Exercises on n-gram language models and context-aware spelling correction

Koutsomarkos Alexandros[1], Lakkas Ioannis[2], Tzoumezi Vasiliki[3], Tsoukalelis Nikolaos[4], and Chalkiopoulos Georgios[5]

[1]p3352106 , [2]p3352110 , [3]p3352121 , [4]p3352123 , [5]p3352124
Emails: akoutsomarkos@aueb.gr , ilakkas@aueb.gr , vtzoumezi@aueb.gr , ntsoukalelis@aueb.gr , gchalkiopoulos@aueb.gr

June 19, 2022

Google Colab Link

1 Exercises on text classification with (mostly) linear models

Develop a sentiment classifier for a kind of texts of your choice (e.g., tweets, product or movie reviews). Use an existing sentiment analysis dataset with at least two classes (e.g., positive/negative or positive/negative/neutral).2 The classes should be mutually exclusive, i.e., this is a single-label multi-class classification problem.

1.1 Introduction

Implementing classifiers to make a sentiment analysis on tweets/posts.

1.2 Dataset

For the purpose of the exercise we used the Twitter Sentiment Analysis Dataset. It contains tweets and the corresponding label (0 - negative, 1 - positive). In Figure (1) we can see how these tweets are distributed to each class for the entirety of the dataset.

1.3 Pre-Processing

Before proceeding with the development of the models, we had to do some data-prepossessing in order to clear the text of the twits and only keep words. The steps we followed may be found below:

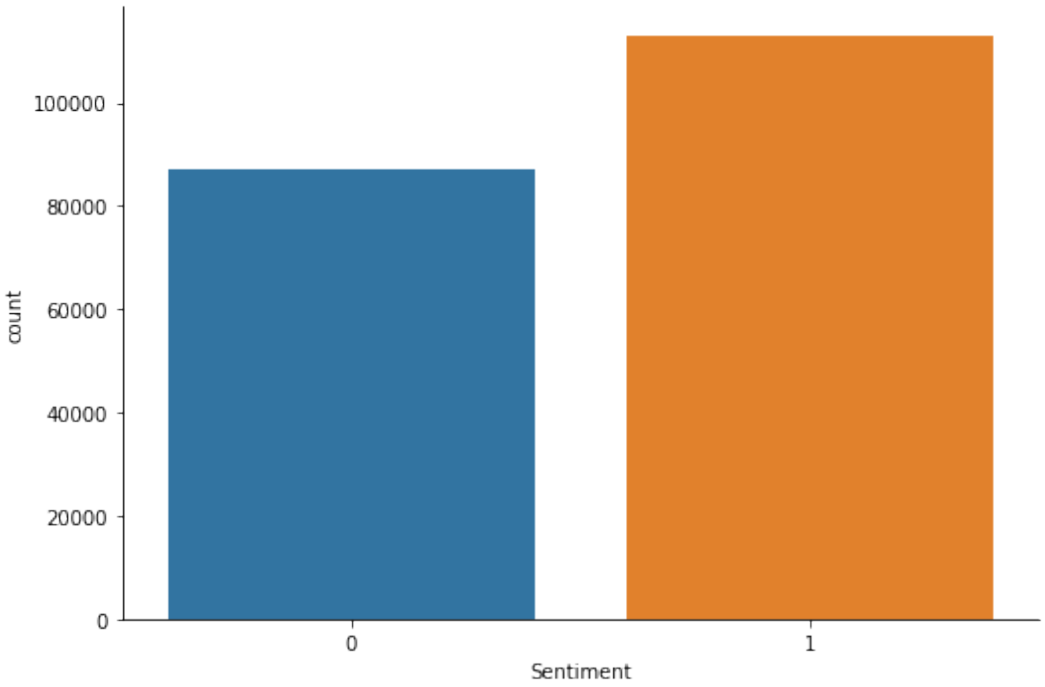

Figure 1: Class distribution for the whole dataset of twits.

