

Introduction to Natural Language Processing (NLP)

LECTURE 1

Lisa Andreevna Chalaguine

Lecture

About me and my Research

What is NLP

Importance/Applications of NLP

Challenges

Practical

Python Libraries needed

Choice of Dataset

Text Cleaning and Preprocessing

- **Stopword removal**
- **Tokenisation**
- **Punctuation Removal**
- **Lemmatisation/Stemming**

Lisa A. Chalaguine

PhD Student (Intelligent Systems Group)

Develop Chatbots that argue with people

Converted from Law to Computer Science

Discovered that being a Penny is better than a being Rachel Zane

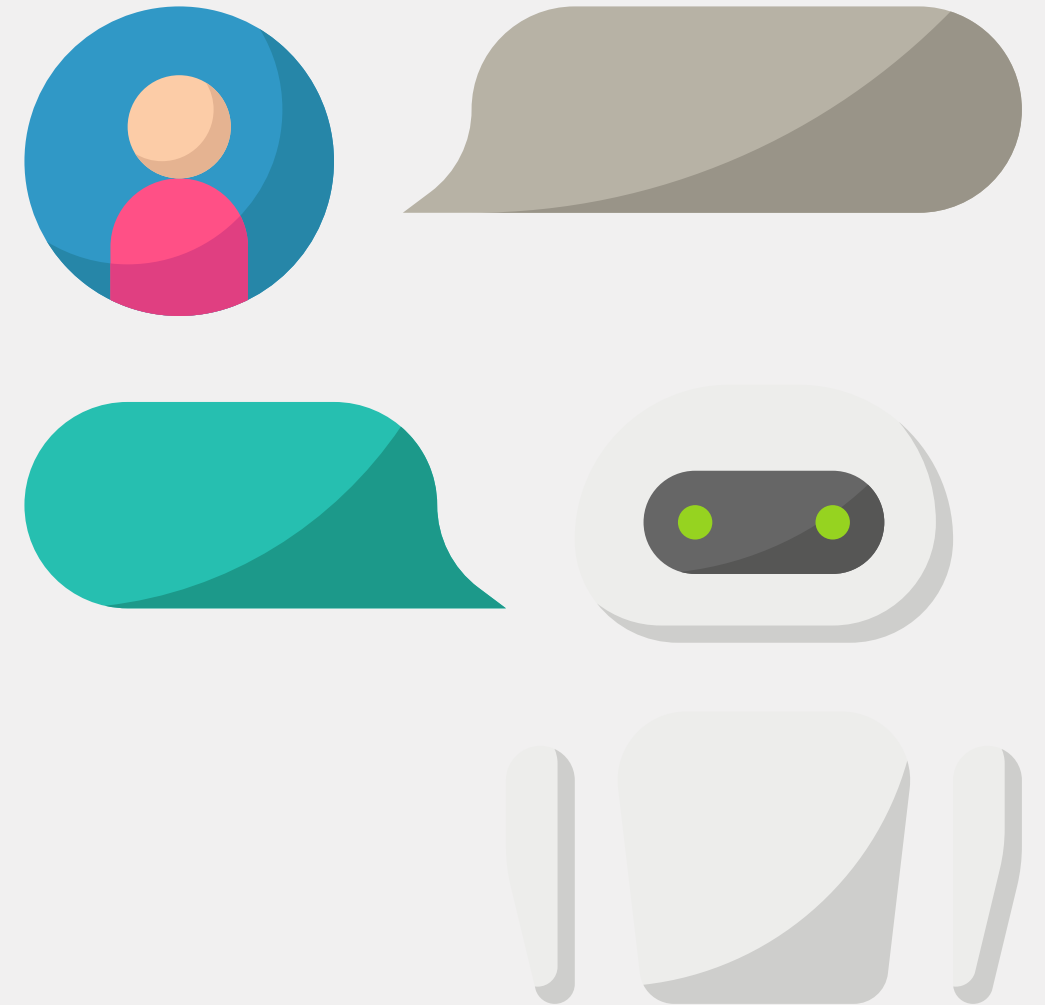
Originally from Belarus

And since August most people finally know this country exists...



Demo of latest Project

Chatbot that tries to convince people to get a COVID-19 vaccine, once one is developed and becomes available



Applications

Sentiment/Opinion Analysis (e.g. social media, product reviews...)

Chatbots/Virtual Assistants (Siri, Alexa, Cortana, Google Assistant...)

- Speech Recognition
- Natural Language Generation

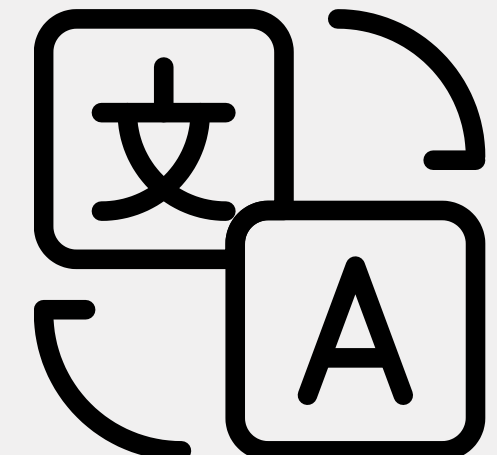
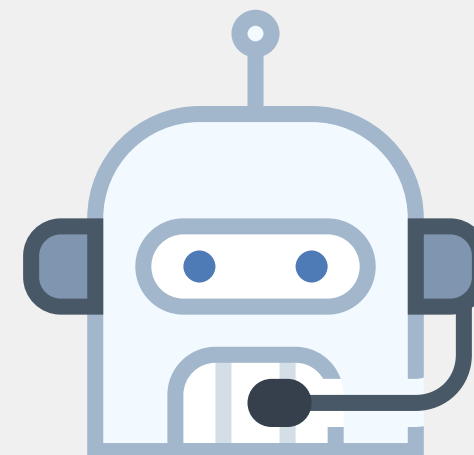
Text Classification (e.g. spam filtering)

Information extraction

Machine Translation

Text Summarisation

Auto-Correct



Challenges of NLP

Unstructured data

Ambiguity of language (same word - different meaning)

Synonymy (same meaning - different words)

Coreference (what/who do pronouns refer to in subsequent sentences)

Irony, Sarcasm...

