Implementing Sentiment Analysis with an ML-based Approach

Vitthal Srinivasan
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Represent a text snippet as a feature vector

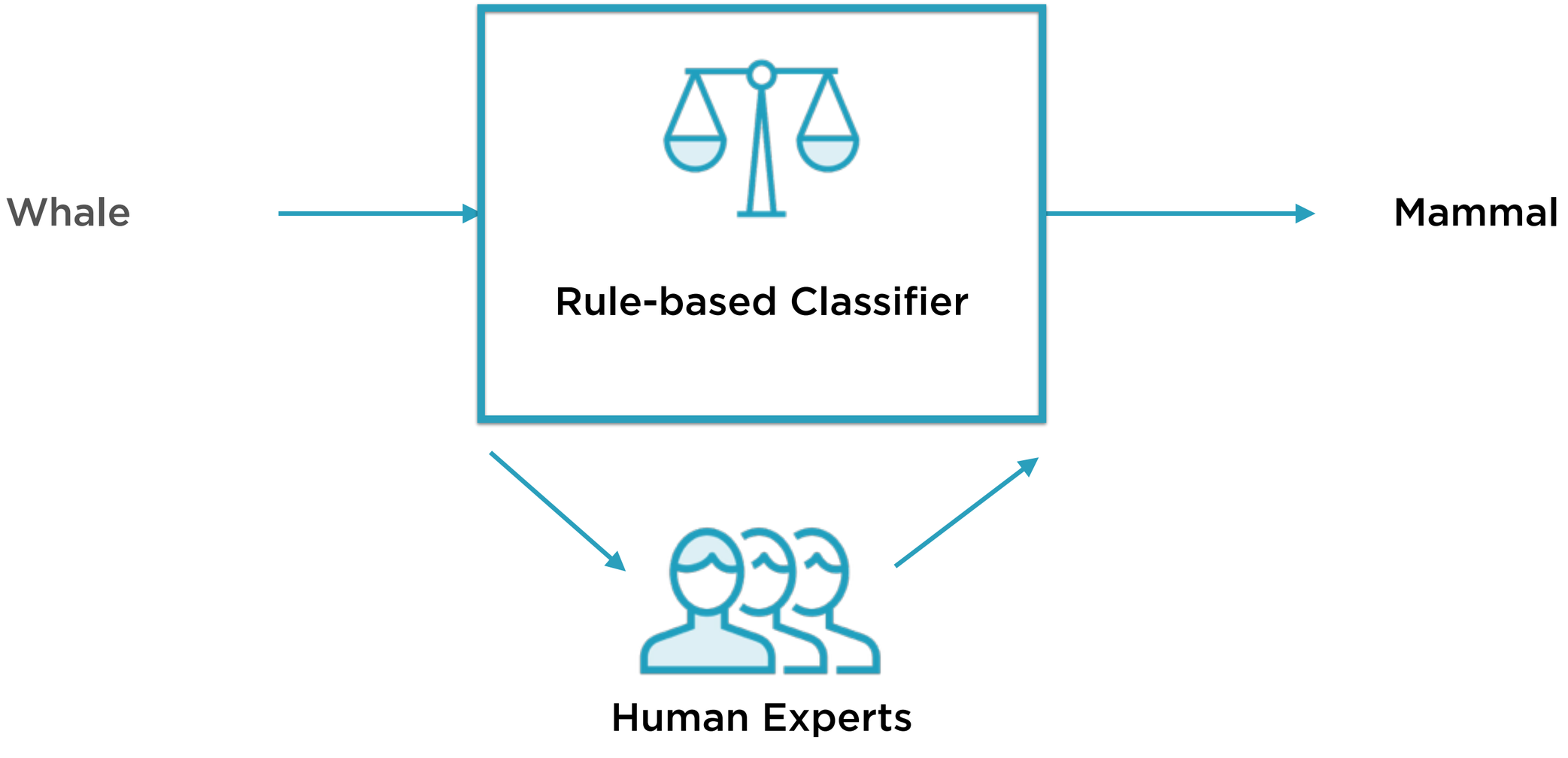
Train a Naive Bayes classifier and use it for review classification

