ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"

7 - Classification for Text Analysis

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Halloween Conference in Quantitative Finance

Bologna - October 26-28, 2021

Subsection 1

Introduction

What is Text Classification

Introduction

- Text Classification is the process of labeling or organizing text data into groups - it forms a fundamental part of Natural Language Processing.
- In the digital age that we live in, we are surrounded by text on our social media accounts, commercials, websites, Ebooks, etc. The majority of this text data is **unstructured**, so classifying this data can be extremely useful.
- The premise of classification is simple: given a categorical target variable, learn patterns that exist between instances composed of independent variables and their relationship to the target.

Introduction

Text Classification: a formal definition

INPUT:

- a document d;
- a fixed set of classes $C = \{C_1, C_2, \ldots, C_n\}$

OUTPUT

- a predicted class $c \in C$

Introduction

Text Classification has a wide array of applications. Some popular uses are:

- Spam detection in emails
- Sentiment analysis of online reviews
- Topic labeling documents like research papers
- Language detection like in Google Translate
- Age/gender identification of anonymous users
- Tagging online content
- Speech recognition used in virtual assistants (like Siri and Alexa)

Introduction

Hand-coded rules

- Rules based on combinations of words or other features, for example: spam if black-list-address OR ("dollars" AND "you have been selected")
- Accuracy can be high if rules carefully refined by expert
- But building and maintaining these rules is expensive

Introduction

- Because **the target is given ahead of time**, classification is a supervised machine learning model because a model can be trained to minimize error between predicted and actual categories in the training data.
- Once a classification model is fit, it assigns categorical labels to new instances based on the patterns detected during training.
- The application problem can be formulated to identify either a yes/no (binary classification) or discrete buckets (multiclass classification).

Introduction

Text Classification: Supervised Machine Learning

INPUT:

- a document d;
- a fixed set of classes $C = \{C_1, C_2, \ldots, C_n\}$
- a training set of m hand-labeled documents $(d_1, c_1), \ldots, (d_m, c_m)$

OUTPUT

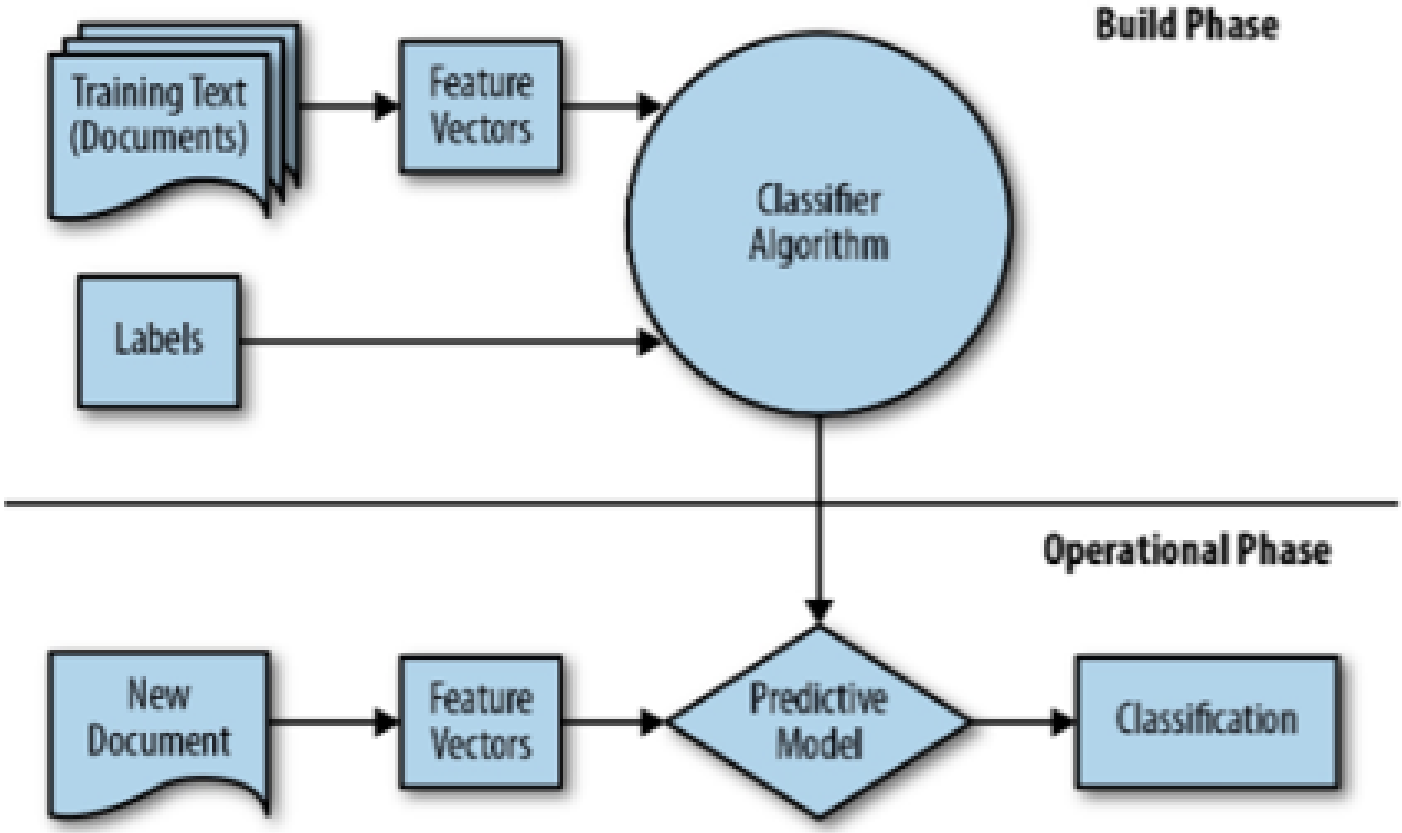
- a learned classifier $\gamma : d \rightarrow c$