What is NLP?

NLP is a field of AI that gives machines the ability to read, understand and derive meaning from human languages

Discipline that focuses on the interaction between data science and human language.

Programming computers to process and analyse large amounts of natural language data.

Any questions so far?

Practical - stuff you need

Firstly you need ...

- A laptop
- Python 3.6+
- Jupyter Notebooks

Python libraries (for now)

- Pandas
- NLTK

Anaconda comes with most the required libraries and Jupyter!!

Python libraries for next week

- scikit-learn
- (requires NumPy & SciPy)

Practical - GitHub page

**I will post all of the materials on the
following GitHub page**

https://github.com/lisanka93/UCL_F2F_NLP101

Try yourself

Select one of the datasets in the dropbox datasets folder and apply some of the preprocessing techniques yourself

For next week

Find a (bigger, more complex) corpus that interests you and that can be used for classification.

NOTE - it will probably not be in .csv format. Your homework is then to write some code to read the corpus into Python

Recap

Last week we covered how to **preprocess** textual data


If you were not here please access the following **GitHub** repository and familiarise yourself with the content from Week 1

https://github.com/lisanka93/UCL_F2F_NLP101

Preprocessing

You should be able to understand what sort of preprocessing is necessary and adequate for different types of textual data.

Preprocecssing includes

- **Removal of unwanted characters**
 - **Normalisation**
 - **Tokenisation**
 - **Stopword Removal**
 - **Lemmatising/Stemming**
- 

Lecture

Word Vectors

Classification Algorithms

Measuring Model Performance

Practical

Choice of Dataset

(for those who weren't here last week)

Training a classifier

- Preprocessing
- Vectorisation
- Training & Cross Validation
- Measuring Model Performance

Representing text in computer-readable format

A computer (obviously) cannot understand words. So we need to somehow convert text to numbers (vectors). There are several ways of doing that

- **Bag of Words**
- **Tf-idf**
- **Pre-trained word vectors**

Bag of words

"apples are great but so are pears, however, sometimes I feel like oranges and on other days I like bananas"

The sentence contains 17 distinct words. The following three sentences can then be represented as vectors using our vocabulary:

- **"I like apples"** --> [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]
- **"Bananas are great. Bananas are awesome"** --> [0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2] **or** [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
- **"She eats kiwis"** --> [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

tf-idf

T

tf-idf stands for "term-frequency-inverse-document-frequency. It indicates what the importance of a word is in order to understand the document or dataset.

Example: Students write essay on the topic "My family". The word "family" will come up in every single essay and is therefore not informative. Other words like "grandmother", "dog", "separated" will carry more information.

tf-idf calculation

^T

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document)

tf gives equal importance to every single word so we need to define some weighing down of the frequent terms while scaling up the rare ones, which decides the importance of each word

IDF(t) = log10(Total number of documents / Number of documents with term t in it)

tf-idf(t) = tf(t) * idf(t)

tf-idf numerical example

Step 1 : Calculate TF

Step 1.1 : Term Count for each document

| Document 1 | | Document 2 | |
|------------|------------|------------|------------|
| Term | Term Count | Term | Term Count |
| this | 1 | this | 1 |
| is | 1 | is | 1 |
| a | 2 | another | 2 |
| sample | 1 | example | 3 |

Step 1.2 : Now calculate total number of words in each document

Document 1 : Total words are = 5

Document 2 : Total words are = 7

Step 1.3 : Now calculate TF

$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$

$$tf("this", d_1) = \frac{1}{5} = 0.2$$

$$tf("this", d_2) = \frac{1}{7} \approx 0.14$$

tf-idf numerical example

Step 2 : Calculate IDF

Step 2.1 : IDF calculation

$IDF(t) = \log(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it})$

So here there are 2 document and term "this" appears in both of them

So IDF is given below.

$$idf("this", D) = \log\left(\frac{2}{2}\right) = 0$$

Step 3 : TF x IDF calculation

$$tfidf("this", d_1) = 0.2 \times 0 = 0$$

$$tfidf("this", d_2) = 0.14 \times 0 = 0$$

zero implies that the word is not very informative

For other words is given below

$$tf("example", d_1) = \frac{0}{5} = 0$$

$$tf("example", d_2) = \frac{3}{7} \approx 0.429$$

$$idf("example", D) = \log\left(\frac{2}{1}\right) = 0.301$$

Step 4: TF X IDF for word example

$$tfidf("example", d_1) = tf("example", d_1) \times idf("example", D) = 0 \times 0.301 = 0$$

$$tfidf("example", d_2) = tf("example", d_2) \times idf("example", D) = 0.429 \times 0.301 \approx 0.13$$

Pre-trained word vectors



Word2Vec is one of the most popular techniques to learn word embeddings using a shallow neural network

developed by Tomas Mikolov in 2013 at Google

"Have a good day" and "Have a great day"
very similar meaning

One-hot encoded vectors of our vocabulary:

Have = [1,0,0,0,0]

a=[0,1,0,0,0]

good=[0,0,1,0,0]

great=[0,0,0,1,0]

day=[0,0,0,0,1]

words are independend of each other

Pre-trained word vectors

objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0

solution: **distributed representations**

we introduce some dependence of one word on the other words. The words in context of this word would get a greater share of this dependence

Word2Vec is a method to construct such an embedding

- Skip Gram
- Common Bag Of Words (CBOW)

CBOW



Will not go deep into the maths. In short: CBOW works by training a neural network on a vocabulary of words where the input are one-hot encoded vectors (context) and the outcome is a target word. The NN updates the weights in the hidden layer in order to predict the target word given the context words. The trained weights are the word embeddings

Example

| | fruit | animal | edible |
|----------|-------|--------|--------|
| Apple | 0.9 | 0.01 | 0.85 |
| Orange | 0.8 | 0.02 | 0.9 |
| Elephant | 0.1 | 0.95 | 0.02 |