1. Lower Case: using the `.lower()` method we converted the input string to the equivalent, only having lower case words. This ensures that there will be no differences in calculations due to differences in lower and capital letters.
2. Clean the text: using regular expressions we removed the following and kept only the words in each tweet.
   (a) underscores
   (b) twitter usernames
   (c) websites
   (d) non words
   (e) single characters
   (f) numbers
   (g) multiple whitespaces
3. Lemmatization: We split each tweet on whitspaces to get words and then we used the lematize methond from nltk to return the lemma of each word.

4. Finally, the words are put together and returned as a sentence.

In Table (1) we see some samples of the orginal tweets and the corresponding processed tweets.

Original text	Processed text
is so sad for my APL friend.....	is so sad for my apl friend
I missed the New Moon trailer...	missed the new moon trailer
omg its already 7:30 :O	omg it already
.. Omgaga. Im sooo im gunna CRy	omgaga im sooo im gunna cry
i think mi bf is cheating on me!!! T_T	think mi bf is cheating on me tt

Table 1: Sample tweets and the corresponding processed text

1.4 Train/Dev/Test split

In order to create our model we had to split the input into three parts. We set a seed for reproducibility purposes and shuffle the tweets.

1. Train: 60% of the input tweets were used to train the models.
2. Development: 25% of the input tweets were used for feature tuning and hyper-parameters tuning.
3. Test: 15% of the input tweets were used in order to test the performance of the model, using previously unseen data.

1.5 Features

In this exercise we choose to experiment with TF-IDF features, BOW(bag of words) and pre-trained word embeddings

1.5.1 TF-IDF

We used uni-gram and bi-gram TF-IDF features, which were produced by the TfidfVectorizer. In total we produced 5000 features by excluding any stopword.

1.5.2 Bag of Words

We used uni-gram and bi-gram features, which were produced by the CountVectorizer. In total we produced 5000 features by excluding any stopword.

1.5.3 Word embedings

We used the Gensim Downloader to download pre-trained glove vectors, based on tweets, from https://nlp.stanford.edu/projects/glove/. Then with this model, we computed Word2Vec embeddings for our dataset. Finally, we used the SimpleImputer for completing missing values. In total we produced 200 features.

1.6 Dimensionality reduction

For dimensionality reduction we use the TruncatedSVD algorithm from sklearn library. We dedcide the value of k or the number of components based on the explained variance by each of the singular values.

