

aws_product_class-Copy1

September 29, 2019

0.1 PRODUCT CATEGORY CLASSIFICATION

0.1.1 Here, we want to develop an automatic and scalable first prototype that helps to correctly categorize a new product in the available categories when it arrives.

This first prototype will help us to identify in what elements we have to go deeper in order to get the best model.

0.2 Summary of requirements

1. Train a model that predicts the product category for Software, Digital Software, and Digital Video Games products using the Amazon Customer Reviews dataset.
2. Evaluate and validate your model.

```
[1]: # I checked warnings, but for the final report I prefer ignore those  
#that really does not affect the results (warnings of libraries, etc)  
import warnings  
warnings.simplefilter('ignore')  
#import platform  
#platform.architecture()[0]
```

```
[2]: #Further improvement: develop an enviroment  
# Basic Modules  
import os  
import pandas as pd  
import numpy as np  
import boto3  
import json  
import matplotlib.pyplot as plt  
#from sagemaker.predictor import json_deserializer
```

```
[3]: #MOdels  
from sklearn.linear_model import LogisticRegression
```

```
[4]: # Modules for tokens  
from keras.preprocessing.text import Tokenizer  
from keras.preprocessing.sequence import pad_sequences
```

```
from keras.utils import to_categorical
```

Using TensorFlow backend.

```
[5]: #my own functions
%load_ext autoreload
%autoreload 2

from utils.py_functions import *
```

0.3 DATA

For our problem we use will three datasets from public datasets of Amazon. This datasets contains Customer Reviews fro three category of products.

- Software
- Digital Software
- Digital video games

```
[6]: '''
!mkdir /tmp/recsys/
!aws s3 cp s3://amazon-reviews-pds/tsv/
    ↪amazon_reviews_us_Digital_Video_Games_v1_00.tsv.gz /tmp/recsys/
!aws s3 cp s3://amazon-reviews-pds/tsv/amazon_reviews_us_Software_v1_00.tsv.gz /
    ↪tmp/recsys/
!aws s3 cp s3://amazon-reviews-pds/tsv/amazon_reviews_us_Digital_Software_v1_00.
    ↪tsv.gz /tmp/recsys/
'''
```

```
[6]: '\n!mkdir /tmp/recsys/\n!aws s3 cp s3://amazon-reviews-
pds/tsv/amazon_reviews_us_Digital_Video_Games_v1_00.tsv.gz /tmp/recsys/\n!aws s3
cp s3://amazon-reviews-pds/tsv/amazon_reviews_us_Software_v1_00.tsv.gz
/tmp/recsys/\n!aws s3 cp s3://amazon-reviews-
pds/tsv/amazon_reviews_us_Digital_Software_v1_00.tsv.gz /tmp/recsys/\n'
```

```
[7]: df_video_games = pd.read_csv('data/amazon_reviews_us_Digital_Video_Games_v1_00.
    ↪tsv', delimiter = '\t', error_bad_lines = False)
print('Number of rows: ', df_video_games.shape)
df_software = pd.read_csv('data/amazon_reviews_us_Software_v1_00.tsv',
    ↪delimiter = '\t', error_bad_lines = False)
print('Number of rows: ', df_software.shape)
df_digital_software = pd.read_csv('data/
    ↪amazon_reviews_us_Digital_Software_v1_00.tsv', delimiter = '\t',
    ↪error_bad_lines = False)
print('Number of rows: ', df_digital_software.shape)
```

Number of rows: (144724, 15)

```
b'Skipping line 8021: expected 15 fields, saw 22\nSkipping line 34886: expected
15 fields, saw 22\nSkipping line 49286: expected 15 fields, saw 22\n'
```

```
Number of rows: (341249, 15)
```

```
Number of rows: (101836, 15)
```

0.4 Dataset columns

- marketplace: 2-letter country
- customer_id: random identifier for a customer
- review_id: unique id for the review
- product_id: ASIN number. Unique id for product
- product_parent: the parent of that ASIN. Multiple ASINs (color or format variations of the same product) can roll up into a single parent parent.
- product_title: title/description of the product
- product_category: broad product category
- star_rating: 1 to 5 stars (review rating)
- helpful_votes: Number of helpful votes for the review
- total_votes: number of total votes the review received
- vine: Was the review written as part of the Vine program?
- verified_purchase: Was the review from a verified purchase?
- review_headline: the title of the review itself
- review_body: text of the review
- review_date: the date of the review was written.

```
[8]: df_video_games.tail()
```

```
[8]:      marketplace  customer_id      review_id  product_id  product_parent \
144719          US      53011810  R2G7DI8NYXZB5R  B001AUEITS      163061733
144720          US      53094564  R3QRKP4DS759BP  B001AU6TQ8      801870836
144721          US      37181147  R24K4C0ZC3093U  B001AUEITS      163061733
144722          US      18614365  R130A3TRCM8IBM  B001AUEITS      163061733
144723          US      28326760  R1PFDIHC9TM6V4  B000AQ7K4I      99419628
```

```

                                product_title \
144719                                Crazy Machines 2 [Download]
144720  Crazy Machines 1 - The Wacky Contraptions Game...
144721                                Crazy Machines 2 [Download]
144722                                Crazy Machines 2 [Download]
144723                                Emperor of the Fading Suns
```

```

      product_category  star_rating  helpful_votes  total_votes  vine \
144719  Digital_Video_Games          4             2           3     N
144720  Digital_Video_Games          1            13          16     N
144721  Digital_Video_Games          3             3           3     N
144722  Digital_Video_Games          1            20          22     N
144723  Digital_Video_Games          4             4           4     N
```

	verified_purchase		review_headline \
144719	N		Worked first try for me
144720	N	The Software May be Great, But I'll Never Know	
144721	N	Some install problems but good otherwise	
144722	N	Do Not Download This!	
144723	N	Suprisingly large scale and complex strategy game	

		review_body	review_date
144719	I was worried due to the 2 reviews I saw here,...		2008-12-25
144720	I downloaded this as a Christmas present for m...		2008-12-24
144721	The previous reviewer is correct in noting tha...		2008-09-10
144722	I downloaded this for my son's birthday yester...		2008-09-01
144723	This game has all the makings of a wonderful t...		2006-08-08

0.5 DROP SOME COLUMNS

```
[9]: df_digital_software = df_digital_software.drop(columns = ['marketplace',
    ↳ 'customer_id', 'product_parent', 'review_id', 'vine', 'review_date'])
df_software = df_software.drop(columns = ['marketplace', 'customer_id',
    ↳ 'product_parent', 'review_id', 'vine', 'review_date'])
df_video_games = df_video_games.drop(columns = ['marketplace',
    ↳ 'customer_id', 'product_parent', 'review_id', 'vine', 'review_date'])
```

```
[10]: df = pd.concat([df_digital_software, df_video_games, df_software], axis=0)
df.tail()
```

```
[10]:      product_id      product_title \
341244  0877794618  Merriam-Webster's Medical Audio Dictionary
341245  0877794618  Merriam-Webster's Medical Audio Dictionary
341246  B00002SV6E      Star Wars: Droid Works
341247  0671573535      Star Trek Borg
341248  0877794618  Merriam-Webster's Medical Audio Dictionary
```

	product_category	star_rating	helpful_votes	total_votes \
341244	Software	5	5	7
341245	Software	5	3	5
341246	Software	4	1	1
341247	Software	1	0	2
341248	Software	3	12	13

	verified_purchase		review_headline \
341244	N	My name is Maeda, and I love this CR-ROM.	
341245	N	I'd rather give 6 stars to this title	
341246	N	Droid building	
341247	N	don't buy this game	

341248 N An Easy-to-Use Medical Dictionary with Excelle...

review_body

341244 I am a medical student. I loved this CD-ROM v...
 341245 I tried a Taber's, but was disappointed: he do...
 341246 You get to build droids even ones in existence...
 341247 We have not been able to even run this game be...
 341248 Merriam-Webster's Medical Audio Dictionary is ...

[11]: df.dtypes

```
[11]: product_id      object
      product_title   object
      product_category object
      star_rating     int64
      helpful_votes   int64
      total_votes     int64
      verified_purchase object
      review_headline  object
      review_body      object
      dtype: object
```

0.5.1 We have products that only appears one. These products may be problematic if the text data does not describe the product correctly.

[12]: df.groupby(['product_id', 'product_title', 'product_category'])['star_rating'].
 ↪count().reset_index().sort_values('star_rating').head(15)

```
[12]:      product_id      product_title \
0      0028650506      BLACK MUSIC OF TWO WORLDS CD
13274  B000LP6JMM      Zonealarm Antivirus Small Business Ed 10U
13275  B000LPAGOY      Magic Math: Grades 1-2: Jacobson + Jennings
30638  B00B1PFFVC  Dell Inspiron 15R (N5110) Driver Recovery and ...
30637  B00B1PF9KO  Dell Inspiron N5050 Driver Update and Drivers ...
30636  B00B1PETSC  Dell Inspiron 8600 Driver Update and Drivers I...
13280  B000LRDOAA      MoviePlus
13281  B000LRGHBS      ImpactPlus 5; High Impact Design
30635  B00B1PESDS  Dell Studio XPS 8100 Driver Update and Drivers...
30634  B00B1PEPJA  Dell Inspiron 660 Driver Update and Drivers In...
13284  B000LT158G      Pro Series Bridge
13285  B000LTM9ZO      North American Birds
13286  B000LTNVIS      3-D Web Animation Pack
13288  B000LU6M6K      Pc-Cillin Antivirus 2007 (Tech Bench)
13290  B000LU6M8S      pc-Cillin Antivirus 2007
```

product_category star_rating

0	Software	1
13274	Software	1
13275	Software	1
30638	Software	1
30637	Software	1
30636	Software	1
13280	Software	1
13281	Software	1
30635	Software	1
30634	Software	1
13284	Software	1
13285	Software	1
13286	Software	1
13288	Software	1
13290	Software	1

0.6 CHECK IF WE HAVE MISSING VALUES

We have missing data in review_headline, review_body. We have to eliminate it or fix this in order to keep going with the analysis.

```
[13]: ## check for missing values
df.isnull().sum()
```

```
[13]: product_id      0
product_title      0
product_category    0
star_rating         0
helpful_votes       0
total_votes         0
verified_purchase   0
review_headline     5
review_body         4
dtype: int64
```

0.6.1 We can see that the na's are not at the same rows.

```
[14]: df[df.isnull().any(axis=1)]
```

```
[14]:   product_id      product_title \
3074  B00452VGX0  The Sims 3 Late Night
11907 B008D7F47Q  FIFA Soccer 13
14470 B00S00IJH8  Sims 4
128119 B004VSTQ2A  Xbox Live Subscription
21683  B00MUTB2SS  McAfee 2015 Antivirus Plus 3 PC (3-Users)
```

55474	B00EOI2ND6	HRB 2011 Basic FFP Test ASIN (Formerly: Micros...
76722	B001GL6QHS	TurboTax Deluxe Federal + State + eFile 2008
178530	B000M9DOTS	MorphVOX Pro - Voice Changer
281897	B00008NNY0	Instant CD/DVD

	product_category	star_rating	helpful_votes	total_votes	\
3074	Digital_Video_Games	1	0	0	
11907	Digital_Video_Games	5	0	0	
14470	Digital_Video_Games	2	0	2	
128119	Digital_Video_Games	1	1	8	
21683	Software	1	1	1	
55474	Software	4	0	0	
76722	Software	4	0	0	
178530	Software	4	0	1	
281897	Software	1	0	4	

	verified_purchase	review_headline	\
3074	Y	NaN	
11907	N	Five Stars	
14470	N	Two Stars	
128119	Y	NaN	
21683	Y	NaN	
55474	N	Four Stars	
76722	Y	Four Stars	
178530	Y	NaN	
281897	N	NaN	

	review_body
3074	Product code does not work.
11907	NaN
14470	NaN
128119	I DID NOT BUY THIS PRODUCT SO I AM CONFUSED AS...
21683	I am giving this one star because I am unable ...
55474	NaN
76722	NaN
178530	Product worked was able to play with applicati...
281897	Shortly after buying the product I had to repl...

0.6.2 Fill values

```
[15]: df.review_headline.fillna(df.review_body, inplace=True)
df.review_body.fillna(df.review_headline, inplace=True)
```

0.7 Check for white strings

```
[16]: blanks = [] # start with an empty list

for i,lb,rv in df[['product_category','review_body']].itertuples(): # iterate_
    ↳over the DataFrame , 'product_title', 'verified_purchase', 'review_headline'
    if type(rv)==str: # avoid NaN values
        if rv.isspace(): # test 'review' for whitespace
            blanks.append(i) # add matching index numbers to the list

print(len(blanks), 'blanks: ', blanks)
```

0 blanks: []

```
[17]: blanks = [] # start with an empty list

for i,lb,rv in df[['product_title', 'review_headline']].itertuples(): #_
    ↳iterate over the DataFrame ,
    if type(rv)==str: # avoid NaN values
        if rv.isspace(): # test 'review' for whitespace
            blanks.append(i) # add matching index numbers to the list

print(len(blanks), 'blanks: ', blanks)
```

0 blanks: []

0.8 IS BALANCED?

```
[18]: df.groupby("product_category")['product_id'].count()/df.shape[0]
```

```
[18]: product_category
Digital_Software      0.173247
Digital_Video_Games   0.246209
Software              0.580544
Name: product_id, dtype: float64
```

```
[19]: dicti = {"Digital_Software": 0, "Digital_Video_Games": 1, "Software": 2}
df['product_category_label'] = df['product_category']
df = df.replace({"product_category": dicti})
```

```
[20]: df.head()
```

```
[20]:   product_id      product_title \
0  B00U7LCE6A      CCleaner Free [Download]
1  B00HRJMM04  ResumeMaker Professional Deluxe 18
2  B00P31G9PQ      Amazon Drive Desktop [PC]
```



```

3 B00FGDEPDY          Norton Internet Security 1 User 3 Licenses
4 B00FZ0FKOU  SecureAnywhere Internet Security Complete 5 De...