Analyse a dataset of 10,000+ movie reviews using this classifier

Compare result with rule-based approaches

The Importance of Feature Extraction

Rule-based Binary Classifier



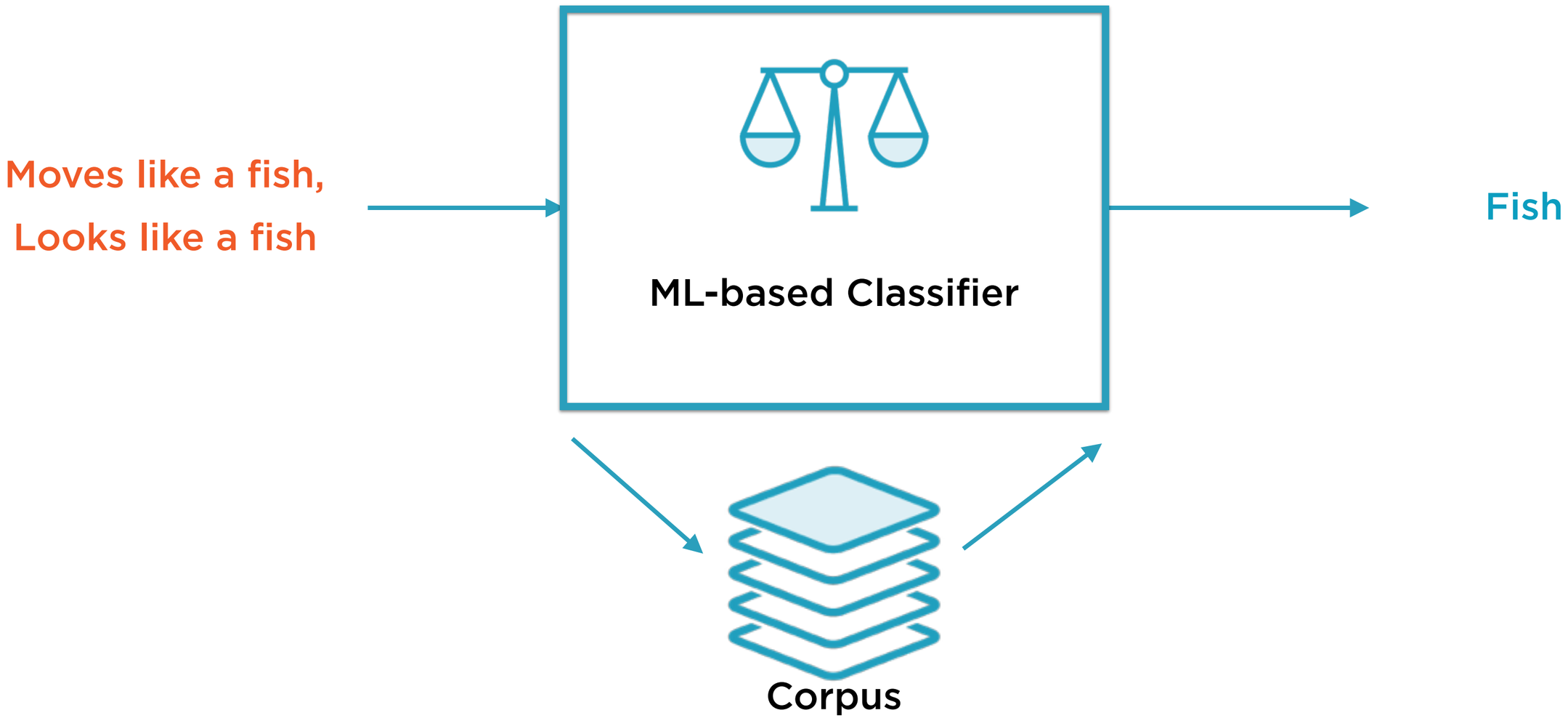
ML-based Binary Classifier

Corpus

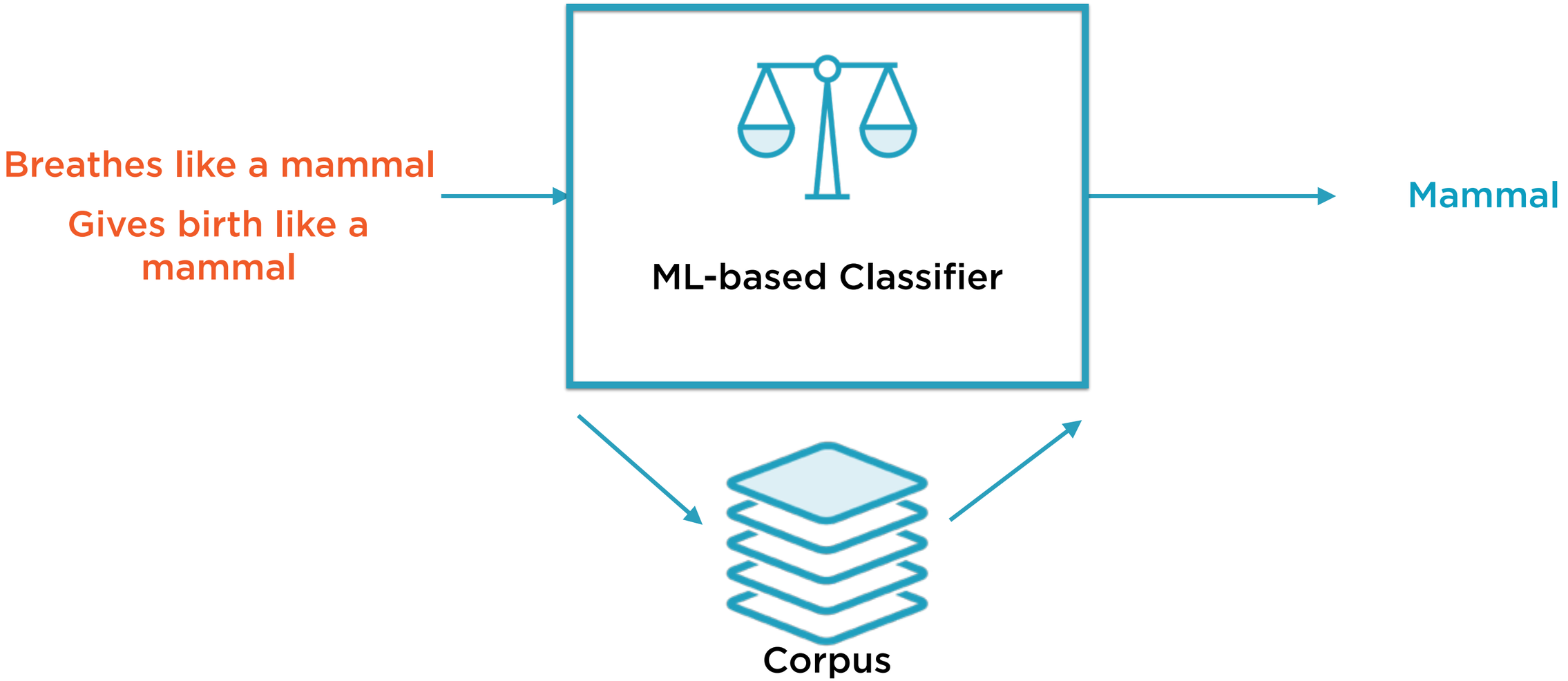
Classification Algorithm

ML-based Classifier

ML-based Binary Classifier



ML-based Binary Classifier



Smart feature selection is key to making ML-based approaches work

“This is not the worst restaurant in the metropolis, not by a long way”

(“This”, “is”,”not”,”the”,”worst”,”restaurant”,”in”,”the”, “metropolis”, “not”,”by”,”a”,”long”,”way”)

Feature Vector: Word Tuple

Presence or absence of a word in a review works fine for Naive Bayes classifiers

Word Tuples

Reviews	Labels
Amazing!	Positive
Worst movie ever	Negative
Two thumbs up	Positive
Part 2 was bad, 3 the worst	Negative
Up there with the greats	Positive

Simply express a document as a tuple of words

Word Tuples

Reviews

Amazing!
Worst movie ever
Two thumbs up
Part 2 was bad, 3 the worst
Up there with the greats

All Words

amazing
worst
movie
ever
two
thumbs
up
Part
was
bad
3
the
there
with
greats

Create a set of all words (all across the corpus)

Word Tuples

	Amazing!	Worst movie ever	Two thumbs up
amazing	1	0	0
worst	0	1	1
movie	0	1	1
ever	0	1	1
two	0	0	1
thumbs	0	0	1
up	0	0	1
Part	0	0	0
was	0	0	0
bad	0	0	0
3	0	0	0
the	0	0	0
there	0	0	0
with	0	0	0
greats	0	0	0

Express each review as a tuple of 1,0 elements

Word Tuples

Reviews	Feature Vector
Amazing!	(1,0,0,0,0,0,0 ...)
Worst movie ever	(0,1,1,1,0,0,0 ...)
Two thumbs up	(0,0,0,0,1,1,1 ...)
Part 2 was bad, 3 the worst	...
Up there with the greats	...

Now compare, measure distance using simple geometry

Feature Selection for Text Documents

Word tuples

Work fine for Naive Bayes classifiers

Term frequency (tf)

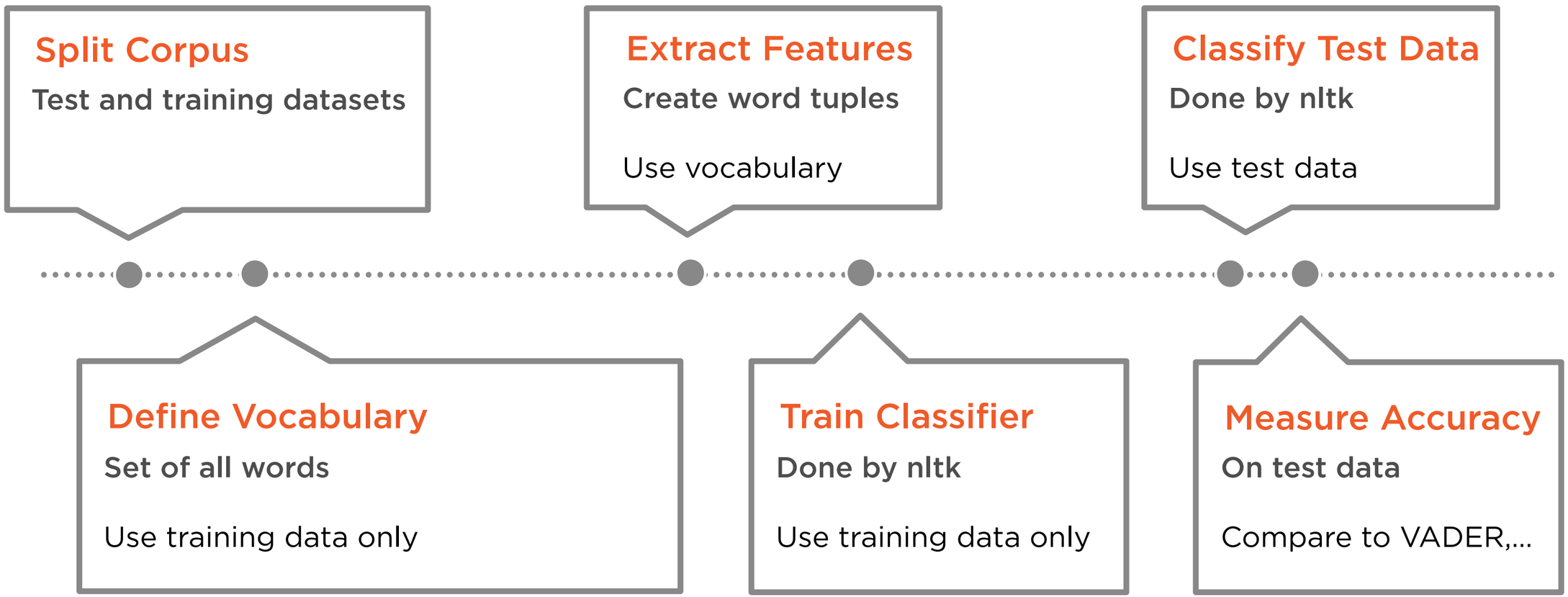
Capture frequency information; useful in SVM