TF-IDF

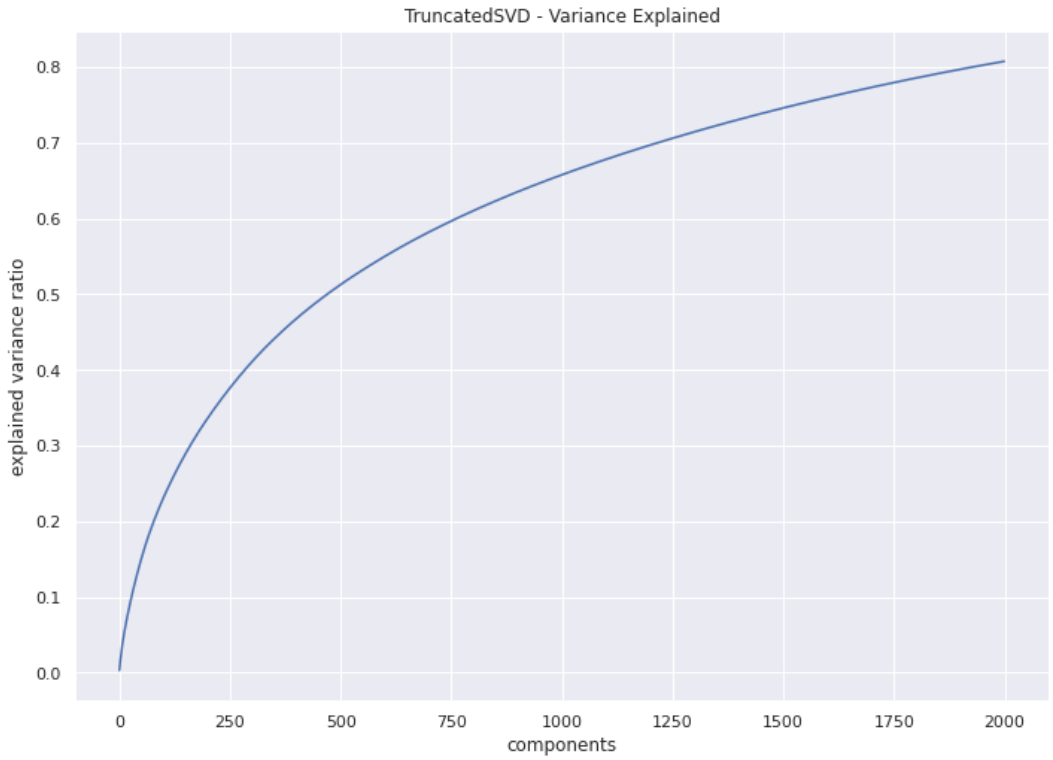

Figure 2: Plot for selecting the optimal k or optimal number of components for Truncated SVD

In Figure (2), we observe that we achieve to explain 80% of the variance by using 2000 components. This reduces our original feature vector length by more than 50%. However, 80% is not an accepted percentage. Due to the fact that our resources

(RAM) are not enough for calculating more components, we decide to disregard the dimensionality reduction step.

BOW
same result as TF-IDF

Word Embedings

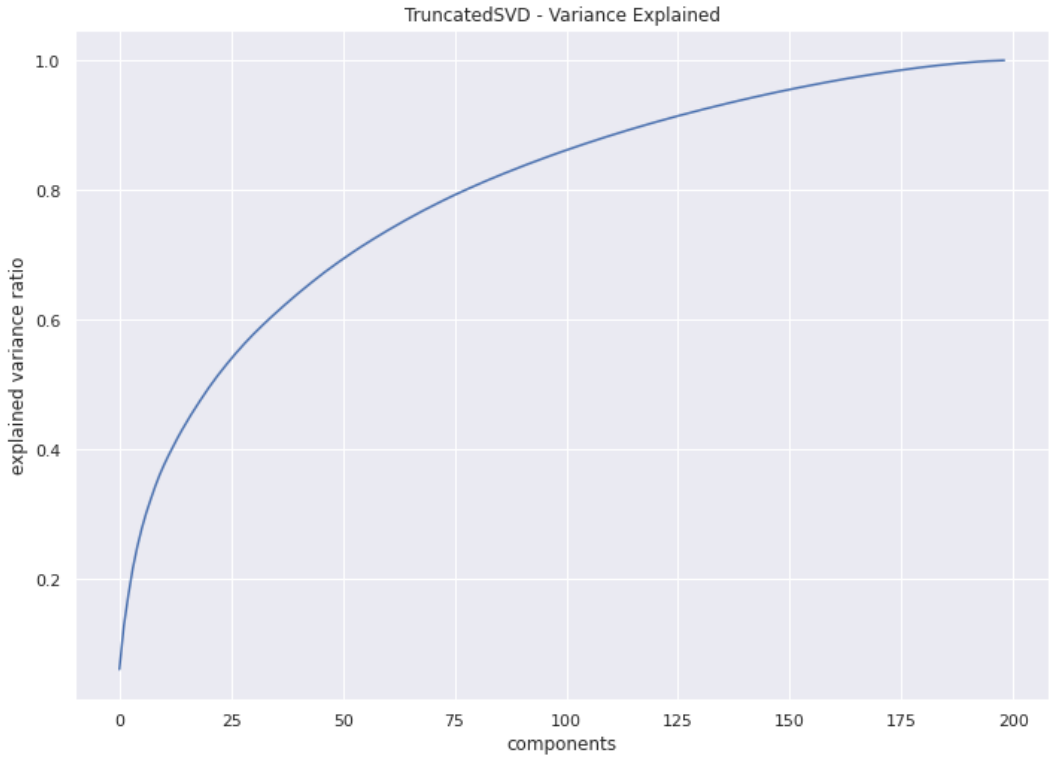

Figure 3: Plot for selecting the optimal k or optimal number of components for Truncated SVD

