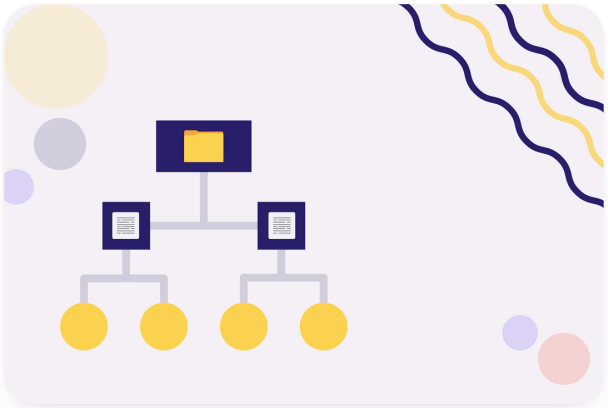


Back to overview

Text Classifiers in Machine Learning: A Practical Guide

Patricia Orza · Content, Community & Events Manager

September 30, 2024

Text classifiers in Machine Learning: A practical guide

Unstructured data accounts for over 80% of all data, with text being one of the most common categories. Because analyzing, comprehending, organizing, and sifting through text data is difficult and time-consuming due to its messy nature, most businesses do not exploit it to its full potential despite all the potential benefits it would bring.

This is where Machine Learning and text classification come into play. Companies may use text classifiers to quickly and cost-effectively arrange all types of relevant content, including emails, legal documents, social media, chatbots, surveys, and more.

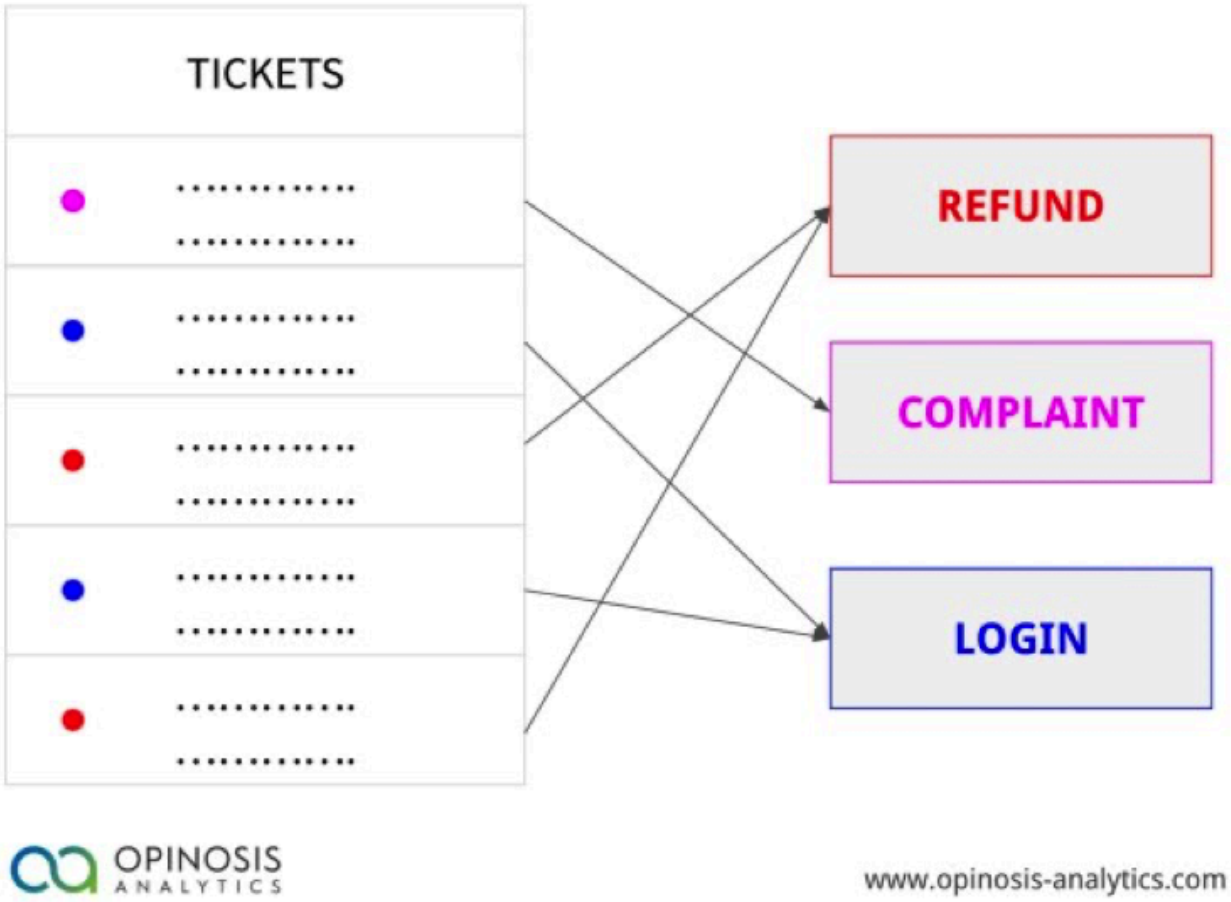
This guide will explore text classifiers in Machine Learning, some of the essential models you need to know, how to evaluate those models, and the potential alternatives to developing your algorithms.

