[BERT for Text Classification with NO model training | by Mauro Di Pietro | Towards Data Science](https://towardsdatascience.com/text-classification-with-no-model-training-935fe0e42180)

**BERT for Text Classification with NO model training**

Use BERT, Word Embedding, and Vector Similarity when you don’t have a labeled training set

**Summary**

Are you struggling to classify text data because you don’t have a labeled dataset? In this article, using BERT and Python, I will explain how to perform a sort of “unsupervised” text classification based on similarity.

Text

Description automatically generated with low confidence

Image by Author

[**NLP (Natural Language Processing)**](https://en.wikipedia.org/wiki/Natural_language_processing) is the field of artificial intelligence that studies the interactions between computers and human languages, in particular how to program computers to process and analyze large amounts of natural language data. NLP is often applied for classifying text data. **Text classification** is the problem of assigning categories to text data according to its content. In order to carry out a classification use case, you need a labeled dataset for machine learning models training. So what happens if you don’t have one?

This kind of situation happens in the real world more often than you think. Nowadays, AI is so hyped that firms want to use it even when they don’t have data. In particular, most non-technical people don’t fully get the concept of “target variable” and how it is used in supervised machine learning. So how can you build a classifier when you have text data but no label? In this tutorial, I am going to explain a strategy that applies W2V and BERT to classify text by word vector similarity.

I will present some useful Python code that can be easily applied in other similar cases (just copy, paste, run) and walk through every line of code with comments so that you can replicate this example (link to the full code below).

**[mdipietro09/DataScience\_ArtificialIntelligence\_Utils](https://github.com/mdipietro09/DataScience_ArtificialIntelligence_Utils/blob/master/natural_language_processing/example_text_classification.ipynb" \t "_blank)**

[Permalink Dismiss GitHub is home to over 50 million developers working together to host and review code, manage…](https://github.com/mdipietro09/DataScience_ArtificialIntelligence_Utils/blob/master/natural_language_processing/example_text_classification.ipynb" \t "_blank)

[github.com](https://github.com/mdipietro09/DataScience_ArtificialIntelligence_Utils/blob/master/natural_language_processing/example_text_classification.ipynb" \t "_blank)

I will use the “**News category dataset**” in which you are provided with news headlines from the year 2012 to 2018 obtained from *HuffPost*and you are asked to classify them with the right category, therefore this is a multiclass classification problem (link below).

**[News Category Dataset](https://www.kaggle.com/rmisra/news-category-dataset" \t "_blank)**

[Identify the type of news based on headlines and short descriptions](https://www.kaggle.com/rmisra/news-category-dataset" \t "_blank)

[www.kaggle.com](https://www.kaggle.com/rmisra/news-category-dataset" \t "_blank)

In particular, I will go through:

* Setup: import packages, read data.
* Preprocessing: clean text data.
* Create Target Clusters: use Word2Vec with *gensim*to build the target variable.
* Feature Engineering: Word Embedding with *transformers*and BERT*.*
* Model Design & Testing: assign observations to clusters by Cosine Similarity and evaluate the performance.
* Explainability: understand how the model produces results.

**Setup**

First of all, I need to import the following packages:

**## for data**import **json**import **pandas** as pd  
import **numpy** as np  
from **sklearn** import metrics, manifold**## for processing**import **re**  
import **nltk## for plotting**  
import **matplotlib**.pyplot as plt  
import **seaborn** as sns**## for w2v**  
import **gensim**import gensim.downloader as gensim\_api**## for bert**  
import **transformers**

The dataset is contained into a json file, so I will first read it into a list of dictionaries with *json*and then transform it into a *pandas*Dataframe.

lst\_dics = []  
with **open**('data.json', mode='r', errors='ignore') as json\_file:  
 for dic in json\_file:  
 lst\_dics.append( json**.loads**(dic) )**## print the first one**  
lst\_dics[0]

Text

Description automatically generated

Image by Author

The original dataset contains over 30 categories, but for the purposes of this tutorial, I will work with a subset of 3: Entertainment, Politics, and Tech.

**## create dtf**  
dtf = pd.DataFrame(lst\_dics)**## filter categories**  
dtf = dtf[ dtf["category"].isin(['**ENTERTAINMENT**','**POLITICS**','**TECH**']) ][["category","headline"]]**## rename columns**  
dtf = dtf.rename(columns={"category":"**y**", "headline":"**text**"})**## print 5 random rows**  
dtf.sample(5)

Table

Description automatically generated

Image by Author

As you can see, the dataset includes a target variable as well. I won’t be using it for modeling, just for performance evaluation.