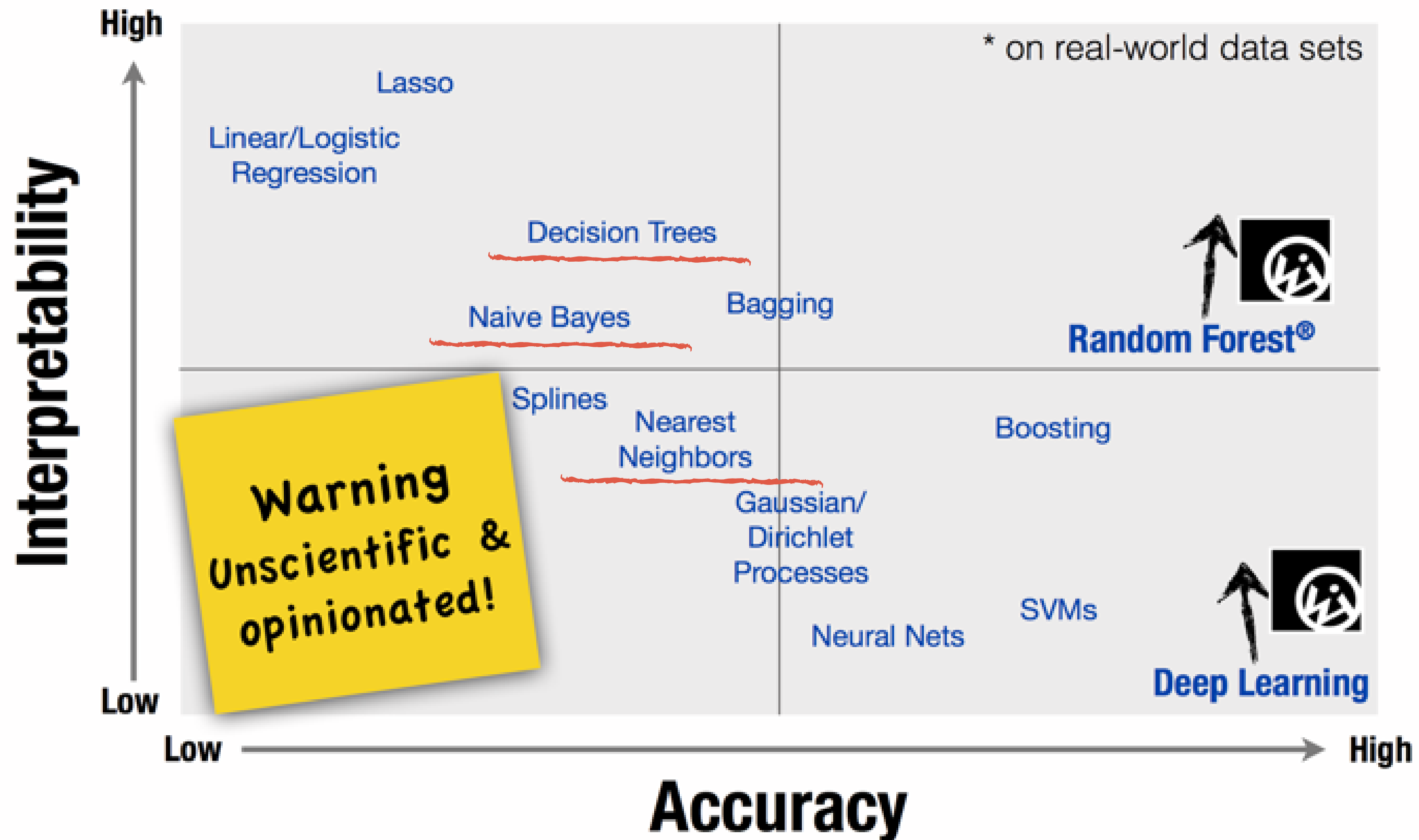
Classification Algorithms

(for today)

- **Naive Bayes**
- **KNN (K-Nearest Neighbour)**
- **Decision Trees**

Note: scikit-learn makes it quite easy to experiment with different algorithms, so feel free to test out any others (e.g. SVM)

ML Algorithmic Trade-Off



(Multinomial) Naive Bayes

- based on Bayes' Theorem (probabilistic algorithm)
- works particularly well with NLP problems

Example

| Text | Tag |
|--------------------------------|------------|
| "A great game" | Sports |
| "The election was over" | Not Sports |
| "Very clean match" | Sports |
| "A clean but forgettable game" | Sports |
| "It was a close election" | Not Sports |

(Multinomial) Naive Bayes

- Since Naive Bayes is a probabilistic classifier, we want to calculate the probability that the sentence “A very close game” is Sports and the probability that it’s Not Sports
- Written mathematically, what we want is $P(\text{Sports} \mid \text{a very close game})$ — the probability that the tag of a sentence is *Sports* given that the sentence is “A very close game”
- We ignore the word order and sentence construction (hence, NAIVE bayes)
- Features will be the counts of each of these words

(Multinomial) Naive Bayes

Example: A very close game

$P(\text{Sports}) = 3/5$ and $P(\text{Not Sports}) = 2/5$

$P(\text{a very close game} \mid \text{Sports}) = P(\text{a} \mid \text{Sports}) * P(\text{very} \mid \text{Sports}) * P(\text{close} \mid \text{Sports}) * P(\text{game} \mid \text{Sports})$

Problem: “close” is not in any Sports texts. Since we are multiplying, we will end up with $P = 0$!

Solution: laplace smoothing: adding 1 to every count, so it’s never 0. To balance this, we add the number of possible words to the divisor (so division will never be greater than 1)

| Word | P(word Sports) | P(word Not Sports) |
|-------|-------------------------|------------------------|
| a | $\frac{2 + 1}{11 + 14}$ | $\frac{1 + 1}{9 + 14}$ |
| very | $\frac{1 + 1}{11 + 14}$ | $\frac{0 + 1}{9 + 14}$ |
| close | $\frac{0 + 1}{11 + 14}$ | $\frac{1 + 1}{9 + 14}$ |
| game | $\frac{2 + 1}{11 + 14}$ | $\frac{0 + 1}{9 + 14}$ |

| Text | Tag |
|--------------------------------|------------|
| “A great game” | Sports |
| “The election was over” | Not Sports |
| “Very clean match” | Sports |
| “A clean but forgettable game” | Sports |
| “It was a close election” | Not Sports |

And Finally...