```

	product_category	star_rating	helpful_votes	total_votes	\
0	0	4	0	0	
1	0	3	0	0	
2	0	1	1	2	
3	0	5	0	0	
4	0	4	1	2	

	verified_purchase	review_headline	\
0	Y	Four Stars	
1	Y	Three Stars	
2	Y	One Star	
3	Y	Works as Expected!	
4	Y	Great antivirus. Worthless customer support	

	review_body	product_category_label
0	So far so good	Digital_Software
1	Needs a little more work...	Digital_Software
2	Please cancel.	Digital_Software
3	Works as Expected!	Digital_Software
4	I've had Webroot for a few years. It expired a...	Digital_Software

0.9 CLEAN TEXT COLUMNS

```

[21]: #clean characters
df = standardize_text(df, "product_title")
df = standardize_text(df, "review_headline")
df = standardize_text(df, "review_body")

```

0.10 Eliminate stop words

```

[22]: #download stopwords
import nltk
nltk.download('stopwords')
nltk.download('wordnet')

```

```

[nltk_data] Downloading package stopwords to
[nltk_data] /home/erikapat/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/erikapat/nltk_data...
[nltk_data] Package wordnet is already up-to-date!

```

[22]: True

[23]: df.tail(15).head()

```
[23]:      product_id      product_title \
341234  B00002CEXA  email games  xcom first alien invasion
341235  B00002S6Z2      dimension  standard edition
341236  B00002S6EQ                        unreal
341237  B00002JV50  microsoft windows  second edition upgrade
341238  B00002S9XK      sesame street  get set to learn ages

      product_category  star_rating  helpful_votes  total_votes \
341234                2            4             25           26
341235                2            5              9            9
341236                2            4              6           12
341237                2            2             21           25
341238                2            5             48           59

      verified_purchase      review_headline \
341234                N      old game but classic
341235                N  d is e best relation database product on deskt...
341236                N  wow!  so beautiful  you almost ate to blow it...
341237                N      expensive bug fix
341238                N      best introduction to games for  year old

      review_body \
341234  along xcom lacks e wizbang graphics of e latest...
341235  e d world opens a pleora of opportunities to c...
341236  is is a great game!  a little lacking on story...
341237  for a company at as enoug revenue to run a sma...
341238  is game was e first game my son was able to pl...

      product_category_label
341234      Software
341235      Software
341236      Software
341237      Software
341238      Software
```

```
[24]: # Import stopwords with nltk.
from nltk.corpus import stopwords
stop = stopwords.words('english')
# Exclude stopwords with Python's list comprehension and pandas.DataFrame.apply.
df['product_title'] = df['product_title'].apply(lambda x: ' '.join([word for
    ↪word in x.split() if word not in (stop)]))
df['review_headline'] = df['review_headline'].apply(lambda x: ' '.join([word
    ↪for word in x.split() if word not in (stop)]))
```

```

df['review_body'] = df['review_body'].apply(lambda x: ' '.join([word for word_
↳in x.split() if word not in (stop)]))
#more cleaning
df['product_title'] = df['product_title'].map(lambda x : clean_text(x))
df['review_headline'] = df['review_headline'].map(lambda x : clean_text(x))
df['review_body'] = df['review_body'].map(lambda x : clean_text(x))
df['product_title'] = df['product_title'].map(lambda x : removeAscendingChar(x))
df['review_headline'] = df['review_headline'].map(lambda x :
↳removeAscendingChar(x))
df['review_body'] = df['review_body'].map(lambda x : removeAscendingChar(x))

```

0.11 Lemmatization

```

[25]: #lemmatization
df['product_title'] = df['product_title'].map(lambda x : lemitizeWords(x))
df['review_headline'] = df['review_headline'].map(lambda x : lemitizeWords(x))
df['review_body'] = df['review_body'].map(lambda x : lemitizeWords(x))

```

```

[26]: #eliminate single letters
df['product_title'] = df['product_title'].str.replace(r'\b\w\b','').str.
↳replace(r'\s+', ' ')
df['review_headline'] = df['review_headline'].str.replace(r'\b\w\b','').str.
↳replace(r'\s+', ' ')
df['review_body'] = df['review_body'].str.replace(r'\b\w\b','').str.
↳replace(r'\s+', ' ')

```

```

[27]: df.tail(15).head()

```

```

[27]:
product_id      product_title \
341234  B00002CEXA  email games xcom first alien invasion
341235  B00002S6Z2           dimension standard edition
341236  B00002S6EQ           unreal
341237  B00002JV50  microsoft windows second edition upgrade
341238  B00002S9XK           sesame street get set learn ages

product_category  star_rating  helpful_votes  total_votes \
341234           2           4           25           26
341235           2           5           9           9
341236           2           4           6          12
341237           2           2          21          25
341238           2           5          48          59

verified_purchase      review_headline \
341234              N      old game classic
341235              N  best relation database product desktop macines

```

341236	N	wow beautiful almost ate blow
341237	N	expensive bug fix
341238	N	best introduction games year old

		review_body \
341234		aloug xcom lacks wizbang grapics latest games ...
341235		world opens pleora opportunities create custo...
341236		great game little lacking story wat care got a...
341237		company enoug revenue run small country expect...
341238		game first game son able play age onwards exce...

	product_category_label
341234	Software
341235	Software
341236	Software
341237	Software
341238	Software

0.12 BASELINE MODEL

First we will develop a simple model, that we are going to use as Baseline.

For our baseline model we will do: * Tokenizing sentences to a list of separate words * Creating a train test split * Inspecting our data a little more to validate results

```
[28]: from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer(r'\w+')

df["token_product_title"] = df["product_title"].apply(tokenizer.tokenize)
df["token_review_body"] = df["review_body"].apply(tokenizer.tokenize)
df["token_review_headline"] = df["review_headline"].apply(tokenizer.tokenize)
```

0.13 Inspecting tokens

```
[29]: sentence_lengths_1 = descriptive_tokens(df, name = "token_product_title")
sentence_lengths_2 = descriptive_tokens(df, name = "token_review_body")
sentence_lengths_3 = descriptive_tokens(df, name = "token_review_headline")
```

Column token_product_title have 2637986 words total, with a vocabulary size of 16285

Max sentence length is 52

Column token_review_body have 22956901 words total, with a vocabulary size of 170124

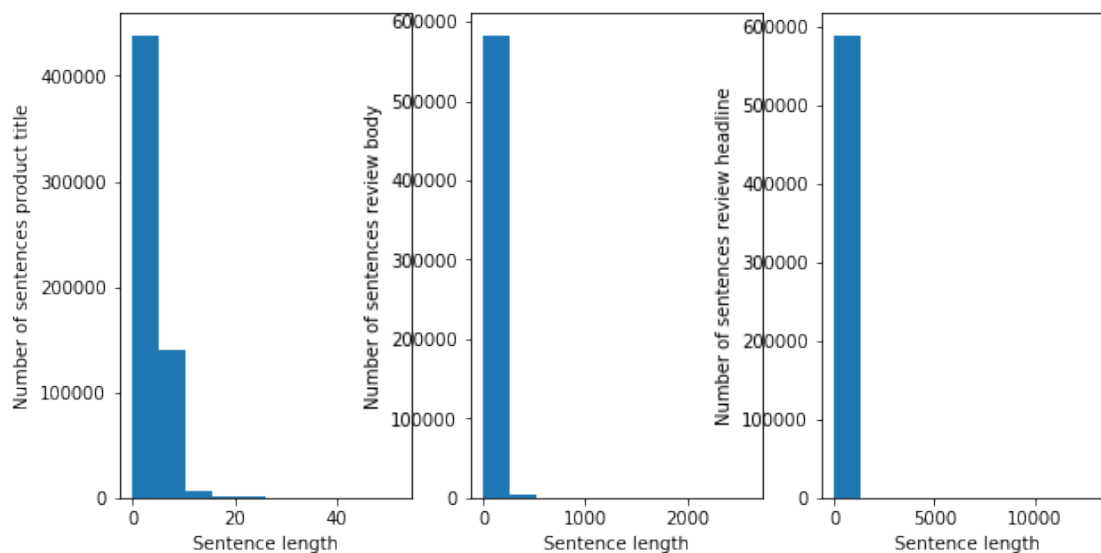
Max sentence length is 2609

Column token_review_headline have 1670103 words total, with a vocabulary size

of 43088

Max sentence length is 13279

```
[30]: %matplotlib inline
fig= plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.xlabel('Sentence length')
plt.ylabel('Number of sentences product title')
plt.hist(sentence_lengths_1)
plt.subplot(1, 3, 2)
plt.xlabel('Sentence length')
plt.ylabel('Number of sentences review body')
plt.hist(sentence_lengths_2)
plt.subplot(1, 3, 3)
plt.xlabel('Sentence length')
plt.ylabel('Number of sentences review headline')
plt.hist(sentence_lengths_3)
plt.show()
```



```
[33]: ## **concatenate columns**

#We will use a concatenation of columns, to see if we include review
→information or not

#
#df['full_text'] = df["product_title"] + " " + df["review_body"] + " " +
→df["review_headline"]
```

```
df['product_title'] = df["product_title"] #+ " " + df["review_body"] # + " " +  
↳ df["review_headline"] #
```

```
[34]: df.head()
```

```
[34]:  product_id                                product_title \  
0  B00U7LCE6A                                ccleaner free download  
1  B00HRJMMOM4                            resumemaker professional deluxe  
2  B00P31G9PQ                                amazon drive desktop pc  
3  B00FGDEPDY                            norton internet security user licenses  
4  B00FZ0FKOU  secureanywere internet security complete device  
  
  product_category  star_rating  helpful_votes  total_votes \  
0                0            4                0            0  
1                0            3                0            0  
2                0            1                1            2  
3                0            5                0            0  
4                0            4                1            2  
  
  verified_purchase                                review_headline \  
0                Y  
1                Y                                ree  
2                Y  
3                Y                                works expected  
4                Y  great antivirus worless customer support  
  
                                review_body  product_category_label \  
0                                far good      Digital_Software  
1                            needs little work      Digital_Software  
2                                please cancel      Digital_Software  
3                            works expected      Digital_Software  
4  ve ad webroot years expired decided purchase r...      Digital_Software  
  
                                token_product_title \  
0                                [ccleaner, free, download]  
1                            [resumemaker, professional, deluxe]  
2                                [amazon, drive, desktop, pc]  
3                            [norton, internet, security, user, licenses]  
4  [secureanywere, internet, security, complete, ...  
  
                                token_review_body \  
0                                [far, good]  
1                            [needs, little, work]  
2                                [please, cancel]  
3                            [works, expected]  
4  [ve, ad, webroot, years, expired, decided, pur...
```

```

                                token_review_headline
0                                []
1                                [ree]
2                                []
3                                [works, expected]
4  [great, antivirus, worless, customer, support]

```

0.14 Enter embeddings

Machine Learning on images can use raw pixels as inputs. A way to represent text is to encode each character individually, this seems quite inadequate to represent and understand language. Our goal is to first create a useful embedding for each sentence in our dataset, and then use these embeddings to accurately predict the relevant category.

The most simplest approach is to use a **bag of words model**, and apply a machine learning algorithm (like, logistic, naive bayes, between others). A bag of words just associates an index to each word in our vocabulary, and embeds each sentence as a list of 0s, with a 1 at each index corresponding to a word present in the sentence.