So we’ve got some raw text data and we are tasked to classify it into the 3 categories (Entertainment, Politics, Tech) we know nothing about. Here’s what I am planning to do:

* clean data and embed it into the vector space,
* create a topic cluster for each category and embed it into the vector space,
* calculate similarities between every text vector and the topic clusters, then assign it to the closest cluster.

Graphical user interface, application

Description automatically generated

Image by Author

That is why I called it “a sort of unsupervised text classification”. It’s a really basic idea, but the execution can be tricky.

Now that’s all set, let’s get started.

**Preprocessing**

The absolute first step is to preprocess the data: cleaning text, removing stop words, and applying lemmatization. I will write a function and apply it to the whole data set.

**'''  
Preprocess a string.  
:parameter  
 :param text: string - name of column containing text  
 :param lst\_stopwords: list - list of stopwords to remove  
 :param flg\_stemm: bool - whether stemming is to be applied  
 :param flg\_lemm: bool - whether lemmitisation is to be applied  
:return  
 cleaned text  
'''**  
def **utils\_preprocess\_text**(text, flg\_stemm=False, flg\_lemm=True, lst\_stopwords=None):  
 **## clean (convert to lowercase and remove punctuations and   
 characters and then strip)**  
 text = re.sub(r'[^\w\s]', '', str(text).lower().strip())  
   
 **## Tokenize (convert from string to list)**  
 lst\_text = text.split() **## remove Stopwords**  
 if lst\_stopwords is not None:  
 lst\_text = [word for word in lst\_text if word not in   
 lst\_stopwords]  
   
 **## Stemming (remove -ing, -ly, ...)**  
 if flg\_stemm == True:  
 ps = nltk.stem.porter.PorterStemmer()  
 lst\_text = [ps.stem(word) for word in lst\_text]  
   
 **## Lemmatisation (convert the word into root word)**  
 if flg\_lemm == True:  
 lem = nltk.stem.wordnet.WordNetLemmatizer()  
 lst\_text = [lem.lemmatize(word) for word in lst\_text]  
   
 **## back to string from list**  
 text = " ".join(lst\_text)  
 return text

That function removes a set of words from the corpus if given. I can create a list of generic stop words for the English vocabulary with *nltk*(we could edit this list by adding or removing words).

lst\_stopwords = **nltk**.corpus.stopwords.words("**english**")  
lst\_stopwords

Text

Description automatically generated

Image by Author

Now I shall apply the function to the whole dataset and store the result in a new column named “*text\_clean*” that I am going to use as a corpus.

dtf["**text\_clean**"] = dtf["text"].apply(lambda x:   
 **utils\_preprocess\_text**(x, flg\_stemm=False, **flg\_lemm=True**,   
 **lst\_stopwords=lst\_stopwords**))dtf.head()

Table

Description automatically generated

Image by Author

We have our preprocessed corpus, consequently the next step is to build the target variable. Basically, we’re here:

Graphical user interface

Description automatically generated with medium confidence

Image by Author

**Create Target Clusters**

The objective of this section is to create some keywords which can represent the context of each category. By performing some text analysis, you can easily discover that the 3 most frequent words are “*movie*”, “*trump*”, and “*apple*” (for a detailed text analysis tutorial you can check [this article](https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d)). I’d suggest starting with those keywords.

Let’s take the Politics category for instance: the word “*trump*” can have different meanings, so we need to add keywords to avoid polysemy problems (i.e. “*donald*”, “*republican*”, “*white house*”, “*obama*”). This task could be carried out manually or you could use the assistance of a pre-trained NLP model. You can load a pre-trained Word Embedding model from [*genism-data*](https://github.com/RaRe-Technologies/gensim-data)like this:

nlp = gensim\_api.load("**glove-wiki-gigaword-300**")

The *gensim*package has a very convenient function that returns the most similar words for any given word into the vocabulary.

nlp.**most\_similar**(["**obama**"], topn=3)



Image by Author

I shall use that to create a dictionary of keywords for each category:

**## Function to apply**  
def **get\_similar\_words**(lst\_words, top, nlp):  
 lst\_out = lst\_words  
 for tupla in nlp.most\_similar(lst\_words, topn=top):  
 lst\_out.append(tupla[0])  
 return list(set(lst\_out))  
**## Create Dictionary {category:[keywords]}**dic\_clusters = {}dic\_clusters["**ENTERTAINMENT**"] = get\_similar\_words([**'celebrity','cinema','movie','music'**],   
 top=30, nlp=nlp)dic\_clusters[**"POLITICS"**] = get\_similar\_words([**'gop','clinton','president','obama','republican'**]  
 , top=30, nlp=nlp)dic\_clusters["**TECH**"] = get\_similar\_words([**'amazon','android','app','apple','facebook',  
 'google','tech'**],   
 top=30, nlp=nlp)  
**## print some**  
for k,v in dic\_clusters.items():  
 print(k, ": ", v[0:5], "...", len(v))

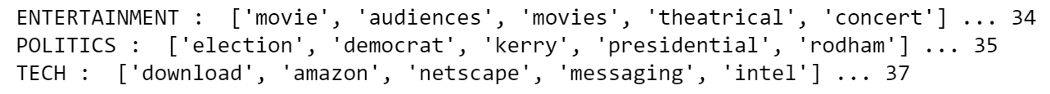


Image by Author

Let’s try to visualize those keywords in a 2D space by applying a dimensionality reduction algorithm (i.e. [TSNE](https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html)). We want to make sure that the clusters are well separated from each other.

**## word embedding**tot\_words = [word for v in **dic\_clusters**.values() for word in v]  
X = nlp[tot\_words]  
 **## pca**  
pca = manifold.**TSNE**(perplexity=40, n\_components=2, init='pca')  
X = pca.fit\_transform(X) **## create dtf**  
dtf = pd.DataFrame()  
for k,v in **dic\_clusters**.items():  
 size = len(dtf) + len(v)  
 dtf\_group = pd.DataFrame(X[len(dtf):size], columns=["x","y"],   
 index=v)  
 dtf\_group["cluster"] = k  
 dtf = dtf.append(dtf\_group)  
 **## plot**  
fig, ax = plt.subplots()  
sns.**scatterplot**(data=dtf, x="x", y="y", hue="cluster", ax=ax)ax.legend().texts[0].set\_text(None)  
ax.set(xlabel=None, ylabel=None, xticks=[], xticklabels=[],   
 yticks=[], yticklabels=[])for i in range(len(dtf)):  
 ax.annotate(dtf.index[i],   
 xy=(dtf["x"].iloc[i],dtf["y"].iloc[i]),   
 xytext=(5,2), textcoords='offset points',   
 ha='right', va='bottom')

Scatter chart

Description automatically generated

Image by Author

Cool, they look isolated enough from each other. The Entertainment cluster is closer to the Tech one than the Politics one, which makes sense as words like “*apple*” and “*youtube*” can appear in both Tech and Entertainment news.

**Feature Engineering**

It’s time to embed the corpus we preprocessed and the target clusters we created in the same vector space. Basically, we’re doing this:

Diagram

Description automatically generated

Image by Author

Yes, I’m using [**BERT**](https://en.wikipedia.org/wiki/BERT_(language_model)) for this. It’s true that you could utilize any Word Embedding model (i.e. Word2Vec, Glove, …), even the one that we already loaded to define keywords, so why bother to use such a heavy and complex language model? That’s because BERT doesn’t apply a fixed embedding, instead it looks at the entire sentence and then assigns an embedding to each word. Therefore, the vector BERT assigns to a word is a function of the entire sentence, so that a word can have different vectors based on the contexts.

I’m going to load the original pre-trained version of BERT with the package *transformers*and give an example of the dynamic embedding:

tokenizer = transformers.**BertTokenizer**.from\_pretrained('**bert-base-  
 uncased'**, do\_lower\_case=True)nlp = transformers.**TFBertModel**.from\_pretrained(**'bert-base-uncased'**)

Let’s use the model to convert the string “*river bank*” into vectors and print the one assigned to the word “*bank*”:

txt = **"river bank"## tokenize**  
idx = tokenizer.encode(txt)  
print("tokens:", tokenizer.convert\_ids\_to\_tokens(idx))  
print("ids :", tokenizer.encode(txt))**## word embedding**  
idx = np.array(idx)[None,:]  
embedding = nlp(idx)  
print("shape:", embedding[0][0].shape)**## vector of the second input word**  
embedding[0][0][2]

Text

Description automatically generated

Image by Author

If you do the same for the string “*financial bank*”, you’ll see that the vector assigned to the word “*bank*” is different because of the context. Please note that the BERT tokenizer inserts special tokens at the beginning and end of sentences and its vector space has a dimension of 768 (to understand better how BERT processes text you can check [this article](https://towardsdatascience.com/text-classification-with-nlp-tf-idf-vs-word2vec-vs-bert-41ff868d1794)).

