NLTK Tutorial

Text Classification

Outline

Announcements (talk tomorrow, project questions?)

Review

Introduction

Using Classifiers

Labeled Texts

The Classifier Interface

Training Classifiers

Features

Feature Selection

Algorithms - back to decision trees

Review

Learning from Observations (Chapter 18, AIMA)

Learning Agents

Inductive Learning

- learn a function f from examples (x, f(x))
- consistent, realizable, simple, occam's razor, . . .

Decision Trees

Text Classification

Divides texts into categories

- news stories into topics

Single-Category Text Classificaiton

- predefined set of categories
- each text belongs to exactly one category
- choose the most likely category for a given text

What about our class project?

What kind of learning (from Ch. 18) is this?

Using Classifiers (Ch. 6)

Features

- which words are present
- the first word
- sentence length
- combining the feature lists

Training Corpus

Training the Classifier

Classifying New Texts

Example Task

Categorize sentences as

- statements
- imperatives (commands)
- questions

Why is this text classification?

What do we do first?

Features

Which aspects of a sentence are relevant to our classification task?

How do we get ideas for this?

Features

Which aspects of a sentence are relevant to our classification task?

How do we get ideas for this?

Examine a training corpus

Let's brainstorm...

Example Features (from Tutorial)

Which words are present in the sentence (bag of words)

The first word of the sentence

The length of the sentence

Using NLTK

Define 3 FDLists (Feature Detector Lists)

Combine them

See code examples

What next?

Training Corpus

Load and label

See code examples

What about our project?

What next?

Training the Classifier

Build a ClassifierTrainer, using the feature detector list

Use the ClassifierTrainer to train a new classifier, using the training corpus

What if we wanted to switch classification algorithms?

See code examples

What next?

Classifying New Texts

Use the classifier we built to classify new texts

See code examples

Now, the details!

Relationship to Other Tasks

Multi-category classification

Clustering

Tagging

Labeled Texts

LabeledText class represents categorized text types

Defined by nltk.classifier module

Obvious member functions, type/token distinction

See code examples

The Classifier Interface

`ClassifierI` for single category text classifiers

Requires two methods

- `classify` returns a labeled text token
- `labels` returns the list of category labels

See code examples

Training Classifers

Training Corpus (usually hand-classified, *assumed* mostly correct)

`ClassiferTrainerI` for building classifiers from training corporoa

- single method: `train`
- input: list of labeled tokens
- returns: new classifer

See code examples

Feature-Based Classification

A way of encoding information used to build classifiers, that provides domain/task independence

Each feature specifies some aspect of a `LabeledText` that is (hopefully) relevant to deciding how likely a label is for that text (classification)

- example feature: whether a document contains the word "ball" and has the label "sports"

Features are defined by *feature detector functions*

- map `LabeledTexts` to feature values
- see code example
- what are the values returned?

Feature Types

Boolean (Binary) Features

Integer Features

Others (not implemented in NLTK)

See code examples

Feature Detectors

`FeatureDetectorI` for implementing feature detector functions

- single method: `detect`
- input: labeled text
- returns: feature value

`FunctionFeatureDetector` is one provided implementation

- created from Python functions using the constructor
- `detect` can then be used

See code examples

Feature Detector Lists

Data structures that represent the feature detector functions for a set of features

- grouping mechanism for feature detectors
- associate unique identifiers with each feature detector
  - feature ids for a feature detector list with N features are $0, 1, \ldots, N\text{-}1$
  - unique within a feature detector list, not globally
- efficient implementation for related feature detectors

Feature Detector List Interface

`FeatureDetectorListI` for implementing feature detector lists

Four methods

- an indexing operator
- a length operator
- a `detect` method
- an addition operator

See code examples

Implementations of `FeatureDetectorListI`

`TextFunctionFDList` implements feature detector lists consisting of boolean features with detector functions of a specific form (see code)

- inputs: a function defined over texts, a list of function values, and a list of labels
- the list contains one feature detector for each (val,L) pair

See code examples

Implementations, continued

`BagOfWordsFDList` checks which words are present in text

- again, it implements a feature detector list containing boolean features whose detector functions have a specific form
- see code example
- inputs: a list of relevant words, and a list of labels
- the list contains one feature detector for each (w,L) pair

See code examples

Comparison

`TextFunctionFDList`

- function defined over texts, a list of function values, and a list of labels
- one feature detector for each (val,L) pair

`BagOfWordsFDList`

- a list of relevant words, and a list of labels
- one feature detector for each (w,L) pair

See reference documentation for other available implementations

Feature Value Lists

Data structures that represent the feature values for a set of features

- contrast with Feature Detector Lists: data structures that represent the feature detector *functions*
- created with `detect` method of a feature detector list
- the feature ids in the feature value list correspond with those in the feature detector list

FeatureValueListI

Four methods

- an indexing operator (old)
- a length operator (old)
- an `assignment` method
- a default method

`default` typically returns the majority value

`assignments` returns (feature-id, feature-value) pairs for the rest

See code examples

Feature Selection

It is often difficult for humans to decide which information is relevant for classification

An alternative is to use all and let the classification algorithm decide, but this has drawbacks (inefficient, over-fitting)

Another alternative is to use statistical techniques to decide which features are relevant, before constructing the classifier

Using Classifiers

Let's revisit how we started

For our purposes, we will view NLTK Classifier Model implementations (e.g., Naive Bayes) as black boxes

Instead we will return to decision trees