0.15 Bag of Words

```

[35]: from sklearn.model_selection import train_test_split
      from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

      def cv(data):
          count_vectorizer = CountVectorizer()

          emb = count_vectorizer.fit_transform(data)

          return emb, count_vectorizer

      df_corpus = df['product_title'] #df['review_headline']#df['full_text']
      #df_corpus = df['review_body', 'review_headline']
      df_labels = df["product_category"]

      X_train, X_test, y_train, y_test = train_test_split(df_corpus, df_labels,
          ↪test_size=0.2,

          ↪random_state=40)

```

```

[36]: y_test.value_counts()

```

```

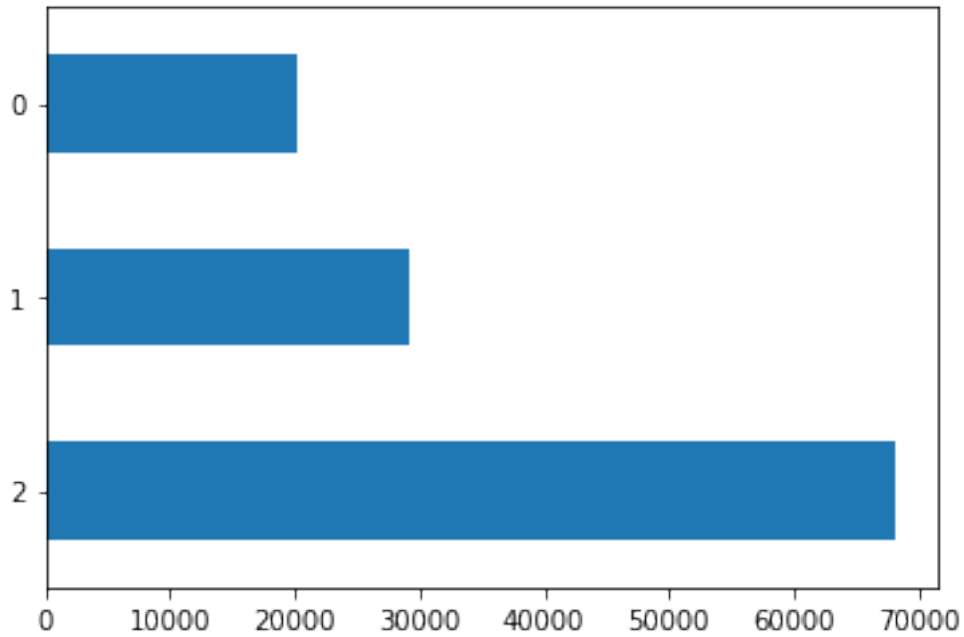
[36]: 2    68194
      1    29104
      0    20264

```

Name: product_category, dtype: int64

```
[37]: y_test.value_counts().plot('barh')
```

```
[37]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc3a06c2050>
```



```
[38]: X_train_counts, count_vectorizer = cv(X_train)
X_test_counts = count_vectorizer.transform(X_test)
```

```
[39]: print(X_train_counts.shape)
```

```
(470247, 15354)
```

```
[40]: #print(X_train_counts.todense())
```

0.16 CHI-SQUARED

We've constructed a matrix, that have a lot of unique words/columns. This data configuratio will take a very long time to make predictions. We want to speed it up, so we'll need to cut down the column count somehow. One way to do this is to pick a subset of the columns that are the most informative.

```
[41]: from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# Find the 1000 most informative columns
```



```
selector = SelectKBest(chi2, k=1000) #further work:check other alternatives
selector.fit(X_train_counts, y_train)
top_words = selector.get_support().nonzero()

# Pick only the most informative columns in the data.
chi_matrix = X_train_counts[:,top_words[0]]
```

```
[42]: print(chi_matrix.shape)
```

```
(470247, 1000)
```

```
[43]: X_train_counts = chi_matrix.copy()
```

```
[44]: print(X_train_counts.shape)
```

```
(470247, 1000)
```

```
[45]: #TOP WORDS FOR TEST
X_test_counts = X_test_counts[:,top_words[0]]
```

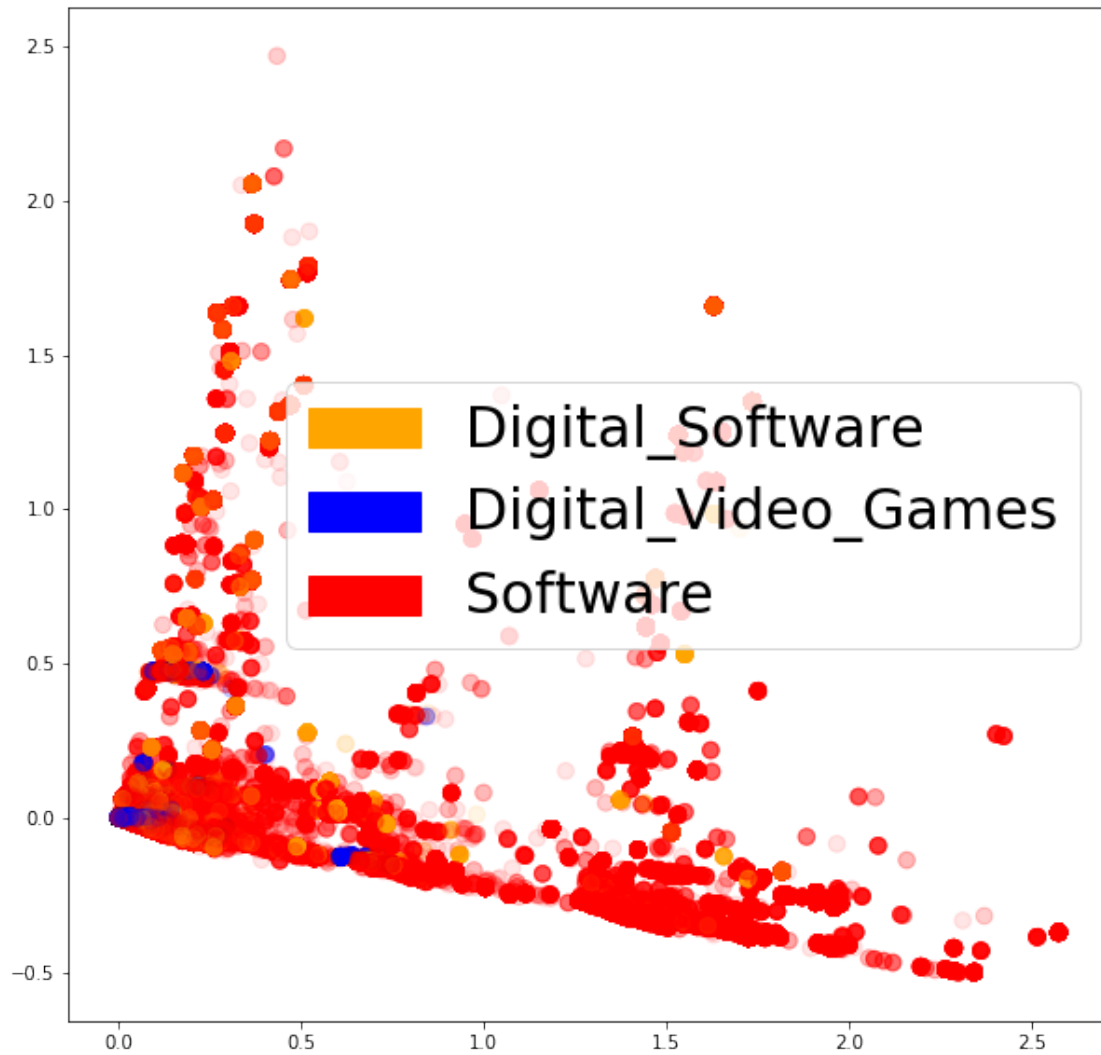
```
[46]: print(X_test_counts.shape)
```

```
(117562, 1000)
```

0.17 Visualizing the embeddings

Here, we apply linear dimensionality reduction to see if we can find separations between the groups.

```
[47]: fig = plt.figure(figsize=(10, 10))
plot_LSA(X_train_counts, y_train)
plt.show()
```



These embeddings don't look very cleanly separated. Let's see if we can still fit a useful model on them.

0.17.1 Fitting a classifier

Starting with a logistic regression is a good idea. It is simple, often gets the job done, and is easy to interpret.

0.17.2 Logistic regression

```
[48]: # balance classes
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(y_train),
                                                  y_train)

classes = np.unique(y_train)
class_weights = dict(zip(classes, class_weights))
class_weights
```

```
[48]: {0: 1.9216030010297651, 1: 1.355725653001211, 2: 0.574056508761971}
```

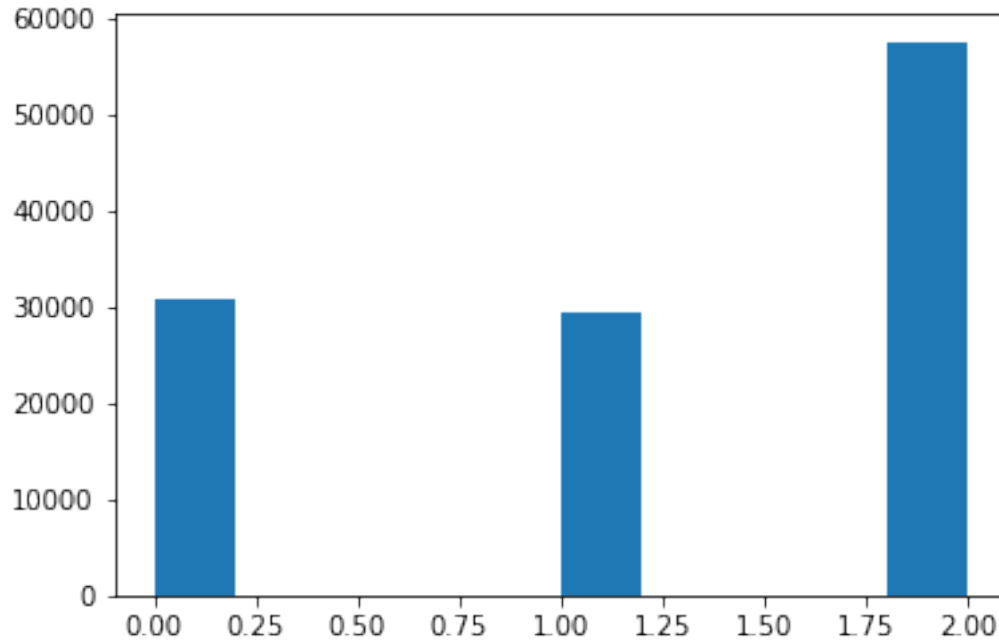
```
[49]: from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=30.0, class_weight=class_weights, #class_weight =
    ↳ 'balanced',
                        solver='newton-cg',
                        multi_class='multinomial', n_jobs=1, random_state=40)
    ↳ #, n_jobs=-1 (default) all, -2 all cpus but one are used
clf.fit(X_train_counts, y_train) #, class_weight=class_weights to balance data
```

```
[49]: LogisticRegression(C=30.0,
                        class_weight={0: 1.9216030010297651, 1: 1.355725653001211,
                                      2: 0.574056508761971},
                        dual=False, fit_intercept=True, intercept_scaling=1,
                        l1_ratio=None, max_iter=100, multi_class='multinomial',
                        n_jobs=1, penalty='l2', random_state=40, solver='newton-cg',
                        tol=0.0001, verbose=0, warm_start=False)
```

```
[50]: y_predicted_counts = clf.predict(X_test_counts)
```

```
[51]: plt.hist(y_predicted_counts)
```

```
[51]: (array([30698.,    0.,    0.,    0.,    0., 29282.,    0.,    0.,
           0., 57582.]),
      array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
      <a list of 10 Patch objects>)
```



0.17.3 Evaluation

Let's start by looking at some metrics to see if our classifier performed well at all.

```
[52]: accuracy, precision, recall, f1 = get_metrics(y_test, y_predicted_counts)
      print("accuracy = %.3f, precision = %.3f, recall = %.3f, f1 = %.3f" %
            ↳(accuracy, precision, recall, f1))
```

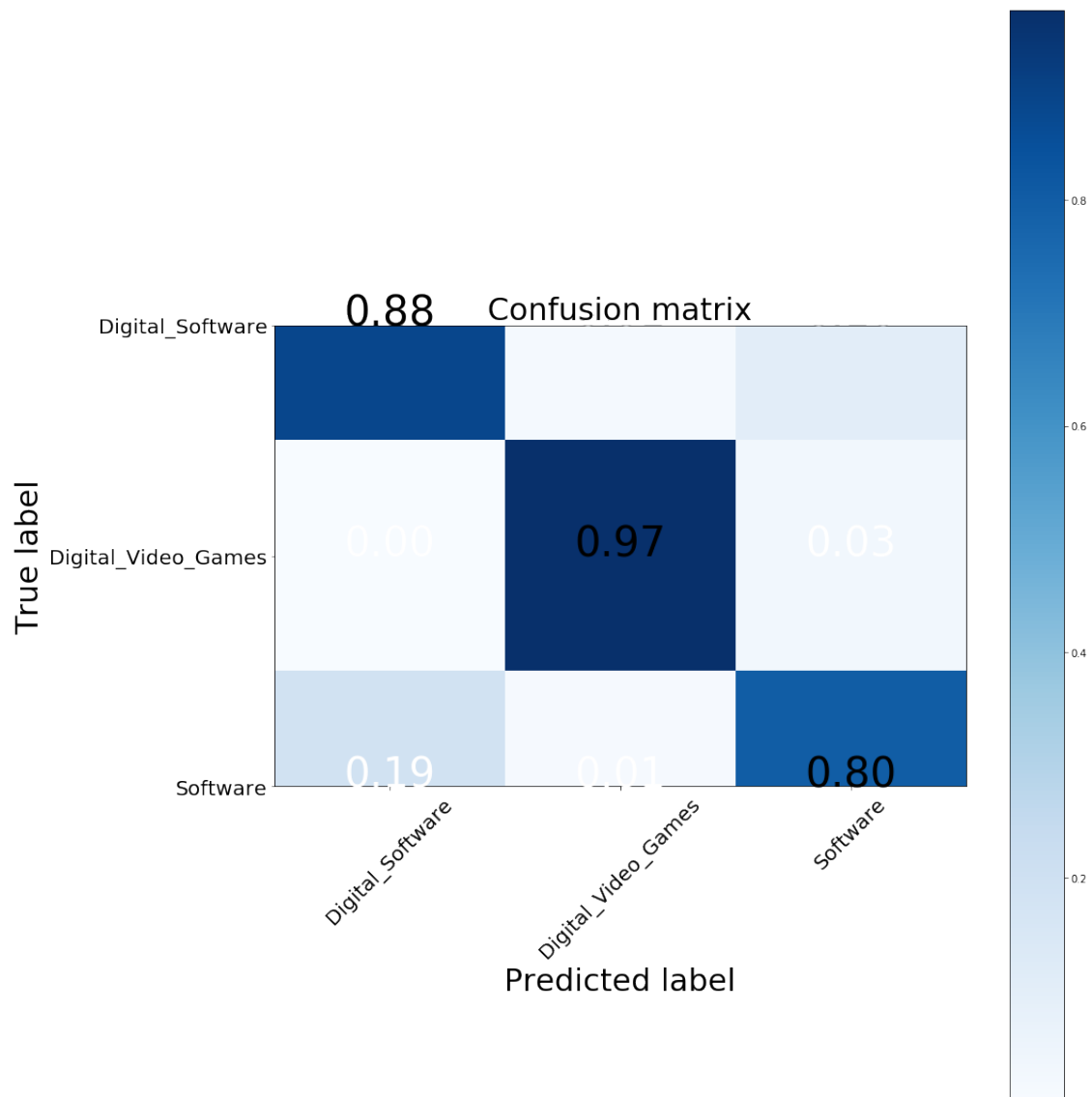
```
accuracy = 0.856, precision = 0.888, recall = 0.856, f1 = 0.863
```