Introduction

All classifier model families have the same basic workflow, and with Scikit-Learn Estimator objects, they can be employed in a procedural fashion and compared using cross-validation to select the best performing predictor. The classification workflow occurs in two phases: a build phase and an operational phase.

- In the build phase, a corpus of documents is transformed into feature vectors. The document features, along with their annotated labels (the category or class we want the model to learn), are then passed into a classification algorithm that defines its internal state along with the learned patterns.
- Once trained or fitted, a new document can be vectorized into the same space as the training data and passed to the predictive algorithm, which returns the assigned class label for the document.

Introduction



source: *Bengfort B. et al. Text Analysis with Python*

Subsection 2

Naive Bayes

Naive Bayes

- Simple ("naive") classification method based on Bayes rule
- Relies on very simple representation of document: Bag of words
- For a document d and a class c

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{1}$$

$$C_{map} = \underset{c \in C}{\text{argmax}} \frac{P(d|c)P(c)}{P(d)} \tag{2}$$

The denominator does not change, it remain static. Therefore, the denominator can be removed and a proportionality can be introduced

$$C_{map} \sim \underset{c \in C}{\text{argmax}} \left[P(d|c)P(c)\right] \tag{3}$$

Naive Bayes

- We can represent a document as a vector of features $(x_1, x_2, \ldots, x_n)$

$$C_{map} \sim \underset{c \in C}{\text{argmax}} \left[P(x_1, x_2, \ldots, x_n | c) P(c) \right] \tag{4}$$

- To estimate P we can just count the relative frequencies in a corpus
- Of course, it could only be estimated if a very, very large number of training examples was available.

Naive Bayes

Now the **naive** assumptions ...

- Bag of Words assumption: Assume position doesn't matter
- Conditional Independence: Assume the feature probabilities $P(x_i|c_j)$ are independent given the class c

$$P(x_1, x_2, \ldots, x_n|c) = \prod_{i=1}^{n} P(x_i|c) \tag{5}$$

$$C_{map} \sim \underset{c \in C}{\operatorname{argmax}}\, P(c) \prod_{i=1}^{n} P(x_i|c) \tag{6}$$

Naive Bayes

- Training process maximum likelihood estimates
- Simply use the frequencies in the data
- Create mega-document for topic j by concatenating all docs in this topic

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$$

$$\hat{P}(w_i|c_j) = \frac{\text{count}(w_i|c_j)}{\sum_{w \in V} \text{count}(w|c_j)}$$

Subsection 3

Practical Examples

Example 1 - Gender Identification

In this first example we will try to build a simple algorithm to understand if a noun passed in input is of masculine or feminine gender. In this, and in the following examples, we will implicitly assume that the language used is English.

FOCUS

- How build a feature in text analysis
- Training Vs validation

Example 2 - The *20 Newsgroup* data set

- In this example we are going to work with the "20 Newsgroup Dataset".
- The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.
- This data set is infact in-built in scikit, so we don't need to download it explicitly.

Example 2 - The *20 Newsgroup* data set

FOCUS

- using sklearn package
- numerical features and vectorization process
- building a ml pipeline with sklearn
- model optimization

Example 2 - Building a Pipeline

- A machine learning pipeline is a way to codify and automate the workflow it takes to produce a machine learning model.
- Machine learning pipelines consist of multiple sequential steps that do everything from data extraction and preprocessing to model training and deployment.
- For data science teams, the production pipeline should be the central product.
- It encapsulates all the learned best practices of producing a machine learning model for the organization use-case and allows the team to execute at scale.
- Whether you are maintaining multiple models in production or supporting a single model that needs to be updated frequently, an end-to-end machine learning pipeline is a must.