Inverse document frequency(idf)

Unusual words assigned more importance

Implementing an ML-based Approach Using Naive Bayes

Naive Bayes on Cornell Movie Data

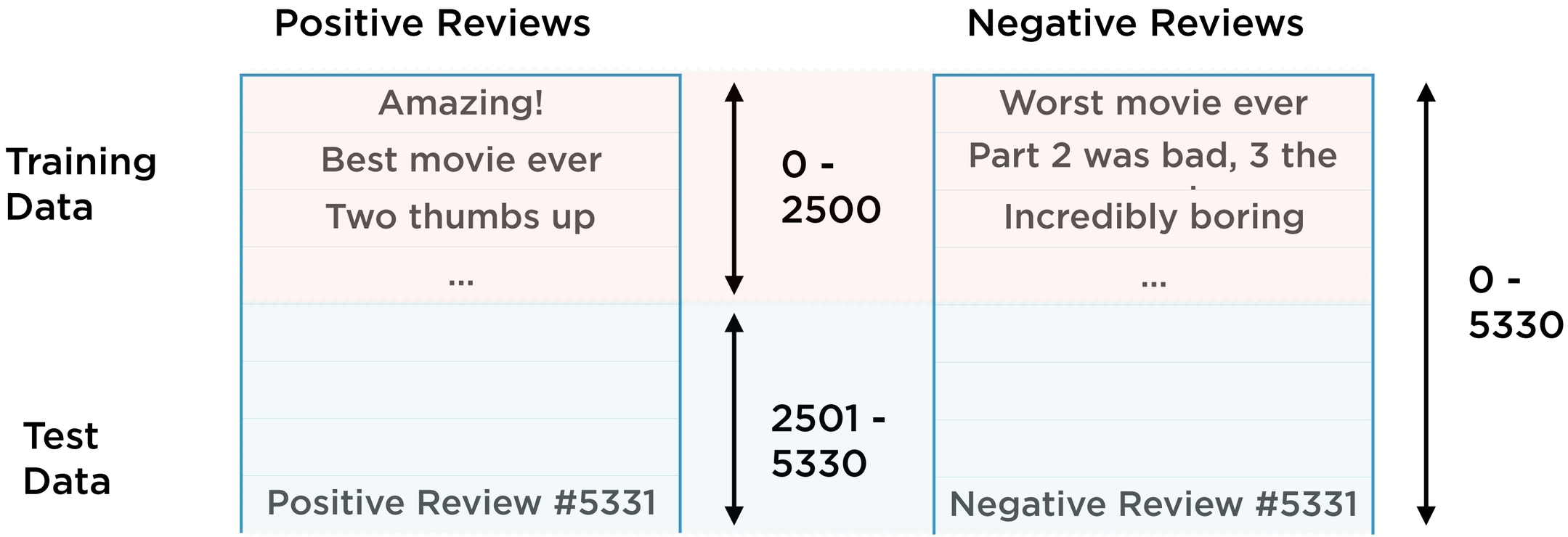


Naive Bayes on Cornell Movie Data

Split Corpus

Test and training datasets

Split Corpus into Test and Training Data



```
negativeReviewsFileName =
"/Users/vitthalsrinivasan/rt-polaritydata/rt-polaritydata/rt-polarity.neg"

with open(negativeReviewsFileName,'r') as f:
    negativeReviews = f.readlines()
```

Read in the Reviews

Exactly as before

```
testTrainingSplitIndex = 2500

testNegativeReviews = negativeReviews[testTrainingSplitIndex+1:]
testPositiveReviews = positiveReviews[testTrainingSplitIndex+1:]

trainingNegativeReviews = negativeReviews[:testTrainingSplitIndex]
trainingPositiveReviews = positiveReviews[:testTrainingSplitIndex]
```

