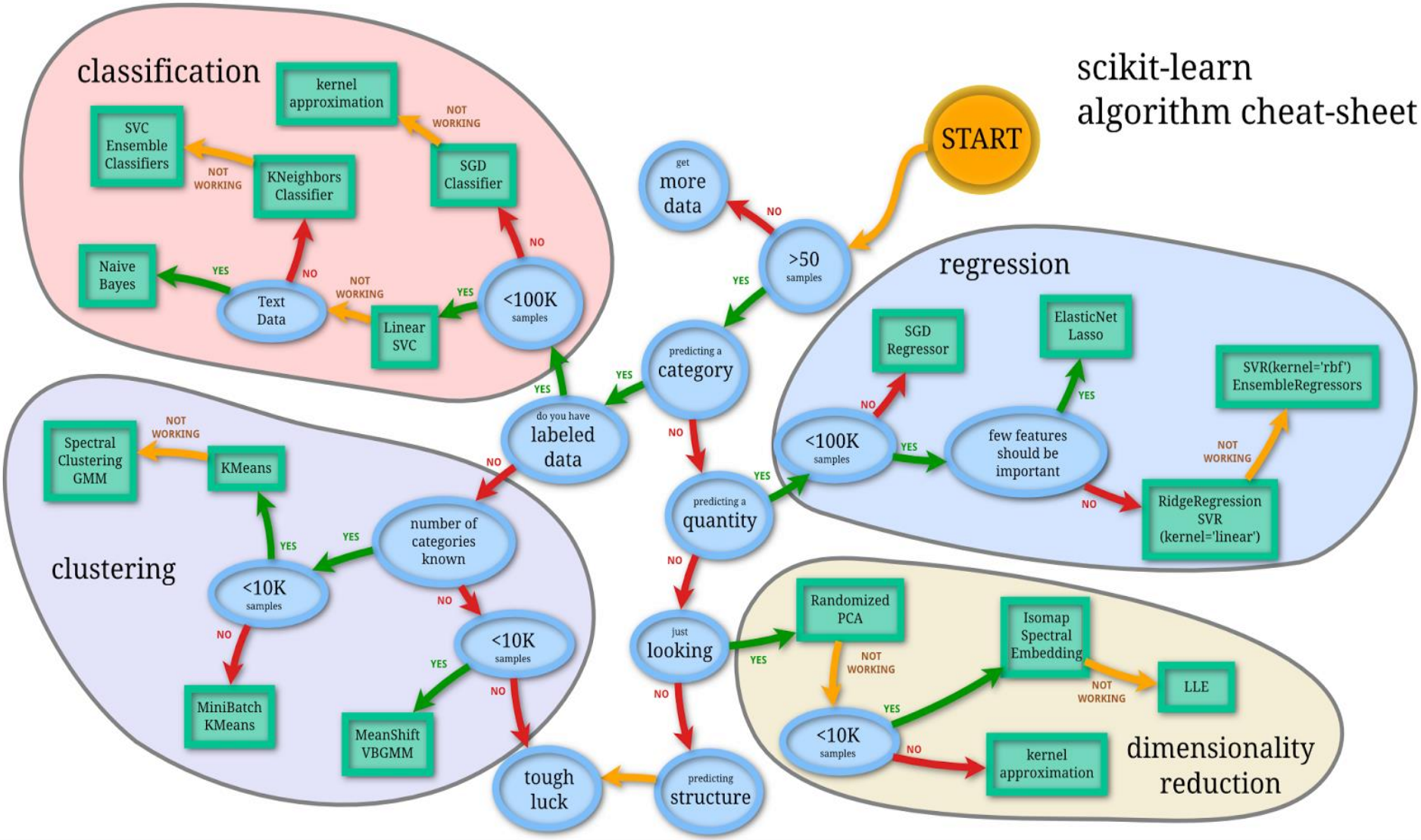


Image by Andreas Müller

Linear SVC

- A linear model for classification
  - Stands for Linear support vector classifier
  - A type of support vector machine (SVM) algorithm