What is a text classifier?

Natural Language Processing (NLP), Sentiment Analysis, spam, and intent detection, and other applications use text classification as a core Machine Learning technique. This crucial feature is especially useful for language identification, allowing organizations and individuals to comprehend things like consumer feedback better and inform future

A text classifier labels unstructured texts into predefined text categories. Instead of users having to review and analyze vast amounts of information to understand the context, text classification helps derive relevant insight.

Companies may, for example, need to classify incoming customer support tickets so that they are sent to the appropriate customer care personnel.



Example of text classification labels for customer support tickets. *Source: https://kavita-ganesan.com/5-real-world-examples-of-text-classification/#.YdRRGWjP23A*

Text classification Machine Learning systems do not rely on rules that have been manually established. It learns to classify text based on

for a given text or input. In highly complicated tasks, the results are more accurate than human rules, and algorithms can incrementally learn from new data.

Classifier vs model – what's the difference?

In some contexts, the terms "classifier" and "model" are synonymous. However, there is a subtle difference between the two.

The algorithm, which is at the heart of your Machine Learning process, is known as a classifier. An SVM, Naïve Bayes, or even a Neural Network classifier can be used. Essentially, it's an extensive "collection of rules" for how you want to categorize your data.

A model is what you have after training your classifier. In Machine Learning language, it is like an intelligent black box into which you feed samples for it to output a label.

We have listed some of the key terminology associated with text classification below to make things more tractable.

Training sample

A training sample is a single data point (x) from a training set to solve a predictive modeling problem. If we want to classify emails, one email in our dataset would be one training sample. People may also use the terms training instance or training example interchangeably.

We are usually interested in modeling a specific process in predictive modeling. We want to learn or estimate a particular function that, for example, allows us to discriminate spam from non-spam email. The proper function f that we want to model is the target function f(x) = y.

Hypothesis

In the context of text classification, such as email spam filtering, the hypothesis would be that the rule we come up with can separate spam from genuine emails. It is a specific function that we estimate is similar to the target function that we are looking to model.

Model

Where the hypothesis is a guess or estimation of a Machine Learning function, the model is the manifestation of that guess used to test it.

Learning algorithm

The learning algorithm is a series of instructions that uses our training dataset to approximate the target function. A hypothesis space is the set of probable hypotheses that a learning algorithm can generate to model an unknown target function by formulating the final hypothesis.