Split Corpus into Training and Test Data

Use rows 0 to 2500 to train our classifier, then test it with rows 2501 to 5330

Naive Bayes on Cornell Movie Data

Split Corpus

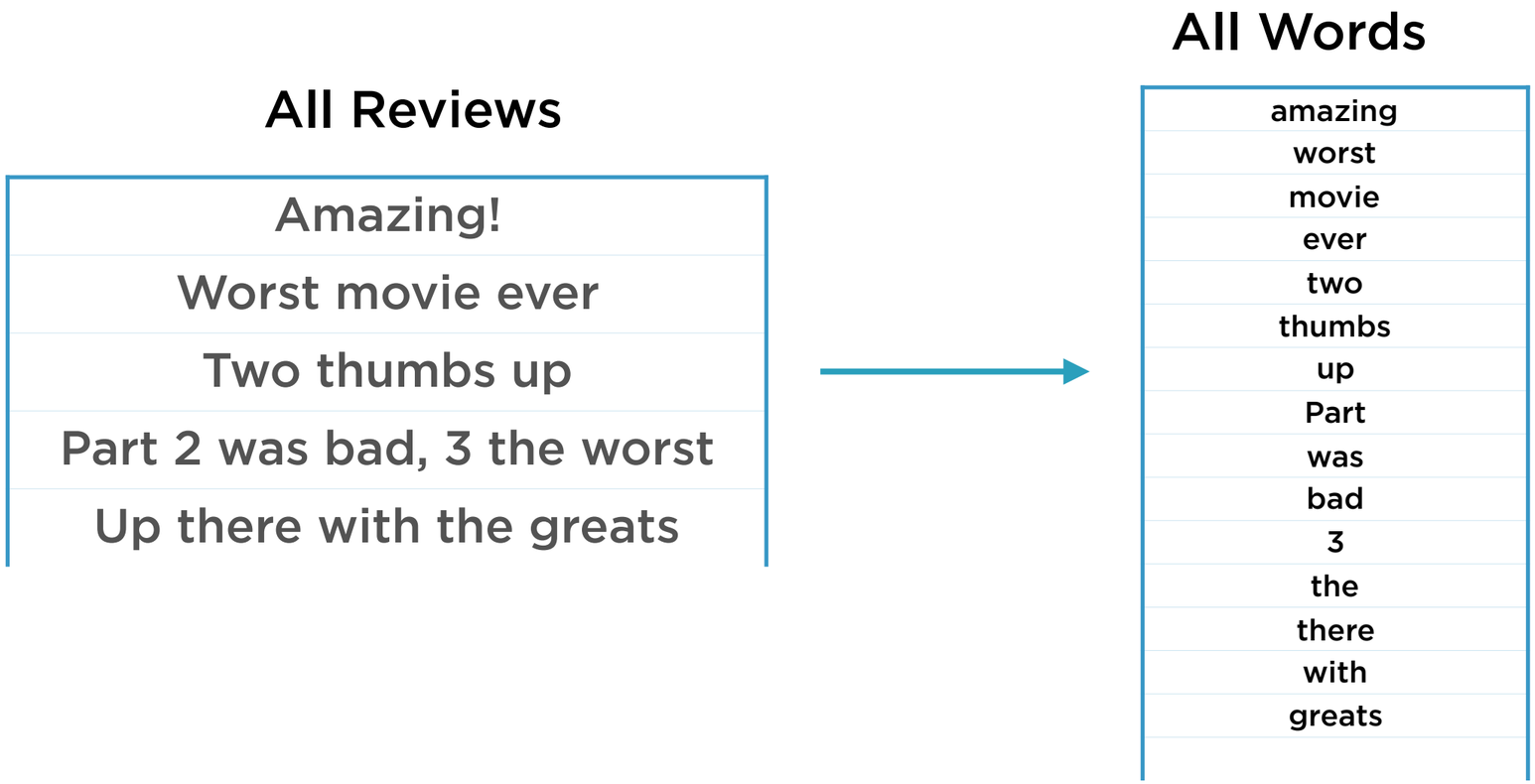
Test and training datasets

Define Vocabulary

Set of all words

Use training data only

Define Vocabulary of Training Data



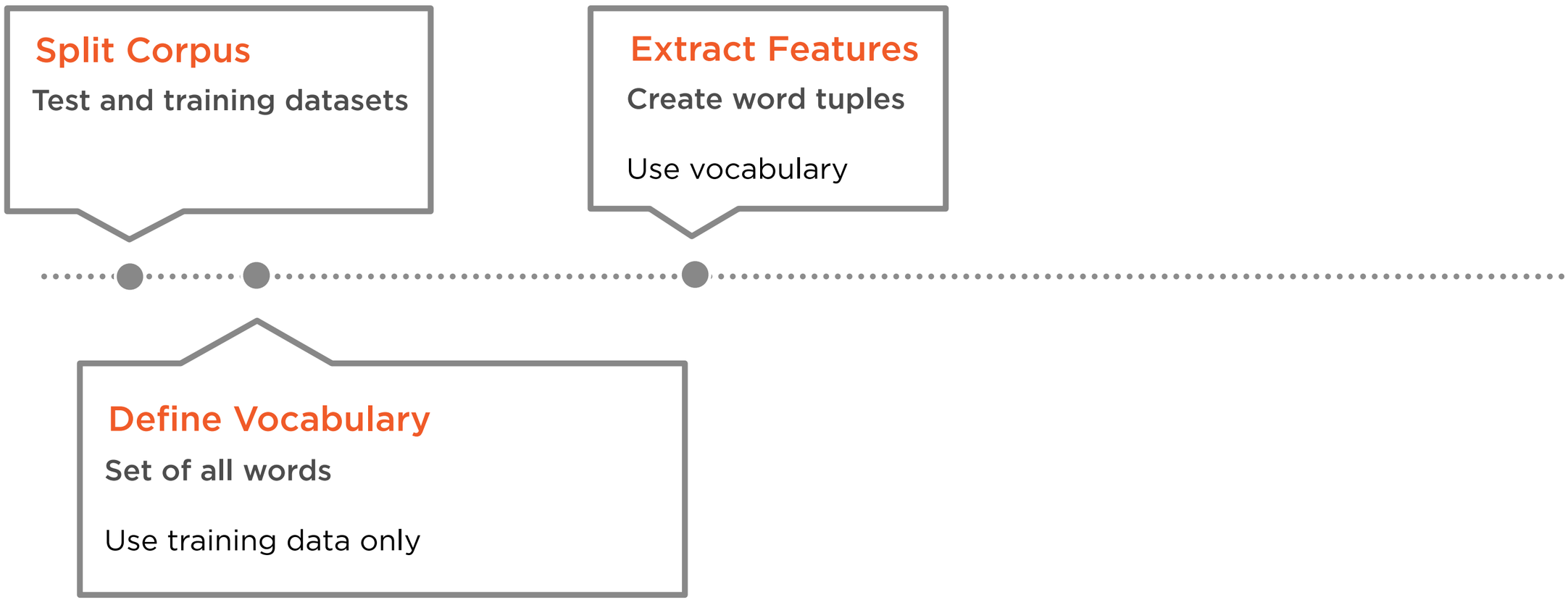
Create a set of all words (in the training data)

```
positiveWordList = [word for line in trainingPositiveReviews for word in line.split()]
negativeWordList = [word for line in trainingNegativeReviews for word in line.split()]
allWordList = [item for sublist in [positiveWordList,negativeWordList] for item in sublist]
vocabulary = list(set(allWordList))
```

Define Vocabulary of Training Data

A list in which each word in the training data occurs exactly once

Naive Bayes on Cornell Movie Data



Extract Features

	Amazing!	Worst movie ever	Two thumbs up
amazing	1	0	0
worst	0	1	1
movie	0	1	1
ever	0	1	1
two	0	0	1
thumbs	0	0	1
up	0	0	1
Part	0	0	0
was	0	0	0
bad	0	0	0
3	0	0	0
the	0	0	0
there	0	0	0
with	0	0	0
greats	0	0	0

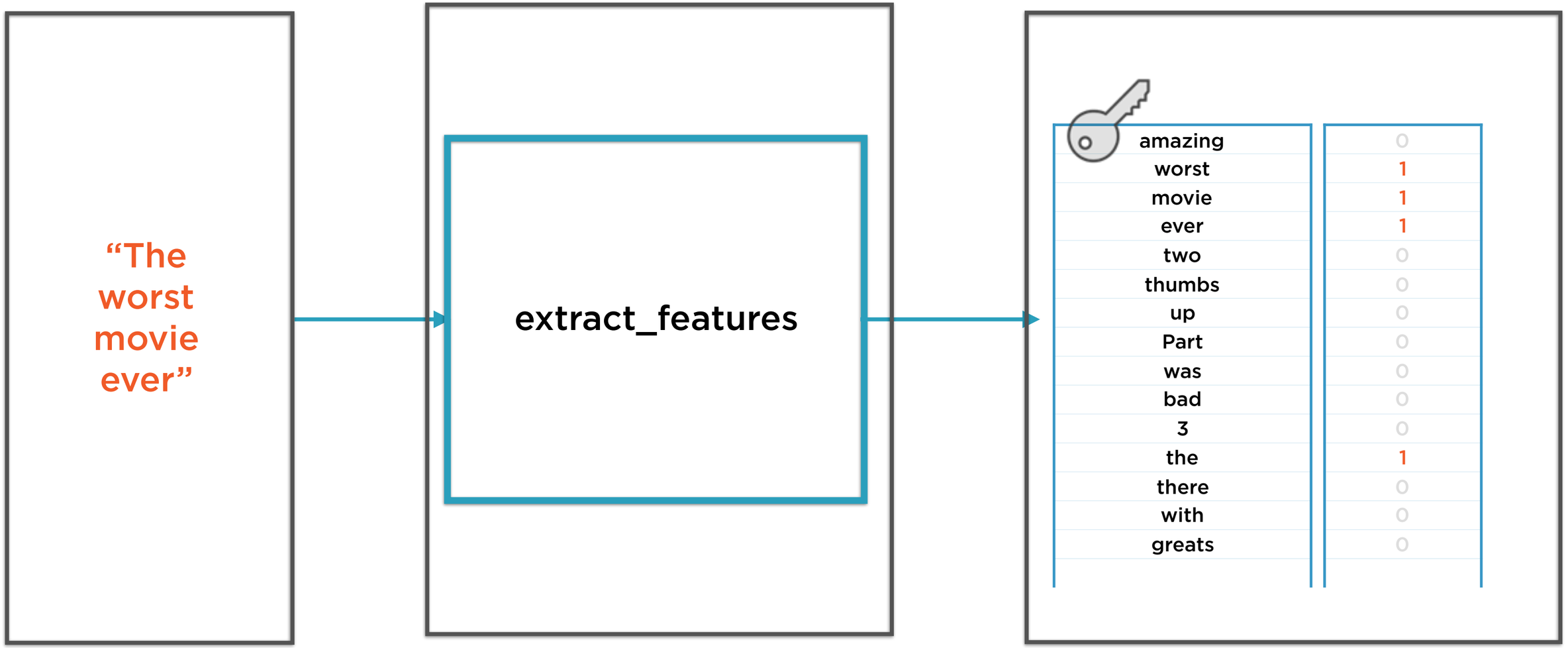
Express each review as a tuple of 1,0 elements

```
def extract_features(review):
  review_words=set(review)
  features={}
  for word in vocabulary:
      features[word]=(word in review_words)
  return features
```