In Figure (3), we observe that we achieve to explain 95% of the variance by using 148 components (from 200 initial features). In this case we decide to use the dimensionality reduction step.

1.7 Model Training

For this part we chose to follow three main pillars in order to implement our analysis.

Dummy Classifier - Logistic Regression - PyCaret

1.7.1 Dummy Classifier

Beginning with the Dummy Classifier we get a "baseline" classification and we can have a first picture. This classifier serves as a simple baseline to compare against other more complex classifiers and this is exactly how we planned to use it in our assignment. In detail we used the 'most frequent' strategy. As a result we made a classifier that assigns the most frequent class of the training data. Finally we printed the train/test accuracy and the confusion matrix (see below).

Train accuracy: 56.46%

Test accuracy: 56.46%

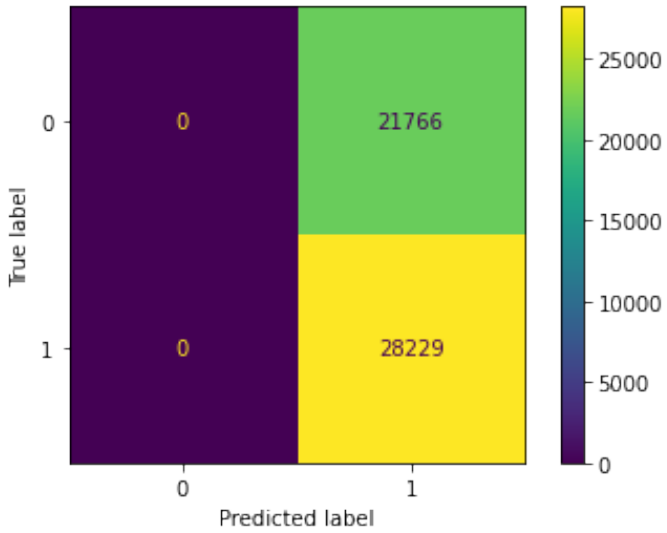

Figure 4: Confusion matrix for the test data using the Dummy Classifier

1.7.2 Logistic Regression

Afterwards we implemented the Logistic Regression classification procedure (two-class logistic regression classifier). This is the model that was tuned in order to obtain the best possible result.

In order to select the model with the best performance (TF-IDF, Bag of Words, Word embedings) and then optimize it using Randomized Search Cross Validation, we ran a 10-fold Cross Validation for the three datasets. As a result the TF-IDF gave the best score.

Consequently we implemented the Randomized Search Cross Validation on these (TF-IDF, Bag of Words, Word embedings) using the development set. Having found our

best hyerparameters we implemented our classifier using these hyperparameters on the train and test dataset (fit, predict). Below are the results of the development, training and test set, their respective confusion matrices and the learning curves.

TF-IDF development f1-score: 81.86%

	precision	recall	f1-score	AUC
0	0.78	0.70	0.74	
1	0.79	0.85	0.82	
accuracy			0.79	0.86
macro avg	0.79	0.78	0.78	
weighted avg	0.79	0.79	0.79	

Table 2: Score table for the development set.

TF-IDF train f1-score: 81.82%

	precision	recall	f1-score	AUC
0	0.79	0.70	0.74	
1	0.79	0.85	0.82	
accuracy			0.79	0.89
macro avg	0.79	0.78	0.78	
weighted avg	0.79	0.79	0.79	

Table 3: Score table for the training set.