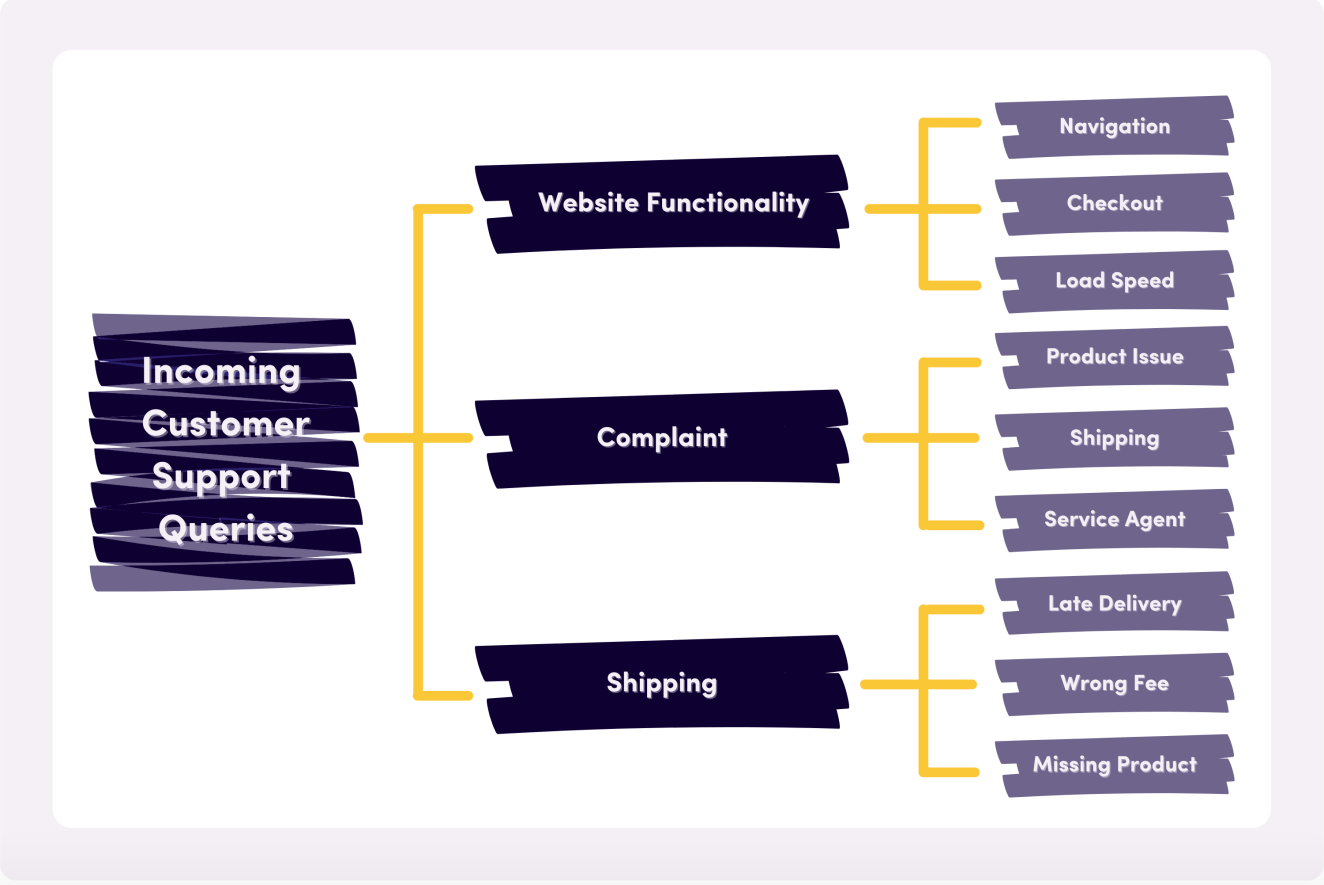
Classifier

A classifier is a hypothesis or discrete-valued function for assigning (categorical) class labels to specific data points. This classifier might be a hypothesis for classifying emails as spam or non-spam in the email

While each of the terms has similarities, there are subtle differences between them that are important to understand in Machine Learning.

Defining your tags

When working on text classification in Machine Learning, the first step is defining your tags, which depend on the business case. For example, if you are classifying customer support queries, the tags may be "website functionality," "shipping," or "complaint." In some cases, the core tags will also have sub-tags that require a separate text classifier. In the customer support example, sub-tags for complaints could be "product issue" or "shipping error." You can create a hierarchical tree for your tags.



In the hierarchical tree above, you will create a text classifier for the first level of tags (Website Functionality, Complaint, Shipping) and a separate classifier for each subset of tags. The objective is to ensure that the subtags have a semantic relation. A text classification process with a clear and obvious structure makes a significant difference in the accuracy of predictions from your classifiers.

You must also avoid overlapping (two tags with similar meanings that could confuse your model) and ensure each model has a single classification criterion. For example, a product can be tagged as a "complaint" and "website functionality," as it is a complaint about the website, meaning the tags don't contradict each other.

Deciding on the right algorithm

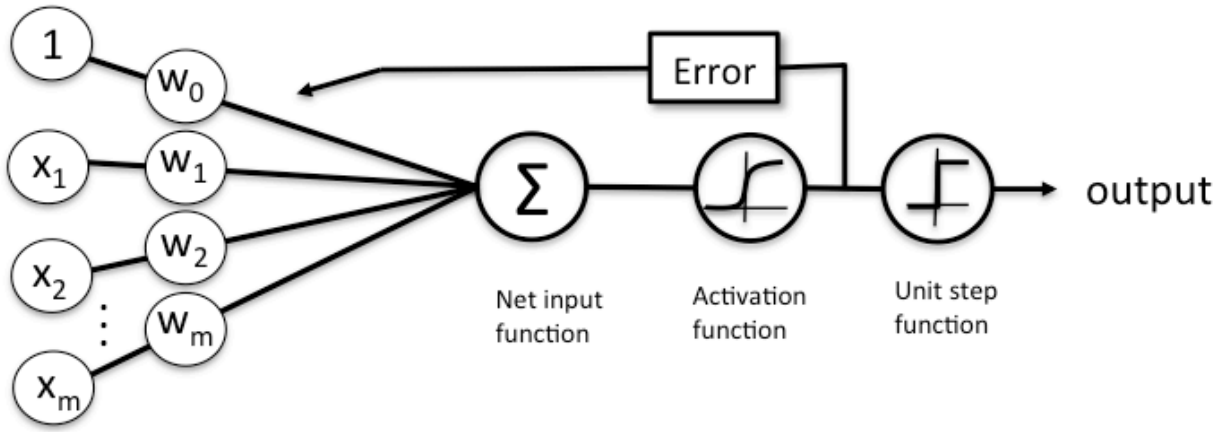
Python is the most popular language when it comes to text classification with Machine Learning. Python text classification has a simple syntax and several open-source libraries available to create your algorithms.

