In a standard classification problem, the algorithm would learn to correctly assign one class label to one example. The “20newsgroup” dataset consists of document news with 20 different categories and each document corresponds to a single category (label). When it comes to the Reuters dataset, this is a multi-class/multi-label classification because some documents have more than one or two-class labels.

Since the train data are less than 100K, I used the non-probabilistic *LinearSVC* and the probabilistic *Multinomial Naïve Bayes* to train my models. Before viewing the evaluation results, I suspected that the LinearSVC would be more effective, since this is a classification task and using probabilistic algorithms to evaluate data provides additional uncertainty in the prediction.

Reuters21578: For this dataset I used the multi-class mode OneVStheRest (LinearSVC does not include multiclass support by default). The probabilistic Multinomial Naïve Bayes can also address multiclass classification problems.

DATA

First, I parsed the files ending with *.sgm*. Each file includes multiple reuters documents and I stored those separately in a list. Initially I split the train-test set with the sklearn module, but I finally changed it to “The Modified Apte ("ModApte") Split” according to the READme file in the reuters dataset so that it can be compared to other baselines. To be more specific, the train set includes articles with the LEWISSPLIT="TRAIN" tag and the test set articles with the LEWISSPLIT="TEST" tag. Articles without labels are ignored due to ambiguity: some documents lack TOPICS categories because no TOPICS apply (i.e., the document is a true negative example for all TOPICS categories), but it appears that other simply were never assigned TOPICS categories by the indexers (according to the READme). This split is recommended if we want to predict the TOPICS field, since the evidence is that a significant number of documents that should have TOPICS do not. I used BeautifulSoup to extract the topics (labels) from each document, as well as the text of the xml type files.

Time for data parsing: 11.46 seconds

FEATURE EXTRACTION

TfidfVectorizer is implemented for feature extraction. After testing (later on) the model with multiple parameters, the best results were achieved by only extending the default token pattern of the vectorizer to *'(?u)\b[A-Za-z]+\b'.* By excluding stop-words and using different n-gram sequence other than default (1,1), the performance declined.

In order to handle properly this multilabel-classification problem where each sample can have multiple labels for possible classes, I used the *MultiLabelBinarizer* from sklearn. The list that contains the document labels may contain nested lists (or tuples) since one document could correspond to multiple labels. This makes process to create a proper matrix difficult. So, if we could imagine that as a dataframe, each column would correspond to one label and each row to a Reuters document. In the final frame output, if a topic existed for a specific document, then the corresponding cell in the frame would be assigned to 1, otherwise to 0.
This binary labeling is transformed to an np array.

Time for feature extraction: 2.57 seconds

MODEL TRAIN

The Multinomial Naïve Bayes model was trained in 0.605 seconds and the OneVSRest LinearSVC model was trained in 3.085 seconds. After testing different parameters, only the *alpha* parameter when assigned the lowest value possible (less than 0.1) seemed to make a difference and improve the model. The default parameters for the non-probabilistic model presented the best results.

The prediction time for both models was 0.21 seconds.

EVALUATION

The tables below present the best results of our two models. When used the stop-words and different than 1,1 n-gram sequence, the results were significantly different, such as 0.39 accuracy and 0.05 F1 score for the probabilistic model and 0.69 accuracy and 0.76 F1 score for the non-probabilistic.

Probabilistic Model

	Precision	Recall	F1-score
Micro avg	0.68	0.78	0.73
Macro avg	0.22	0.26	0.23

Accuracy of the probabilistic model: 0.69

Hamming loss of the probabilistic model: 0.005

Zero-one loss of the probabilistic model: 0.301

Non-Probabilistic Model

	Precision	Recall	F1-score
Micro avg	0.95	0.76	0.85
Macro avg	0.46	0.27	0.32

Accuracy of the non-probabilistic model: 0.79

Hamming loss of the non-probabilistic model: 0.002

Zero-one loss of the probabilistic model: 0.204

The non-probabilistic model seems to be more efficient for this multi-label classification problem and the F1 score also differs in micro and macro avg. In the non- probabilistic model, precision is high, meaning that it identifies many labels correctly. I think though that high precision and low recall could also mean high bias, especially since we know that the dataset in imbalanced. The macro averaging gives equal weight to every class and the classifier is forced to distinguish all classes and not rely on the distribution of the classes (so it pays attention to distinguishing classes). It seems that both classifiers struggle to identify small classes. They are more effective though on the large classes and more instances are predicted correctly. Also, in the probabilistic model, recall is higher than precision and the reverse is true for the non-probabilistic model. High precision means that the output is relevant but high recall means that more results are relevant.