- A linear classifier separates classes using a line, a plane, or a hyperplane

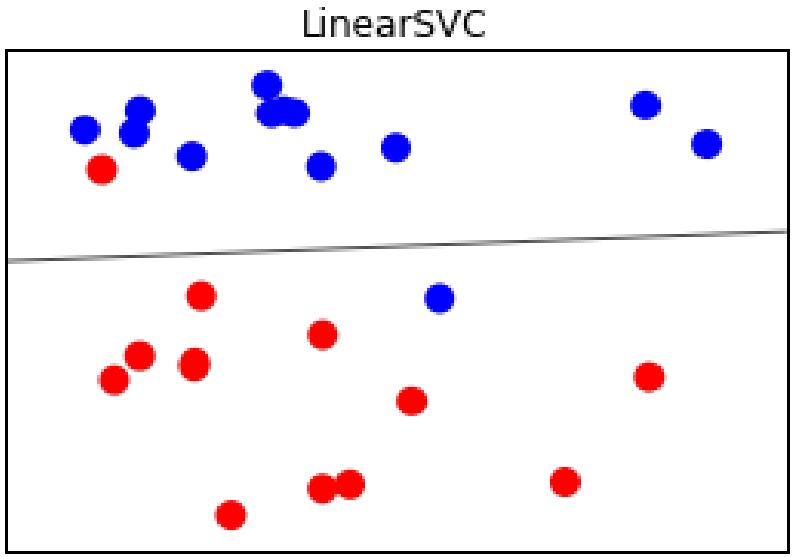


Image by Andreas Müller (2016)

The beauty of sklearn: simplicity

- To do machine learning with sklearn, you really only need to know a few commands.

1. Import your desired classifier
   - `from sklearn.svm import LinearSVC`

2. create an instance of your machine learning algorithm
   - `classifier  = LinearSVC()`

3. Fit, predict, and score!
   - `classifier.fit (X_train, y_train)`
   - `classifier.predict (X_test, y_test)`
   - `classifier.score(X_test)`

Evaluating your classifier

- Our classifier is currently about 67% accurate
  - This is pretty good, considering we are only using one feature!
- Can also be visualized in a confusion matrix

	p' (Predicted)	n' (Predicted)
p (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

Image by Marcel Caraciolo (2010)

Our confusion matrix

	Predicted news label	Predicted romance label
Actually a news label	**342**	158
Actually a romance label	168	**332**

- Our model predicts more news labels than romance labels
  - This is understandable, as our training data has more instances of news noun counts than romance noun counts

Great! When and how can I use this?

- Imagine you wanted to perform sentiment analysis on a text, perhaps on a set of movie reviews.

- The simplest way of determining sentiment is by calculating the mean polarity of a document, and providing those numbers as features for a model
  - “Great story, great actors, great movie.”

- Can also distinguish between expectation and reality (a high standard deviation suggests this)
  - “I thought this movie would be terrible. It was great.”

But of course, it's not always that easy

- Consider the following:
  - "I did not like this movie. It is not wonderfully written. It has no good actors. I would never recommend this movie."
  - "I did like this movie. It is wonderfully written. It has good actors. I would recommend this movie."
- What's the problem?
  - "No", "Not", and "Never" are often considered *stop words* – words that are ignored because they are too frequent & often unhelpful
    - a, the
  - So it is very likely that simple sentiment analysis systems would consider these **both** to be positive reviews!

Wow, we've done a lot of things! Let's recap

- Explored text analysis and the basics of NLP
  - First passes of tools in NLTK (concordance, word tokenization, pos tagging)
  - Frequency distributions as a means of counting stuff
- Represented textual analysis information as a set of machine-understandable features
- Used scikit-learn and pandas to build a classifier for a specific text analysis question
  - Building training and testing sets of data
  - Training and evaluating a machine learning model

“This is awesome! How can I learn more?”

- Come to Digital Fellows workshops!
- Come to Python Users Group!