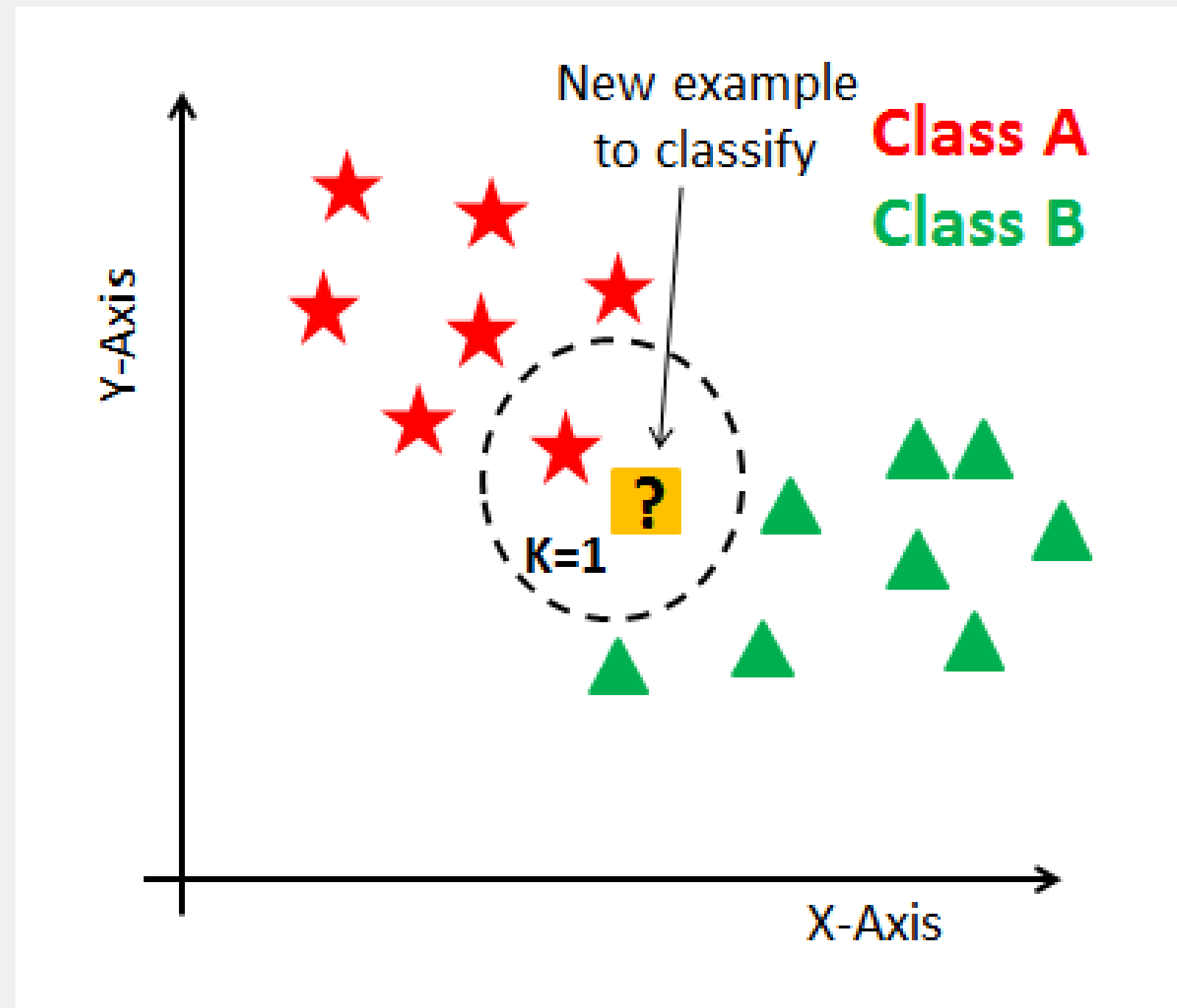
Now we just multiply all the probabilities, and see who is bigger:

$$\begin{aligned} &P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports) \times \\ &P(Sports) \\ &= 2.76 \times 10^{-5} \\ &= 0.0000276 \end{aligned}$$

$$\begin{aligned} &P(a|Not Sports) \times P(very|Not Sports) \times P(close|Not Sports) \times \\ &P(game|Not Sports) \times P(Not Sports) \\ &= 0.572 \times 10^{-5} \\ &= 0.00000572 \end{aligned}$$

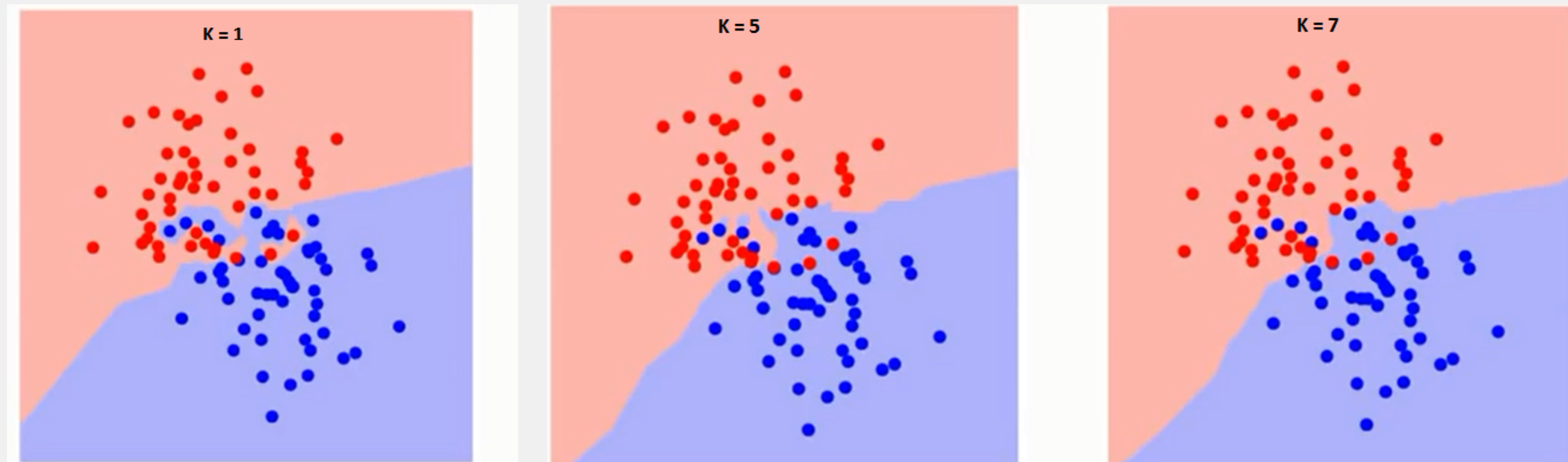
Excellent! Our classifier gives “A very close game” the **Sports** tag.