0.17.4 Inspection

A metric is one thing, but in order to make an actionable decision, we need to actually inspect the kind of mistakes our classifier is making. Let's start by looking at the confusion matrix.

```
[53]: cm = confusion_matrix(y_test, y_predicted_counts)
      fig = plt.figure(figsize=(15, 15))
      plot = plot_confusion_matrix(cm, classes=['Digital_Software',
            ↳'Digital_Video_Games', 'Software'], normalize=True, title='Confusion matrix')
      plt.show()
      print(cm)
```

Normalized confusion matrix



```
[[17864  293  2107]
 [   74 28161   869]
 [12760  828 54606]]
```

```
[54]: #IMPORTANCE OF WORDS
importance = get_most_important_features(count_vectorizer, clf, 10)
```

```
[55]: importance[1]['tops'] #tops for video games
```

```
[55]: [(5.8721997659747975, 'advertisement'),
 (6.056004926710933, 'auriculoerapy'),
 (6.067608435417419, 'acpa'),
```

```
(6.189084280002864, 'armonica'),
(6.366940460936444, 'annihilation'),
(6.654794575584947, 'anvil'),
(6.662276675378382, 'ajq'),
(6.749679684006429, 'aboard'),
(6.856134258149394, 'al'),
(8.879801571394873, 'arcxt')]
```

```
[56]: importance[0]['tops'] #digital software
```

```
[56]: [(4.732064914147096, 'arcticlean'),
(4.916537265320087, 'accessory'),
(5.022995206385733, 'annie'),
(5.149171414134906, 'abuse'),
(5.250643295788913, 'albian'),
(5.3368857002896295, 'anywere'),
(5.450814857466169, 'ability'),
(5.960461665146921, 'accurate'),
(6.0128726237316785, 'amorpium'),
(6.061294136122159, 'auorware')]
```

```
[57]: importance[2]['tops'] #software
```

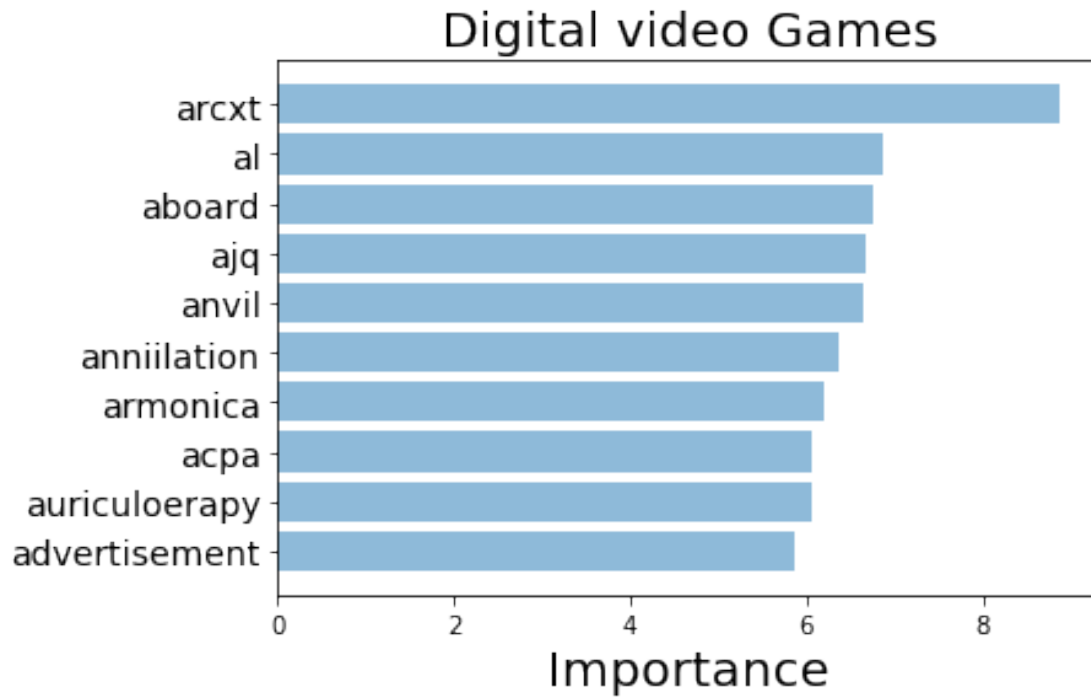
```
[57]: [(4.728202879859752, 'accountz'),
(4.747692774415891, 'amapi'),
(4.798641640556464, 'amr'),
(4.84254274779429, 'admirals'),
(4.855943234965648, 'atari'),
(5.335865550472419, 'adapter'),
(5.635358372442225, 'arrangement'),
(5.651636066332626, 'affiliate'),
(5.92872288647246, 'artsy'),
(6.144311096751719, 'attack')]
```

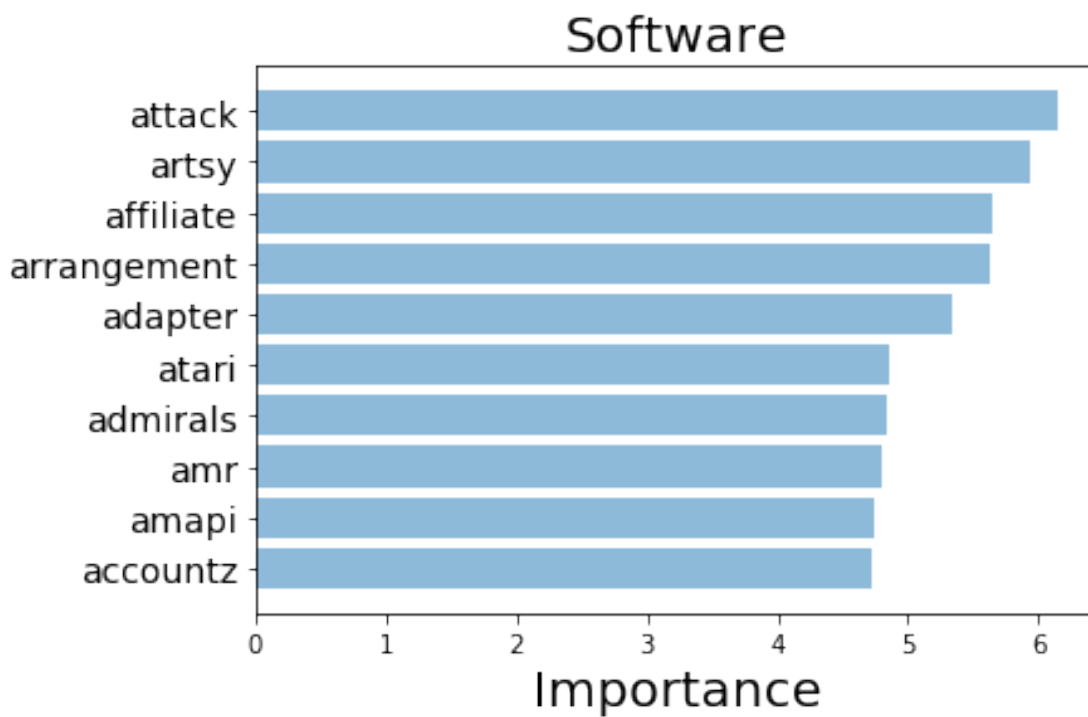
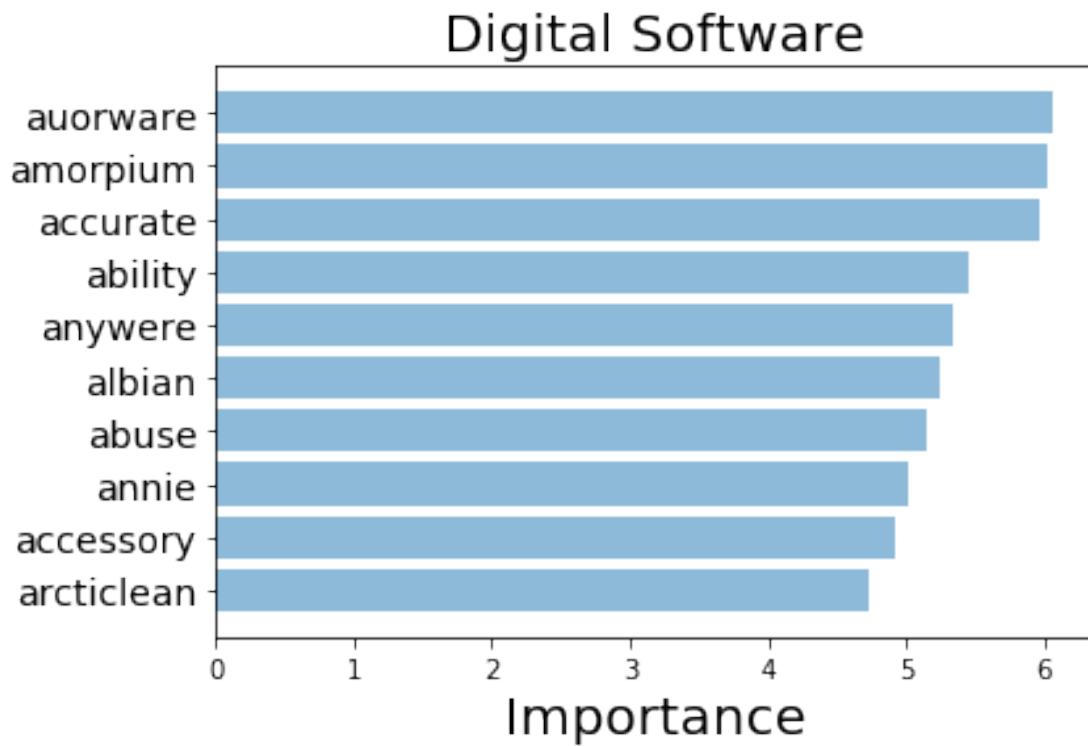
```
[58]: #video games (1)
top_scores_video = [a[0] for a in importance[1]['tops']]
top_words_video = [a[1] for a in importance[1]['tops']]
#digital software (0)
top_scores_digital = [a[0] for a in importance[0]['tops']]
top_words_digital = [a[1] for a in importance[0]['tops']]
##software
top_scores_software = [a[0] for a in importance[2]['tops']]
top_words_software = [a[1] for a in importance[2]['tops']]

print("Most important words")
plot_important_words(top_scores_video, top_words_video, name = 'Digital video_
↳Games')
```

```
plot_important_words(top_scores_digital, top_words_digital, name = 'Digital_↪Software')  
plot_important_words(top_scores_software, top_words_software, name = 'Software')
```