Example 2 - Building a Pipeline

- The purpose of a Pipeline is to chain together multiple estimators representing a fixed sequence of steps into a single unit.
- All estimators in the pipeline, except the last one, must be transformers - that is, implement the transform method, while the last estimator can be of any type, including predictive estimators.
- Pipelines provide convenience; fit and transform can be called for single inputs across multiple objects at once.
- Pipelines also provide a single interface for grid search of multiple estimators at once.

Example 2 - Build a Pipeline with SciKit

- Most importantly, pipelines provide operationalization of text models by coupling a vectorization methodology with a predictive model.
- Pipelines are constructed by describing a list of (key, value) pairs where the key is a string that names the step and the value is the estimator object.
- Pipelines can be created either by using the make_pipeline helper function, which automatically determines the names of the steps, or by specifying them directly.
- Generally, it is better to specify the steps directly to provide good user documentation, whereas make_pipeline is used more often for automatic pipeline construction.

Example 2 - Build a Pipeline with SciKit

- The Pipeline can then be used as a single instance of a complete model.
- Calling model.fit is the same as calling fit on each estimator in sequence, transforming the input and passing it on to the next step.
- Other methods like fit_transform behave similarly.

```
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB

model = Pipeline([
    ('normalizer', TextNormalizer()),
    ('vectorizer', GensimVectorizer()),
    ('bayes', MultinomialNB()),
])
```

Example 2 - Build a Pipeline with SciKit

- The pipeline will also have all the methods the last estimator in the pipeline has. If the last estimator is a transformer, so too is the pipeline.
- If the last estimator is a classifier, as in the example above, then the pipeline will also have predict and score methods so that the entire model can be used as a classifier.

```
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB

model = Pipeline([
    ('normalizer', TextNormalizer()),
    ('vectorizer', GensimVectorizer()),
    ('bayes', MultinomialNB()),
])
```

Subsection 4

Sentiment Analysis with Keras

From *Deep Learning with Python by F. Chollet*

Sentiment Analysis with Keras

- The IMDb Movie Reviews dataset is a binary sentiment analysis dataset consisting of 50,000 reviews from the Internet Movie Database (IMDb) labeled as positive or negative.
- The dataset contains an even number of positive and negative reviews.
- Only highly polarizing reviews are considered.
- A negative review has a score $= 4$ out of 10, and a positive review has a score $= 7$ out of 10.
- No more than 30 reviews are included per movie.
- The dataset contains additional unlabeled data.

The IMDb Movie Reviews Dataset

- Just like the MNIST dataset, the IMDB dataset comes packaged with Keras.
- It has already been preprocessed: the reviews (sequences of words) have been turned into sequences of integers, where each integer stands for a specific word in a dictionary.
- The following code will load the dataset (when you run it the first time, about 80 MB of data will be downloaded to your machine).

Listing 3.1 Loading the IMDB dataset

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

The IMDb Movie Reviews Dataset

- **One-hot encode**
- All lists are transformed into vectors of 0s and 1s.
- This would mean, for instance, turning the sequence [3, 5] into a 10,000-dimensional vector that would be all 0s except for indices 3 and 5, which would be 1s.
- Then you could use as the first layer in your network a Dense layer, capable of handling floating-point vector data.

Listing 3.2 Encoding the integer sequences into a binary matrix

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Creates an all-zero matrix of shape (len(sequences), dimension)