KNN



KNN

- with a given K value we can make boundaries of each class
- These boundaries will segregate the classes
- let's see the effect of the value for K on the class boundaries



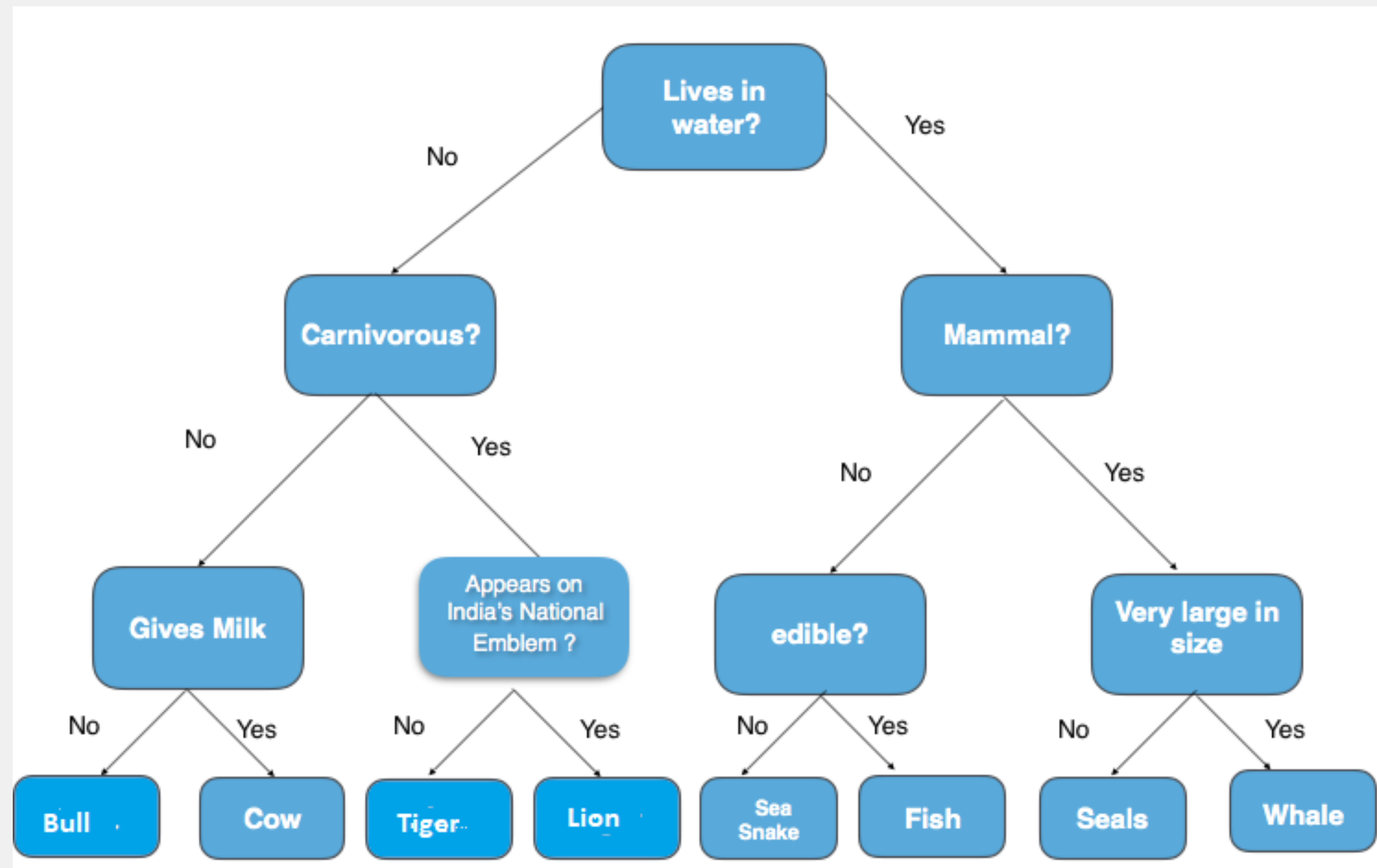
OVERFITTING!

Algorithm

- Initialize K to your chosen number of neighbors
- For each example in the data
 - Calculate the distance between the query example and the current example from the data
 - Add the distance and the index of the example to an ordered collection
- Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
- Pick the first K entries from the sorted collection
- Get the labels of the selected K entries
- If regression, return the mean of the K labels/If classification, return the mode of the K labels