Below are the standard algorithms to help pick the best one for your text classification project.

Logistic regression

Despite the word "regression" in its name, logistic regression is a supervised learning method usually employed to handle binary "classification" tasks. Although "regression" and "classification" are incompatible terms, the focus of logistic regression is on the word "logistic," which refers to the logistic function that performs the classification operation in the algorithm. Because logistic regression is a

regression can solve. It's even employed as a Neural Network layer activation function.



Schematic of a logistic regression classifier.

The logistic function, commonly known as the sigmoid function, is the foundation of logistic regression. It takes any real-valued integer and translates it to a value between 0 and 1.

A linear equation is used as input, and the logistic function and log odds are used to complete a binary classification task.

Naïve Bayes

Creating a text classifier with Naïve Bayes is based on Bayes Theorem. The existence of one feature in a class is assumed to be independent of the presence of any other feature by a Naïve Bayes classifier. They're probabilistic, which means they calculate each tag's probability for a

tight game" is Sports and the probability that it is Not Sports because Naïve Bayes is a probabilistic classifier. Then we choose the largest. P (Sports | a very close game) is the probability that a sentence's tag is Sports provided that the sentence is "A very tight game," written mathematically.

All of the features of the sentence contribute individually to whether it is about Sports, hence the term "Naïve."

The Naïve Bayes model is simple to construct and is especially good for huge data sets. It is renowned for outperforming even the most advanced classification systems due to its simplicity.

Stochastic Gradient Descent

Gradient descent is an iterative process that starts at a random position on a function's slope and goes down until it reaches its lowest point. This algorithm comes in handy when the optimal locations cannot be obtained by simply equating the function's slope to 0.

Suppose you have millions of samples in your dataset. In that case, you will have to use all of them to complete one iteration of the Gradient Descent, and you will have to do this for each iteration until the minima are reached if you use a traditional Gradient Descent optimization technique. As a result, it becomes computationally prohibitively expensive to carry out.

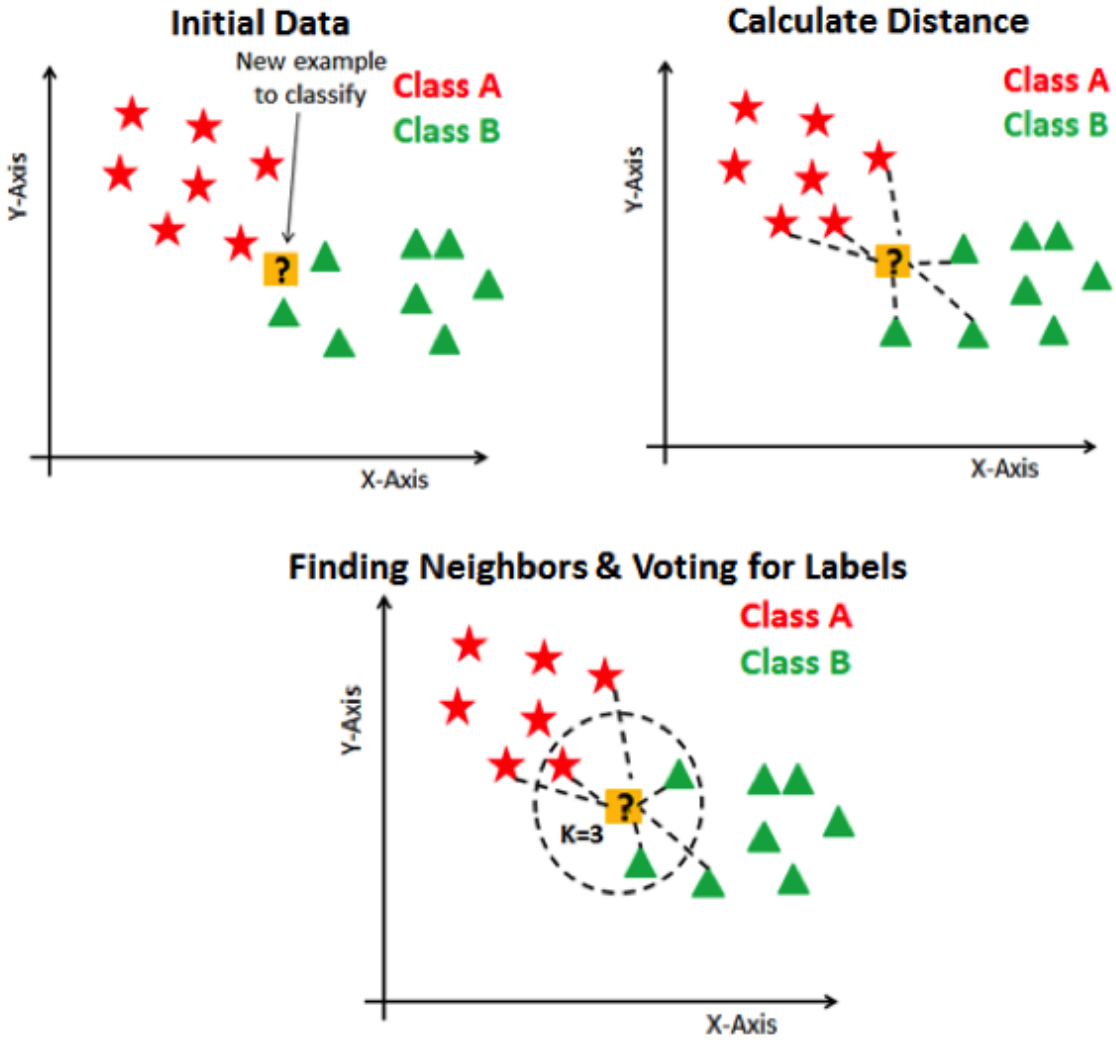
Stochastic Gradient Descent is used to tackle this problem. Each iteration of SGD is performed with a single sample, i.e., a batch size of one. The selection is jumbled and chosen at random to execute the iteration.

