Back to all blogs ⟶

CONTENTS

All production ML models need monitoring. NLP models are no exception. But, monitoring models that use text data can be quite different from, say, a model built on tabular data.

In this tutorial, we will dive into a specific example. We will explore issues affecting the performance of NLP models in production, imitate them on an example toy dataset, and show how to monitor and debug them.

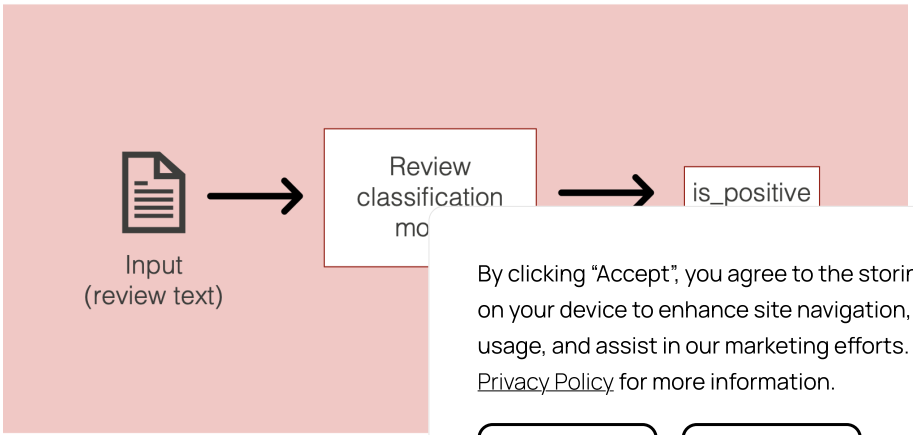
We will work with a drug review dataset and go through the following steps:

- Train a simple review classification model, and evaluate its quality on a validation dataset;
- Imitate data quality issues, test their impact on the model accuracy, and explore how one can identify them in advance;
- Apply the model to the new data, and explore how to detect and debug model quality decay when applied to previously unseen inputs.

We will use the Evidently open-source Python library to evaluate and debug model issues.

You can reproduce the steps and explore additional details in the example Jupyter notebook.

The use case



Let's imagine that you want to **classify reviews of medications.**

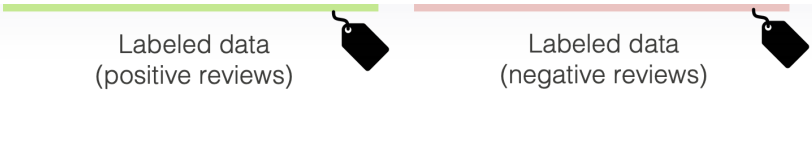
This NLP use case is common in e-commerce. For example, users might leave reviews on the online pharmacy website. You might want to assign a category to each review based on its content, such as "side effects," "ease of use," or "effectiveness." Once you create a model, you can automatically classify each newly submitted review. Tags will improve the user experience, helping find relevant content faster.

You might use a similar classification model in other scenarios. For example, to surface relevant information and enrich user experience in a healthcare-focused chat app. In this case, you'd likely classify the reviews in batches and store them in some database. You can retrieve them on demand to surface the content to the user.

Let's take this use case as an inspiration and start with a simpler classification model. Our goal is to predict whether **the overall review sentiment** is highly positive or negative.

Creating a model

To solve this problem, you first need a labeled dataset.



For illustration purposes, we will work with a drug review dataset from the UCI repository.

> **Disclaimer:** this created model is used solely for research and educational purposes to illustrate the ML model evaluation and monitoring process. It should not be used in any other form or purpose or to inform any actual decisions.

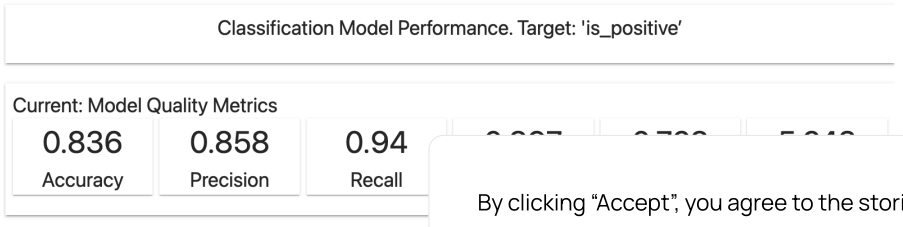
The dataset is fairly large. We will start with one particular subset: reviews of painkiller medications. We will split them into two parts. 60% goes to the "training" partition. The other 40% is the "validation" part.

We will train a model to distinguish between reviews with ratings "1" (negative review) and "10" (positive review), making it a simple binary classification problem.

In practice, you often have limited labeled data. It is not unusual to start with a dataset representing only a subset of the data to which the model might eventually be applied.

Once we train the model, we can evaluate its accuracy on the validation dataset. Here is what we got: the accuracy on the validation dataset is 0.836. Let's consider it to be good enough for our demo purposes.

We can expect similar quality in production on similar data. If the accuracy falls considerably below this level, we should react and dig deeper into what is happening.



Note: this is a simple demo. If you a
forget about cross-validation to m
about your model quality.

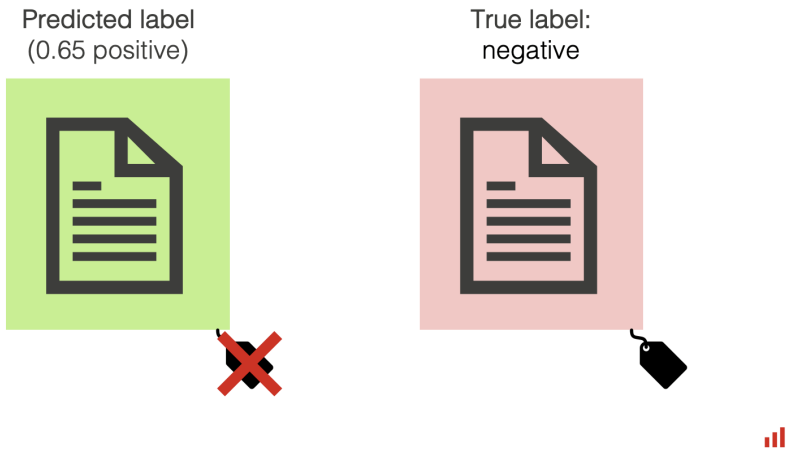
Model in pra

Model in production

Once we put the model in production, we apply it to the new, unseen data.

In the e-commerce example, we will likely wrap the model as an API. We will call the model once a new review is submitted on the website and assign a category to display based on the model's response. In the chat app scenario, we will likely perform batch scoring. We will write the new predictions with assigned labels to a database.

In both cases, you typically do not get immediate feedback. There is no quick way to know if the predicted labels are correct. However, you do need *something* to keep tabs on the model's performance to ensure it works as expected.



There are different ways to understand if the model is doing well:

- **You can have a feedback mechanism directly in the website UI.** For example, you can allow the review authors or readers to report incorrectly assigned categories and suggest a better one. If you get a lot of reports or corrections, you can react to this and investigate.
- **Manual labeling as quality control.** In the simplest form, the model creator can look at some of the model predictions to see if it behaves as expected. You can also engage external labelers from time to time to label a portion of the data. This way you can directly evaluate the quality of the mo assigned labels.

In both cases, the model checks ar address any model quality issues a the accuracy.

While you might often accept som

cost of error is tolerable), trying to detect the issues in advance is a good practice.

The two frequent culprits of model quality decay are data quality issues and changes in the input data distributions. Let's explore how one can detect those!

Example 1: data quality and integrity

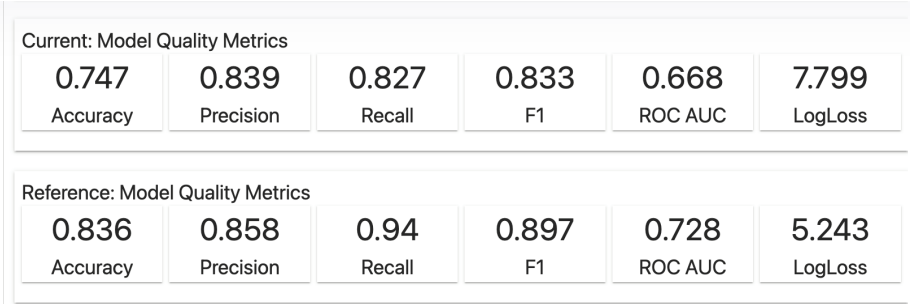
Data quality issues come in all shapes and sizes. For example, you might have some bugs in the input data processing that leak HTML tags into the text of the reviews. Data might also be corrupted due to wrong encoding, the presence of special symbols, text in different languages, emojis, and so on. There might be bugs in the feature transformation code, post-processing, or cleaning steps that you run as part of a scoring pipeline.

In our case, we artificially changed the dataset. We took the same validation dataset and made a few changes: injected random HTML tags and translated some reviews into French. The goal was to "break" the dataset, imitating the data quality issues.

You can see the complete cod

Now, let's check the model quality

Classification Model Pe



The model quality is below what was seen in the initial validation on the "clean" dataset. The accuracy is only 0.747.

How can we troubleshoot this decay? Had it occurred in practice, our next step would be to dig into the model's performance and data to understand what is happening. Let's have a look!

We'll use Evidently Python library. It contains various evaluation metrics and tests and helps generate interactive reports for different scenarios.

In this case, we will create a custom report by combining several evaluations we want to run to understand data changes.

To apply Evidently, we first need to prepare the data and map the schema so that Evidently can parse it correctly. This is called "column mapping." We re-use it across all our evaluations since the data schema stays the same.

Here is how we point to the columns with the predictions, target values and specify that the column with reviews should be treated as the text column:

```
column_mapping = ColumnMapping()

column_mapping.target = 'is_positive'
column_mapping.prediction = 'predict_proba'
column_mapping.text_features = ['review']
```

Next, we generate the report. To do that, we need to do the following:

- pass our original validation da
comparison) and the modifie
- specify the types of evaluatio
in the report,
- call the visual report to explor

In our case, we choose the evaluat

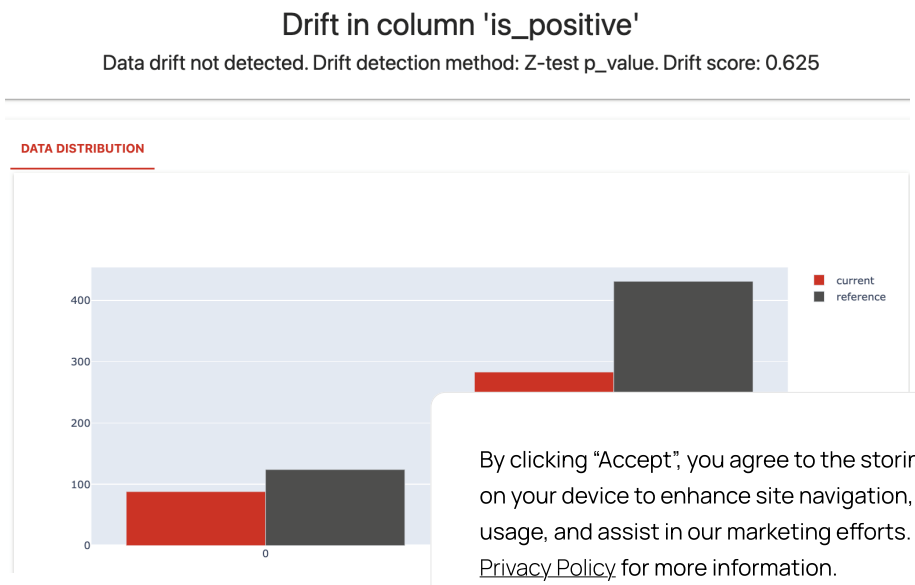
First, we want to see if the model outputs have changed. Second, we want to see if the input text has changed.

There are a few ways to evaluate the similarity between text datasets. One way is to compare the descriptive statistics of the text data (such as length of text, the share of out-of-vocabulary words, and the share of non-letter symbols) and explore if they have shifted between the two datasets. This option is available in Evidently as the **Text Descriptors Drift** metric. We will include it in the combined report in addition to evaluating drift in the model predictions and target.

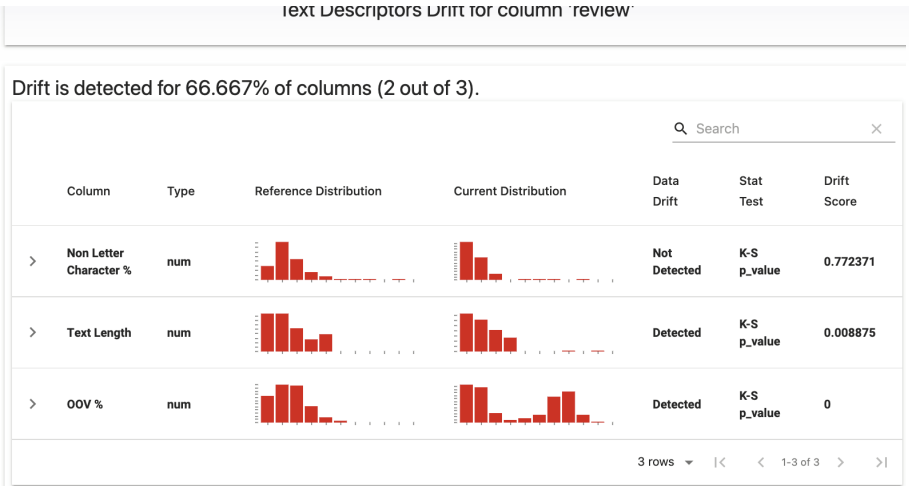
Here is how you can call it:

```
data_drift_report = Report(
   metrics=[
       ColumnDriftMetric('is_positive'),
       ColumnDriftMetric('predict_proba'),
       TextDescriptorsDriftMetric(column_name='review'),
   ]
)
data_drift_report.run(reference_data=reference,
                      current_data=valid_disturbed,
                      column_mapping=column_mapping)
data_drift_report
```

Once we display the report, one can see no drift in the true labels or predicted probabilities.



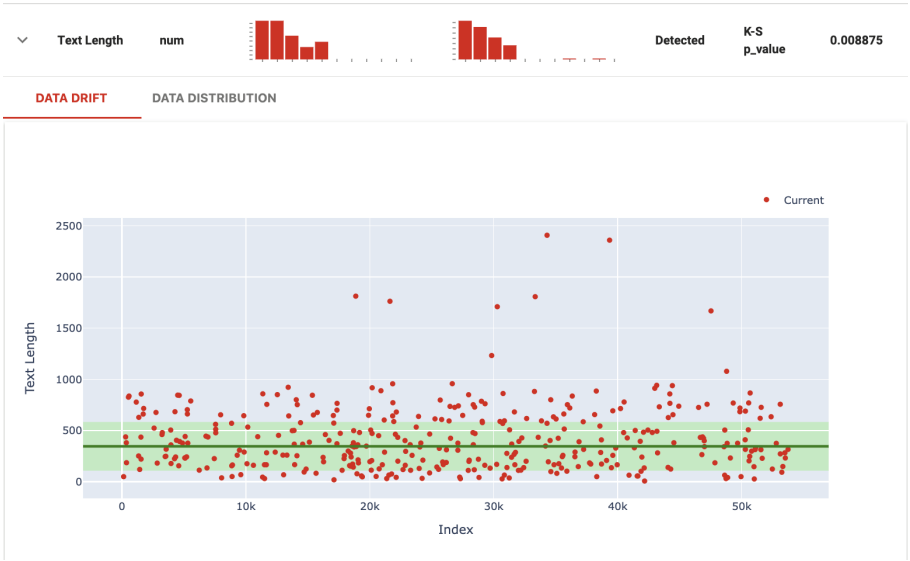
But some input text properties are



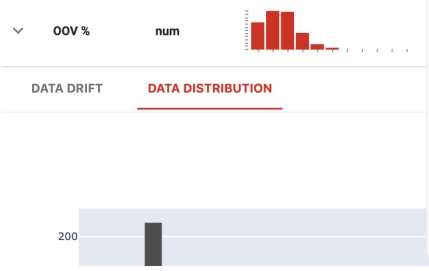
Under the hood, Evidently calculates these descriptors and applies different statistical tests and distance metrics to examine if there is a significant shift between the two datasets.

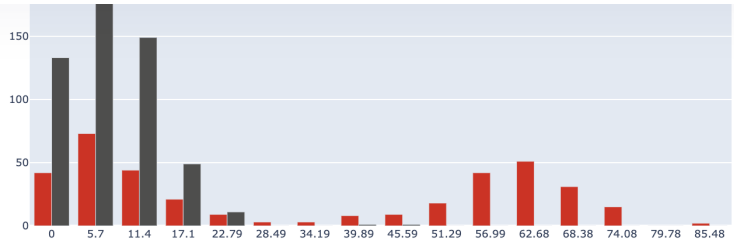
In particular, it points out a **change in the distribution of text length**. If we expand the details in the report, we can see some additional plots that help understand the shift.

Some reviews are now suspiciously long:



The vocabulary has also shifted. Multiple reviews contain over 30% of out-of-vocabulary words:





These findings can help us find examples of changes to understand what is going on. For instance, we can query our dataset for all the long reviews with over 1000 words and reviews with over 30% of out-of-vocabulary words.

Once we surface the examples, we can quickly see what is going on here:

- Texts containing HTML tags directly in the body are passed to the model
- The reviews are in a new, unexpected language

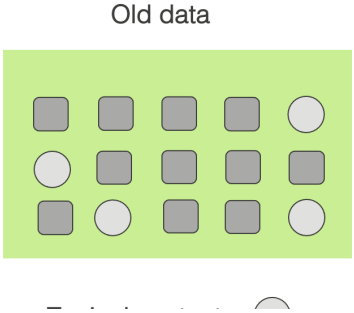
Here is one of the query results:

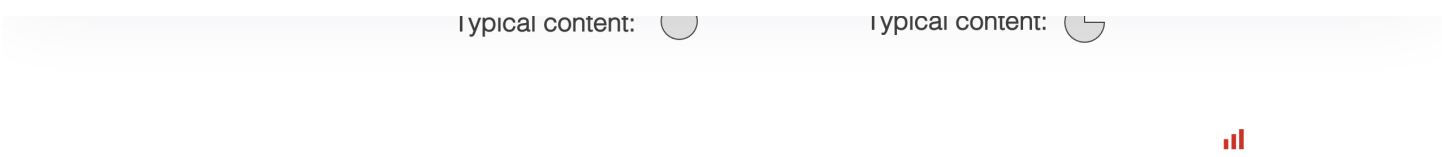
```
valid_disturbed[valid_disturbed['oov_share'] > 30].head()
```

	review	is_positive	predict_proba	text_length	oov_share
19899	"Chaque fois que j'ai des douleurs dues à un e...	1	0.0	649	72.641509
1558	"Je prends du dilaudid pour une grave maladie ...	1	1.0	857	62.162162
48578	"Je prends ce médicament depuis 3 ans pour sou...	1	1.0	342	74.193548
18574	"Pendant que j'étais aux urgences, le diagnost...	1	1.0	349	63.157895
21625	"I have been on 4-5, 15 mg oxycodones a day fo...	1	1.0	1763	42.380952

Knowing what exactly has happened, we can now resolve the issue with the data engineering team (to sort out the pipelines) and the product team (to make sure that the reviews in French are expected, and it is time to create a separate model for those).

Example 2: data distribution change





There is another type of change that might occur in production: change in the **content** of the texts the model is tasked to analyze. Such a shift can ultimately lead to model quality degradation or **model drift**. It can come in different forms.

One is **concept drift,** when some concepts the model learns evolve. For example, some words or symbols can gradually change their meaning. Maybe some emoji previously representative of a "positive" review is now frequently used with the opposite intention. Or perhaps there is a second new drug on the market with the same active ingredient, which converts one "concept" into two different ones.

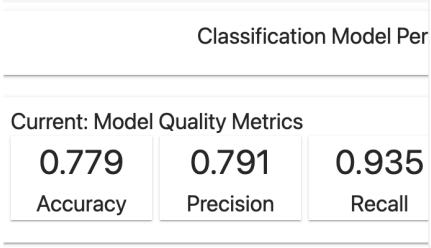
Another is **data drift,** when the model is applied to new data different from the training. The relationships the model has learned still hold. But it hasn't seen anything related to the patterns on the latest data and thus cannot score it that well. For example, you will observe data drift if you apply the model trained to classify medical reviews to other products.

Understanding the difference between data and concept drift is useful when interpreting the changes. However, to detect them, we would typically use the same approach. If you already have the labels, **the true model quality** (e.g., accuracy) is the best measure of model drift. If you do not have the labels or want to debug the quality drop, you can look at the **change in the input data and prediction** and then interpret it using your domain understanding.

Let's return to our example dataset and see how the model drift can look in practice.

We will now apply our model to a new, unseen dataset. We will use a different category of drug reviews: they are no longer related to the painkiller medication but instead to the anti-depressant drugs. We may still expect reasonable quality: reviewers could use overlapping words to describe whether or not some medication works.

The model does not fail completely

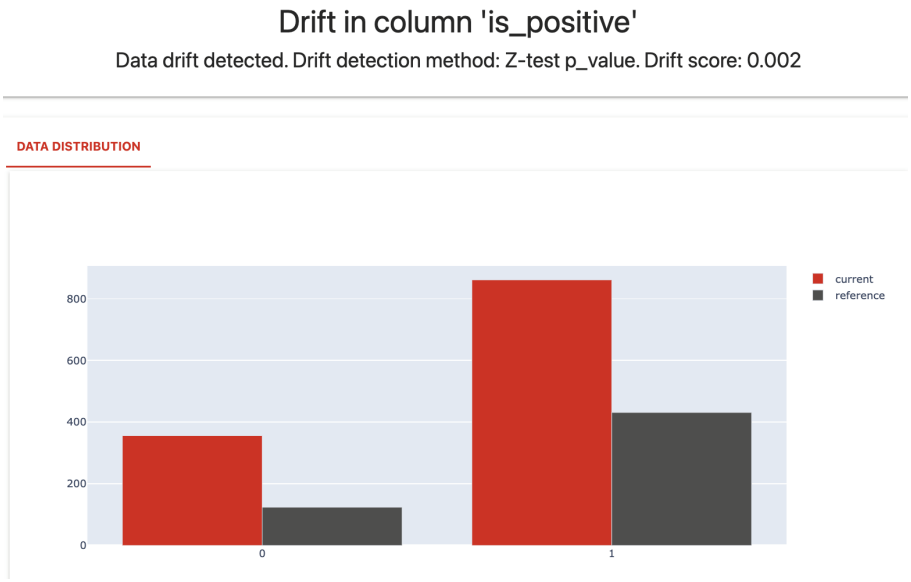


Reference: Model Quality Metrics

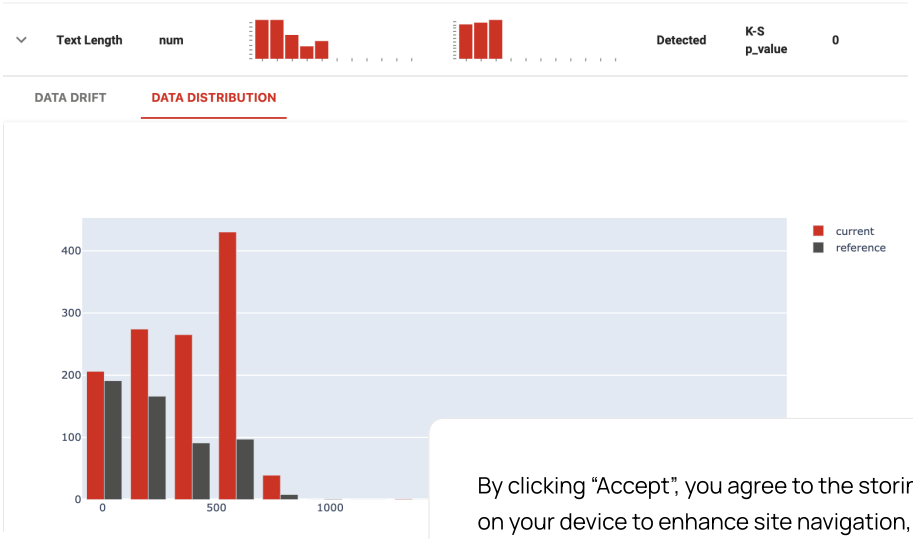
0.836	0.858	0.94	0.897	0.728	5.243
Accuracy	Precision	Recall	F1	ROC AUC	LogLoss

This is lower than expected. Let's investigate!

We can again generate the drift report and will immediately notice some changes. Notably, the distribution of labels has drifted.



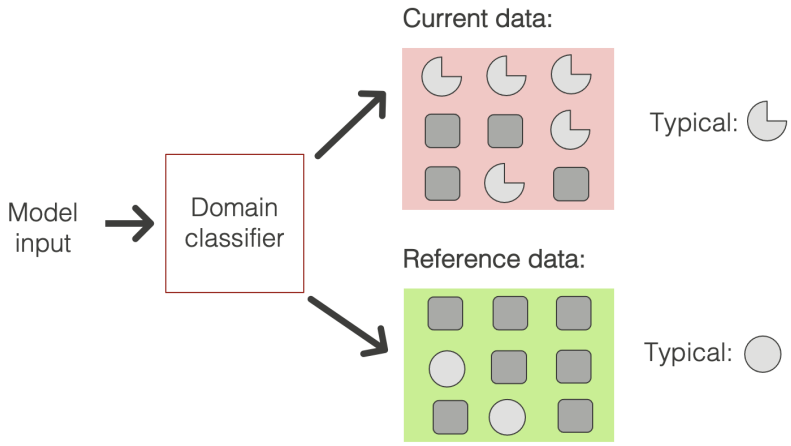
Reviews are also longer in the current dataset, and OOV words appear more often. But there is nothing as obvious as in the case above.



We can try something else to debu
comparing text stats, look to evalu
changed.

There are many methods to detect data drift. With tabular data, you'd typically look at the distributions of the individual features in the dataset. With text data, this approach is not so convenient: you probably do not want to count the distribution of each word in the dataset. There are simply too many, and the results will be hard to interpret.

Evidently applies a different approach for text drift detection: a domain classifier. It trains a background model to distinguish between the reference and the current dataset. The ROC AUC of the binary classifier shows if the drift is detected. If a model can reliably identify the reviews that belong to the current or reference dataset, the two datasets are probably sufficiently different.



This approach, among others, is described in the paper "Failing loudly: An Empirical Study of Methods for Detecting Dataset Shift."

It is not without caveats. If you have some temporal information in the new dataset (for example, each review includes the date), the model might quickly learn to distinguish between the datasets. This might be simply because one of them contains the word "March" and another "February," or due to the mention of the Black Friday promotions. However, we can evaluate this by looking at the top features of the domain classifier model and some examples.

If the text data drift is detected, Evidently will automatically provide some helpful information:

- **Typical words in the current** are most indicative when prec belongs to.
- **Examples of texts** from curre the easiest for a classifier to l probabilities being very close

To use this method, we will create a new report and include the metric that helps detect drift in a given column. For columns containing text data, domain classifier is the default method.

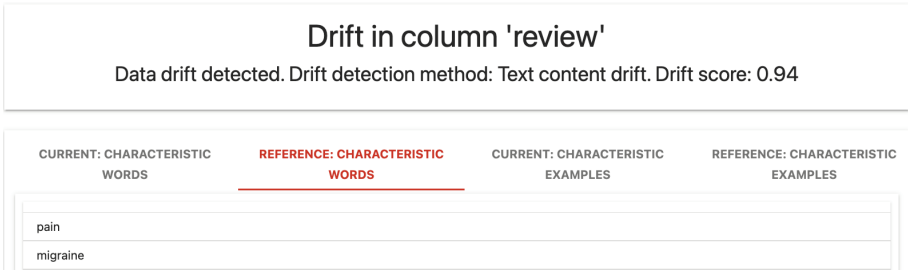
```
data_drift_dataset_report = Report(metrics=[
   ColumnDriftMetric(column_name='review')
])

data_drift_dataset_report.run(reference_data=reference,
                              current_data=new_content,
                              column_mapping=column_mapping)
data_drift_dataset_report
```

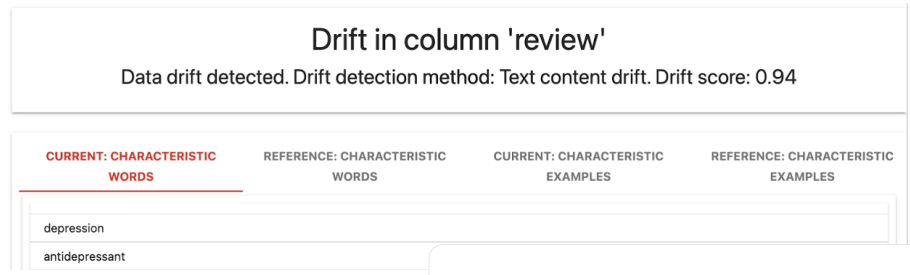
Here is what it shows for our dataset.

First, it does indeed detect the distribution drift. The classifier model is very confident and has a ROC AUC of 0.94. Second, the top distinctive features very explicitly point to the possible change in the contents of the text.

The reference dataset contains words like “pain” and “migraine.”



The current dataset has words like “depression” and “antidepressant.”



The same is clear from the specific
different groups of drugs, and the
describe whether a particular med
mood” differs from “relieve pain,” m
classify the review’s sentiment.

Once we identify the reason for model drift, we can devise a solution: typically, retrain the model using the newly labeled data.

How to apply this in production?

In this toy example, we showed the debugging workflow. We measured the factual model accuracy and dug deeper to identify the reasons for the quality drop.

In practice, you can perform data quality checks proactively. For example, you can implement this early quality control step in your batch scoring pipeline. You can test your data to surface potential issues before you get the actual labels or even score the model.

If you detect issues like the HTML tags in the body of the review, you can take immediate action to resolve them: by updating and re-running the pre-processing pipeline.

You can do the same for data drift checks. Every time you get a new batch of data, you can evaluate its key characteristics and how similar it is to the previous batch.

If you detect drift and see that it is indeed due to new types of content or topics appearing, you can also take proactive steps. In this case, it most likely means initiating a new labeling process and subsequent model retraining.

What’s next?

What's next?

Evaluating text data drift can involve other challenges. For example, you might need to monitor drift in embeddings instead of raw text data. You can also run additional tests and evaluations, for example, related to the model's robustness and fairness.

Sign up here if you want to get updates on the new hands-on tutorials and new feature releases.

WRITTEN BY

Natalia Tarasova
Senior ML engineer
Guest author

Elena Samuylova
Co-founder and CEO
Evidently AI

SHARE ON

#data-drift #data-quality #nlp #text-data #code-example #llm

You might also like