Decision Trees

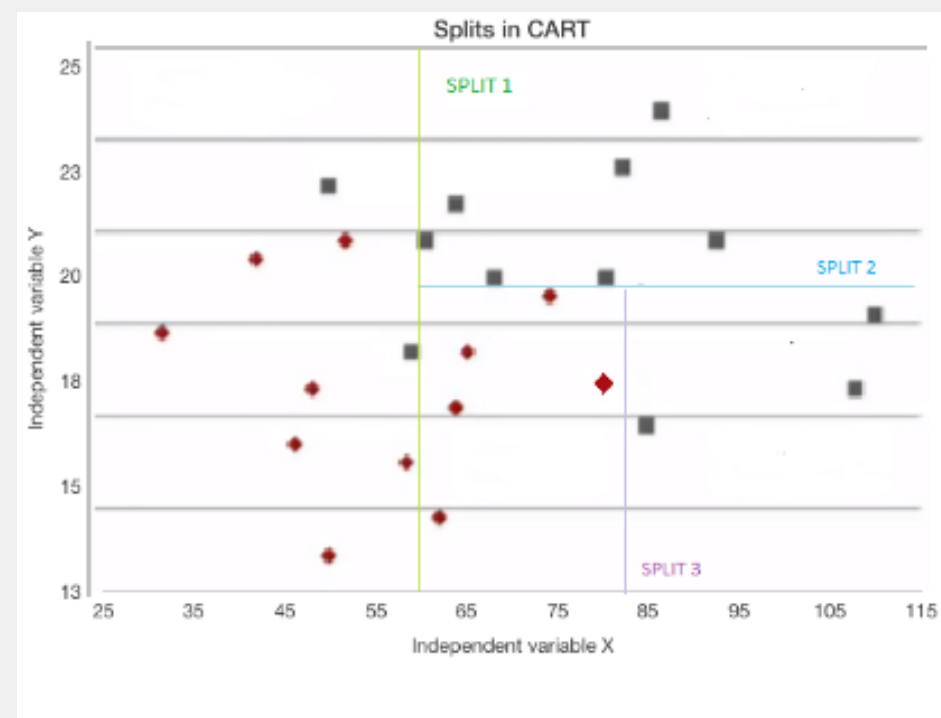
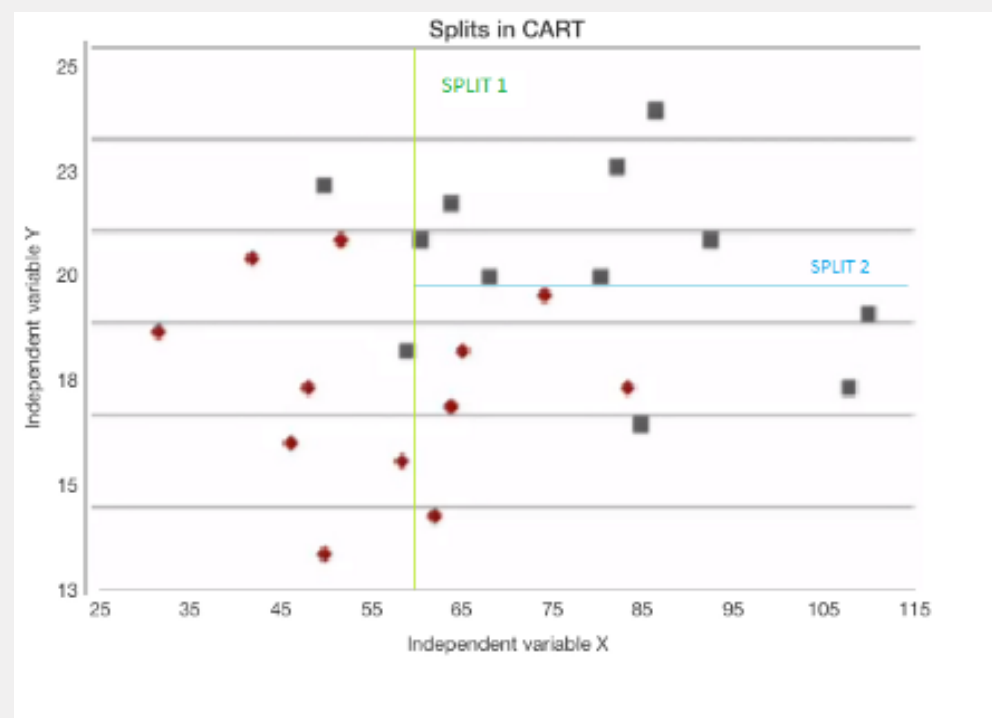
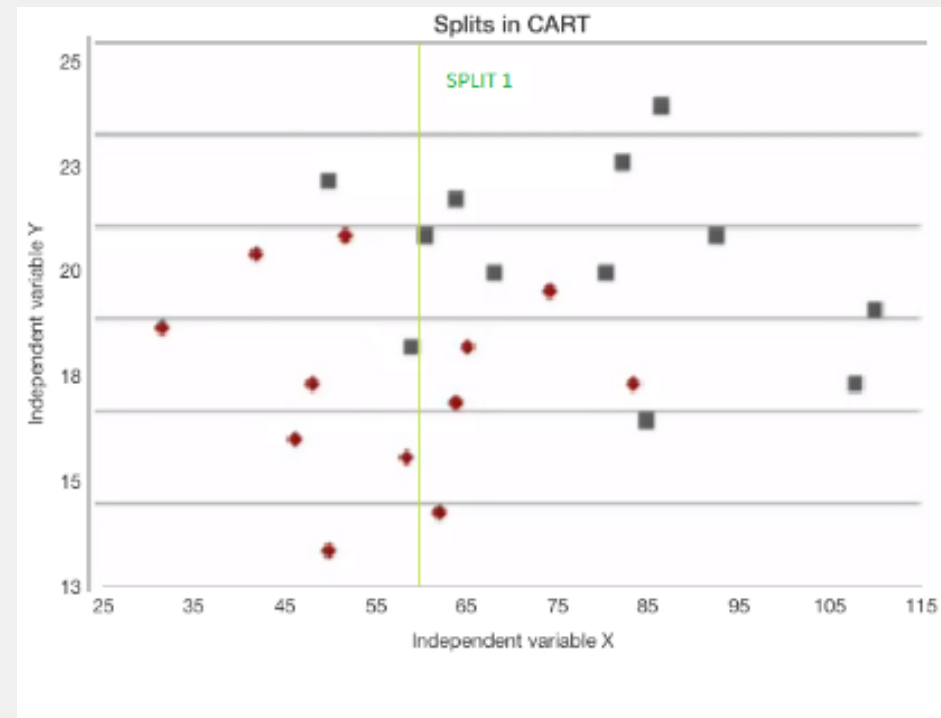
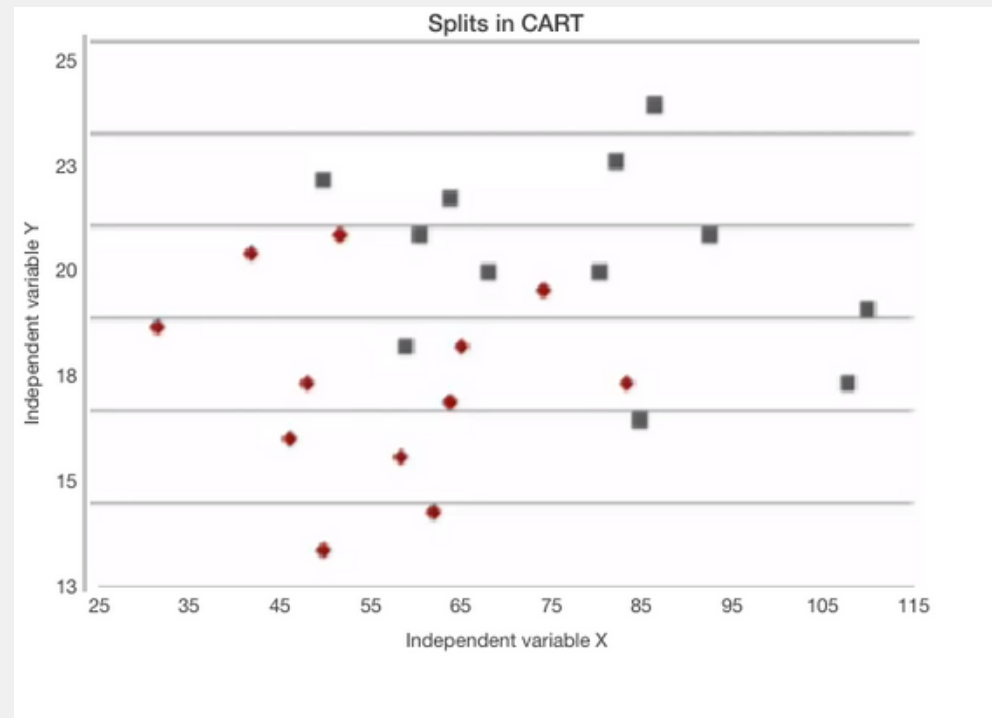
- Advantage: very easy to interpret
- Disadvantage: easy to overfit