Hamming loss is very low in both models, meaning that not many labels (individually) are predicted incorrectly. Hamming loss is more forgiving as a metric because it does not penalize the whole set of labels for a document if a single label is incorrect (like the zero-one-loss).

Thus, I could not resist in not predicting the zero-one loss as well. This metric is higher (as expected) in both models, but again the non-probabilistic presents better results. Still, I believe the result is pretty good. I used hamming loss and not log loss because this is a multi-label and multi-class classification problem.

I tried to find a baseline from the “The Modified Apte ("ModApte") Split” but that was not possible. Hopefully, the results can be compared with those of other peers that chose the corresponding split method.

20newsgroups:

To start with the code, I used the corresponding function from sklearn to fetch the dataset and created a training set and a test set for my pipeline. Then, I instantiated the TfidfVectorizer from sklearn to convert my raw documents (from both sets) to a matrix of TF-IDF features. I applied the *fit_transform* method on the train set to learn vocabulary and idf and the *transform* method on the test set because in this case we only need to transform documents to document-term matrix. After testing the parameters of the Vectorizer, it seemed that the existence of stop-words in the documents did not affect the accuracy of the models. Similarly, using the n-grams parameter did not affect the non-probabilistic model, however the results of the probabilistic model were less accurate the more the n-gram range increased. After evaluation and class prediction, I built a text report showing the main classification metrics based on the target values (ground truth data) from the test set, the evaluated vectors from the test set and the names matching the labels (target names). We need intel from the test set because scores are meaningful if computed in unseen data. Concerning the models’ hyperparameters, when trying to tune those, the models’ performance decreased, thus mainly the default parameters we kept. For instance, in both models the regularization parameter C affected the performance only when C<1. Other parameters such as loss, dual or random state did not seem to affect the performance when changed. Only the *alpha* (smoothing) parameter in the Naïve Bayes model provides better results when alpha<1 (the less, the better) as in the previous dataset.

In the classification report, we can see the classes (labels) in each row and the metrics with the class weights in each column. The class weights tell our model how important a class is.

After multiple hyperparameter tests, the LinearSVC was pretty stable with the model's default parameters in the correctly predicted observations whereas the Naïve Bayes accuracy varied depending on the n-grams and alpha parameter. Here are the best results:

Probabilistic Model	Non-Probabilistic Model
Accuracy: 0.70	Accuracy: 0.70
F1 score: 0.68	F1 score: 0.68
Precision: 0.70	Precision: 0.71
Recall: 0.69	Recall: 0.68

Time for probabilistic modeling: 0.112 seconds
Time for non-probabilistic modeling: 1.148 seconds

Zero-one loss of the probabilistic model: 0.298
Zero-one loss of the non-probabilistic model: 0.307

I did not expect the probabilistic model to be as effective as the non-probabilistic model. I also expected the accuracy of the non-probabilistic model to be higher. However, accuracy alone does not mean that a model is great or not. Zero-one loss is probably a more indicative metric , which in this case shows that almost 30% of the labels are wrongly predicted. Precision and recall attempt to minimize false positives and false negatives respectively. All metrics of the table seem to be almost at the same level, which I believe means that bias is low in these models. These results are almost the same with my peers' results (draft submission).

If I was to compare all 4 pipelines, I would say the 20newsgroups models appear to be more stable and less biased even though the models from the Reuters dataset present same or higher accuracy. It is important to mention though that multi-label/multi-class prediction is more demanding and thus, much more data is needed for generating a stable model. The 20newsgroups models use many more articles both for training and testing than those of the Reuters dataset (which is a more demanding task).