TF-IDF test f1-score: 79.30%

	precision	recall	f1-score	AUC
0	0.75	0.66	0.70	
1	0.76	0.83	0.79	
accuracy			0.76	0.86
macro avg	0.75	0.75	0.75	
weighted avg	0.76	0.76	0.75	

Table 4: Score table for the test set.

BOW development f1-score: 82.16%

	precision	recall	f1-score	AUC
0	0.79	0.70	0.74	
1	0.79	0.86	0.82	
accuracy			0.79	0.89
macro avg	0.79	0.78	0.78	
weighted avg	0.79	0.79	0.79	

Table 5: Score table for the development set.

BOW train f1-score: 81.31%

	precision	recall	f1-score	AUC
0	0.78	0.68	0.73	
1	0.78	0.85	0.81	
accuracy			0.78	0.88
macro avg	0.78	0.77	0.77	
weighted avg	0.78	0.78	0.78	

Table 6: Score table for the training set.

BOW test f1-score: 79.30%

	precision	recall	f1-score	AUC
0	0.76	0.65	0.70	
1	0.76	0.84	0.80	
accuracy			0.76	0.86
macro avg	0.76	0.75	0.75	
weighted avg	0.76	0.76	0.75	

Table 7: Score table for the test set.

Word Embeddings development f1-score: 76.92%

	precision	recall	f1-score	AUC
0	0.71	0.62	0.66	
1	0.73	0.81	0.77	
accuracy			0.73	0.83
macro avg	0.72	0.71	0.72	
weighted avg	0.73	0.73	0.72	

Table 8: Score table for the development set.

Word Embeddings train f1-score: 76.75%

	precision	recall	f1-score	AUC
0	0.71	0.62	0.66	
1	0.73	0.81	0.77	
accuracy			0.71	0.83
macro avg	0.72	0.71	0.71	
weighted avg	0.72	0.72	0.72	

Table 9: Score table for the training set.

Word Embeddings test f1-score: 76.5%

	precision	recall	f1-score	AUC
0	0.71	0.61	0.66	
1	0.73	0.81	0.77	
accuracy			0.72	0.83
macro avg	0.72	0.71	0.71	
weighted avg	0.72	0.72	0.72	

Table 10: Score table for the test set.

Next we produced the respective learning curves for the 3 datasets. In the report we chose to show only the TF-IDF learing curves, as TF-IDF was the model that gave us the best average score (the rest of the learning curves have been produced and can be found in our ipynb file)

