PROJECT

PATTERN CLASSIFICATION

Done During Internship

About company:

The **NYCE Group,** founded in 1997, represents a group of top quality of reputed professionals, offering the best services available in Indian financial service industry. The management of this group companies is open to change and all decisions are taken with the active participation of the experienced personnel with diverse talents. This has been the key factor in the emergence of the group as the leading financial service provider in Delhi. The group now looks forward to growing from strength to strength in the years to come.

We offer a diverse range of financial services which includes institutional and retail brokerage of equities, derivatives, commodities, currencies, online trading, ETFs, Global Indices, IPOs, Mutual Funds, financial planning, Principal Strategies and research. Our aim is to make the business process a smooth and a pleasant experience for all our clients. We believe in cultivating relationships that are of mutual benefit, both equitable and profitable for all parties concerned.

The flagship was started in the year 1997 in the name NYCE Pharma Pvt. Ltd. in Delhi. The resulting growth of business has put NYCE Pharma Pvt. Ltd. among the top echelons of both Pharmaceutical Companies and the Medical fraternity. The attitude of the management to keep an open mind to diversification in profitable fields of business and to keep pace with the needs of time the company diversified in 2004 with a new division of online commodity trading by becoming Trading Cum Clearing member of Multi Commodity Exchange of India Ltd. (MCX). Further we have successfully entered and consolidated our presence in the financial market by becoming the member of National Stock Exchange of India Ltd. (NSE), MCX Stock Exchange (MCX-SX) and United Stock Exchange (USE). The NYCE Group now comprises of **NYCE Securities & Derivatives Ltd.** and **NYCE Commodities & Derivatives Ltd.**

Enriched with a long and varied experience in the financial market, our extremely skilled and dedicated team of research experts has successfully advised our clients to reach the pinnacle of success. Our assets are our values, people and reputation. Our continued success depends upon unswerving adherence to these standards. We take great pride in having established a high degree of professionalism in our methodology of work. We possess uncompromising determination to achieve excellence in everything that we undertake. Though we may be involved in a wide variety and heavy volume of activities, we would rather opt to be the 'BEST' than the 'biggest'.

**VISION**

Our **Vision**We are extremely tech-savvy and have an innovative financial vision. Our business strategy plans and execute state-of-the-art trading in equities, futures, commodities, currencies and other derivatives to support and to deepen the group's trading expertise in depth and breadth.

Through continuous innovation and high-tech R&D we wish to become 'premier' financial and technological service provider, renowned for its expertise in Trading, Research based investment solutions, Cutting-edge technological know-how and most valued for its advice and flawless execution to provide best value for money to investors.

Though we may be involved in a wide variety and heavy volume of activities, we would rather opt to be the 'BEST' than the 'biggest'

**Our CRM Policy: Customer is King**

We are committed to providing world-class products and services which exceed the expectations of our customers, achieved by teamwork and a process of continuous improvement.

"A Customer is the most Important Visitor on our premises. He is not dependent on us, but we are dependent on him. He is not an interruption in our work. He is the purpose of it. He is not an outsider in our business. He is part of it. We are not doing him a favour by serving him. He is doing us a favour by giving us an opportunity to do so."   
- Mahatma Gandhi

**Our Buisness Model**

* Largely trading for our own group
* Specialist in Jobbing & Arbitrage
* Strict Risk Management
* Well-defined procedures of decision-making
* All actions based purely on in-house research
* Use of State-of-the-Art Technology and continuous up-gradations to match global standards
* Qualified & experienced personnel for operations
* Professionally managed group
* Business activity encompasses
* Ethical practices & transparency in all our dealings
* Customers interest above our own
* Always deliver what we promise

**RESEARCH FIELD:**

With the **NYCE (Nice Yielding Competent Efficient)** approach, our Research team offers timely Research reports covering investment summary, trend of world markets, sector trends, commodity trends. We have a team of highly experienced analysts, who cover stocks, commodity, currency and special reports.

Our teams of skilled financial professionals constantly monitor a whole gamut of investment opportunities in companies to cater the need of Investors, who are continuously in need of opportunities for striking rich rewards on their investment. We have one of the most advanced process and technology resources to provide complete research solutions. We offer proactive and timely world class research based advice and guidance to our clients so that they can take informed decisions.

Our investment philosophy focuses on how to maximize client's returns on the stocks bought/sold by suggesting exit/stop-loss or re-entry points in the respective stocks/positions, while keeping in mind their investment objectives, personal financial situation, time horizon and risk taking ability. We want our clients to take advantage of each and every tick movement. Our views are purely based on analysis and are independent, unbiased and balanced.

**NATURAL LANGUAGE PROCESSING:**

Natural language processing (NLP) is the ability of a computer program to understand human speech as it is spoken. NLP is a component of artificial intelligence (AI).

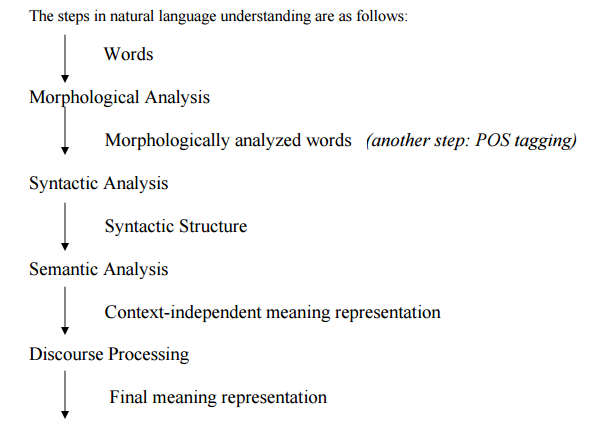
By “natural language” it means a language that is used for everyday communication by humans; languages such as English,

Hindi, or Portuguese. In contrast to artificial languages such as programming languages and mathematical notations, natural languages have evolved as they pass from generation to generation, and are hard to pin down with explicit rules.

Taking Natural Language Processing—or NLP for short—in a wide sense to cover any kind of computer manipulation of natural language. At one extreme, it could be as simple as counting word frequencies to compare different writing styles. At the other extreme, NLP involves “understanding” complete human utterances, at least to the extent of being able to give useful responses to them. Natural Language Processing is used everywhere—in search engines, spell checkers, mobile phones, computer games, and even in your washing machine. Technologies based on NLP are becoming increasingly widespread.

For example, phones and handheld computers support predictive text and handwriting recognition; web search engines give access to information locked up in unstructured text; machine translation allows us to retrieve texts written in Chinese and read them in Spanish. By providing more natural human-machine interfaces, and more sophisticated access to stored information, language processing society.[2]

NLP is important for scientific, economic, social, and cultural reasons. NLP is experiencing rapid growth as its theories and methods are deployed in a variety of new language technologies. For this reason it is important for a wide range of people to have a working knowledge of NLP. Within industry, this includes people in human-computer interaction, business information analysis, and web software development. Within academia, it includes people in areas from humanities computing and corpus linguistics through to computer science and artificial intelligence.



**PATTERN RECOGNITION:**

Pattern Recognition (PR) can be defined as “the act of taking in raw data and taking an action based on the category of the pattern” [16]. PR is heavily related to perception and, therefore, the most straightforward application of PR is the design and construction of systems able to imitate (to some extent) the human senses. The benefits obtained by the application of such systems are clear and huge. On the one hand, there are environments where using human beings is not possible or it is too risky (the outer space, the ocean depths, the inner earth, etc.). On the other, the productivity that can be achieved by a human operator is clearly limited. For that reason, in these extreme environments, or when a high throughput is required, automatic systems seem to be the only solution. In addition to these examples, PR can be useful to better understand how biological systems recognize patterns in nature. A typical PR process usually consists of three steps:

• Preprocessing: A signal or stimulus is captured from the real world. This stimulus can contain a set of patterns to be recognized along with some useless data. In this step, a segmentation process is usually carried out in order to separate the different patterns captured. In addition, the noise carried by the signal is removed or limited.

• Feature extraction: Once the input is segmented and clean, the relevant information is extracted. The point here is to achieve a suitable representation for the upcoming recognition process.

• Recognition: The final step consists in interpreting the input pattern. From all the ways in which a pattern can be characterized (for instance, by means of a complex linguistic description or by a set of nominal features), labeling the pattern as an instance of a class is, maybe, the most convenient way for an automatic processing. To this end, this label has to somehow summarize the relevant information included into the pattern. This classification can be performed on a previously 1 Chapter 1. Introduction defined set of classes (supervised classification) or, alternatively, can be intended to group the patterns into a set of unknown classes that will be discovered during the classification task itself (unsupervised classification). Nevertheless, regarding the recognition process as a mere classification task can turn out in a very constrained point of view.

In the case of Natural Language Processing (NLP), discussed in section1.2, it is more appropriate to consider the final PR step as an interpretation process. This step can be approached following different techniques. A deductive approximation could be used when the knowledge needed to perform the recognition is available and can be, as well, properly formalized and represented. However, this knowledge is normally not available or it is extremely vague or imprecise and, because of this, inductive techniques are often more appropriate here. These techniques are based on learning a model from a set of samples (training samples) that, somehow, captures the information needed to solve the problem. This model attempts to extract general patterns from the training samples in order to recognize future inputs. Statistical PR is one of the most representative examples of this inductive approximation. In this case, the model is actually a (or a set of) probability distribution that relates the possible inputs to the recognition outcomes. Here, each (properly represented) pattern can be seen as a point in a d-dimensional space which has to be scored according to this probability distribution. In the case of NLP, a pattern is actually defined by a set of relationships among these points.

**NATURAL LANGUAGE TOOLKIT(NLTK)**

NLTK is the Natural Language Toolkit, a comprehensive Python library for natural language processing and text analytics. Originally designed for teaching, it has been adopted in the industry for research and development due to its usefulness and breadth of coverage. Python's Natural Language Toolkit (NLTK) suite of libraries has rapidly emerged as one of the most efficient tools for Natural Language Processing.

NLTK was originally created in 2001 as part of a computational linguistics course in the Department of Computer and Information Science at the University of Pennsylvania. Since then it has been developed and expanded with the help of dozens of contributors. It has now been adopted in courses in dozens of universities, and serves as the basis of many research projects.

NLTK was designed with four primary goals in mind:

SIMPLICITY

To provide an intuitive framework along with substantial building blocks, giving users a practical knowledge of NLP without getting bogged down in the tedious house-keeping usually associated with processing annotated language data.

CONSISTENCY

To provide a uniform framework with consistent interfaces and data structures, and easily guessable method names.

EXTENSIBILITY

To provide a structure into which new software modules can be easily accommodated, including alternative implementations and competing approaches to the same task.

MODULARITY

To provide components that can be used independently without needing to understand the rest of the toolkit.

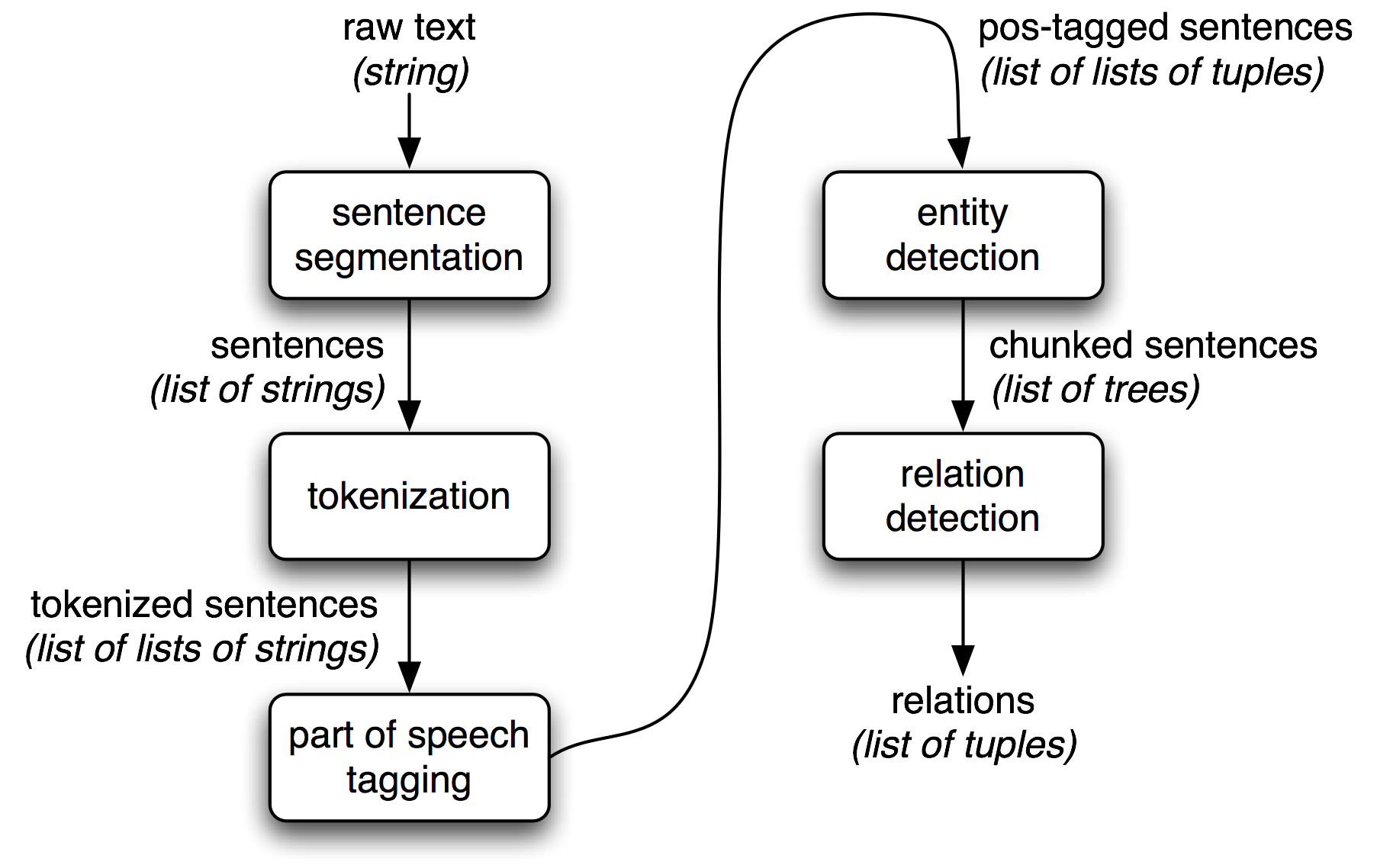
**Contrasting** with these goals are three non-requirements—potentially useful qualities that we have deliberately avoided.

1.While the toolkit provides a wide range of functions, it is not encyclopedic; it is a toolkit, not a system, and it will continue to evolve with the field of NLP.

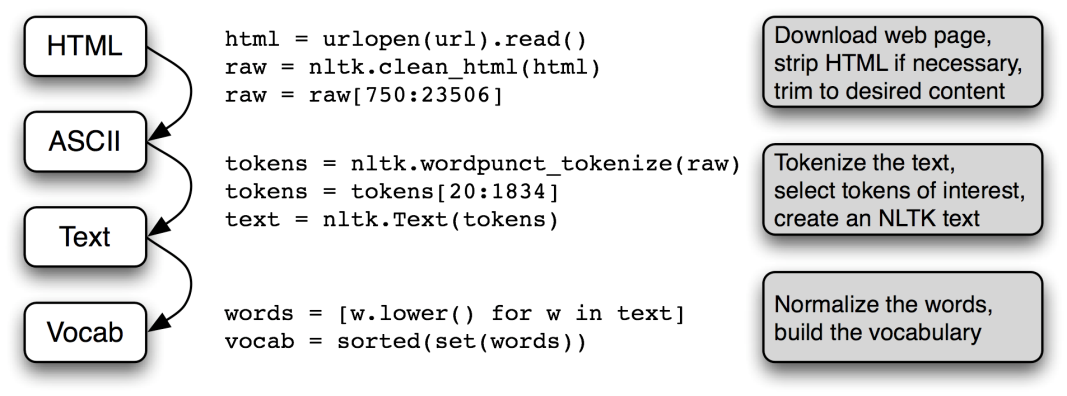
**2.**While the toolkit is efficient enough to support meaningful tasks, it is not highly optimized for runtime performance; such optimizations often involve more complex algorithms, or implementations in lower-level programming languages such as C or C++. This would make the software less readable and more difficult to install.

**3.**TRY to avoid clever programming tricks, as clear implementations are preferable to ingenious yet indecipherable ones.

ARCHiTECTURE -



### *The NLP Pipeline*



**CLASSIFIER:**

Classes and interfaces for labeling tokens with category labels (or “class labels”). Typically, labels are represented with strings (such as 'health' or 'sports'). Classifiers can be used to perform a wide range of classification tasks. For example, classifiers can be used...

* to classify documents by topic
* to classify ambiguous words by which word sense is intended
* to classify acoustic signals by which phoneme they represent
* to classify sentences by their author

### Features

In order to decide which category label is appropriate for a given token, classifiers examine one or more ‘features’ of the token. These “features” are typically chosen by hand, and indicate which aspects of the token are relevant to the classification decision. For example, a document classifier might use a separate feature for each word, recording how often that word occurred in the document.

### Featuresets

The features describing a token are encoded using a “featureset”, which is a dictionary that maps from “feature names” to “feature values”. Feature names are unique strings that indicate what aspect of the token is encoded by the feature. Examples include 'prevword', for a feature whose value is the previous word; and 'contains-word(library)' for a feature that is true when a document contains the word 'library'. Feature values are typically booleans, numbers, or strings, depending on which feature they describe.

Featuresets are typically constructed using a “feature detector” (also known as a “feature extractor”). A feature detector is a function that takes a token (and sometimes information about its context) as its input, and returns a featureset describing that token.

nltk.classify.megam module

A set of functions used to interface with the external [megam](http://www.umiacs.umd.edu/~hal/megam/index.html) maxent optimization package. Before megam can be used, you should tell NLTK where it can find the megam binary, using the config\_megam() function. Typical usage:

**>>> from** **nltk.classify** **import** megam

**>>>** megam.config\_megam() *# pass path to megam if not found in PATH*

[Found megam: ...]

Use with MaxentClassifier. Example below, see MaxentClassifier documentation for details.

nltk.classify.MaxentClassifier.train(corpus, ‘megam’)

nltk.classify.megam.**call\_megam**(*args*)[[source]](http://www.nltk.org/_modules/nltk/classify/megam.html#call_megam)

Call the megam binary with the given arguments.

nltk.classify.megam.**config\_megam**(*bin=None*)[[source]](http://www.nltk.org/_modules/nltk/classify/megam.html#config_megam)

Configure NLTK’s interface to the megam maxent optimization package.

|  |  |
| --- | --- |
| **Parameters:** | **bin** (*[str](http://www.nltk.org/api/nltk.sem.html" \l "nltk.sem.logic.AnyType.str" \o "nltk.sem.logic.AnyType.str)*) – The full path to the megam binary. If not specified, then nltk will search the system for a megam binary; and if one is not found, it will raise aLookupError exception. |

nltk.classify.megam.**parse\_megam\_weights**(*s*, *features\_count*, *explicit=True*)[[source]](http://www.nltk.org/_modules/nltk/classify/megam.html#parse_megam_weights)

Given the stdout output generated by megam when training a model, return a numpy array containing the corresponding weight vector. This function does not currently handle bias features.

nltk.classify.megam.**write\_megam\_file**(*train\_toks*, *encoding*, *stream*, *bernoulli=True*,*explicit=True*)[[source]](http://www.nltk.org/_modules/nltk/classify/megam.html#write_megam_file)

Generate an input file for megam based on the given corpus of classified tokens.

|  |  |
| --- | --- |
| **Parameters:** | * **train\_toks** (*list(tuple(dict, str))*) – Training data, represented as a list of pairs, the first member of which is a feature dictionary, and the second of which is a classification label. * **encoding** (*[MaxentFeatureEncodingI](http://www.nltk.org/api/nltk.classify.html" \l "nltk.classify.maxent.MaxentFeatureEncodingI" \o "nltk.classify.maxent.MaxentFeatureEncodingI)*) – A feature encoding, used to convert featuresets into feature vectors. May optionally implement a cost() method in order to assign different costs to different class predictions. * **stream** ([*stream*](http://www.nltk.org/api/nltk.html#nltk.data.SeekableUnicodeStreamReader.stream)) – The stream to which the megam input file should be written. * **bernoulli** – If true, then use the ‘bernoulli’ format. I.e., all joint features have binary values, and are listed iff they are true. Otherwise, list feature values explicitly. If bernoulli=False, then you must call megamwith the -fvals option. * **explicit** – If true, then use the ‘explicit’ format. I.e., list the features that would fire for any of the possible labels, for each token. Ifexplicit=True, then you must call megam with the -explicit option. |

nltk.classify.naivebayes module

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of [feature](https://en.wikipedia.org/wiki/Feature_vector) values, where the class labels are drawn from some finite set. It is not a single [algorithm](https://en.wikipedia.org/wiki/Algorithm) for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is [independent](https://en.wikipedia.org/wiki/Independence_(probability_theory)) of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible [correlations](https://en.wikipedia.org/wiki/Correlation_and_dependence) between the color, roundness and diameter features.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) setting. In many practical applications, parameter estimation for naive Bayes models uses the method of [maximum likelihood](https://en.wikipedia.org/wiki/Maximum_likelihood); in other words, one can work with the naive Bayes model without accepting [Bayesian probability](https://en.wikipedia.org/wiki/Bayesian_probability) or using any Bayesian methods.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible [efficacy](https://en.wikipedia.org/wiki/Efficacy) of naive Bayes classifiers.[[5]](https://en.wikipedia.org/wiki/Naive_Bayes_classifier#cite_note-5) Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as [boosted trees](https://en.wikipedia.org/wiki/Boosted_trees) or [random forests](https://en.wikipedia.org/wiki/Random_forests).[[6]](https://en.wikipedia.org/wiki/Naive_Bayes_classifier#cite_note-6)

An advantage of naive Bayes is that it only requires a small amount of training data to estimate the parameters necessary for classification.

A classifier based on the Naive Bayes algorithm.  **naive Bayes classifiers** are a family of simple [probabilistic classifiers](https://en.wikipedia.org/wiki/Probabilistic_classifier) based on applying [Bayes' theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem" \o "Bayes' theorem) with strong (naive) [independence](https://en.wikipedia.org/wiki/Statistical_independence) assumptions between the features. In order to find the probability for a label, this algorithm first uses the Bayes rule to express P(label|features) in terms of P(label) and P(features|label):

P(label) \* P(features|label)

P(label|features) = ——————————

P(features)

The algorithm then makes the ‘naive’ assumption that all features are independent, given the label:

P(label) \* P(f1|label) \* ... \* P(fn|label)

P(label|features) = ——————————————–

P(features)

Rather than computing P(featues) explicitly, the algorithm just calculates the denominator for each label, and normalizes them so they sum to one:

P(label) \* P(f1|label) \* ... \* P(fn|label)

P(label|features) = ——————————————–

SUM[l]( P(l) \* P(f1|l) \* ... \* P(fn|l) )

*class*nltk.classify.naivebayes.**NaiveBayesClassifier**(*label\_probdist*, *feature\_probdist*)[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#NaiveBayesClassifier)

Bases: [nltk.classify.api.ClassifierI](http://www.nltk.org/api/nltk.classify.html" \l "nltk.classify.api.ClassifierI" \o "nltk.classify.api.ClassifierI)

A Naive Bayes classifier. Naive Bayes classifiers are paramaterized by two probability distributions:

* P(label) gives the probability that an input will receive each label, given no information about the input’s features.
* P(fname=fval|label) gives the probability that a given feature (fname) will receive a given value (fval), given that the label (label).

If the classifier encounters an input with a feature that has never been seen with any label, then rather than assigning a probability of 0 to all labels, it will ignore that feature.

The feature value ‘None’ is reserved for unseen feature values; you generally should not use ‘None’ as a feature value for one of your own features.

**classify**(*featureset*)[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#NaiveBayesClassifier.classify)

**labels**()[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#NaiveBayesClassifier.labels)

**most\_informative\_features**(*n=100*)[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#NaiveBayesClassifier.most_informative_features)

Return a list of the ‘most informative’ features used by this classifier. For the purpose of this function, the informativeness of a feature (fname,fval) is equal to the highest value of P(fname=fval|label), for any label, divided by the lowest value of P(fname=fval|label), for any label:

max[ P(fname=fval|label1) / P(fname=fval|label2) ]

**prob\_classify**(*featureset*)[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#NaiveBayesClassifier.prob_classify)

**show\_most\_informative\_features**(*n=10*)[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#NaiveBayesClassifier.show_most_informative_features)

*classmethod***train**(*labeled\_featuresets*, *estimator=<class 'nltk.probability.ELEProbDist'>*)[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#NaiveBayesClassifier.train)

|  |  |
| --- | --- |
| **Parameters:** | **labeled\_featuresets** – A list of classified featuresets, i.e., a list of tuples (featureset, label). |

nltk.classify.naivebayes.**demo**()[[source]](http://www.nltk.org/_modules/nltk/classify/naivebayes.html#demo)

**Implementation**

Now that we know how to use a bunch of algorithmic classifiers, like a child in the candy isle, told they can only pick one, we may find it difficult to choose just one classifier. The good news is, you don't have to! Combining classifier algorithms is is a common technique, done by creating a sort of voting system, where each algorithm gets one vote, and the classification that has the votes votes is the chosen one.

To do this, we want our new classifier to act like a typical NLTK classifier, with all of the methods. Simple enough, using object oriented programming, we can just be sure to inherit from the NLTK classifier class. To do this, we'll import it:

from nltk.classify import ClassifierI

from statistics import mode

We also import mode, as it will be our method for choosing the most popular vote.

Now, let's build our classifier class:

class VoteClassifier(ClassifierI):

def \_\_init\_\_(self, \*classifiers):

self.\_classifiers = classifiers

We're calling our class the VoteClassifier, and we're inheriting from NLTK's ClassifierI. Next, we're assigning the list of classifiers that are passed to our class to self.\_classifiers.

Next, we want to go ahead and create our own classify method. We want to call it classify, so that we can invoke .classify later on, like a traditional NLTK classifier would allow.

def classify(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

return mode(votes)

Easy enough, all we're doing here is iterating through our list of classifier objects. Then, for each one, we ask it to classify based on the features. The classification is being treated as a vote. After we are done iterating, we then return the mode(votes), which is just returning the most popular vote.

This is all we really need, but I think it would be useful to have another parameter, confidence. Since we have algorithms voting, we can also tally the votes for and against the winning vote, and call this "confidence." For example, 3/5 votes for positive is weaker than 5/5 votes. As such, we can literally return the ratio of votes as a sort of confidence indicator. Here's our confidence method:

def confidence(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

choice\_votes = votes.count(mode(votes))

conf = choice\_votes / len(votes)

return conf

Now, let's put everything together:

import nltk

import random

from nltk.corpus import movie\_reviews

from nltk.classify.scikitlearn import SklearnClassifier

import pickle

from sklearn.naive\_bayes import MultinomialNB, BernoulliNB

from sklearn.linear\_model import LogisticRegression, SGDClassifier

from sklearn.svm import SVC, LinearSVC, NuSVC

from nltk.classify import ClassifierI

from statistics import mode

class VoteClassifier(ClassifierI):

def \_\_init\_\_(self, \*classifiers):

self.\_classifiers = classifiers

def classify(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

return mode(votes)

def confidence(self, features):

votes = []

for c in self.\_classifiers:

v = c.classify(features)

votes.append(v)

choice\_votes = votes.count(mode(votes))

conf = choice\_votes / len(votes)

return conf

documents = [(list(movie\_reviews.words(fileid)), category)

for category in movie\_reviews.categories()

for fileid in movie\_reviews.fileids(category)]

random.shuffle(documents)

all\_words = []

for w in movie\_reviews.words():

all\_words.append(w.lower())

all\_words = nltk.FreqDist(all\_words)

word\_features = list(all\_words.keys())[:3000]

def find\_features(document):

words = set(document)

features = {}

for w in word\_features:

features[w] = (w in words)

return features

#print((find\_features(movie\_reviews.words('neg/cv000\_29416.txt'))))

featuresets = [(find\_features(rev), category) for (rev, category) in documents]

training\_set = featuresets[:1900]

testing\_set = featuresets[1900:]

#classifier = nltk.NaiveBayesClassifier.train(training\_set)

classifier\_f = open("naivebayes.pickle","rb")

classifier = pickle.load(classifier\_f)

classifier\_f.close()

print("Original Naive Bayes Algo accuracy percent:", (nltk.classify.accuracy(classifier, testing\_set))\*100)

classifier.show\_most\_informative\_features(15)

MNB\_classifier = SklearnClassifier(MultinomialNB())

MNB\_classifier.train(training\_set)

print("MNB\_classifier accuracy percent:", (nltk.classify.accuracy(MNB\_classifier, testing\_set))\*100)

BernoulliNB\_classifier = SklearnClassifier(BernoulliNB())

BernoulliNB\_classifier.train(training\_set)

print("BernoulliNB\_classifier accuracy percent:", (nltk.classify.accuracy(BernoulliNB\_classifier, testing\_set))\*100)

LogisticRegression\_classifier = SklearnClassifier(LogisticRegression())

LogisticRegression\_classifier.train(training\_set)

print("LogisticRegression\_classifier accuracy percent:", (nltk.classify.accuracy(LogisticRegression\_classifier, testing\_set))\*100)

SGDClassifier\_classifier = SklearnClassifier(SGDClassifier())

SGDClassifier\_classifier.train(training\_set)

print("SGDClassifier\_classifier accuracy percent:", (nltk.classify.accuracy(SGDClassifier\_classifier, testing\_set))\*100)

##SVC\_classifier = SklearnClassifier(SVC())

##SVC\_classifier.train(training\_set)

##print("SVC\_classifier accuracy percent:", (nltk.classify.accuracy(SVC\_classifier, testing\_set))\*100)

LinearSVC\_classifier = SklearnClassifier(LinearSVC())

LinearSVC\_classifier.train(training\_set)

print("LinearSVC\_classifier accuracy percent:", (nltk.classify.accuracy(LinearSVC\_classifier, testing\_set))\*100)

NuSVC\_classifier = SklearnClassifier(NuSVC())

NuSVC\_classifier.train(training\_set)

print("NuSVC\_classifier accuracy percent:", (nltk.classify.accuracy(NuSVC\_classifier, testing\_set))\*100)

voted\_classifier = VoteClassifier(classifier,

NuSVC\_classifier,

LinearSVC\_classifier,

SGDClassifier\_classifier,

MNB\_classifier,

BernoulliNB\_classifier,

LogisticRegression\_classifier)

print("voted\_classifier accuracy percent:", (nltk.classify.accuracy(voted\_classifier, testing\_set))\*100)

print("Classification:", voted\_classifier.classify(testing\_set[0][0]), "Confidence %:",voted\_classifier.confidence(testing\_set[0][0])\*100)

print("Classification:", voted\_classifier.classify(testing\_set[1][0]), "Confidence %:",voted\_classifier.confidence(testing\_set[1][0])\*100)

print("Classification:", voted\_classifier.classify(testing\_set[2][0]), "Confidence %:",voted\_classifier.confidence(testing\_set[2][0])\*100)

print("Classification:", voted\_classifier.classify(testing\_set[3][0]), "Confidence %:",voted\_classifier.confidence(testing\_set[3][0])\*100)

print("Classification:", voted\_classifier.classify(testing\_set[4][0]), "Confidence %:",voted\_classifier.confidence(testing\_set[4][0])\*100)

print("Classification:", voted\_classifier.classify(testing\_set[5][0]), "Confidence %:",voted\_classifier.confidence(testing\_set[5][0])\*100)

So at the end here, we're running a few classification examples against text. All of our output:

Original Naive Bayes Algo accuracy percent: 66.0

Most Informative Features

thematic = True pos : neg = 9.1 : 1.0

secondly = True pos : neg = 8.5 : 1.0

narrates = True pos : neg = 7.8 : 1.0

layered = True pos : neg = 7.1 : 1.0

rounded = True pos : neg = 7.1 : 1.0

supreme = True pos : neg = 7.1 : 1.0

crappy = True neg : pos = 6.9 : 1.0

uplifting = True pos : neg = 6.2 : 1.0

ugh = True neg : pos = 5.3 : 1.0

gaining = True pos : neg = 5.1 : 1.0

mamet = True pos : neg = 5.1 : 1.0

wanda = True neg : pos = 4.9 : 1.0

onset = True neg : pos = 4.9 : 1.0

fantastic = True pos : neg = 4.5 : 1.0

milos = True pos : neg = 4.4 : 1.0

MNB\_classifier accuracy percent: 67.0

BernoulliNB\_classifier accuracy percent: 67.0

LogisticRegression\_classifier accuracy percent: 68.0

SGDClassifier\_classifier accuracy percent: 57.99999999999999

LinearSVC\_classifier accuracy percent: 67.0

NuSVC\_classifier accuracy percent: 65.0

voted\_classifier accuracy percent: 65.0

Classification: neg Confidence %: 100.0

Classification: pos Confidence %: 57.14285714285714

Classification: neg Confidence %: 57.14285714285714

Classification: neg Confidence %: 57.14285714285714

Classification: pos Confidence %: 57.14285714285714

Classification: pos Confidence %: 85.71428571428571

.

Applications

It is true that application was one of the most important elements for PR theory. Pattern Recognition has been developed for many years, and the technology of PR has been applied in many fields such as artificial intelligence, computer engineering, nerve biology, medicine image analysis, archaeology, geologic reconnoitering, space navigation, armament technology and so on. Detailed applications, such as below: Pattern recognition applications Pattern recognition is used in any area of science and engineering that studies the structure of observations. It is now frequently used in many applications in manufacturing industry, healthcare, and the military. Examples include the following. Optical character recognition (OCR) is becoming an integral part of document scanners, and is also used frequently in banking and postal applications. Printed characters can now be accurately recognized, and the improving performance of automatic recognition of handwritten cursive characters has diminished significantly the need of human interaction for OCR tasks. Automatic speech recognition is very important for user interaction with machines. Commercial systems for automatic response to flight queries, telephone directory assistance, and telebanking are available. Often the systems are tuned to a specific speaker for better recognition accuracy. Computer vision deals with the recognition of objects as well as the identification and localization of their threedimensional environments. This capability is required, for example, by robots to operate in dynamic or unknown environments. This can be useful for applications ranging from manufacturing to household cleaning, and even for rescue missions. Personal identification systems that use biometrics are very important for security applications in airports, ATMs, shops, hotels, and secure computer access. Recognition can be based on face, fingerprint, iris, or voice, and can be combined with the automatic verification of signatures and PIN codes. Recognition of objects on earth from the sky (by satellites) or from the air (by aeroplanes and cruise missiles) is called remote sensing. It is important for cartography, agricultural inspection, detection of minerals and pollution, and target recognition. Many tests for medical diagnosis utilize pattern recognition systems, from counting blood cells and recognition of cell tissues through microscopes to the detection of tumours in magnetic resonance scans and the inspection of bones and joints in X-ray images. Many large databases are stored on the repositories accessible via the internet or otherwise in local computers. They may have a clear structure such as bank accounts, a weak structure such as consumer behaviour, or no obvious structure such as a collection of images. Procedures for finding desired items (database retrieval) as well as learning or discovering structures in databases (data mining) are becoming more and more important. Web search engines and recommender systems are two example applications • Computer vision The first vision system presented was supposing the objects with geometric shapes and optimized edges extracted from images.[26,27,28] • Computer aided diagnosis Medical imaging, EEG, EEG signal analysis Designed to assist physicians, such as: X-ray mammography Highlighting potential tambours on a mammogram • Character recognition Automated mail sorting, processing bank checks; Scanner captures an image of the text; Image is converted into constituent characters • Speech recognition Human computer interaction, Universal access; Microphone records acoustic signal; Speech signal is classified into phonemes and words • Safety Face recognition Identifying fingerprints • Astronomy Classifying galaxies by shape Astronomical telescope image analysis Automatic spectroscopy • Bioinformatics DNA sequences analysis DNA micro array data analysis[29] Research of heredity • Agriculture Output analysis Soil evaluating Extraction mineral characterization in coffee and sugar [30] • Geography Earthquake analysis Rocks classification • Engineering Fault diagnosis for vehicle system Recognition of automobile Type Improve the safety performance of automobile • Military affairs Aviation photography analysis Automatism Aim recognition

Future work:

-> We can improve the accuracy.

->we'll try to implement it for more complex classes.

->