Algorithm

- **repeatedly partitioning the data into multiple sub-spaces, so that the outcomes in each final sub-space is as homogeneous as possible**
- **recursive partitioning**
 - **Given a subset of training data, find the best feature for predicting the labels on that subset.**
 - **Find a split on that feature that best separates the labels, and split into two new subsets**
 - **Repeat steps one and two recursively until you meet a stopping criterion**

Example



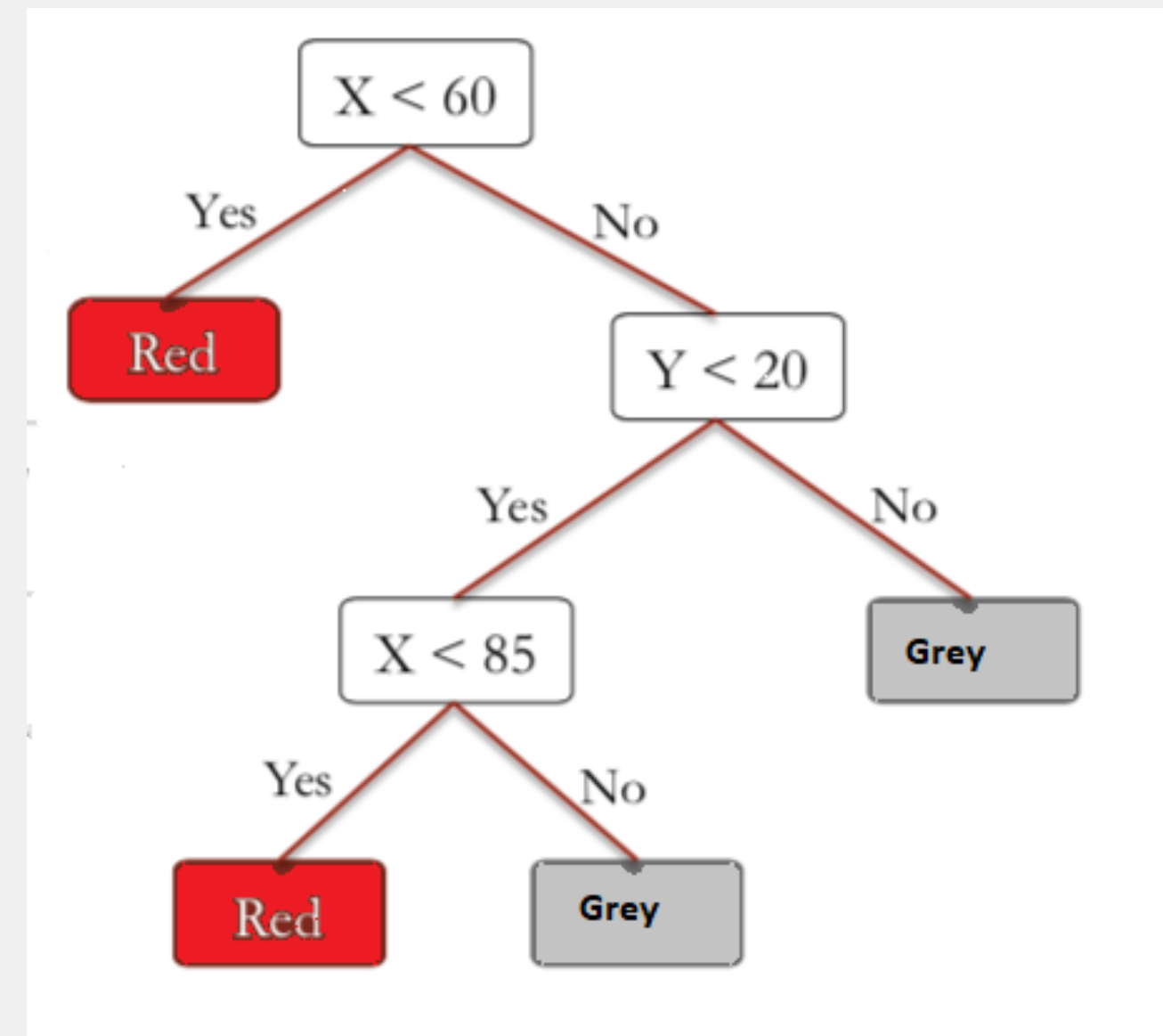
Ways of splitting:

- Gini
- Entropy
- Info Gain

Example

If a new observation fell into any of the subsets, it would now be decided by the majority of the observations in that particular subset

The tree for the splits looks like this:



Measuring Model Performance

As usual, depends on problem...