Most important words





0.18 TFIDF Bag of Words

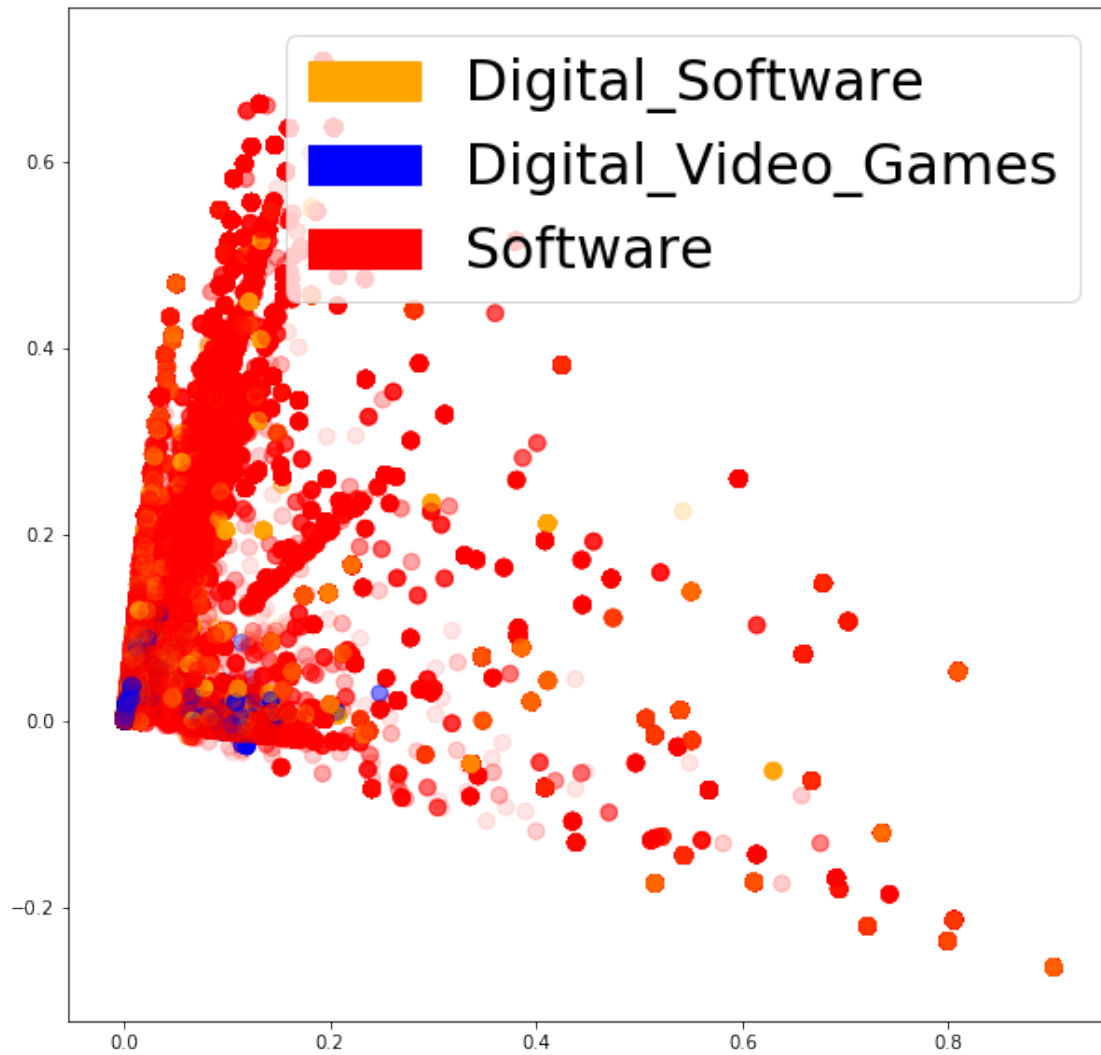
Let's try a slightly more subtle approach. Now, we will use a TF-IDF (Term Frequency, Inverse Document Frequency) which means weighing words by how frequent they are in our dataset, discounting words that are too frequent (as of, the and others), because they just add noise.

```
[59]: def tfidf(data):  
      tfidf_vectorizer = TfidfVectorizer()  
  
      train = tfidf_vectorizer.fit_transform(data)  
  
      return train, tfidf_vectorizer  
  
X_train_tfidf, tfidf_vectorizer = tfidf(X_train)  
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
[60]: print(X_train_tfidf.shape)
```

```
(470247, 15354)
```

```
[61]: fig = plt.figure(figsize=(10, 10))  
      plot_LSA(X_train_tfidf, y_train)  
      plt.show()
```



```
[62]: clf_tfidf = LogisticRegression(C=30.0, class_weight= class_weights,#
    ↪ 'balanced',
    solver='newton-cg',
    multi_class='multinomial', n_jobs=1, random_state=40)
clf_tfidf.fit(X_train_tfidf, y_train)

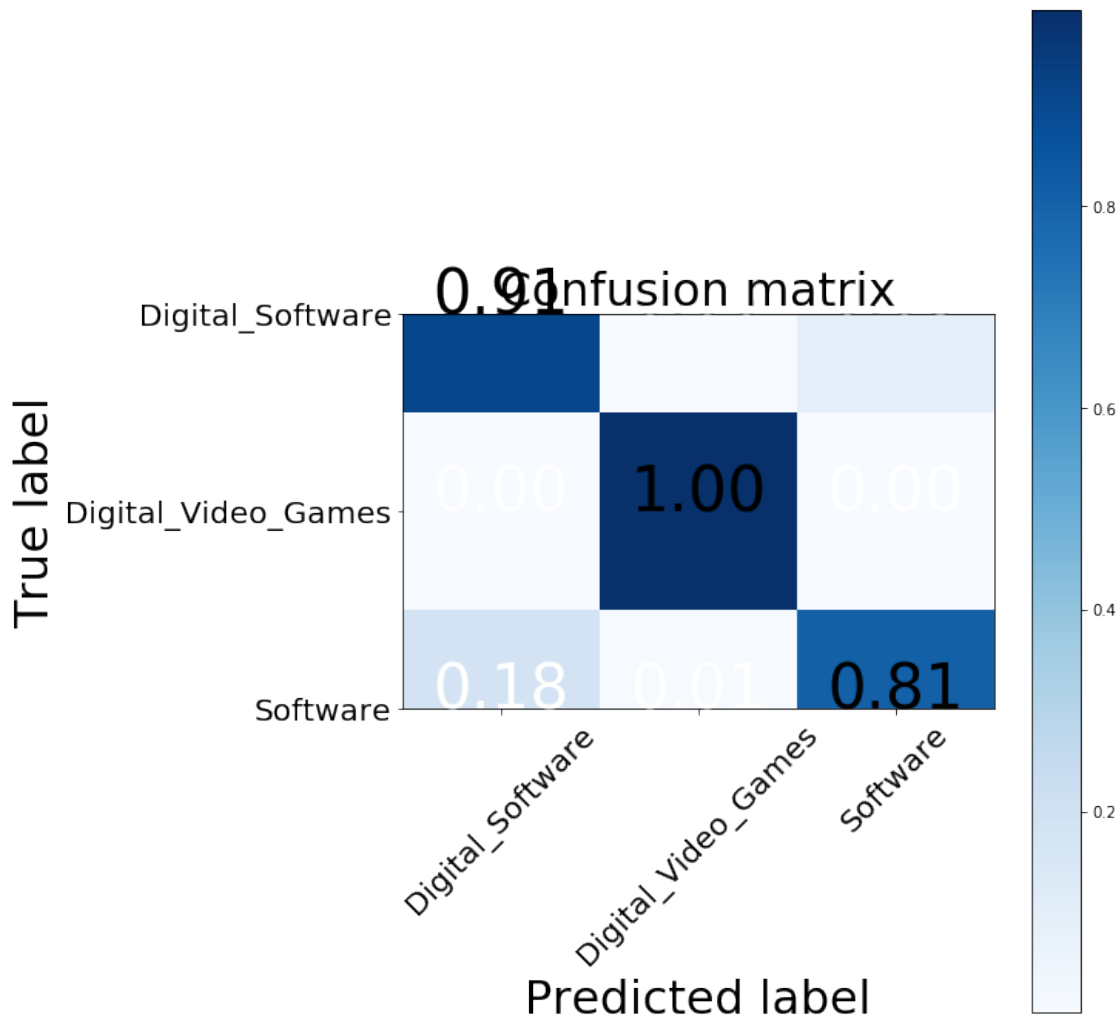
y_predicted_tfidf = clf_tfidf.predict(X_test_tfidf)
```

```
[63]: accuracy_tfidf, precision_tfidf, recall_tfidf, f1_tfidf = get_metrics(y_test,#
    ↪ y_predicted_tfidf)
print("accuracy = %.3f, precision = %.3f, recall = %.3f, f1 = %.3f" %
    ↪ (accuracy_tfidf, precision_tfidf, recall_tfidf, f1_tfidf))
```

```
accuracy = 0.873, precision = 0.907, recall = 0.873, f1 = 0.881
```

```
[64]: cm2 = confusion_matrix(y_test, y_predicted_tfidf)
fig = plt.figure(figsize=(10, 10))
plot = plot_confusion_matrix(cm2, classes=['Digital_Software', 'Digital_Video_Games', 'Software'], normalize=True, title='Confusion matrix')
plt.show()
print("TFIDF confusion matrix")
print(cm2)
print("BoW confusion matrix")
print(cm)
```

Normalized confusion matrix



TFIDF confusion matrix

```
[[18421    55  1788]
 [   21 28962   121]
 [12482   411 55301]]
```

```
BoW confusion matrix
[[17864  293 2107]
 [   74 28161  869]
 [12760  828 54606]]
```

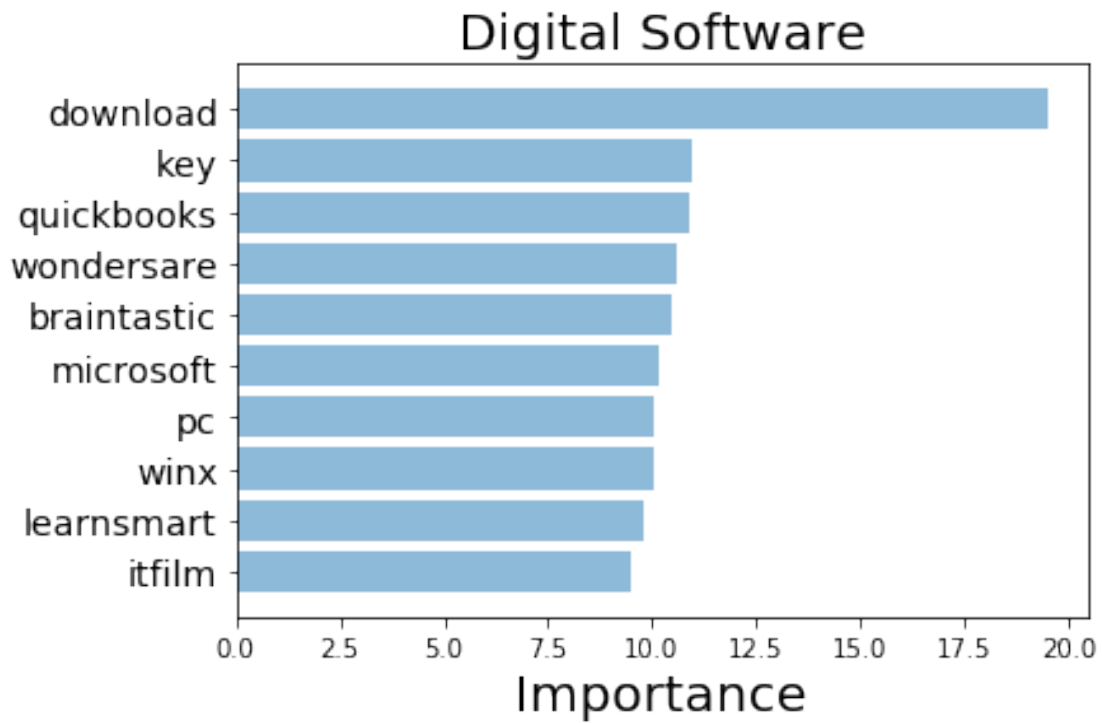
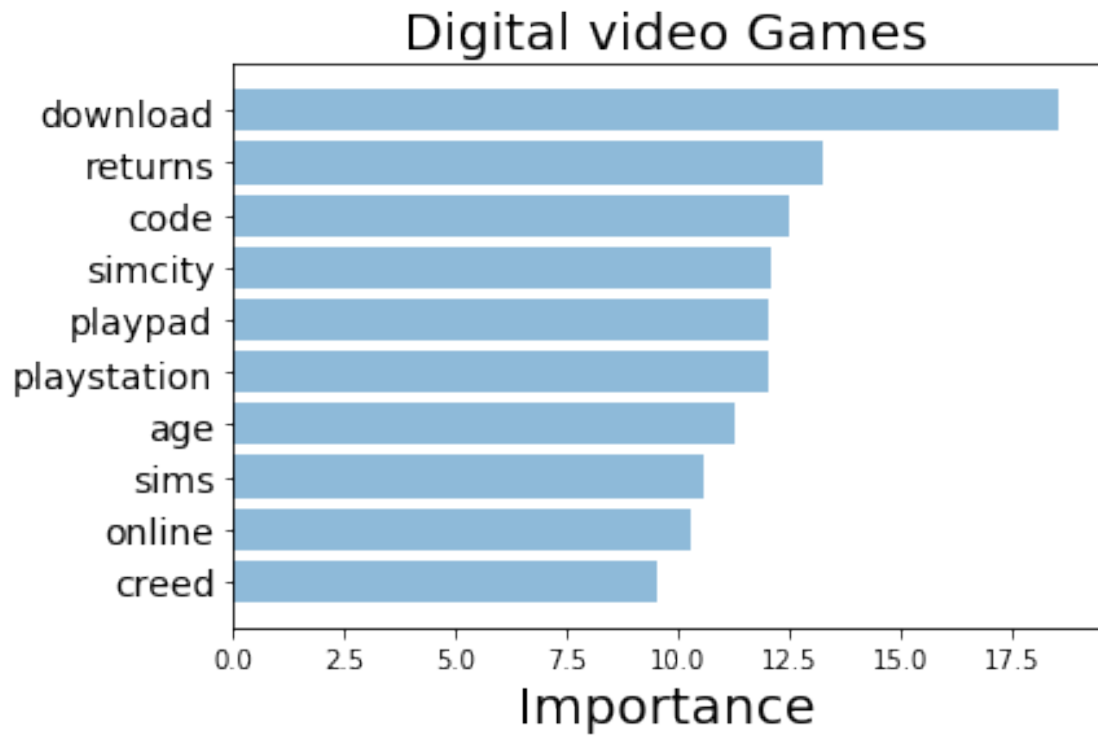
0.18.1 Looking at important coefficients of the model