The neighborhood of data samples is determined by their closeness/proximity. Depending on the problem to be solved, there are numerous methods for calculating the proximity/distance between data points. Straight-line distance is the most well-known and popular (Euclidean Distance).

Neighbors, in general, have comparable qualities and behaviors, which allows them to be classified as members of the same group. The primary idea behind this simple supervised learning classification technique is as follows. For the K in the KNN technique, we analyze the unknown data's K-Nearest Neighbors and aim to categorize and assign it to the group that appears most frequently in those K neighbors. When K=1, the unlabeled data is given the class of its nearest neighbor.

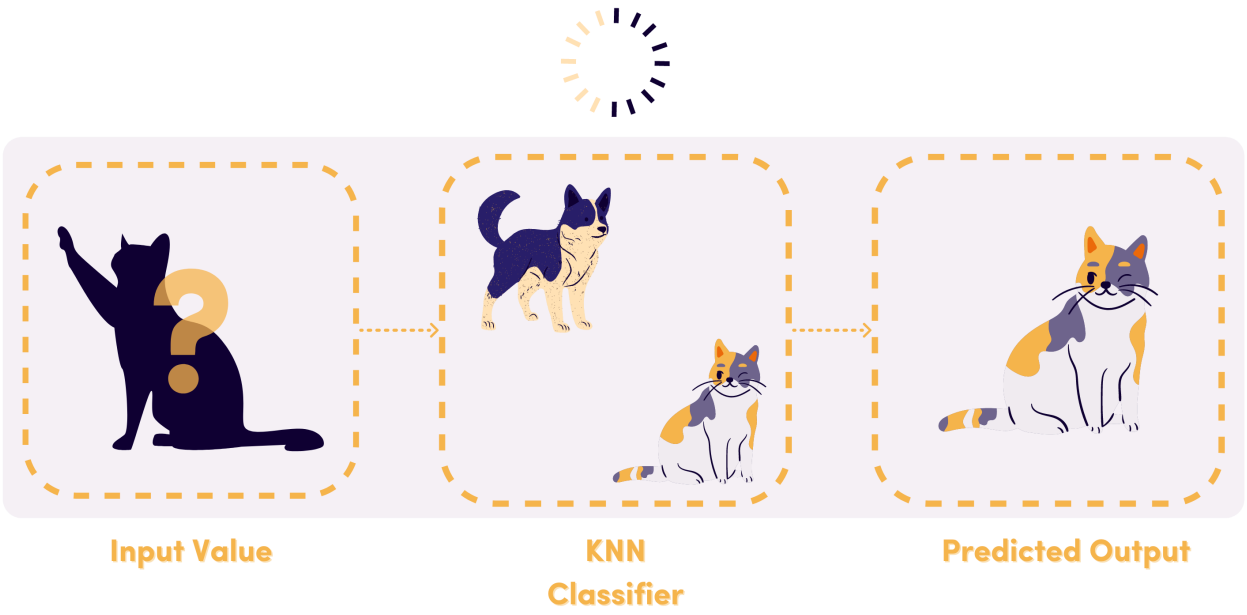
The KNN classifier works on the idea that an instance's classification is most similar to the classification of neighboring examples in the vector space. KNN is a computationally efficient text classification approach that does not rely on prior probabilities, unlike other text categorization methods such as the Bayesian classifier. The main computation is sorting the training documents to discover the test document's K nearest neighbors.

The example below from Datacamp uses the Sklearn Python toolkit for text classifiers.



Example of Sklearn Python toolkit being used for text classifiers.
Source:https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

As a basic example, imagine we are trying to label images as either a cat or a dog. The KNN model will discover similar features within the dataset and tag them in the correct category.

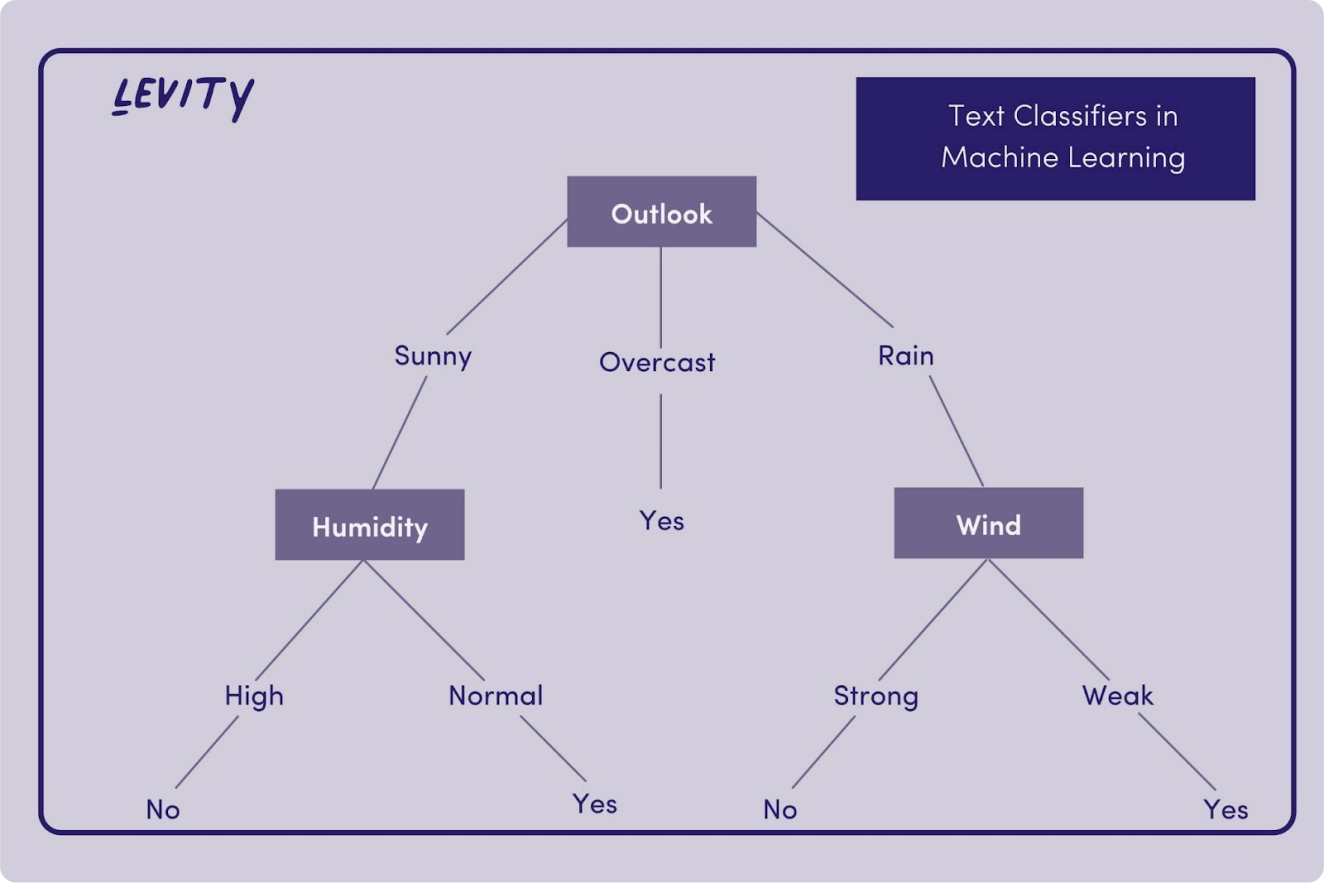


Example of KNN classifier labeling images in either a cat or a dog

Decision tree

One of the difficulties with neural or deep architectures is determining what happens in the Machine Learning algorithm that causes a classifier to select how to classify inputs. This is a significant issue in Deep Learning. We can achieve incredible classification accuracy, yet we have no idea what factors a classifier employs to reach its classification choice. On the other hand, decision trees can show us a graphical picture of how the classifier makes its decision.

A decision tree generates a set of rules that can be used to categorize data given a set of attributes and their classes. A decision tree is simple to understand as and users can visualize the data with minimal data

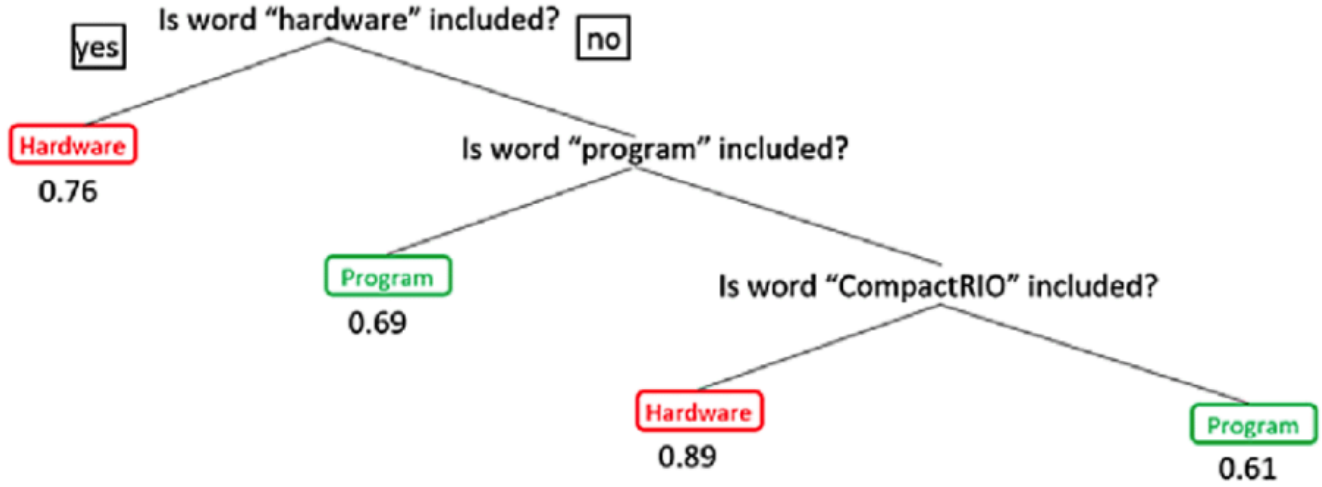


Text classifiers in Machine Learning: Decision tree

Random forest

The random forest Machine Learning technique solves regression and classification problems through ensemble learning. It combines several different classifiers to find solutions to complex tasks. A random forest is essentially an algorithm consisting of multiple decision trees, trained by bagging or bootstrap aggregating.

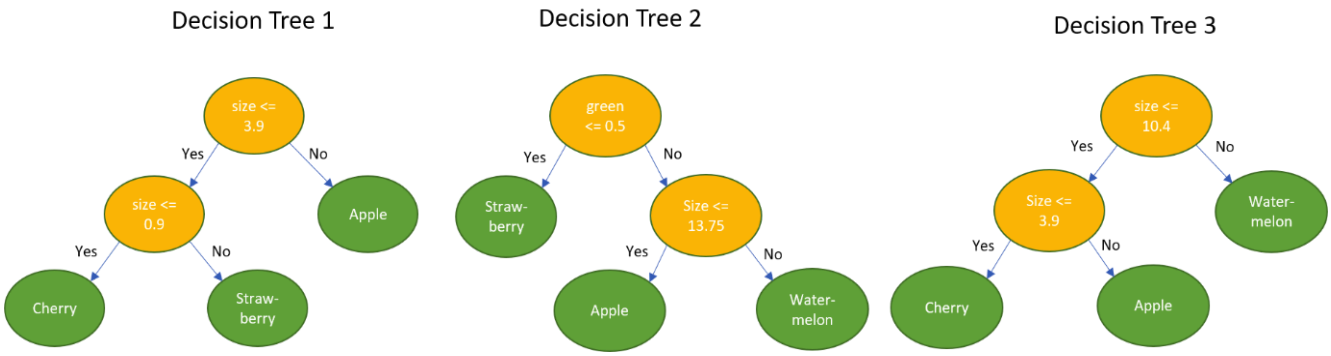
A random forest text classification model predicts an outcome by taking the decision trees' mean output. As you increase the number of trees, the accuracy of the prediction improves.



Text classifiers in Machine Learning: Random forest. *Source: https://medium.com/rapids-ai/accelerating-random-forests-up-to-45x-using-cuml-dfb782a31bea*

Support Vector Machine

For two-group classification issues, a Support Vector Machine (SVM) is a supervised Machine Learning model that uses classification techniques. SVM models can categorize new text after being given labeled training data sets for each category.



Support Vector Machine. *Source: https://www.simplilearn.com/tutorials/data-science-tutorial/svm-in-r*

They have two critical advantages over newer algorithms like Neural

suited to text classification issues, where it's not unusual to only have access to a few thousand categorized samples.

Evaluating the performance of your model

When you've finished building your model, the most crucial question is: how effective is it? As a result, the most vital activity in a Data Science project is evaluating your model, which determines how accurate your predictions are.

Typically, a text classification model will have four outcomes, true positive, true negative, false positive, or false negative. A false negative, as an example, might be if the actual class tells you that an image is of a fruit, but the predicted class says it is a vegetable. The other terms work in the same way.

After understanding the parameters, there are three core metrics to evaluate a text classification model.

Accuracy

The most intuitive performance metric is accuracy, which is just the ratio of successfully predicted observations to all observations. If our model is accurate, one would believe that it is the best. Yes, accuracy is a valuable statistic, but only when the datasets are symmetric and the values of false positives and false negatives are almost equal. As a result, other parameters must be considered while evaluating your model's performance.

The ratio of accurately predicted positive observations to total expected positive observations is known as precision. For example, this measure would answer how many of the images identified as fruit actually were fruit. A low false-positive rate is related to high precision.

Recall

A recall is defined as the proportion of accurately predicted positive observations to all observations in the class. Using the fruit example, the recall will answer how many images we label out of those images that are genuinely fruit.

Learn more about precision vs recall in Machine Learning.

F1 Score

The weighted average of Precision and Recall is the F1 Score. As a result, this score considers both false positives and false negatives. Although it is not as intuitive as accuracy, F1 is frequently more valuable than accuracy, especially if the class distribution is unequal. When false positives and false negatives have equivalent costs, accuracy works well. It's best to look at both Precision and Recall if the cost of false positives and false negatives is considerably different.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

It is sometimes useful to reduce the dataset into two dimensions and plot the observations and decision boundary with classifier models. You can visually inspect the model to evaluate the performance better.

No-code AI involves using a development platform with a visual, code-free, and typically drag-and-drop interface to deploy AI and Machine Learning models. Non-technical people may quickly classify, evaluate, and develop accurate models to make predictions with no coding AI.

Building AI models (i.e. training Machine Learning models) takes time, effort, and practice. No-code AI reduces the time it takes to construct AI models to minutes, allowing businesses to incorporate Machine Learning into their processes quickly. According to Forbes, 83% of companies think AI is a strategic priority for them, yet there is a shortage of Data Science skills.

There are several no-code alternatives to building your models from scratch.

HITL – Human in the Loop

Human-in-the-Loop (HITL) is a subset of AI that creates Machine Learning models by combining human and machine intelligence. People are involved in a continuous and iterative cycle where they train, tune, and test a particular algorithm in a classic HITL process.

To begin, humans assign labels to data. This provides a model with high-quality (and large-volume) training data. From this data, a Machine Learning system learns to make decisions.

The model is then fine-tuned by humans. This can happen in a variety of ways, but the most typical is for humans to assess data to correct for overfitting, teach a classifier about edge cases, or add new categories to the model's scope.

Finally, users can score a model's outputs to test and validate it, especially

The constant feedback loop allows the algorithm to learn and produce better results over time.

Multiple labelers

Use and change various labels to the same product based on your findings. You will avoid erroneous judgments if you use HITL. For example, you will prevent an issue by labeling a red, round item as an apple when it is not.

Consistency in classification criteria

As discussed earlier in this guide, a critical part of text classification is ensuring models are consistent and labels don't start to contradict one another. It is best to start with a small number of tags, ideally less than ten, and expand on the categorization as the data and algorithm become more complex.

Summary

Text classification is a core feature of Machine Learning that enables organizations to develop deep insights that inform future decisions.

- Many types of text classification algorithms serve a specific purpose, depending on your task.
- To understand the best algorithm to use, it is essential to define the problem you are attempting to solve.
- As data is a living organism (and so, subject to constant change),

- No-code Machine Learning is an excellent alternative to building models from scratch but must be actively managed with methods like Human in the Loop for optimum results.

Using a no-code ML solution like Levity will take away the problem of deciding on the proper structure and building your text classifiers yourself. It will let you use the best of what both human and ML power offer and create the best text classifiers for your business.