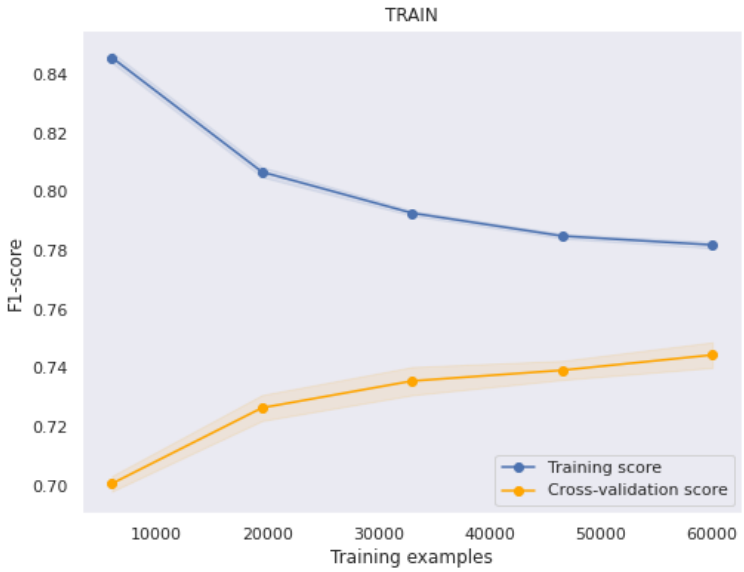

Figure 5: Learning curve for the training data using the Logistic Resgression

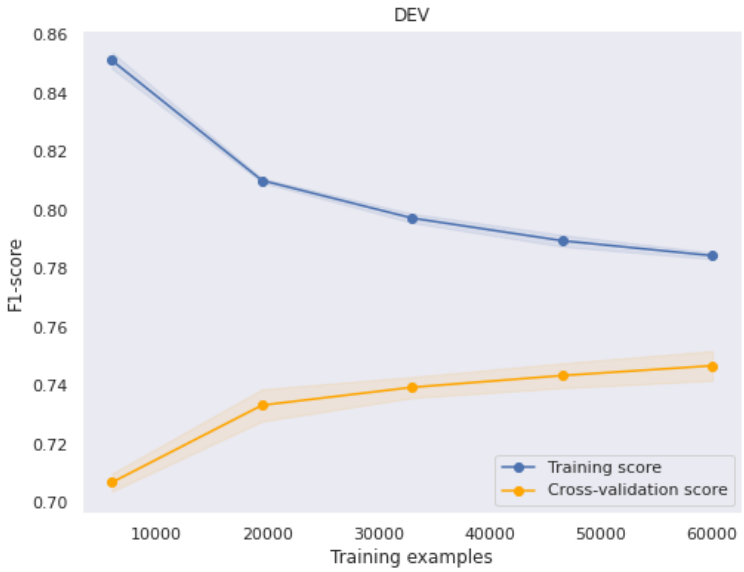

Figure 6: Learning curve for the dev data using the Logistic Resgression

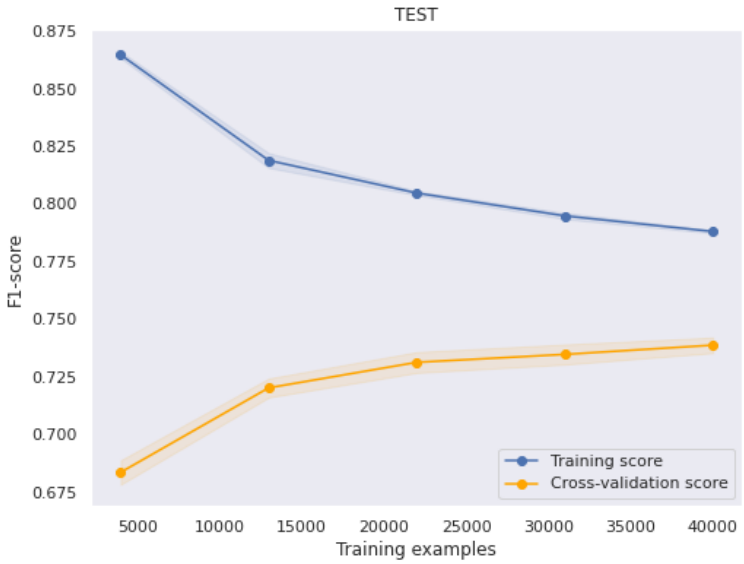

Figure 7: Learning curve for the test data using the Logistic Resgression

1.7.3 Optimization of Logistic Regression with BOW

We also implemented the best hypermeters we found in our previous step (during the development phase) for the optimization of the BOW model, using logistic regression but not only for the 5000 features used before.

More specifically our input data is now a set with all the potential features that the Count Vectorizer can generate. By doing this were able to reach the optimal performance.

Indicative is the f1 score for this optimization.

BOW optimized test f1-score: 92.75%

1.7.4 PyCaret

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. It is an end-to-end machine learning and model management tool that speeds up the experiment cycle exponentially and makes you more productive.

Due to the limitations of thi exercise (Google Colab performance issues and time restrictions) this sections will serve as a demonstration on how the development would have been approached. For that reason a very small sample of the total data will be used (2k rows), and the development process will be demonstrated.

Using the automated pipeline, we got the best results (based on the F-1 score) for the `Gradient Boosting Classifier`. The Output of the model comparison may be found in Figure (8).