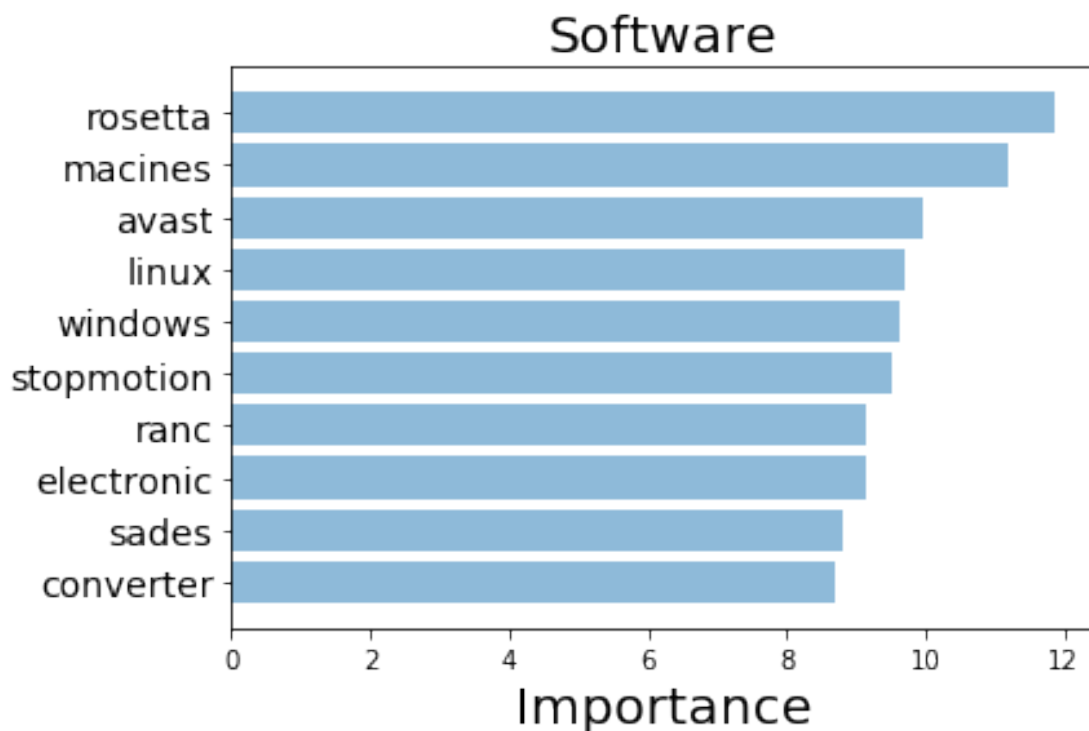
```
[65]: importance_tfidf = get_most_important_features(tfidf_vectorizer, clf_tfidf, 10)
```

```
[66]: #video games (1)
top_scores_video = [a[0] for a in importance_tfidf[1]['tops']]
top_words_video = [a[1] for a in importance_tfidf[1]['tops']]
#digital software (0)
top_scores_digital = [a[0] for a in importance_tfidf[0]['tops']]
top_words_digital = [a[1] for a in importance_tfidf[0]['tops']]
##software
top_scores_software = [a[0] for a in importance_tfidf[2]['tops']]
top_words_software = [a[1] for a in importance_tfidf[2]['tops']]

print("Most important words")
plot_important_words(top_scores_video, top_words_video, name = 'Digital video_
↳Games')
plot_important_words(top_scores_digital, top_words_digital, name = 'Digital_
↳Software')
plot_important_words(top_scores_software, top_words_software, name = 'Software')
```

Most important words





The words the model picked up look much more relevant! Although our metrics on our held out validation set haven't increased much, we have much more confidence in the terms our model is using.

0.19 word2vec

0.19.1 Capturing semantic meaning

Our first models have managed to pick up on high signal words. However, it is unlikely that we will have a training set containing all relevant words. To solve this problem, we need to capture the semantic meaning of words.

0.19.2 Enter word2vec

Word2vec is a model that was pre-trained on a very large set of sentences, and provides embeddings that map words that are similar close to each other. A quick way to get a sentence embedding for our classifier, is to average word2vec scores of all words in our sentence.

```
[67]: #!pip install gensim  
#!pip install --upgrade pip
```

```
#!wget https://github.com/mmihaltz/word2vec-GoogleNews-vectors/blob/master/
↪GoogleNews-vectors-negative300.bin.gz
```

Note: The `load_word2vec_format()` method also has an optional `limit` argument which will only load the supplied number of vectors – so you could use `limit=500000` to cut the memory requirements by about 5/6ths. (And, since the GoogleNews and other vector sets are usually ordered from most- to least-frequent words, you'll get the 500K most-frequent words. Lower-frequency words generally have much less value and even not-as-good vectors, so it may not hurt much to ignore them.)

```
[68]: import gensim
word2vec = gensim.models.KeyedVectors.
↪load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True,
↪limit = 100000)
```

1 data to introduce in the model.

```
[69]: df.head(2)
```

```
[69]:  product_id          product_title  product_category  star_rating \
0  B00U7LCE6A      ccleaner free download                0          4
1  B00HRJ MOM4  resumemaker professional deluxe                0          3

   helpful_votes  total_votes  verified_purchase  review_headline \
0              0            0                  Y
1              0            0                  Y          ree

   review_body  product_category_label \
0      far good      Digital_Software
1  needs little work      Digital_Software

   token_product_title      token_review_body \
0  [ccleaner, free, download]      [far, good]
1  [resumemaker, professional, deluxe]  [needs, little, work]

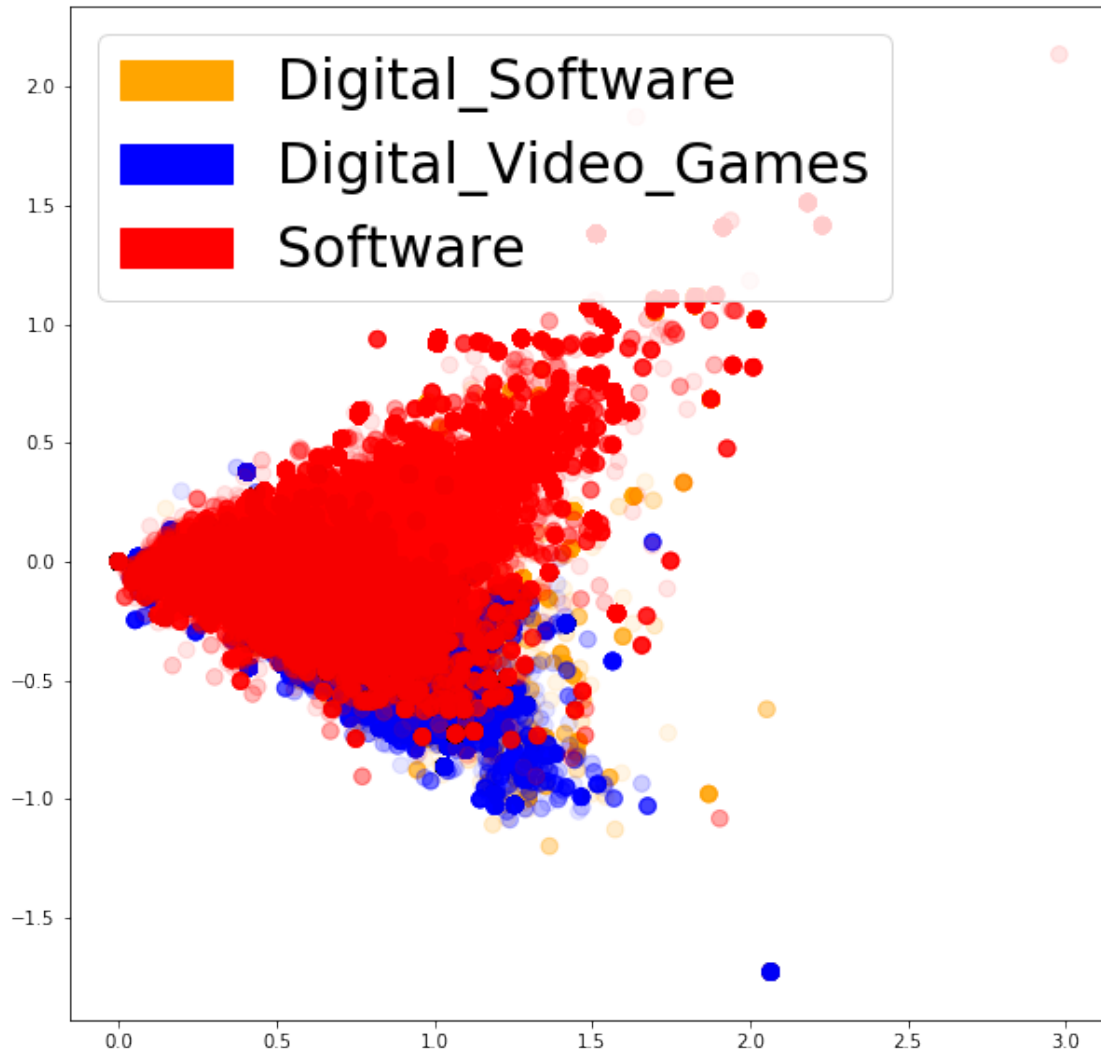
   token_review_headline
0      []
1      [ree]
```

```
[70]: list_labels = df["product_category"].tolist()
df_corpus2 = df[["product_category", 'product_title', 'token_product_title']] #
↪introduce more columns
```

```
[71]: embeddings = get_word2vec_embeddings(word2vec, df_corpus2, name =
↪'token_product_title')
```

```
X_train_word2vec, X_test_word2vec, y_train_word2vec, y_test_word2vec = ↵
↵train_test_split(embeddings, list_labels,
↵
↵                test_size=0.2, random_state=40)
```

```
[72]: fig = plt.figure(figsize=(10, 10))
plot_LSA(embeddings, list_labels)
plt.show()
```



These look a little bit more separated, let's see how our logistic regression does on them!

```
[73]: clf_w2v = LogisticRegression(C=30.0, class_weight = class_weights, ↵
↵    #class_weight='balanced',
↵
↵    solver = 'newton-cg',
```



```

        multi_class = 'multinomial', random_state = 40)
clf_w2v.fit(X_train_word2vec, y_train_word2vec)
y_predicted_word2vec = clf_w2v.predict(X_test_word2vec)

```

```

[74]: accuracy_word2vec, precision_word2vec, recall_word2vec, f1_word2vec =
    ↪ get_metrics(y_test_word2vec, y_predicted_word2vec)
    print("accuracy = %.3f, precision = %.3f, recall = %.3f, f1 = %.3f" %
    ↪ (accuracy_word2vec, precision_word2vec,
    ↪ recall_word2vec, f1_word2vec))