- Also there are books!
  - “Natural Language Processing with Python”, Bird, Klein, & Loper (2009)
  - “Introduction to Machine Learning with Python”, Müller & Guido (2016)

Acknowledgements

- Some slides adapted from Matt Huenerfauth's 'Language Technologies' class (spring of 2012)
- Some slides adapted from Andrew Rosenberg's 'Machine Learning' class (spring 2011)
- More slides adapted from Andrew Rosenberg's "Methods in Computational Linguistics II" class (2015)
- Even more slides adapted from Andrew Rosenberg's "Natural Language Processing, Machine Learning, and the Web" course (Spring 2015)
- Additional images from "Introduction to Machine Learning with Python" (Müller & Guido, 2016)
- Additional help: Sam Raker, Andreas Müller

Thank you!

- Problems? Questions? Corrections? Additions?

ADDITIONAL MATERIALS

That we didn’t have time to get through

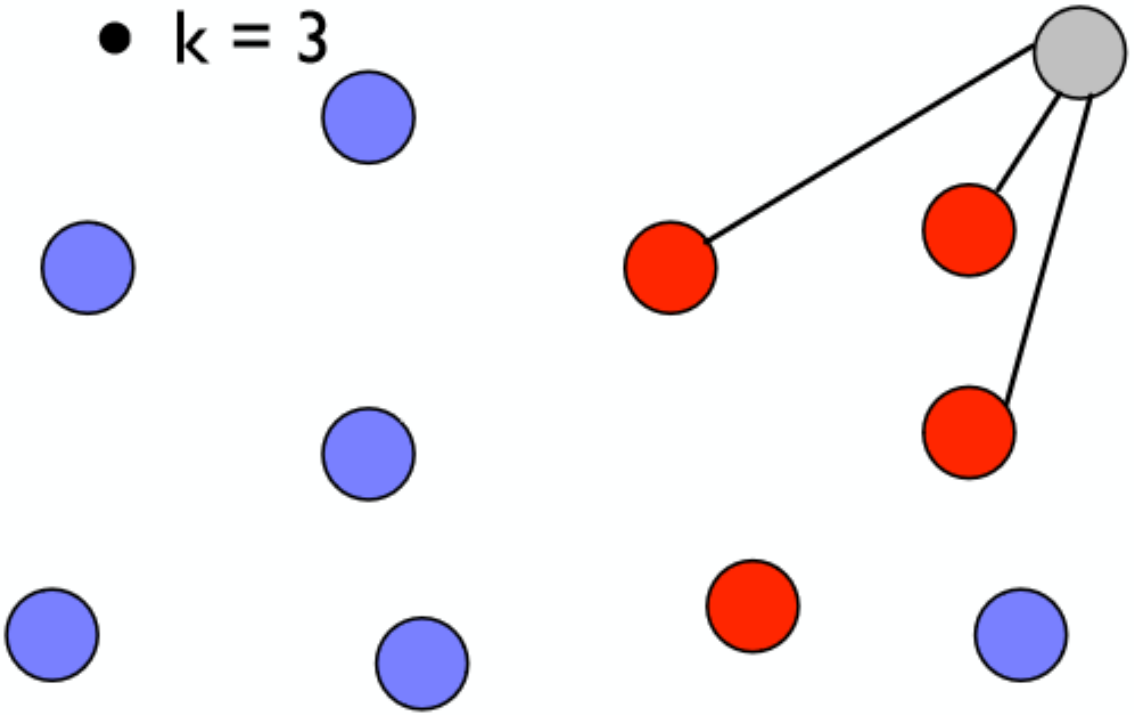
So what comes next?

- Evaluate your features – were they effective for this task?
  - Can we distinguish between news and romance sentences just by counting how many nouns are in each sentence?
    - Yes! …to some degree!
  - Are there other features that we could add to our model that would better represent our sentences, and improve classification?
    - Almost certainly!

- Evaluate your classifier - is it optimal for this task?
  - Tuning of parameters
  - Using a different classification algorithm
    - What happens if we use the k nearest neighbor algorithm?

k nearest neighbors – Classification example

- One way we could identify it is by seeing what it's "nearest" to, and give it the same label
  - Ex. 3 "nearest" things
  - Graphical example!



Changing classifiers in sklearn

1. Import your desired classifier
   - `from sklearn.neighbors import KNeighborsClassifier`

2. create an instance of your machine learning algorithm
   - `classifier  = KNeighborsClassifier()`

3. Fit, predict, and score! (Oh look! It's the same!)
   - `classifier.fit (X_train, y_train)`
   - `classifier.predict (X_test, y_test)`
   - `classifier.score(X_test)`

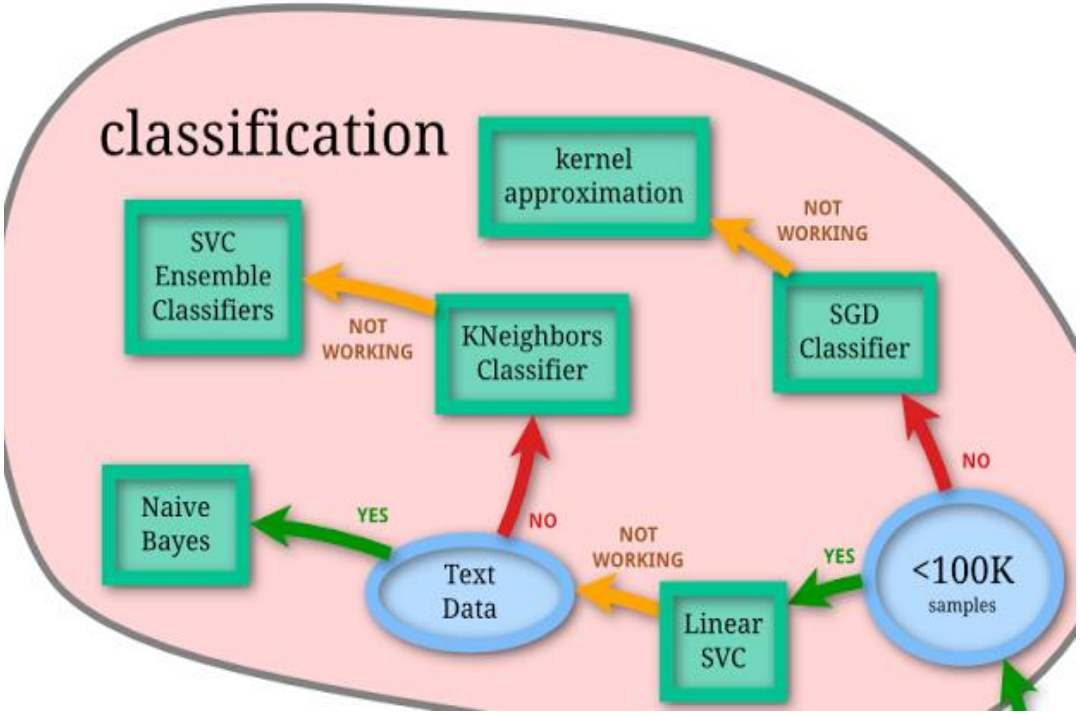
So how did kNN do?

- Ouch! 54% accurate!
- We are over-predicting news labels

	Predicted news label	**Predicted romance label**
Actually a news label	**449**	51
Actually a romance label	404	**96**

Different algorithms are good at different things

- This demonstrates that k-nearest neighbors isn't really the best algorithm for this task
- Not really a surprise….



So what comes next?

- Evaluate your features – were they effective for this task?
  - What are some other features we could add to our model that would better represent our sentences, and improve classification?
- Earlier, we looked at a frequency distribution of modal words (“can” “might” “should” “could”) in our news and romance categories
  - Do you think adding how many modal words a sentence contains would improve our model?

Adding new features

- Create the new feature vector that you would like to add
  - Make sure that you are preserving the test/train split
- Add columns to your DataFrame

```
df.insert(<location>,<"name">,<features>)
```

  - Note: DataFrames are mutable

- Update X_train, y_train, X_test, and y_test to reflect the new features

```
X_train = train_data[train_data.columns[:2]]
y_train = train_data["labels"]
```

Retrain and re-evaluate!