Feature Extraction

A function that takes in a review and returns a feature vector

Feature Extraction



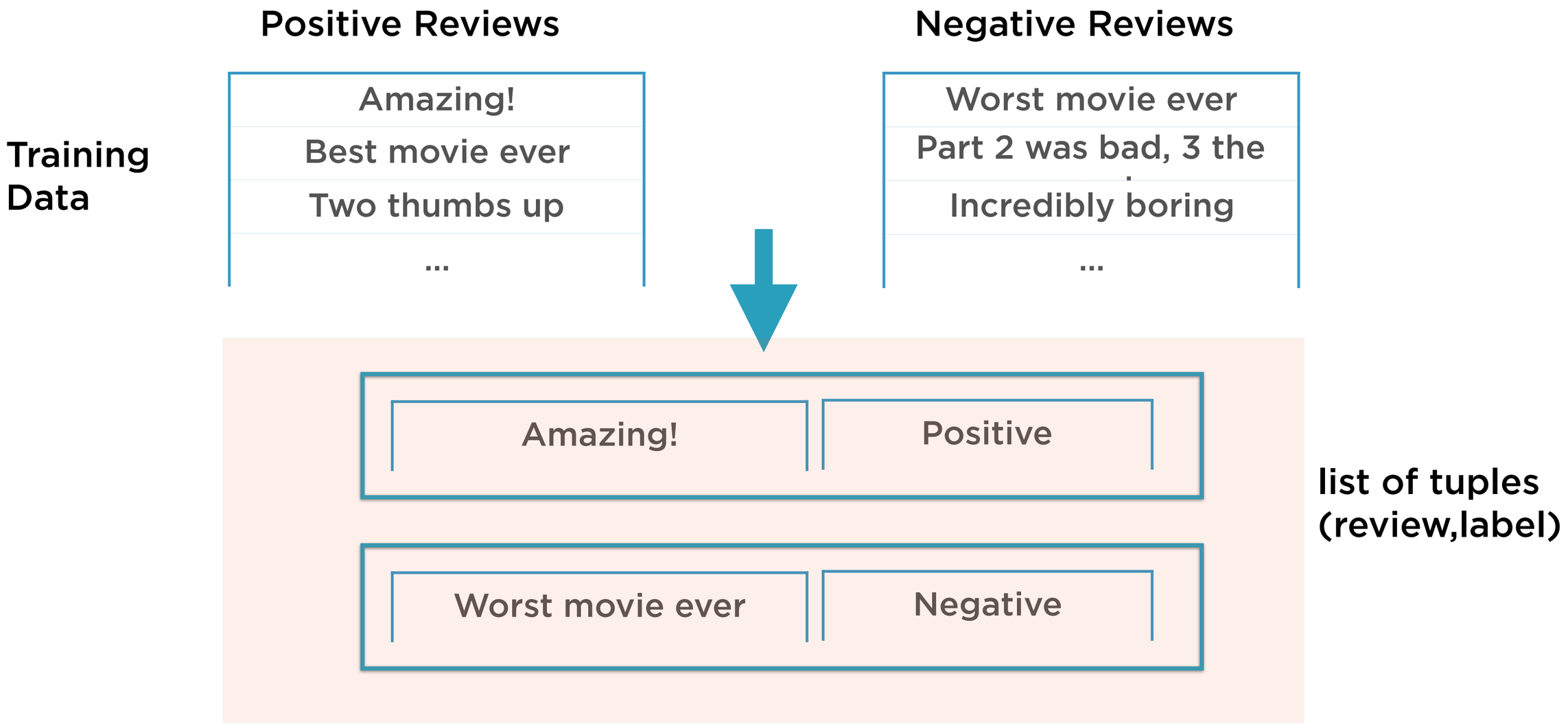
Transforming Data to Input into nltk

```
negTaggedTrainingReviewList = [{'review':oneReview.split(),'label':'negative'} for oneReview in trainingNegativeReviews]
posTaggedTrainingReviewList = [{'review':oneReview.split(),'label':'positive'} for oneReview in trainingPositiveReviews]
fullTaggedTrainingData = [item for sublist in [negTaggedTrainingReviewList,posTaggedTrainingReviewList] for item in sublist]
trainingData = [(review['review'],review['label']) for review in fullTaggedTrainingData]
```

Setting up the Training Data

A list of tuples: first element in each tuple is the review, second element is the label

Split Corpus into Test and Training Data



Feature Vectors and Labels

Amazing!	Positive
Worst movie ever	Negative

list of tuples
(review,label)

Feature Vector	Labels
(1,0,0,0,0,0,0 ...)	Positive
(0,1,1,1,0,0,0 ...)	Negative
(0,0,0,0,1,1,1 ...)	Positive
...	...
...	...

Feature Vectors and Labels

Reviews	Feature Vector	Labels
Amazing!	(1,0,0,0,0,0,0 ...)	Positive
Worst movie ever	(0,1,1,1,0,0,0 ...)	Negative
Two thumbs up	(0,0,0,0,1,1,1 ...)	Positive
Part 2 was bad, 3 the worst	...	...
Up there with the greats	...	...

The Naive Bayes classifier needs only feature vectors and labels to be trained

Feature Vectors and Labels

Feature Vector	Labels
(1,0,0,0,0,0,0 ...)	Positive
(0,1,1,1,0,0,0 ...)	Negative
(0,0,0,0,1,1,1 ...)	Positive
...	...
...	...

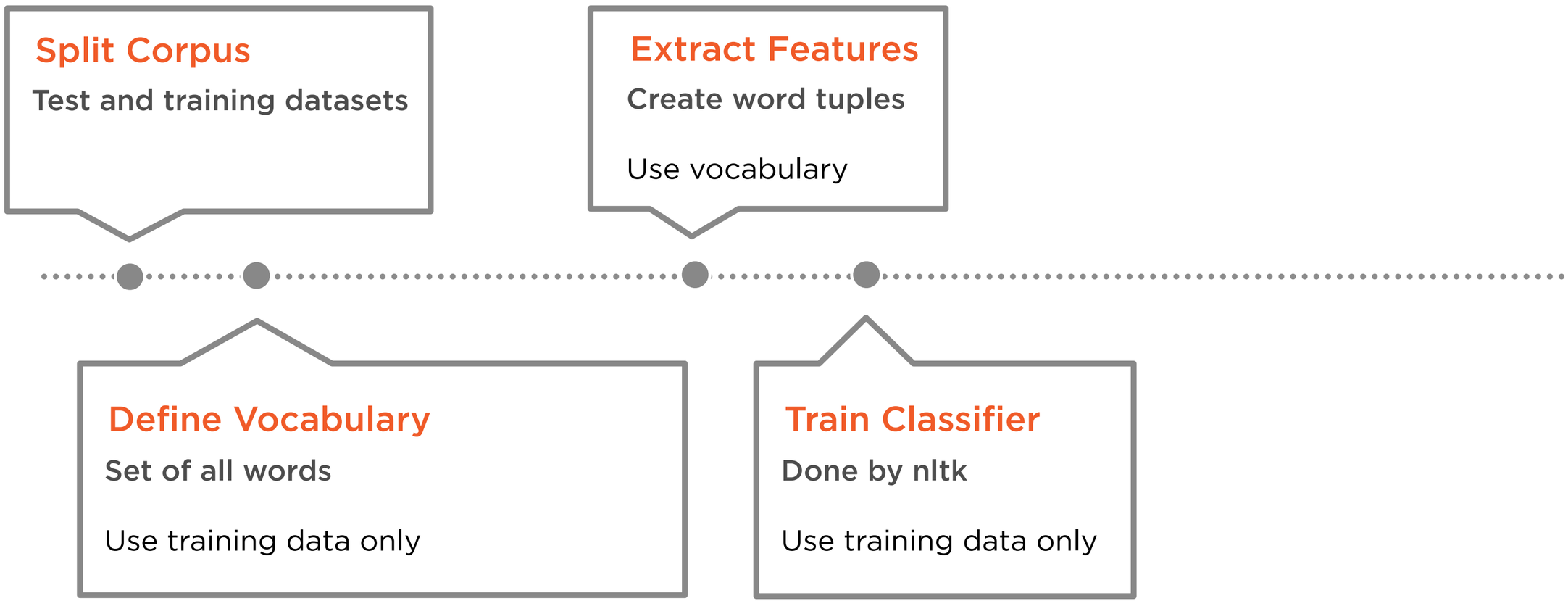
The Naive Bayes classifier needs only feature vectors and labels to be trained

```
trainingFeatures =
nltk.classify.apply_features(extract_features, trainingData)
```