```

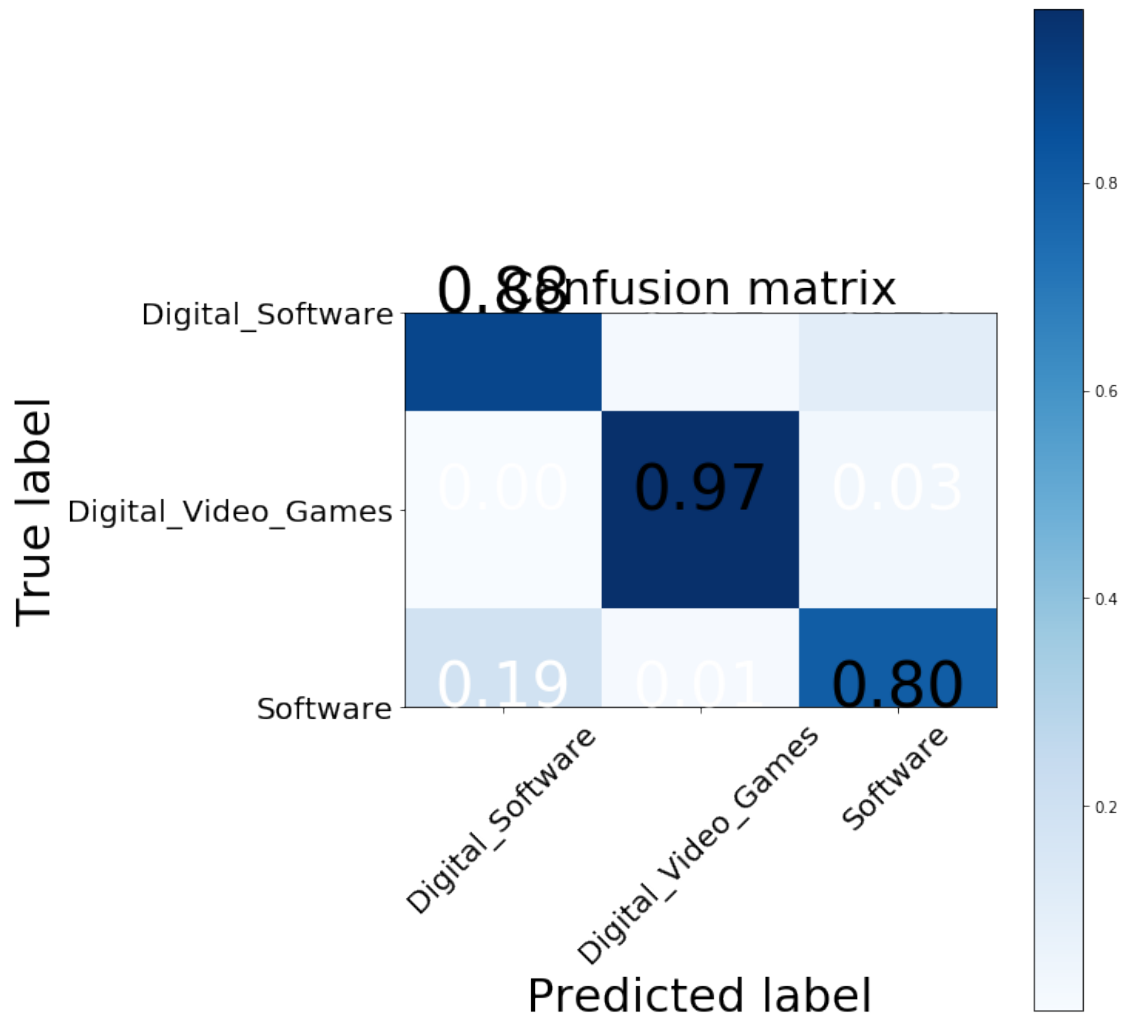
accuracy = 0.809, precision = 0.849, recall = 0.809, f1 = 0.814

```

[75]: cm_w2v = confusion_matrix(y_test_word2vec, y_predicted_word2vec)
    fig = plt.figure(figsize=(10, 10))
    plot = plot_confusion_matrix(cm, classes=['Digital_Software',
    ↪ 'Digital_Video_Games', 'Software'], normalize=True, title='Confusion matrix')
    plt.show()
    print("Word2Vec confusion matrix")
    print(cm_w2v)
    print("TFIDF confusion matrix")
    print(cm2)
    print("BoW confusion matrix")
    print(cm)

```

Normalized confusion matrix



Word2Vec confusion matrix

```
[[17182  682 2400]
 [  338 28115  651]
 [13552 4849 49793]]
```

TFIDF confusion matrix

```
[[18421    55 1788]
 [   21 28962  121]
 [12482   411 55301]]
```

BoW confusion matrix

```
[[17864   293 2107]
 [   74 28161  869]
 [12760   828 54606]]
```

```
[76]: # **More values for the logistic**
```

```
[77]: dicti_n = {0:"Digital_Software", 1:"Digital_Video_Games", 2:"Software"}
```

```
[78]: #clf_w2v.score(X, y)
arr = clf_w2v.predict_proba(X_test_word2vec)
df_predict= pd.DataFrame(data=arr)
df_predict.columns = ["Digital_Software", "Digital_Video_Games", "Software"]
arr = clf_w2v.predict(X_test_word2vec)
df_predict['predicted_label'] = pd.DataFrame(data=arr.flatten())
df_predict['prediction_title'] = df_predict['predicted_label'].copy()
df_predict = df_predict.replace({"prediction_title": dicti_n})
arr = np.array(y_test_word2vec)
df_predict['real_label'] = pd.DataFrame(data=arr.flatten())
df_predict["max_pred"] = df_predict[["Digital_Software",
↪ 'Digital_Video_Games', 'Software']].max(axis=1)
print(df_predict.shape)
df_predict.head()
```

(117562, 7)

```
[78]:   Digital_Software  Digital_Video_Games  Software  predicted_label  \
0          0.025940          0.000846  0.973214             2
1          0.162306          0.463053  0.374641             1
2          0.169370          0.000059  0.830571             2
3          0.528857          0.000017  0.471126             0
4          0.008877          0.004264  0.986858             2

      prediction_title  real_label  max_pred
0          Software             2  0.973214
1  Digital_Video_Games             2  0.463053
2          Software             2  0.830571
3  Digital_Software             2  0.528857
4          Software             2  0.986858
```

```
[79]: df_predict[df_predict['max_pred'] < .60].tail(20)
```

```
[79]:   Digital_Software  Digital_Video_Games  Software  predicted_label  \
117366          0.586714          0.000583  0.412702             0
117370          0.408311          0.104081  0.487608             2
117379          0.484080          0.004732  0.511189             2
117396          0.528857          0.000017  0.471126             0
117414          0.588009          0.411903  0.000087             0
117427          0.162306          0.463053  0.374641             1
117434          0.162306          0.463053  0.374641             1
117447          0.042814          0.385320  0.571866             2
117449          0.405033          0.004700  0.590267             2
117468          0.527834          0.015643  0.456522             0
117469          0.162306          0.463053  0.374641             1
```

117506	0.162306	0.463053	0.374641	1
117537	0.528857	0.000017	0.471126	0
117540	0.586714	0.000583	0.412702	0
117545	0.162306	0.463053	0.374641	1
117546	0.474616	0.000683	0.524701	2
117552	0.553615	0.000045	0.446340	0
117553	0.520785	0.012557	0.466657	0
117554	0.591094	0.000003	0.408903	0
117555	0.528857	0.000017	0.471126	0

	prediction_title	real_label	max_pred
117366	Digital_Software	2	0.586714
117370	Software	2	0.487608
117379	Software	2	0.511189
117396	Digital_Software	2	0.528857
117414	Digital_Software	0	0.588009
117427	Digital_Video_Games	1	0.463053
117434	Digital_Video_Games	1	0.463053
117447	Software	2	0.571866
117449	Software	2	0.590267
117468	Digital_Software	2	0.527834
117469	Digital_Video_Games	2	0.463053
117506	Digital_Video_Games	1	0.463053
117537	Digital_Software	2	0.528857
117540	Digital_Software	0	0.586714
117545	Digital_Video_Games	2	0.463053
117546	Software	2	0.524701
117552	Digital_Software	2	0.553615
117553	Digital_Software	0	0.520785
117554	Digital_Software	2	0.591094
117555	Digital_Software	2	0.528857

```
[80]: df_predict_long = pd.melt(df_predict, id_vars=['predicted_label',
↪ 'prediction_title'])
df_predict_long.head()
```

```
[80]:
```

	predicted_label	prediction_title	variable	value
0	2	Software	Digital_Software	0.025940
1	1	Digital_Video_Games	Digital_Software	0.162306
2	2	Software	Digital_Software	0.169370
3	0	Digital_Software	Digital_Software	0.528857
4	2	Software	Digital_Software	0.008877

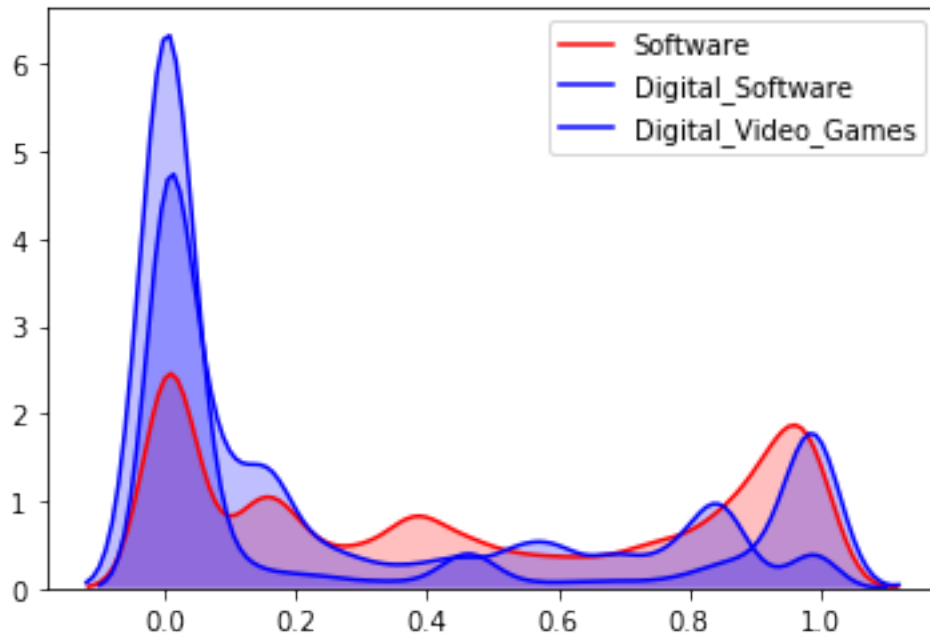
```
[81]: import seaborn as sns
sns.kdeplot(df_predict_long.loc[(df_predict_long['variable']=='Software'),
'value'], color='r', shade=True, Label='Software')
```

```

sns.kdeplot(df_predict_long.
    ↳loc[(df_predict_long['variable']=='Digital_Software'),
          'value'], color='b', shade=True, Label='Digital_Software')
sns.kdeplot(df_predict_long.
    ↳loc[(df_predict_long['variable']=='Digital_Video_Games'),
          'value'], color='b', shade=True, Label='Digital_Video_Games')

```

[81]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc2ed287950>



1.0.1 Further inspection

In order to provide some explainability, we can leverage a black box explainer such as LIME.

```

[82]: from lime import lime_text
      from sklearn.pipeline import make_pipeline
      from lime.lime_text import LimeTextExplainer

      df_corpus = df['product_title'] #df['review_headline']#df['full_text']
      #df_corpus = df['review_body', 'review_headline']
      df_labels = df["product_category"]

      X_train_data, X_test_data, y_train_data, y_test_data =
      ↳train_test_split(df_corpus, df_labels, test_size=0.2,

```

```

↪random_state=40)
vector_store = word2vec
def word2vec_pipeline(examples):
    global vector_store
    tokenizer = RegexpTokenizer(r'\w+')
    tokenized_list = []
    for example in examples:
        example_tokens = tokenizer.tokenize(example)
        vectorized_example = get_average_word2vec(example_tokens, vector_store,
↪generate_missing=False, k=300)
        tokenized_list.append(vectorized_example)
    return clf_w2v.predict_proba(tokenized_list)

c = make_pipeline(count_vectorizer, clf)

```

```

[83]: def explain_one_instance(instance, class_names):
        explainer = LimeTextExplainer(class_names=class_names)
        exp = explainer.explain_instance(instance, word2vec_pipeline,
↪num_features=6, top_labels=2)
        return exp

def visualize_one_exp(features, labels, index, class_names):
    exp = explain_one_instance(features[index], class_names = class_names)
    print('Index: %d' % index)
    print('True class: %s' % class_names[labels[index]])
    exp.show_in_notebook(text=True)

```

```

[84]: class_names = ['Digital_Software', 'Digital_Video_Games', 'Software']
visualize_one_exp(X_test_data, y_test_data, 65, class_names)

```

```

Index: 65
True class: Software
<IPython.core.display.HTML object>

```

```

[85]: visualize_one_exp(X_test_data, y_test_data, 60, class_names)

```

```

Index: 60
True class: Digital_Software
<IPython.core.display.HTML object>

```

```

[86]: visualize_one_exp(X_test_data, y_test_data, 62, class_names)

```

```

Index: 62
True class: Digital_Software

```

<IPython.core.display.HTML object>

```
[87]: visualize_one_exp(X_test_data, y_test_data, 68, class_names)
```

Index: 68

True class: Digital_Video_Games

<IPython.core.display.HTML object>

```
[88]: visualize_one_exp(X_test_data, y_test_data, 58, class_names)
```

Index: 58

True class: Software

<IPython.core.display.HTML object>

```
[89]: visualize_one_exp(X_test_data, y_test_data, 56, class_names)
```

Index: 56

True class: Digital_Software

<IPython.core.display.HTML object>

```
[90]: visualize_one_exp(X_test_data, y_test_data, 52, class_names)
```