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.6362	0.6749	0.9217	0.6244	0.7442	0.1876	0.2396	2.776
ada	Ada Boost Classifier	0.6290	0.6655	0.9093	0.6213	0.7379	0.1737	0.2168	1.326
dummy	Dummy Classifier	0.5747	0.5000	1.0000	0.5747	0.7299	0.0000	0.0000	0.023
lightgbm	Light Gradient Boosting Machine	0.6411	0.6567	0.8383	0.6442	0.7282	0.2252	0.2449	0.707
lr	Logistic Regression	0.6369	0.6915	0.7414	0.6655	0.7009	0.2420	0.2453	3.138
ridge	Ridge Classifier	0.6340	0.0000	0.7152	0.6707	0.6914	0.2422	0.2440	0.328
rf	Random Forest Classifier	0.6376	0.6839	0.6878	0.6833	0.6853	0.2578	0.2582	1.878
et	Extra Trees Classifier	0.6283	0.6687	0.6841	0.6734	0.6778	0.2379	0.2387	2.637
svm	SVM - Linear Kernel	0.6290	0.0000	0.6789	0.6899	0.6727	0.2362	0.2422	0.662
knn	K Neighbors Classifier	0.6004	0.6263	0.6641	0.6495	0.6556	0.1788	0.1798	2.758
dt	Decision Tree Classifier	0.5982	0.6174	0.5758	0.6794	0.6217	0.1986	0.2031	0.468
nb	Naive Bayes	0.5282	0.5507	0.4007	0.6450	0.4915	0.0954	0.1053	0.195
qda	Quadratic Discriminant Analysis	0.5096	0.5341	0.3694	0.6271	0.4611	0.0639	0.0729	2.210

Figure 8: Model comparison using PyCaret

Using the integrated optimization function we optimized the model and trained the model using all the available data. Moreover, the option of generating plots is available as seen in Figure (9).

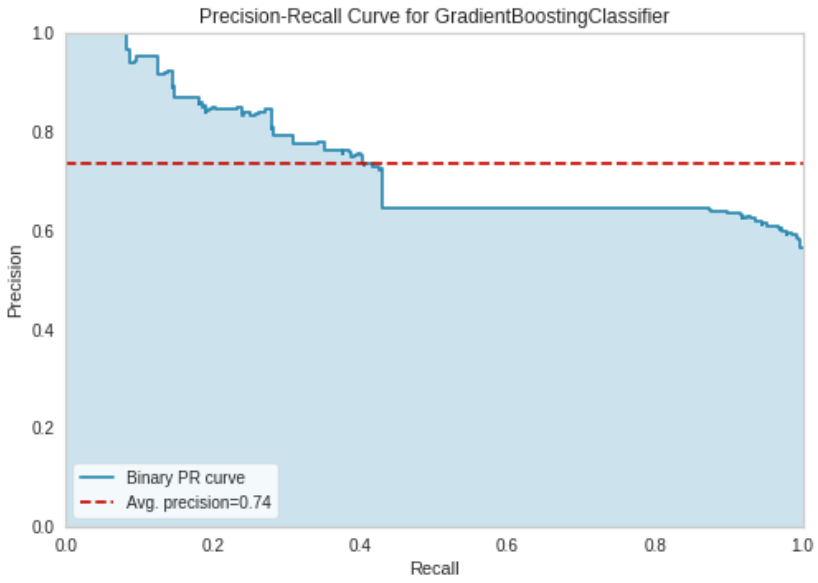

Figure 9: Precision-Recall Curve for GradientBoostingClassifier

Using only a small percentage of the test set (200 rows) we performed predictions

and got the results presented in Figure (10).

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Gradient Boosting Classifier	0.55	0.4955	0.8716	0.5556	0.6786	0.0386	0.0515

Figure 10: Test set prediction scores.

The results would be much better if more resources were available. The purpose of this section was to show our approach, rather than achieve a high score.