Feature Extraction: nltk Takes Charge

Input training data and a function object, output is in correct feature vector form

Naive Bayes on Cornell Movie Data



Train Classifier

Training Data

Classification Algorithm

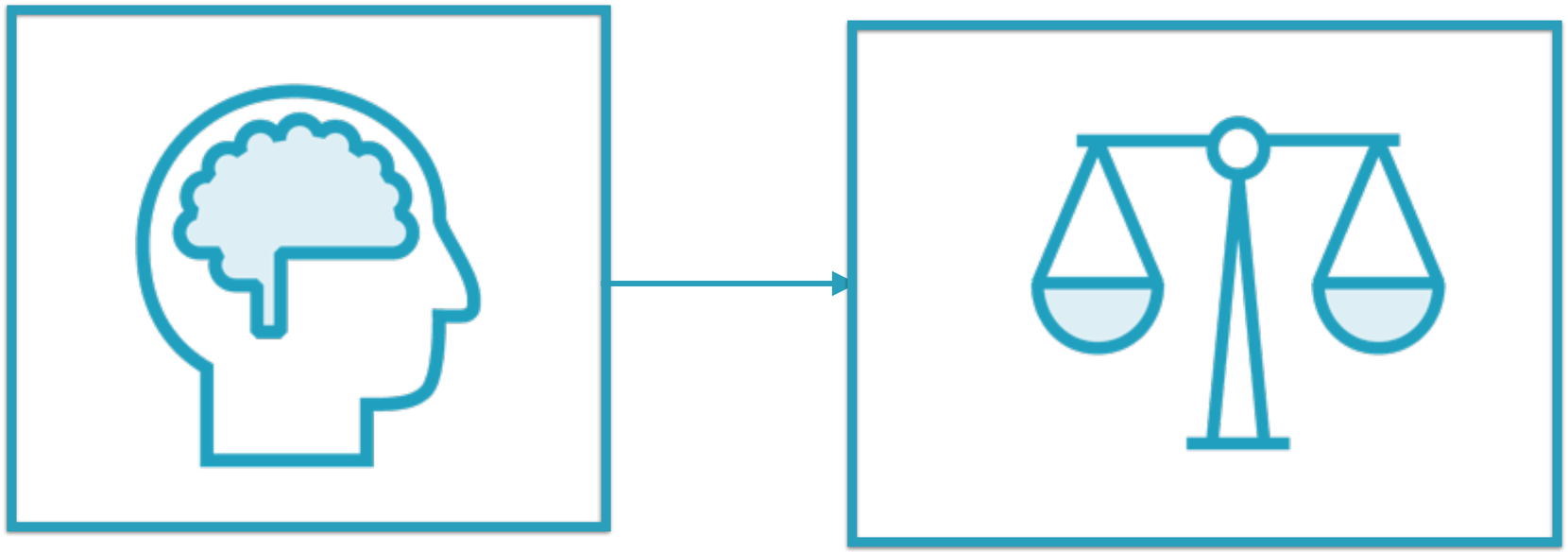
ML-based Classifier

Feature Vectors and Labels

Feature Vector	Labels
(1,0,0,0,0,0,0 ...)	Positive
(0,1,1,1,0,0,0 ...)	Negative
(0,0,0,0,1,1,1 ...)	Positive
...	...
...	...

Classification Algorithm

Feature Vectors and Labels



Classification Algorithm

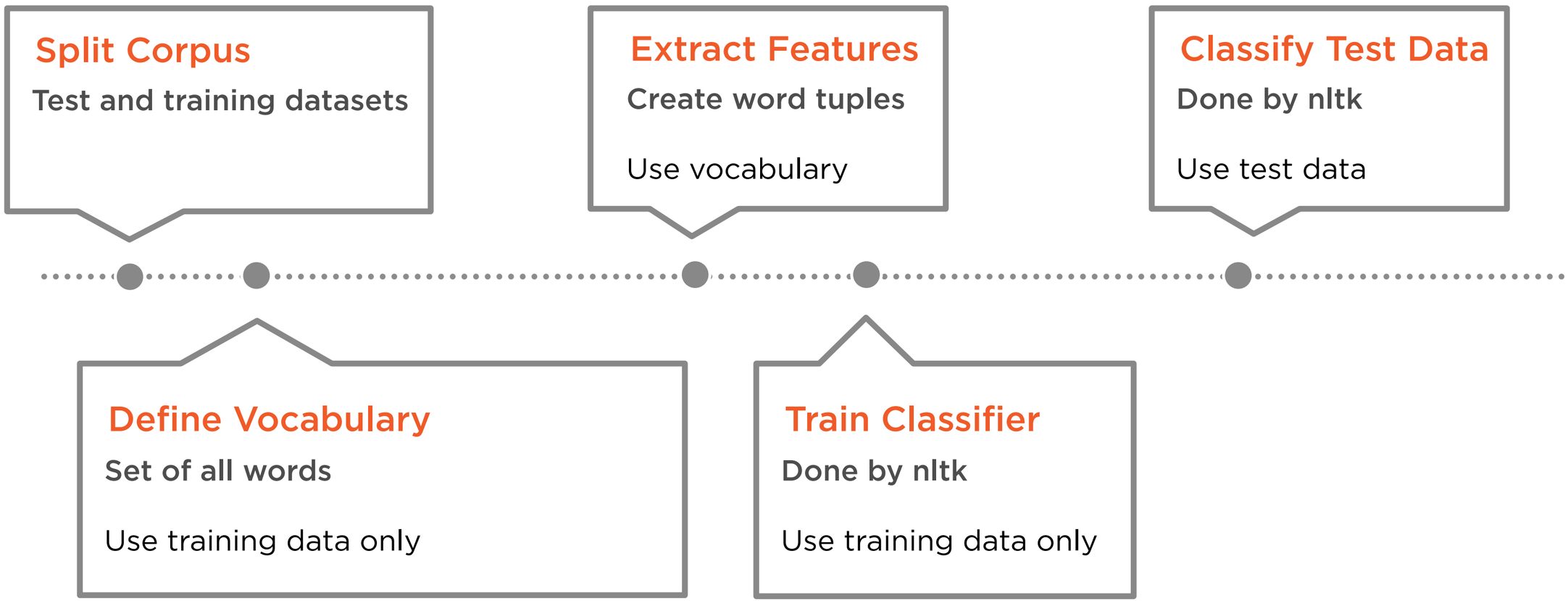
ML-based Classifier

```
trainedNBClassifier=
nltk.NaiveBayesClassifier.train(trainingFeatures)
```

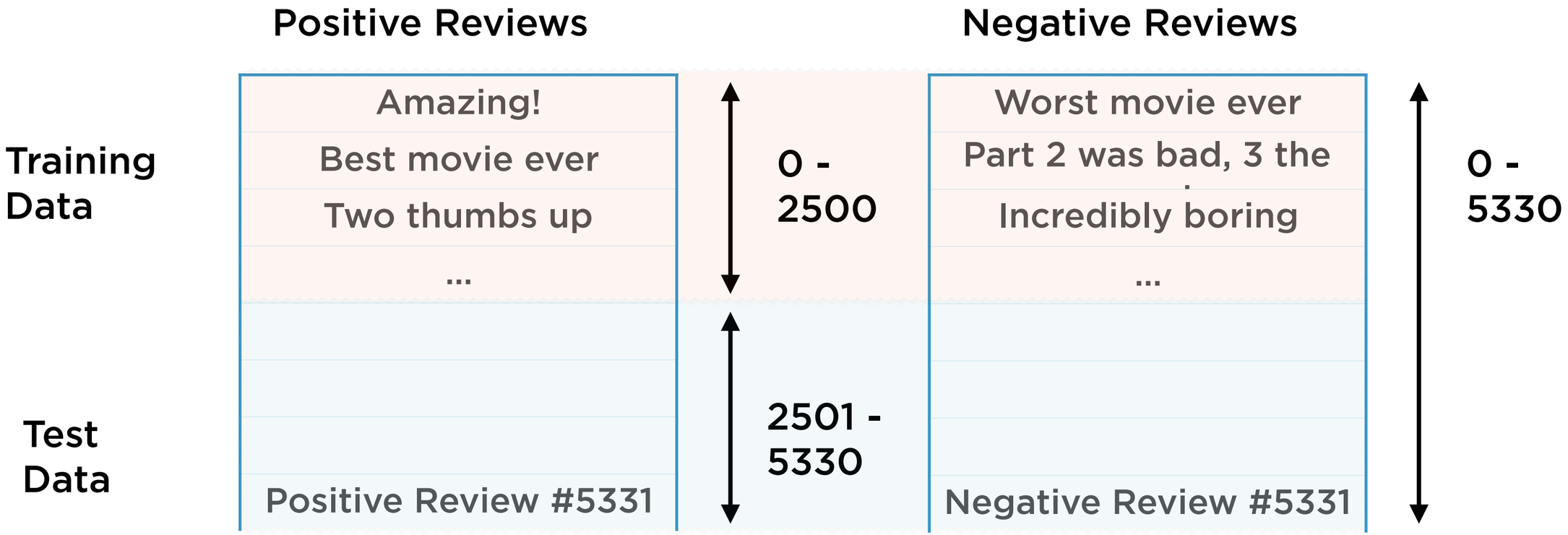
Training the Classifier: nltk Takes Charge

Input feature vectors and labels, output is a ready-to-use classifier

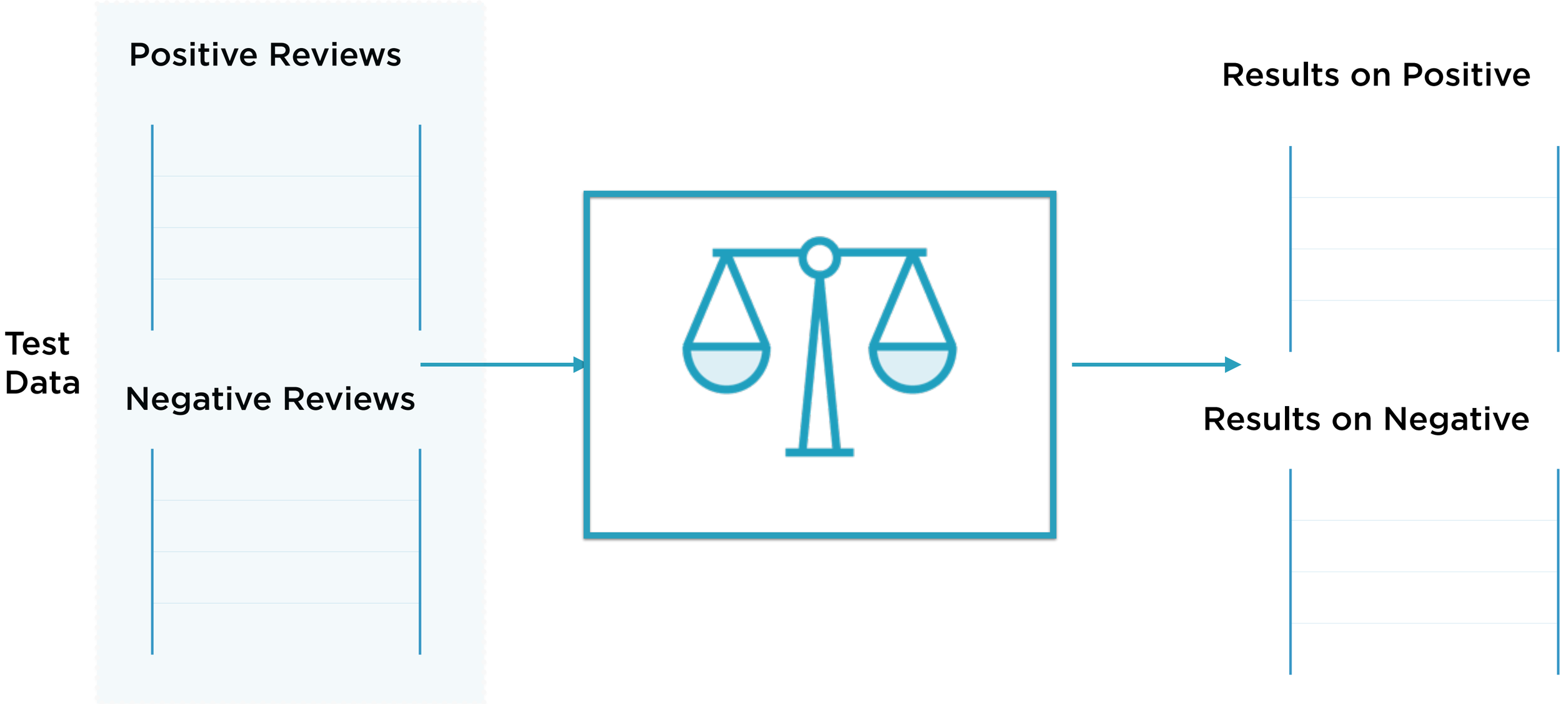
Naive Bayes on Cornell Movie Data



Split Corpus into Test and Training Data



Split Corpus into Test and Training Data



```
trainedNBClassifier.classify(problemFeatures)
```

Using the Classifier

Input a feature vector, output is an assigned label

```
def naiveBayesSentimentCalculator(review):
  problemInstance = review.split()
  problemFeatures = extract_features(problemInstance)
  return trainedNBClassifier.classify(problemFeatures)
```