Index: 52

True class: Software

<IPython.core.display.HTML object>

```
[91]: label_to_text = {
      0: "Digital_Software",
      1: "Digital_Video_Games",
      2: "Software"
    }
    random.seed(40)
    #importance of words
    sorted_contributions = get_statistical_explanation(X_test_data.tolist(), 100,
    ↪word2vec_pipeline, label_to_text)
```

```
[92]: sorted_contributions
```

```
[92]: {'Digital_Software': {'bottom': deluxe      -0.007775
    essentials  -0.006164
    quickbooks  -0.002955
    ome         -0.002803
```

turbotax	-0.002682
efile	-0.002677
publiser	-0.002596
potosop	-0.002360
openoffice	-0.002255
norton	-0.002086
internet	-0.001617
converter	-0.001433
back	-0.000929
wi	-0.000665
premiere	-0.000517
premier	-0.000449
programpc	-0.000447
basic	-0.000426
tax	-0.000378
software	-0.000306
mac	0.000513
avast	0.001191
video	0.001226
financial	0.001455
elements	0.001517
user	0.002185
federal	0.002207
pc	0.003165
industry	0.003212
adobe	0.003285
microsoft	0.005154
pro	0.005687
apace	0.006107
antivirus	0.011239
state	0.011652
security	0.012200
quicken	0.015116
free	0.017359
block	0.018185
fed	0.021634
download	0.022492
dtype: float64, 'tops': download	0.022492
fed	0.021634
block	0.018185
free	0.017359
quicken	0.015116
security	0.012200
state	0.011652
antivirus	0.011239
apace	0.006107
pro	0.005687

microsoft	0.005154	
adobe	0.003285	
industry	0.003212	
pc	0.003165	
federal	0.002207	
user	0.002185	
elements	0.001517	
financial	0.001455	
video	0.001226	
avast	0.001191	
mac	0.000513	
software	-0.000306	
tax	-0.000378	
basic	-0.000426	
programpc	-0.000447	
premier	-0.000449	
premiere	-0.000517	
wi	-0.000665	
back	-0.000929	
converter	-0.001433	
internet	-0.001617	
norton	-0.002086	
openoffice	-0.002255	
potosop	-0.002360	
publiser	-0.002596	
efile	-0.002677	
turbotax	-0.002682	
ome	-0.002803	
quickbooks	-0.002955	
essentials	-0.006164	
deluxe	-0.007775	
dtype: float64}, 'Software': {'bottom': microsoft -0.003474		
command	-0.002850	
contractor	-0.002428	
middle	-0.002116	
ome	-0.001500	
...		
deluxe	0.005151	
dragon	0.005897	
games	0.006495	
version	0.014777	
old	0.015534	
Length: 127, dtype: float64, 'tops': old		0.015534
version	0.014777	
games	0.006495	
dragon	0.005897	
deluxe	0.005151	

```

...
ome          -0.001500
middle       -0.002116
contractor   -0.002428
command      -0.002850
microsoft    -0.003474
Length: 127, dtype: float64}, 'Digital_Video_Games': {'bottom': typing
-0.005492
year         -0.004149
deluxe       -0.003546
overkill     -0.002858
sims         -0.001620

...
game         0.004918
limited       0.008361
subscription 0.009551
card         0.010809
download     0.015530
Length: 74, dtype: float64,
'tops': download      0.015530
card                0.010809
subscription        0.009551
limited              0.008361
game                0.004918

...
sims         -0.001620
overkill     -0.002858
deluxe       -0.003546
year         -0.004149
typing       -0.005492
Length: 74, dtype: float64}}

```

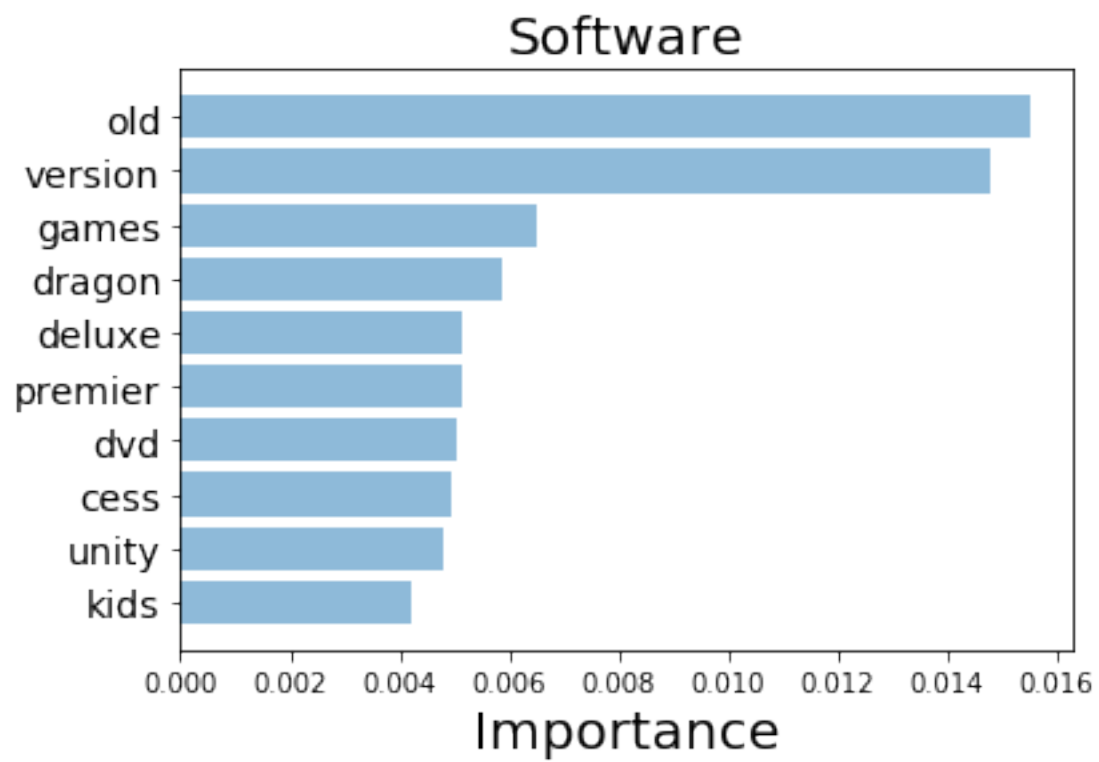
```

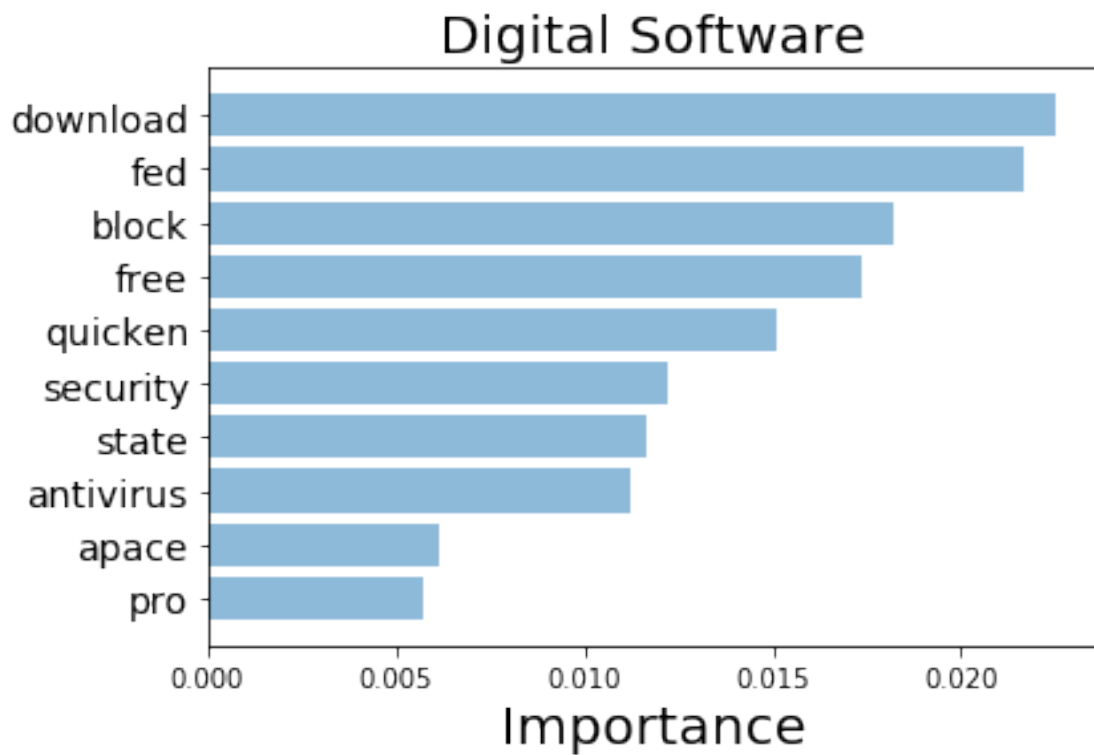
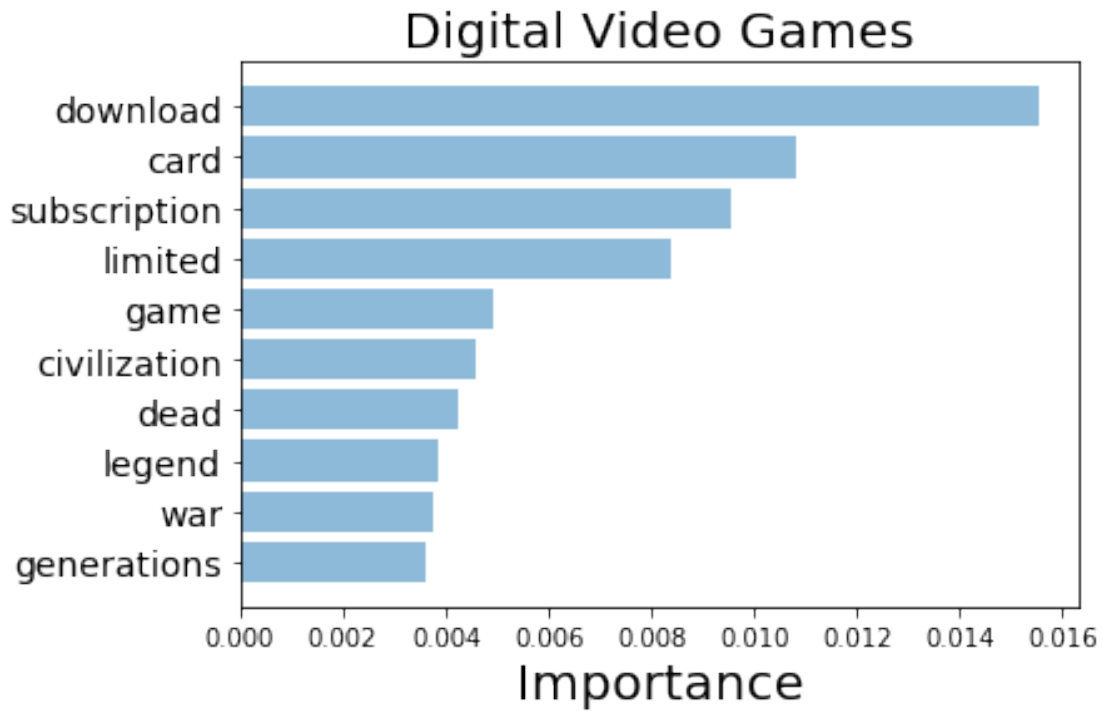
[93]: #Software
top_words_Software = sorted_contributions['Software']['tops'][:10].index.
↳ tolist()
top_scores_Software = sorted_contributions['Software']['tops'][:10].tolist()
top_words_GAMES = sorted_contributions['Digital_Video_Games']['tops'][:10].
↳ index.tolist()
top_scores_GAMES = sorted_contributions['Digital_Video_Games']['tops'][:10].
↳ tolist()
top_words_Digital = sorted_contributions['Digital_Software']['tops'][:10].index.
↳ tolist()
top_scores_Digital = sorted_contributions['Digital_Software']['tops'][:10].
↳ tolist()

plot_important_words(top_scores_Software, top_words_Software, "Software")
plot_important_words(top_scores_GAMES, top_words_GAMES, "Digital Video Games")

```

```
plot_important_words(top_scores_Digital, top_words_Digital, 'Digital Software')
```





1.1 Conclusion:

We get better results with **TFIDF Bag of Words** option. Also, aggregate review info to the model does not help to the improvement of the results of the model that only consider 'product title'. Further analysis are required to get a better model.

1.2 NEXT STEPS

1.2.1 Based in the results there are several things to do to improve the results:

1. Improve the text treatment, there are words that have misspelling or not sense. More time cleaning it is necessary. Also, it is necessary to spend more time cleaning reviews (eliminate adjectives, verbs, keep only nouns that can help to identify the category).
2. Use `n_grams > 1`. A word by itself may mean nothing, specially when we are treating with names of products.
3. This first approach was mostly exploratory, for this reason we only use logistic regression, but there are models that may help to improve the results (based tree methodologies, deep leaning, bayesian methods, between others), additionally would be interesting combine different methodologies using emsemble methodologies like (max voting, averaging, bagging or boosting). Compare the results with ROC curves and information criteria as AIC and BIC.
4. We can try to improve the results combining text with numerical variables. Use number of votes to give more weights to those reviews with more votes may help to find better keywords for the products. Include a label for relevant/irrelevant information based in the available information.
5. Use Spark to speed the results.
6. Use methodologies as SMOTE to balance the data.
7. Explore other alternative to Word2Vect like: fastText GloVe or ELMO. Consider other alternatives like pre-trained NLP models to get better results, as for example BERT, ULMFIT, between others.
8. Use BlazingText's implemetation of Word2Vec.
9. Use H2OAutoML algorithm to if we can improve the results.
10. INclude more categories to the analysis.
11. Implement the ideas wrote at the beggining to fight misclassification issues.
12. During the treatment we eliminate numbers, because we think this may aggregate noise to result. We may recheck this treatment and evaluate which may remain in the trining dataset.

[]:

[]:

[]:

[]: