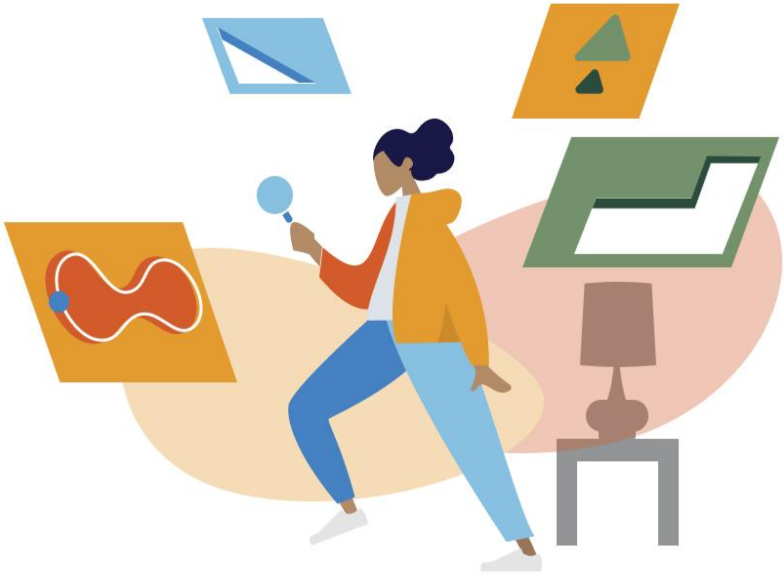
Book Popularity

CodeOp Data Science Bootcamp

By Andy Pereira



October 2022

Content

THE PROBLEM

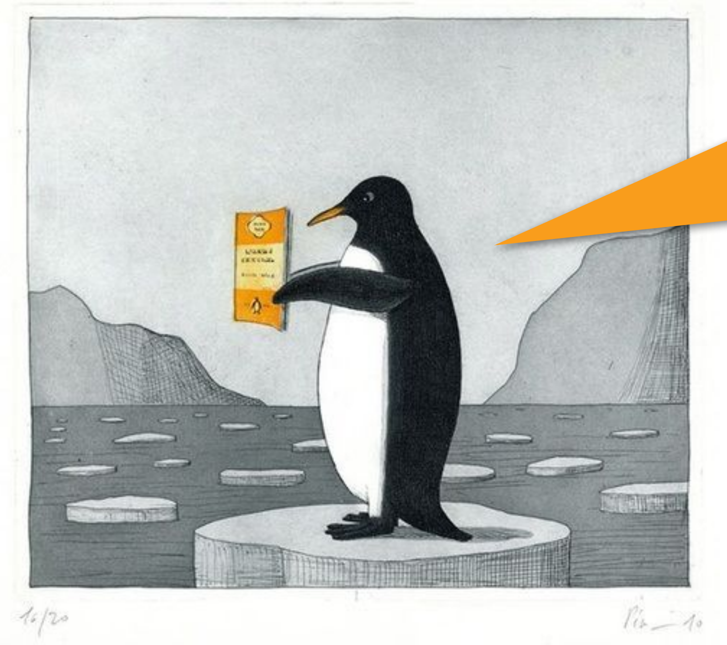
THE SOLUTION

CONCLUSION

The Problem

The Problem

The user problem



As a **publisher**,
when **marketing** a new book, I want to determine the **title** and **description**
that will **maximise the number of readers**.

The Solution

The Solution

We found a solid dataset

- Available on Kaggle
- Created by scraping data using the Goodreads API
- 10 million books available


```
{
   "Id": "5107",
   "Name": "The Catcher in the Rye",
   "RatingDist1": "1:133165",
   "RatingDist2": "2:224884",
   "RatingDist3": "3:553476",
   "RatingDist4": "4:808278",
   "RatingDist5": "5:891037",
   "pagesNumber": 277,
   "RatingDistTotal": "total:2610840",
   "PublishMonth": 30,
   "PublishDay": 1,
   "Publisher": "Back Bay Books",
   "CountsOfReview": 44046,
   "PublishYear": 2001,
   "Language": "eng",
   "Authors": "J.D. Salinger",
   "Rating": 3.8,
   "ISBN": "0316769177",
  "Count of text reviews": 55539,
   "Description": "The hero-narrator of The Catcher in the Rye is an ancient child of sixteen, a native New
Yorker named Holden Caulfield. Through
   circumstances that tend to preclude adult, secondhand description, he leaves his prep school in
Pennsylvania and goes underground in New York City for
   three days. "
}
```

Our plan for the dataset:

- We will apply **NLP** techniques to understand whether a book's **Name** and **Description** can **predict the number of reviews.**
    - From the 10 million books available, only **1.8 million contained the column "Description".**
- We decided to work with the **number of reviews** (vs. rating) as an indicator of a book's success, because it is **more directly related to the number of readers**.
- We started with simple **regression models**, but the outliers were triggering an **overestimation of the total amount of reviews**.

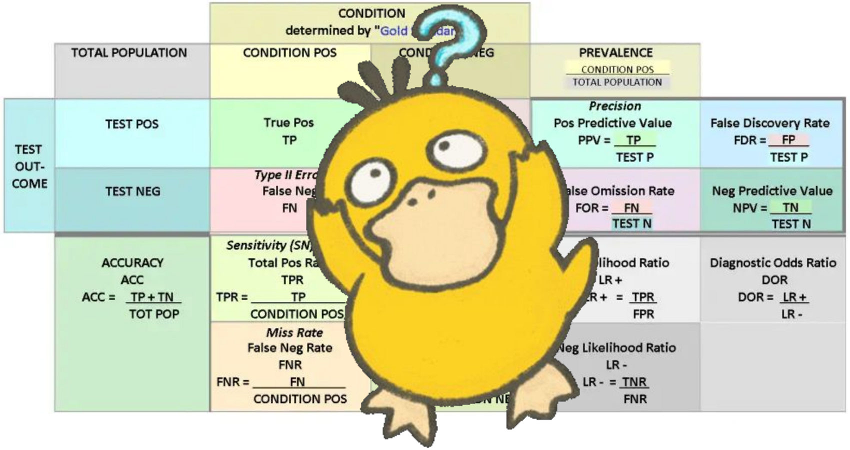
So we changed the task into a classification problem.

- We defined **two classes**: False (**unpopular**) and True (**popular**). The objective of the model was to predict whether a book will be popular.

- We defined a **minimum threshold of 500 reviews** to indicate popularity.

- After defining the threshold, we found out that our dataset is **unbalanced** as **only 7% of the books were popular.**

We selected precision as our main metric.



- Optimising for **recall** would reduce the **opportunity cost** of books that could have been a success, but we didn’t invest on (false negatives).
- **Higher precision**, will stops us, as a Publisher, from incurring in the **real loss** of investing in books that won’t be successful (false positives).

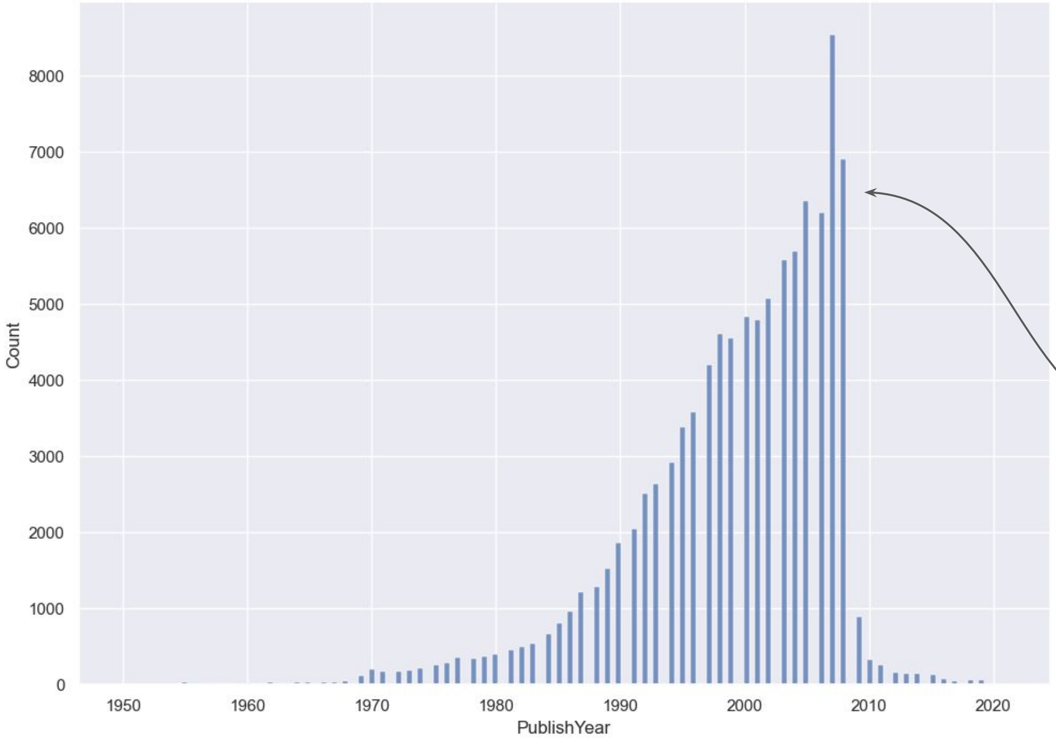
The Solution

Out of curiosity, we started with a Logistic Regression, using PublishYear and PagesNumber as features.

The results were underwhelming:

	precision	recall	f1-score	support
False	0.93	1.00	0.96	95391
True	0.00	0.00	0.00	7129
accuracy			0.93	102520
macro avg	0.47	0.50	0.48	102520
weighted avg	0.87	0.93	0.90	102520

We decided not to use publishing year, the data was noisy



What happened here?!

So we went back to our original features: Title and Description

- The first step was to vectorise both features with CountVectorizer.
- We got a precision score of **0.54.**
- However the model was very **compute-intensive**, which made it hard to iterate upon.
- Additionally, we discovered there was a **leak**. We had **duplicated books**, which was probably increasing the precision erroneously.

The Solution

We tuned the hyperparameters and reduced dimensionality

```
column_trans = ColumnTransformer([
    ('description_bow', CountVectorizer(stop_words={'english'},
                                        max_df=0.05,
                                        min_df=50), 'Description'),
    ('title_bow', CountVectorizer(stop_words={'english'},
                                  max_df=0.05,
                                  min_df=50), 'Name'),
])
model = Pipeline([
    ('column_transformer', column_trans),
    ('svd', TruncatedSVD(n_components=50)),
    ('classifier', linear_model.LogisticRegression(random_state=42)),
])
```

- We set up initial hyperparameters for **stop_words**=english, **max_df**=0.05, and **min_df**=50.
- We tried using **PCA**, but it's not sparse matrix friendly, so we went with **TruncatedSVD** for dimensionality reduction.
- We configured the model as a **Scikit-Learn Pipeline** (avoids data leakage and helps us tune hyperparameters).
- We also did **hyperparameter tuning using cross-validation**. We explored countvectorizer parameters as well as SVD number of components.

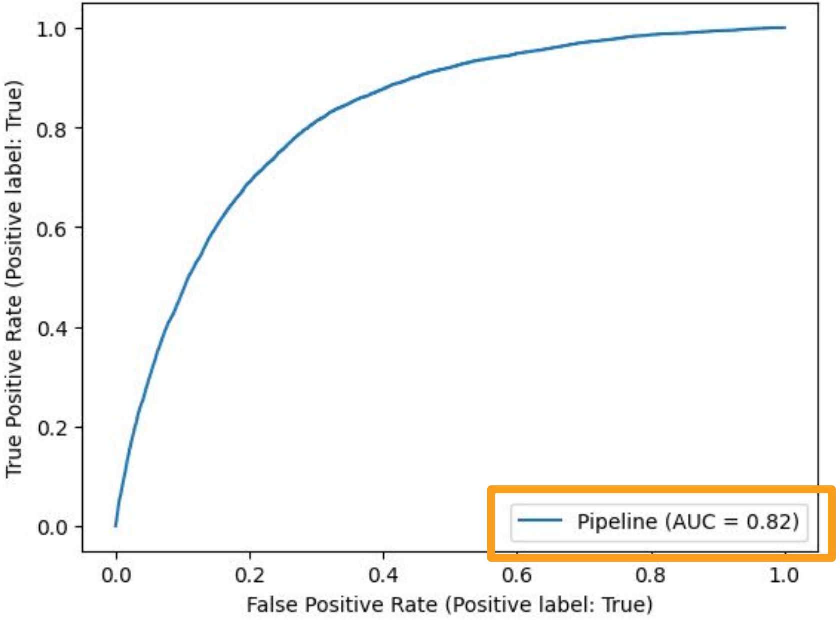
The Solution

And here's our best model:

We got a precision of **0.4** in our validation step and **0.41** in our test set!

	precision	recall	f1-score	support
False	0.93	1.00	0.96	95391
True	0.41	0.04	0.08	7129
accuracy			0.93	102520
macro avg	0.67	0.52	0.52	102520
weighted avg	0.90	0.93	0.90	102520

And here's our beautiful ROC curve:



After obtaining our best model, we proceeded to dig deeper on the errors, focussing on FPs:

We noticed that the **median number of total reviews for FPs was significantly higher than for TFs**.

49

median total reviews for **false positives**

3

median total reviews for **true falses**

Conclusion

Final Thoughts

- We showed the potential of this open and public dataset to create business applications with the use of ML.
- It’s possible to have a model with **41% precision**.
- Even in the case of false positives, **the model seems to indicate more popular books over unpopular ones**.
- As the next steps, we recommend to invest time into **further error analysis** before looking into more complex models or feature engineering.
- Link to the full project on GitHub.