A screenshot of a computer

Description automatically generated with medium confidence

Image by Author

Having said that, the plan is to use BERT Word Embedding to represent each text with an array (shape: number of tokens x 768) and then summarize each article into a mean vector.

A picture containing table

Description automatically generated

Image by Author

So the final feature matrix will be an array with shape: number of documents (or mean vectors) x 768.

**## function to apply**def **utils\_bert\_embedding**(txt, tokenizer, nlp):  
 idx = tokenizer.encode(txt)  
 idx = np.array(idx)[None,:]   
 embedding = nlp(idx)  
 X = np.array(embedding[0][0][1:-1])  
 return X**## create list of news vector**  
lst\_mean\_vecs = [**utils\_bert\_embedding**(txt, tokenizer, nlp)**.mean(0)**   
 for txt in dtf["**text\_clean**"]]**## create the feature matrix (n news x 768)**  
X = np.array(lst\_mean\_vecs)

We can do the same with the keywords in the target clusters. In fact, each label is identified by a list of words that help BERT to understand the context within the clusters. Hence, I’m going to create a dictionary label : cluster mean vector.

dic\_y = {k:**utils\_bert\_embedding**(v, tokenizer, nlp)**.mean(0)** for k,v  
 in dic\_clusters.items()}

We started with just some text data and 3 strings (*“Entertainment”, “Politics”, “Tech”*) and now we have a feature matrix and a target variable… ish.

**Model Design & Testing**

Finally, it’s time to build a model that classifies the news based on the similarity to each target cluster.

Graphical user interface

Description automatically generated with medium confidence

Image by Author

I am going to use [**Cosine Similarity**](https://en.wikipedia.org/wiki/Cosine_similarity), a measure of similarity based on the cosine of the angle between two non-zero vectors, which equals the inner product of the same vectors normalized to both have length 1. You can easily use the cosine similarity implementation of [*scikit-learn*](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html)*,*which takes 2 arrays (or vectors) and returns an array of scores (or a single score). In this case, the output is going to be a matrix with shape: number of news x number of labels (3, Entertainment/Politics/Tech). To put it another way, each row will represent an article and contain one similarity score for each target cluster.

In order to run the usual evaluation metrics (Accuracy, AUC, Precision, Recall, …), we have to rescale the scores in each row so that they sum to 1 and decide a category to label the article with. I’m going to choose the one with the highest score, but it could be wise to set some minimum thresholds and leave out predictions with really low scores.

A picture containing table

Description automatically generated

Image by Author

**#--- Model Algorithm ---### compute cosine similarities**  
similarities = np.array(  
 [metrics.pairwise.**cosine\_similarity**(X, y).T.tolist()[0]   
 for y in dic\_y.values()]  
 ).T**## adjust and rescale**  
labels = list(dic\_y.keys())  
for i in range(len(similarities)): **### assign randomly if there is no similarity**  
 if sum(similarities[i]) == 0:  
 similarities[i] = [0]\*len(labels)  
 similarities[i][np.random.choice(range(len(labels)))] = 1 **### rescale so they sum = 1**  
 similarities[i] = similarities[i] / sum(similarities[i]) **## classify the label with highest similarity score**predicted\_prob = similarities  
predicted = [labels[np.argmax(pred)] for pred in predicted\_prob]

Just like in classic supervised use cases, we have an object with predicted probabilities (here they’re adjusted similarity scores) and another with predicted labels. Let’s check how we did:

y\_test = dtf[**"y"**].values  
classes = np.unique(y\_test)  
y\_test\_array = pd.get\_dummies(y\_test, drop\_first=False).values **## Accuracy, Precision, Recall**  
accuracy = metrics.accuracy\_score(y\_test, predicted)  
auc = metrics.roc\_auc\_score(y\_test, predicted\_prob,   
 multi\_class="ovr")  
print("Accuracy:", round(accuracy,2))  
print("Auc:", round(auc,2))  
print("Detail:")  
print(metrics.classification\_report(y\_test, predicted))  
 **## Plot confusion matrix**  
cm = metrics.confusion\_matrix(y\_test, predicted)  
fig, ax = plt.subplots()  
sns.heatmap(cm, annot=True, fmt='d', ax=ax, cmap=plt.cm.Blues,   
 cbar=False)  
ax.set(xlabel="Pred", ylabel="True", xticklabels=classes,   
 yticklabels=classes, title="Confusion matrix")  
plt.yticks(rotation=0)  
fig, ax = plt.subplots(nrows=1, ncols=2) **## Plot roc**  
for i in range(len(classes)):  
 fpr, tpr, thresholds = metrics.roc\_curve(y\_test\_array[:,i],   
 predicted\_prob[:,i])  
 ax[0].plot(fpr, tpr, lw=3,   
 label='{0} (area={1:0.2f})'.format(classes[i],   
 metrics.auc(fpr, tpr))  
 )  
ax[0].plot([0,1], [0,1], color='navy', lw=3, linestyle='--')  
ax[0].set(xlim=[-0.05,1.0], ylim=[0.0,1.05],   
 xlabel='False Positive Rate',   
 ylabel="True Positive Rate (Recall)",   
 title="Receiver operating characteristic")  
ax[0].legend(loc="lower right")  
ax[0].grid(True)  
 **## Plot precision-recall curve**for i in range(len(classes)):  
 precision, recall, thresholds = metrics.precision\_recall\_curve(  
 y\_test\_array[:,i], predicted\_prob[:,i])  
 ax[1].plot(recall, precision, lw=3,   
 label='{0} (area={1:0.2f})'.format(classes[i],   
 metrics.auc(recall, precision))  
 )  
ax[1].set(xlim=[0.0,1.05], ylim=[0.0,1.05], xlabel='Recall',   
 ylabel="Precision", title="Precision-Recall curve")  
ax[1].legend(loc="best")  
ax[1].grid(True)  
plt.show()

Chart, treemap chart

Description automatically generated

Chart, line chart, scatter chart

Description automatically generated

Image by Author

Okay, I’m the first to say that it’s not the best Accuracy I’ve ever seen. On the other hand, it’s not bad at all considering that we didn’t train any model and we even made up the target variable. The main issue is over 4k Politics observations classified as Entertainment, but these performances can be easily improved by fine-tuning the keywords for those two categories.

**Explainability**

Let’s try to understand what led our algorithm to classify news with a category instead of the others. Let’s take a random observation from the corpus:

i = 7**txt\_instance** = dtf[**"text\_clean"**].iloc[i]print("True:", y\_test[i], "--> Pred:", predicted[i], "|   
 Similarity:", round(np.max(predicted\_prob[i]),2))  
print(txt\_instance)

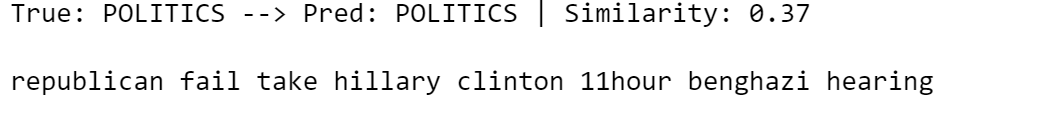


Image by Author

It’s a Politics observation properly classified. Probably, the words “*republican*” and “*clinton*” gave BERT the right hint. I will visualize the mean vector of the article in a 2D space and plot the top similarities with the target cluster.

**## create embedding Matrix**y = np.concatenate([embedding\_bert(v, tokenizer, nlp) for v in   
 dic\_clusters.values()])  
X = embedding\_bert(txt\_instance, tokenizer,  
 nlp).mean(0).reshape(1,-1)  
M = np.concatenate([y,X]) **## pca**  
pca = manifold.**TSNE**(perplexity=40, n\_components=2, init='pca')  
M = pca.fit\_transform(M)  
y, X = M[:len(y)], M[len(y):] **## create dtf clusters**  
dtf = pd.DataFrame()  
for k,v in dic\_clusters.items():  
 size = len(dtf) + len(v)  
 dtf\_group = pd.DataFrame(y[len(dtf):size], columns=["x","y"],   
 index=v)  
 dtf\_group["cluster"] = k  
 dtf = dtf.append(dtf\_group) **## plot clusters**  
fig, ax = plt.subplots()  
sns.**scatterplot**(data=dtf, x="x", y="y", hue="cluster", ax=ax)  
ax.legend().texts[0].set\_text(None)  
ax.set(xlabel=None, ylabel=None, xticks=[], xticklabels=[],   
 yticks=[], yticklabels=[])  
for i in range(len(dtf)):  
 ax.annotate(dtf.index[i],   
 xy=(dtf["x"].iloc[i],dtf["y"].iloc[i]),   
 xytext=(5,2), textcoords='offset points',   
 ha='right', va='bottom') **## add txt\_instance**ax.scatter(x=X[0][0], y=X[0][1], c="red", linewidth=10)  
 ax.annotate("x", xy=(X[0][0],X[0][1]),   
 ha='center', va='center', fontsize=25) **## calculate similarity**sim\_matrix = metrics.pairwise.**cosine\_similarity**(X, y) **## add top similarity**  
for row in range(sim\_matrix.shape[0]): **### sorted {keyword:score}**  
 dic\_sim = {n:sim\_matrix[row][n] for n in   
 range(sim\_matrix.shape[1])}  
 dic\_sim = {k:v for k,v in sorted(dic\_sim.items(),   
 key=lambda item:item[1], reverse=True)} **### plot lines**  
 for k in dict(list(dic\_sim.items())[0:5]).keys():  
 p1 = [X[row][0], X[row][1]]  
 p2 = [y[k][0], y[k][1]]  
 ax.plot([p1[0],p2[0]], [p1[1],p2[1]], c="red", alpha=0.5)plt.show()

Chart, scatter chart

Description automatically generated

Image by Author

Let’s zoom a bit on the cluster of interest:

Text

Description automatically generated with low confidence

Image by Author

Overall, we can say the mean vector is pretty similar to the Politics cluster. Let’s break down the article into tokens to see which ones “activated” the right cluster.

**## create embedding Matrix**y = np.concatenate([embedding\_bert(v, tokenizer, nlp) for v in   
 dic\_clusters.values()])  
X = embedding\_bert(txt\_instance, tokenizer,  
 nlp).mean(0).reshape(1,-1)  
M = np.concatenate([y,X]) **## pca**  
pca = manifold.**TSNE**(perplexity=40, n\_components=2, init='pca')  
M = pca.fit\_transform(M)  
y, X = M[:len(y)], M[len(y):] **## create dtf clusters**  
dtf = pd.DataFrame()  
for k,v in dic\_clusters.items():  
 size = len(dtf) + len(v)  
 dtf\_group = pd.DataFrame(y[len(dtf):size], columns=["x","y"],   
 index=v)  
 dtf\_group["cluster"] = k  
 dtf = dtf.append(dtf\_group) **## add txt\_instance**tokens = tokenizer.convert\_ids\_to\_tokens(  
 tokenizer.encode(txt\_instance))[1:-1]  
dtf = pd.DataFrame(X, columns=["x","y"], index=tokens)  
dtf = dtf[~dtf.index.str.contains("#")]  
dtf = dtf[dtf.index.str.len() > 1]  
X = dtf.values  
ax.scatter(x=dtf["x"], y=dtf["y"], c="red")  
for i in range(len(dtf)):  
 ax.annotate(dtf.index[i],   
 xy=(dtf["x"].iloc[i],dtf["y"].iloc[i]),   
 xytext=(5,2), textcoords='offset points',   
 ha='right', va='bottom') **## calculate similarity**sim\_matrix = metrics.pairwise.**cosine\_similarity**(X, y) **## add top similarity**  
for row in range(sim\_matrix.shape[0]): **### sorted {keyword:score}**  
 dic\_sim = {n:sim\_matrix[row][n] for n in   
 range(sim\_matrix.shape[1])}  
 dic\_sim = {k:v for k,v in sorted(dic\_sim.items(),   
 key=lambda item:item[1], reverse=True)} **### plot lines**  
 for k in dict(list(dic\_sim.items())[0:5]).keys():  
 p1 = [X[row][0], X[row][1]]  
 p2 = [y[k][0], y[k][1]]  
 ax.plot([p1[0],p2[0]], [p1[1],p2[1]], c="red", alpha=0.5)plt.show()

Scatter chart

Description automatically generated

Image by Author

As we thought, there are words in the text which are clearly linked to the Politics cluster, but some others are more similar to the Entertainment general context.

Chart, line chart

Description automatically generated

A picture containing diagram

Description automatically generated

Image by Author

**Conclusion**

This article has been a tutorial to demonstrate **how to perform text classification when a labeled training set isn't available**.

I used a pre-trained Word Embedding model to build a set of keywords to contextualize the target variable. Then I transformed those words and the corpus in the same vector space with the pre-trained BERT language model. Finally, I calculated the Cosine Similarity between text and keywords to determine the context of each article and I used that information to label the news.

This strategy isn’t the most effective but it’s definitely efficient as it allows you to deliver good results quickly. Moreover, this algorithm can be used as a baseline for a supervised model, once a labeled dataset is obtained.

I hope you enjoyed it! Feel free to contact me for questions and feedback or just to share your interesting projects.