Contents

1 Implementation 3
2 Some theory 7
3 Indices and tables 9
Python Module Index 11
Find the code on GitHub.

This is a simple Naïve Bayes classifier implementation in pure Python.
Simple Naive Bayes classifier implementation.

It focuses on being:

- simple to use,
- easy to save and restore (in a file, database...),
- incrementally trainable,
- compatible with sparse matrices represented as dictionaries.

```python
class simple_naivebayes.SimpleNaiveBayes
    Naive Bayes classifier class.
```

**Note:** The counters of features and labels are publicly exposed as you might want to modify them or using them directly.

### feats_label_counts
```
dict of dicts of ints
```

Store conditional probabilities for a given feature to appear in any class. For example, if `feat1` were to appear 4 times in `label1` and 2 times in `label2`:

```python
{  
    "label1": {  
        "feat1": 4  
    },  
    "label2": {  
        "feat1": 2  
    }  
}
```

### seen_examples
```
dict of ints
```

Counts the number of examples seen for every class. For example, if we trained against 3 good instances and 5 bad ones:

```python
{  
    "good": 3,  
    "bad": 5  
}
```

### seen_features
```
dict of ints
```

Counts the number of times a feature has been seen during training. For example, if we’ve seen 3 times `foo` and 2 times `bar`:

```python
Simple Naive Bayes Documentation, Release 1.0

```json
{
    "foo": 3,
    "bar": 2
}
```

`__init__()`
Create a SimpleNaiveBayes object and initializes it with empty probabilities.

It then needs to be trained to become useful. See `unserialize()` and `train()` methods.

`__weakref__`
list of weak references to the object (if defined)

`classify(features)`
Classify a set of features using the current model.

Parameters
- `features` *(list of words)* – A set of features to classify.

Returns
- *(string, float)* tuple: A tuple containing the best matching class, and the posterior probability as a float.

The posterior probability is returned as a log (base 2) value.

**Note:** This method performs Laplace smoothing to avoid errors for unseen features.

`features()`
Returns list of strings: List of all features seen during training.

`features_for_label(label)`
Parameters
- `label` *(string)* – Label to work on.

Returns list of strings: List of all features seen for `label`.

`labels()`
Returns list of strings: A list containing the labels seen during training.

`seen_features_for_label(label)`
Parameters
- `label` *(string)* – Label to work on.

Returns The sum of the occurrences of every feature of `label`.

Return type `int`

`serialize()`
Outputs a serializable object representing the classifier.

Returns dictionary representing the classifier.

Return type `dict`

**Note:** The returned object is ready to be `json.dumps`-ed, or serialized in any other way.

`total_documents()`
Returns Total number of documents seen during training.

Return type `int`

`total_seen_features()`
Returns Sum of occurrences of all features seen during training.

Return type `int`
**train** *(examples)*

Trains the classifier against a set of examples.

**Parameters**

- **examples** *(list of tuples)* – A list or set of already classified examples. E.g.:

  ```
  [(["hello", "world"], 'my_first_label'),
   (["this", "is", "sparta"], 'my_second_label')
  ]
  ```

**Note:** This method mutates the classifier and takes into account previous training sessions.

---

**classmethod** **unserialize** *(data)*

Create a classifier from an object created from `serialize()`.

**Parameters**

- **data** *(dict)* – An object generated by `serialize()` used to initialize the new object.

**Returns**

An instance of this class initialized using the data parameter.

**Return type**

SimpleNaiveBayes
This is a naive bayes classifier, so our goal is, given a vector of features \( f \), to find the label \( l \) which minimizes the posterior probability \( p(l|f) \).

To evaluate \( p(l|f) \), we first apply Bayes’ theorem:

\[
p(l|f) = \frac{p(f|l) \times p(l)}{p(f)}
\]

We now need to make the naïve assumption: features are conditionally independents. Therefore:

\[
p(f) = p(f_1) \times p(f_2) \times ... \times p(f_n)
\]

We can rewrite the posterior probability as such:

\[
p(l|f) = \frac{p(l) \times p(f_1|l) \times p(f_2|l) \times ... \times p(f_n|l)}{p(f_1) \times p(f_2) \times ... \times p(f_n)}
\]

Let’s use the following notations:

- \( C \) is the set of training examples.
- \( C_l \) is the set of training examples of label \( l \).
- \( T \) is the set of occurences of features among the training examples.
- \( T_k \) is set of occurences of the \( k \)-th feature in the training examples.
- \( T_{k,l} \) is the same as \( T_k \) but restructed to the label \( l \).
- \( T_{*,l} \) is the set of occurences of features for the label \( l \).
- \(|S|\) denotes the cardinality of the set \( S \).
- \( N \) is size of \( f \).

Now, by definition we have:

\[
p(l) = \frac{|C_l|}{|C|}
\]

\[
p(f_k|l) = \frac{|T_{k,l}|}{|T_{*,l}|}
\]

\[
p(f_k) = \frac{|T_k|}{|T|}
\]

However, we may need to evaluate the posterior probability of a vector containing an unseen feature \( f_k \). This would yield \( p(f_k) = p(f_k|l) = 0 \), which leads to \( p(l|f) = 0 \). To avoid this problem, we use Laplace smoothing.
(or add-one smoothing). Those two probabilities now become

\[
p(f_k | l) = \frac{|T_{k,l}| + 1}{|T^*| + |T^*|}\]

\[
p(f_k) = \frac{|T_k| + 1}{|T| + |T^*|}
\]

where \(|S|^*\) denotes the cardinal of unique occurrences of the set \(S\).

By putting all that together, the posterior probability becomes

\[
p(l | f) = \frac{|C_l|}{|C|} \times \prod_{k=1}^{N} \frac{|T_{k,l}| + 1}{|T^*| + |T^*|} \times \prod_{k=1}^{N} \frac{|T| + |T^*|}{|T_k| + 1}
\]

Because dealing with very small values inside \([0, 1]\) may lead to floating precision problems, we compute \(\log(p(l | f))\) (also known as log-likelihood) instead:

\[
\log(p(l | f)) = \log(|C_l|) - \log(|C|)
\]

\[
+ \sum_{k=1}^{N} \log(|T_{k,l}| + 1) - N \log(|T^*|)
\]

\[
+ N \log(|T| + |T^*|) - \sum_{k=1}^{N} \log(|T_k| + 1)
\]

That’s what the SimpleNaiveBayes.classify() function computes, for every known classes.
Indices and tables

- genindex
- modindex
- search
S

simple_naivebayes, 3
Symbols

__init__() (simple_naivebayes.SimpleNaiveBayes method), 4
__weakref__ (simple_naivebayes.SimpleNaiveBayes attribute), 4

classify() (simple_naivebayes.SimpleNaiveBayes method), 4

features() (simple_naivebayes.SimpleNaiveBayes method), 4
features_for_label() (simple_naivebayes.SimpleNaiveBayes method), 4

labels() (simple_naivebayes.SimpleNaiveBayes method), 4

seen_examples (simple_naivebayes.SimpleNaiveBayes attribute), 3
seen_features (simple_naivebayes.SimpleNaiveBayes attribute), 3
seen_features_for_label() (simple_naivebayes.SimpleNaiveBayes method), 4
serialize() (simple_naivebayes.SimpleNaiveBayes method), 4

train() (simple_naivebayes.SimpleNaiveBayes method), 4

unserialize() (simple_naivebayes.SimpleNaiveBayes class method), 5

total_documents() (simple_naivebayes.SimpleNaiveBayes method), 4
total_seen_features() (simple_naivebayes.SimpleNaiveBayes method), 4