Sets specific indices of results[i] to 1s

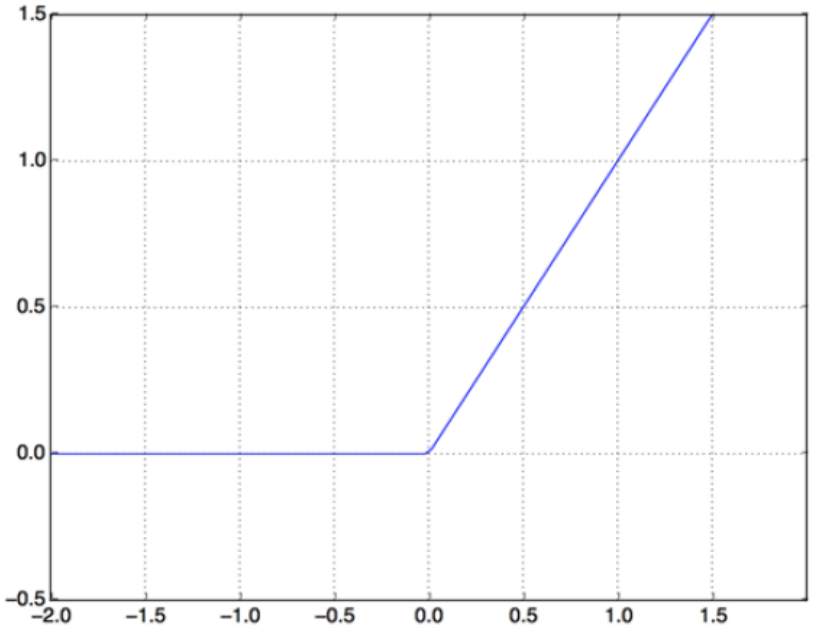
Vectorized training data

Vectorized test data

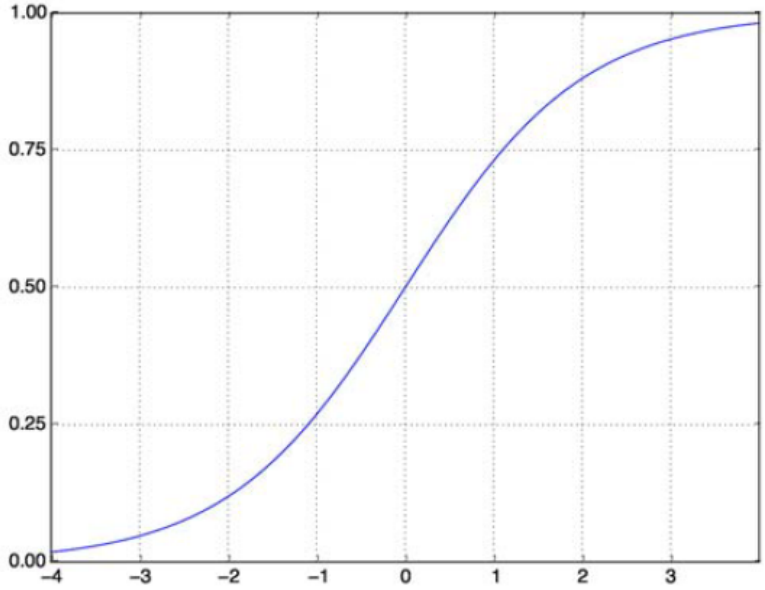
Preparing the data

- The input data is vectors, and the labels are scalars (1s and 0s).
- A type of network that performs well on such a problem is a simple stack of fully connected (Dense) layers with relu activations.
- The argument being passed to each Dense layer (16) is the number of hidden units of the layer.
- The intermediate layers will use relu as their activation function, and the final layer will use a sigmoid activation so as to output a probability (a score between 0 and 1, indicating how likely the review is to be positive).

Building our Network

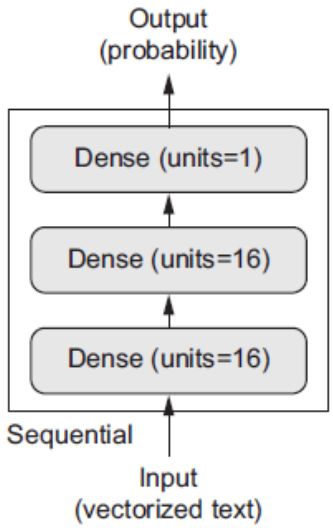


The rectified linear unit function



The sigmoid function

Building our Network



The three-layer network

Listing 3.3 The model definition

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Building our Network

- Finally, we need to choose a loss function and an optimizer.
- Because we are facing a binary classification problem and the output of your network is a probability (we end our network with a single-unit layer with a sigmoid activation), it is best to use the binary_crossentropy loss.
- It is not the only viable choice: you could use, for instance, the well known mean squared error but crossentropy is usually the best choice when you're dealing with models that output probabilities.
- Crossentropy is a quantity from the field of Information Theory that measures the distance between probability distributions or, in this case, between the ground-truth distribution and your predictions.

Loss Function and Optimizer

- **RMSprop** is unpublished optimization algorithm designed for neural networks, first proposed by Geoff Hinton in lecture 6 of the online course **Neural Networks for Machine Learning**;
- **Adaptive learning rate** methods are an optimization of gradient descent methods with the goal of minimizing the objective function of a network by using the gradient of the function and the parameters of the network.

Listing 3.5 Configuring the optimizer

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Example

- Now that we have described the construction of the neural network, let's see how it works in practice ...
- Using **07-DLP-classifying-movie-reviews.ipynb** Notebook