Accuracy: Good measure when target variable classes in the data are nearly balanced

Precision and Recall: If data is unbalanced

| | | Actual | |
|-----------|--------------|--------------|--------------|
| | | Positives(1) | Negatives(0) |
| Predicted | Positives(1) | TP | FP |
| | Negatives(0) | FN | TN |

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

| | | Actual | |
|-----------|--------------|--------------|--------------|
| | | Positives(1) | Negatives(0) |
| Predicted | Positives(1) | TP | FP |
| | Negatives(0) | FN | TN |

$$\text{Precision} = \frac{TP}{TP + FP}$$

| | | Actual | |
|-----------|--------------|--------------|--------------|
| | | Positives(1) | Negatives(0) |
| Predicted | Positives(1) | TP | FP |
| | Negatives(0) | FN | TN |

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1 Score

Unpractical to carry around precision and recall every time we make a model for solving a classification problem

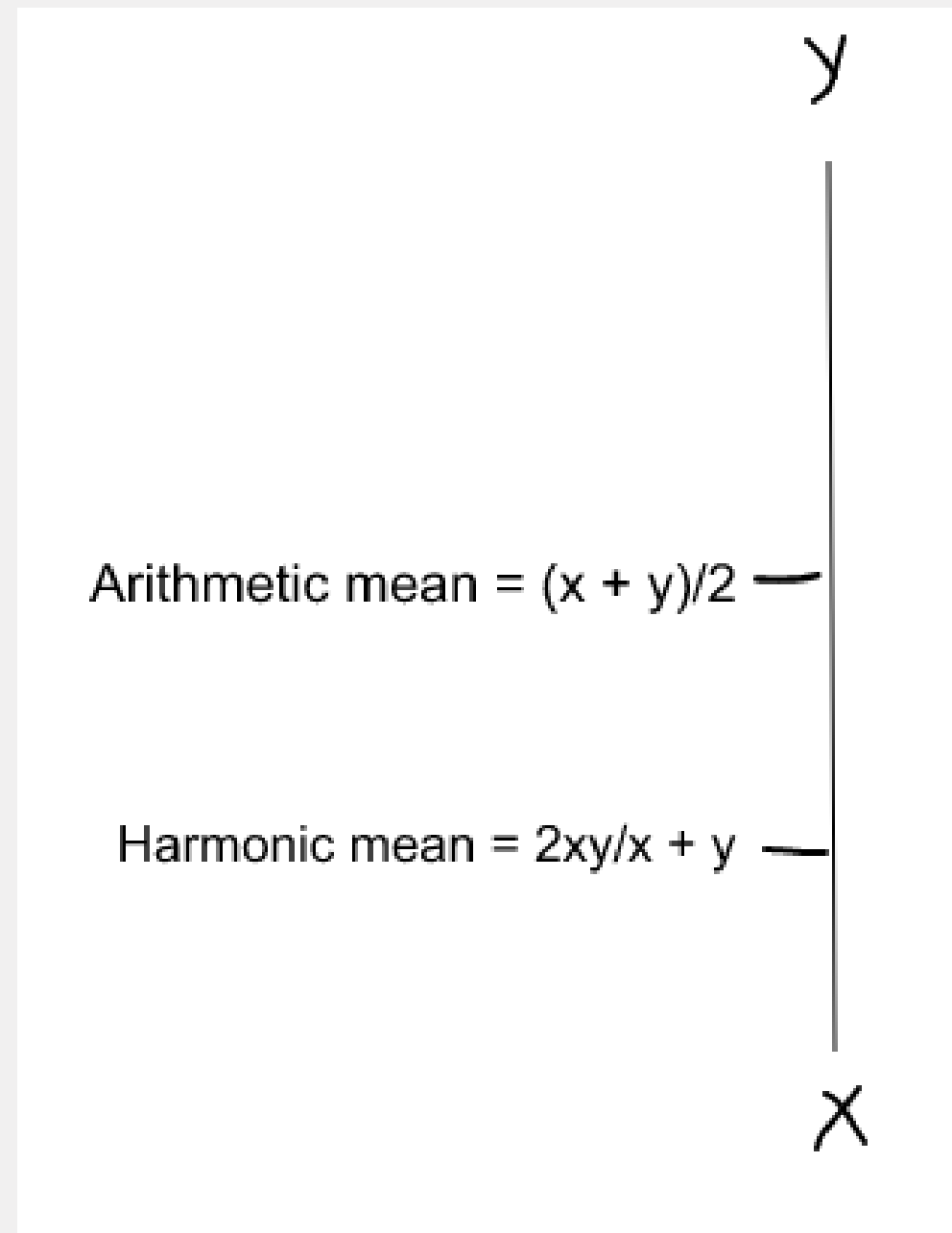
Solution: Single Score that represents Precision and Recall

Example: 100 credit card transactions: 97 legit, 3 fraud.
Model predicts everything as fraud.

F1 Score = $2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$

$$= 2 * 3 * 100 / 103 = 5\%$$

Harmonic Mean



Harmonic mean is kind of an average when x and y are equal. But when x and y are different, then it's closer to the smaller number as compared to the larger number.

For next week

I have decided to stick with what I know and have experience in and will show you how to host a chatbot on Facebook. We will train a bot using the chatterbot library and we will cover sentence similarity algorithms

If there is anything else you are particularly interested in, please tell me in [this Google Form](#) and I will make sure to compile a list of useful resources or maybe give another lecture series next term (because I am just looking for reason to distract me from writing my thesis...)