- How do we do?
  - No real improvement – still at 67%
  - Demonstrates that this is not a particular effective feature for this task

- It is important to keep in mind the problem that you are trying to solve, and to find good ways to model and represent that problem.
  - What works well for one text problem may not work well for another

A word to the wise

- When building a machine learning model, you want to be careful not to put in too many features that are very specific to the training corpus itself
  - Building a model that achieves high accuracy on the training set but doesn't generalize well to new data is called *overfitting*
- However, you also don't want to build too simple a model by not accounting for variation in your training set
  - Using too simple a model is called *underfitting*

Overfitting and Underfitting

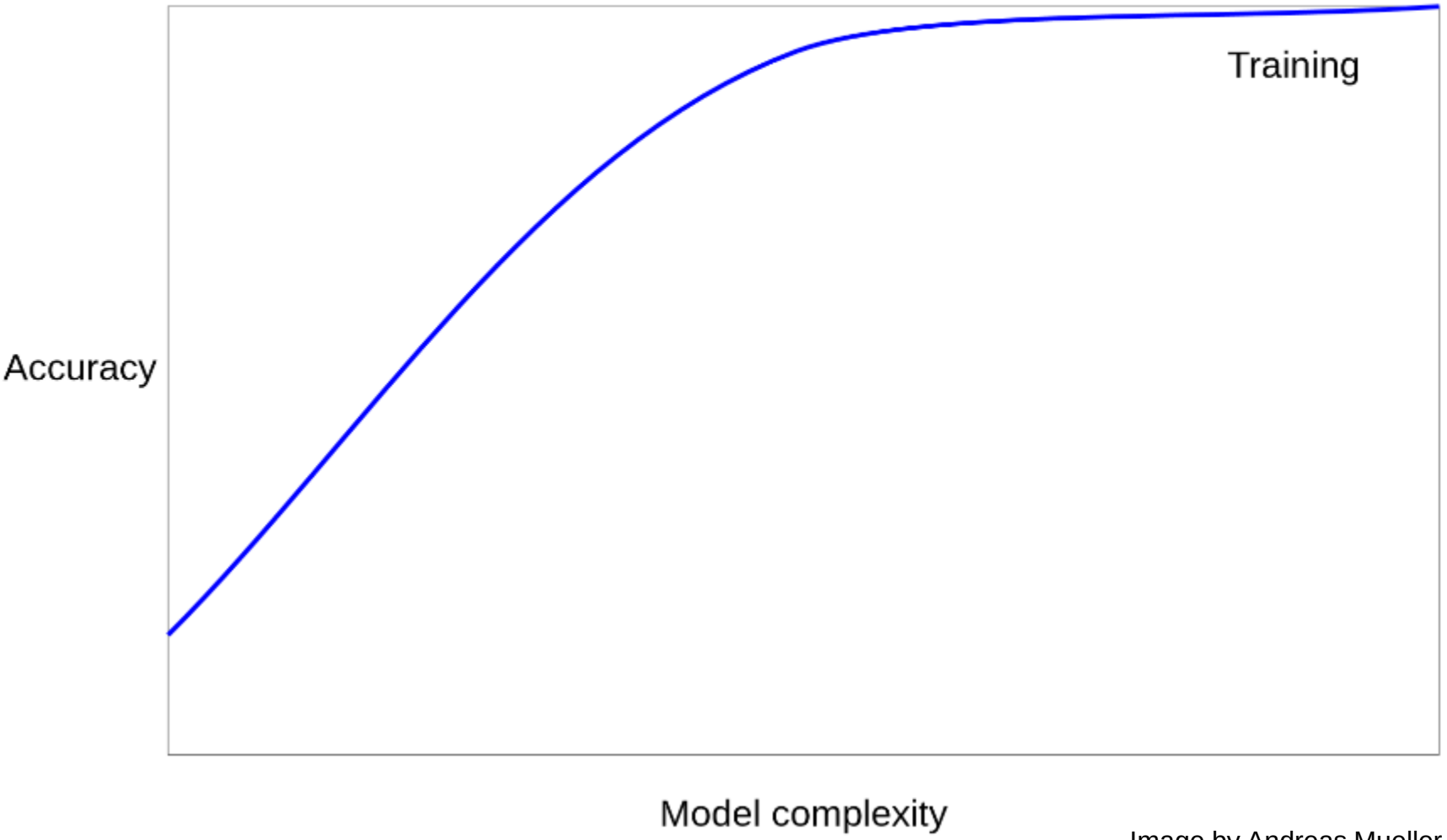


Image by Andreas Mueller, 2016

Overfitting and Underfitting

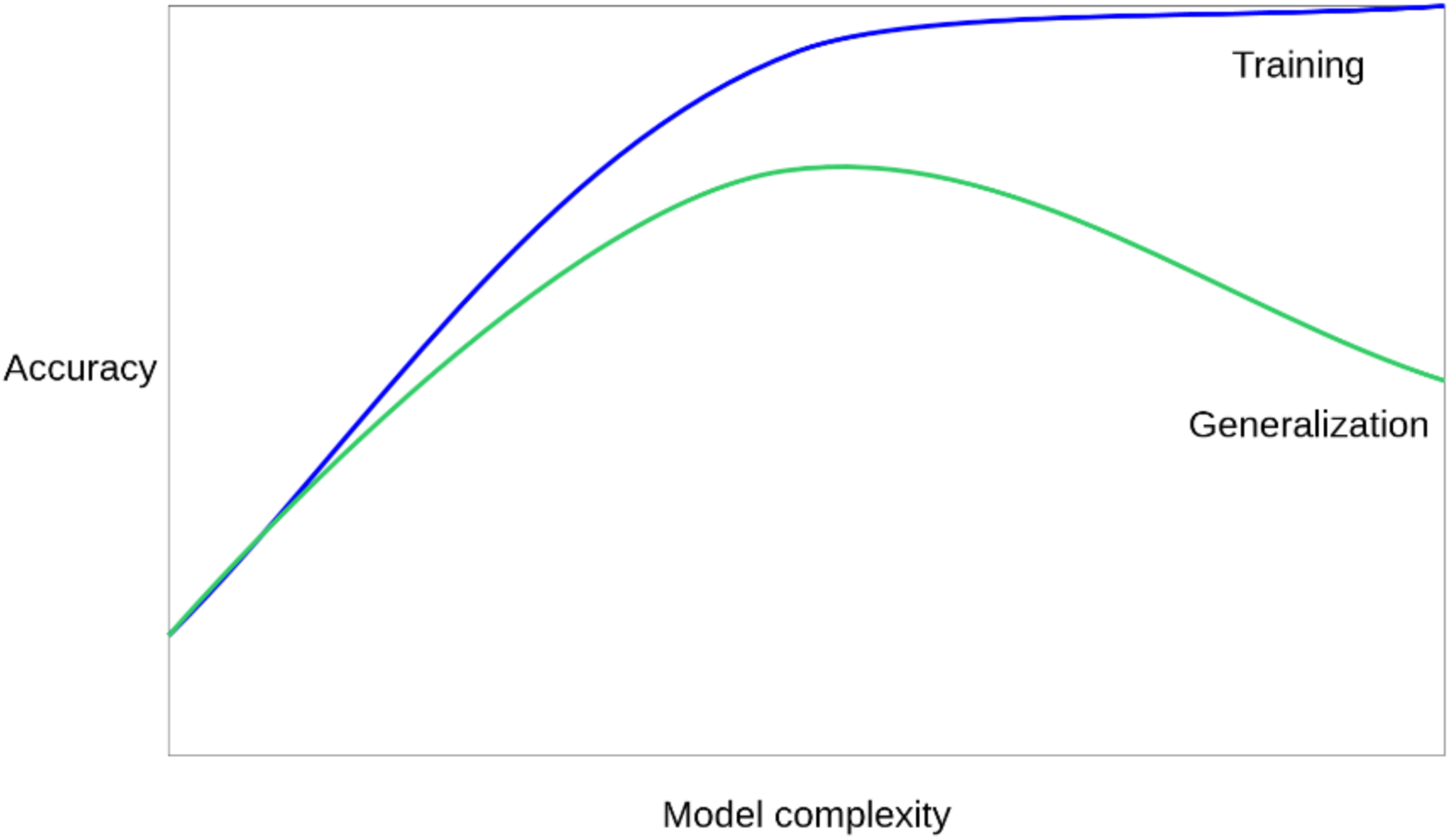


Image by Andreas Mueller, 2016

Overfitting and Underfitting

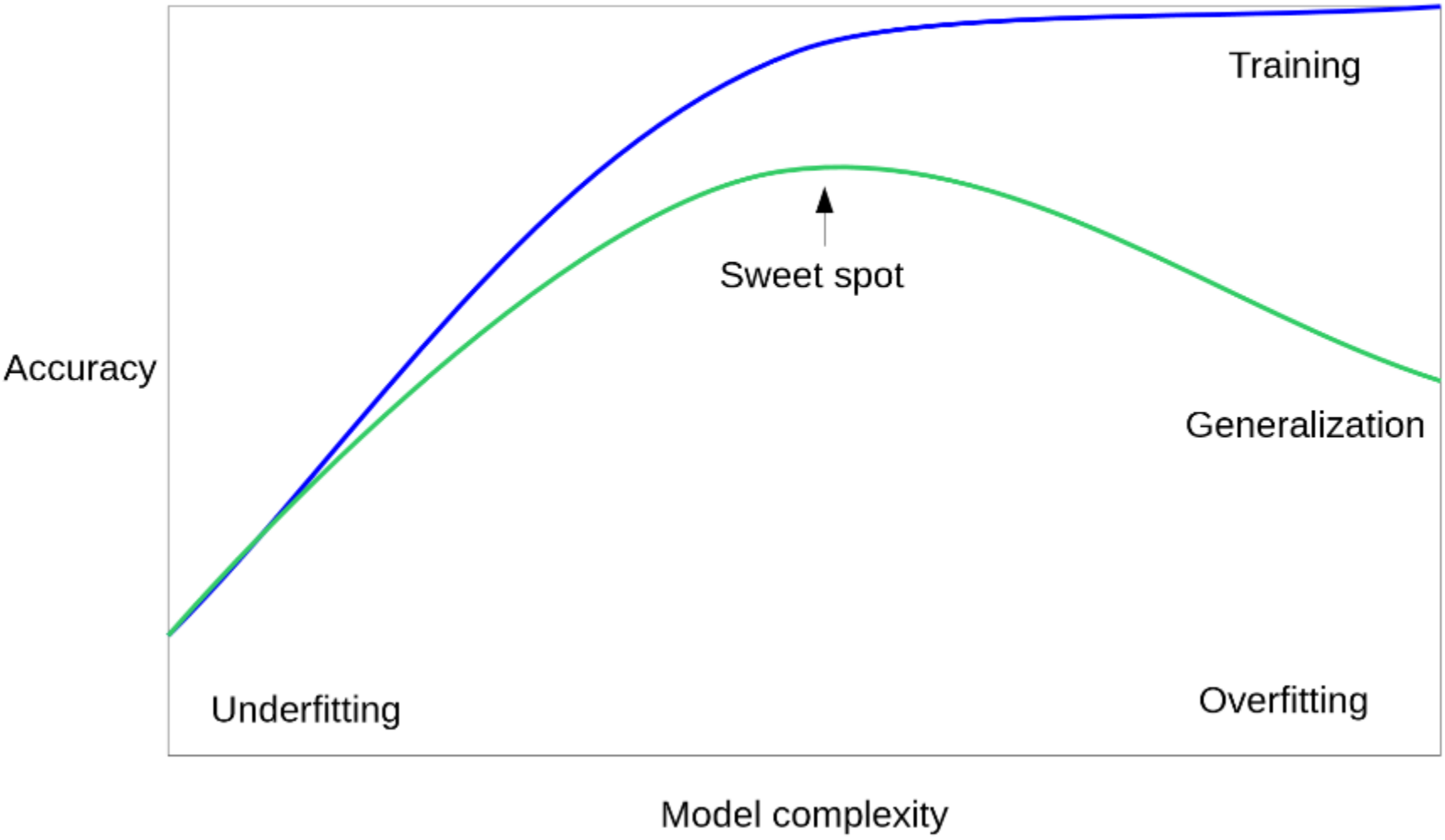


Image by Andreas Mueller, 2016