Using the Classifier

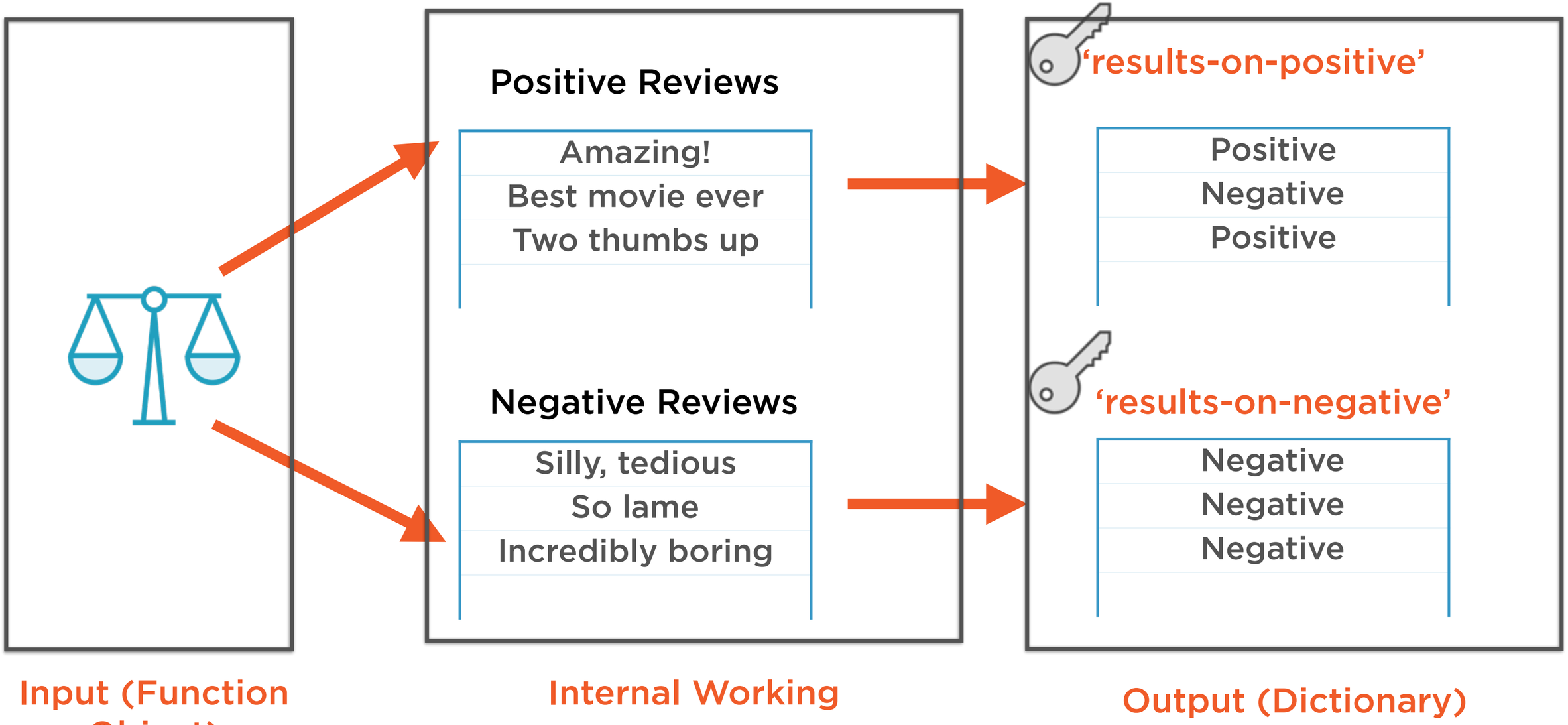
Create a simple wrapper function to abstract the details

```
def getTestReviewSentiments(naiveBayesSentimentCalculator):
  testNegResults = [naiveBayesSentimentCalculator(review) for review in testNegativeReviews]
  testPosResults = [naiveBayesSentimentCalculator(review) for review in testPositiveReviews]
  labelToNum = {'positive':1,'negative':-1}
  numericNegResults = [labelToNum[x] for x in testNegResults]
  numericPosResults = [labelToNum[x] for x in testPosResults]
  return {'results-on-positive':numericPosResults, 'results-on-negative':numericNegResults}
```

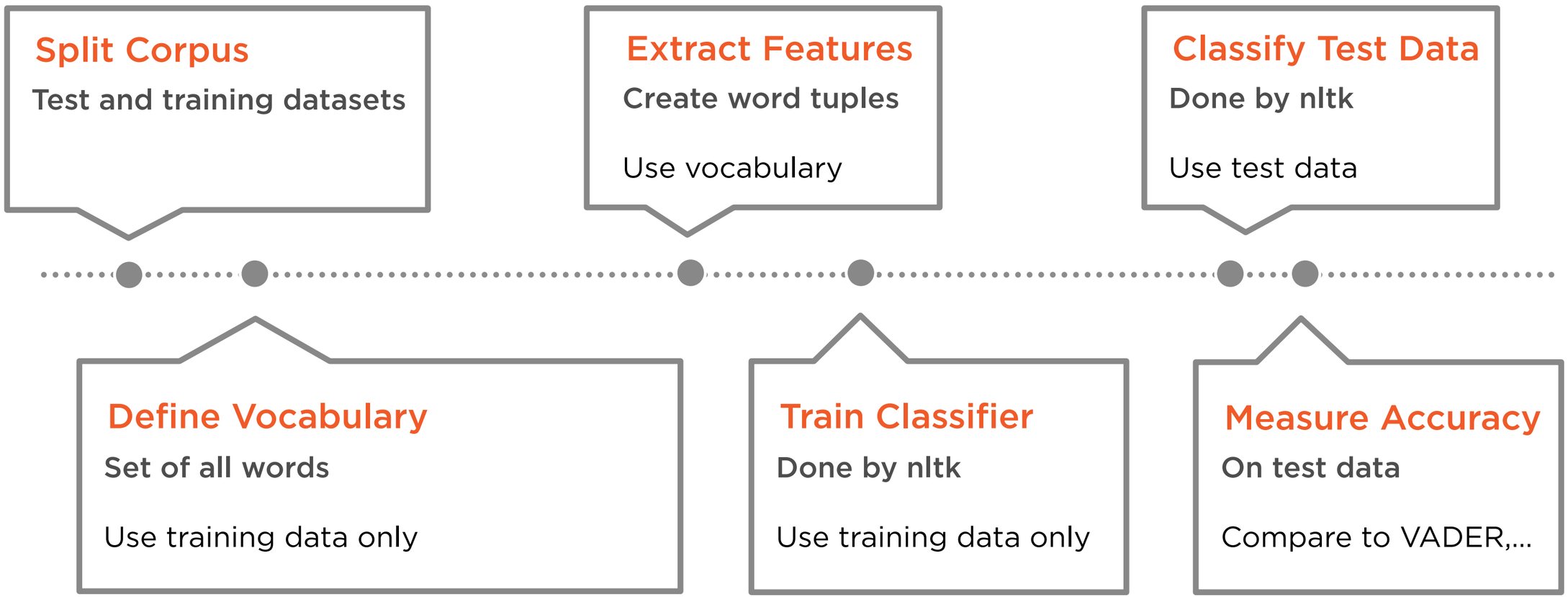
Invoke Using a Test Harness

Facilitates code reuse, as before

Code Reuse


Positive Reviews
Amazing!
Best movie ever
Two thumbs up
Negative Reviews
Silly, tedious
So lame
Incredibly boring
'results-on-positive'
Positive
Negative
Positive
'results-on-negative'
Negative
Negative
Negative
Input (Function Object)
Internal Working
Output (Dictionary)

Naive Bayes on Cornell Movie Data



% of positive reviews that classifier classified correctly

```
pctTruePositive =
float(sum(x > 0 for x in positiveReviewsResult))/len(positiveReviewsResult)
```

Calculate Accuracy on Positive Reviews

Percentage of positive reviews that VADER classified as positive (correctly)

Numerator = number of positive reviews where classifier assigned positive polarity

```
pctTruePositive =
float(sum(x > 0 for x in positiveReviewsResult))/len(positiveReviewsResult)
```

Calculate Accuracy on Positive Reviews

Percentage of positive reviews that VADER classified as positive (correctly)

Denominator = total number of positive reviews

```
pctTruePositive =
float(sum(x > 0 for x in positiveReviewsResult))/len(positiveReviewsResult)
```

Calculate Accuracy on Positive Reviews

Percentage of positive reviews that VADER classified as positive (correctly)

Demo

Use Naive Bayes to classify movie reviews

Comparing VADER, Sentiwordnet and Naive Bayes

Naive Bayes Wins on Overall Accuracy

Naive Bayes

75%

Sentiwordnet

60%

VADER

55%

Negative Reviews Only

Naive Bayes

77%

Sentiwordnet

43%

VADER

40%

Positive Reviews Only

Naive Bayes

74%

Sentiwordnet

76%

VADER

69%

Simple, Powerful and Robust

Naive Bayes

Easy to use, easy to understand, powerful

VADER

Easy to use, great for some uses, not for others

Sentiwordnet

Hard to use, but great as foundation

Summary

ML-based approaches differ in feature vector representations

Naive Bayes is easy to use in Python via nltk

It easily outperforms rule-based approaches in our application