Text analysis Project (Fabio Costantino)

My role in the Company

As a Content Analyst in the Localization Team, I am responsible for migrating data in and out of our Content Management System called STEP. These are mostly text data - a.k.a. content, which are then published daily in our website for all customers to consume.

As a Localization team, we mainly deal with translated text in 12 different languages, therefore also managing the flow of translation files in and out of the company. In order to translate our content, I have daily contact with external translation agencies and manage the use of a Translation Management System called XTM, which helps organize all files and workflows in a single tool.

Besides, we also provide our manager(s) and the rest of the team with reports of many kind: from tailored collection of data coming from different sources, to reports on products (or families of products) performance.

Some of the tools I use on a daily/weekly basis to perform my job are: **STEP Content Management System**, **XTM Translation Management System**, **Microsoft SQL Server Management Studio**, **Report Builder** from **Adobe Analytics**, **DOMO** as a BI dashboard tool, **Office package**, **Jupyter Notebook** (with **Python**).

Brief

Create a tool that can automatically detect the presence in product pages of text in foreign language

Project explanation

Business Problem

The website of our company has a conspicuous number of product pages, translated from English into 12 foreign languages. It may sometimes happen that, for technical reason or for human error, some pages are displaying in the wrong language (e.g. English text showing in the French website), which can negatively impact SEO scores.

Current Process

At the moment we rely on the feedback of customers and colleagues stumbling upon pages with the wrong language implemented. We can also create reports from our Content Management System (CMS) to try and spot mistakes, but it is not uncommon that many pages escape this system for technical reason. This happens because the CMS clearly doesn't distinguish between languages, but relies on a system of flags (mainly binomial flags such as `translated` / `not translated`) that are often selected and unselected in a wrong way.

Current issues

Google often actively *threatens* companies with cancellation of their paid advertising profile, or its algorithm simply penalises in case a substantial amount of foreign text is displayed in a website. An approch that mainly relies on sporadic feedback will drag issues for months before being spotted by chance, causing the SEO scores for certain products to sink over time.

Proposed solution

My solution is to create a script that can analyze the text of selected parts of the website (e.g. a list of products in one family or for one brand), in order to proactively perform periodical checks with few simple clicks. The solution should make this *house-keeping task* quite effortless, with minimal changes in the code.

The problem will be approached in two parts: firstly I will code a tool that can extract a dataset with text in a given language and in English; secondly I will create a way of detecting foreign languages. I decided to build a neural network to classify text based on presence of stopwords in a given language (in this particular case, English and Italian). The decision of using a neural network is based on the potential amount of future data that the algorithm should be able to deal with (over a million pages).

Pros and Cons

This system should allow the check of hundreds of products in few minutes. However, it becomes exponentially more computationally expensive for massive amounts of pages, therefore it requires a limited amount of pages to be analyzed each time - i.e. a few hundreds.

Assumptions

Given that our CMS can prevent translated content to go automatically out for translation again if nothing has changed in the text, the cost of a *False Positive* is actually null. Therefore, I would assume that **False Negatives** are the least acceptable error for the purpouse of this analysis.

Hypothesis

My hypothesis is that I can obtain a model that can classify over 85% of the observations in the dataset, maintaining the number of **False Negatives** as low as possible.

0. Importing libraries

In [1]:
```
# To manipulate data
import pandas as pd
import numpy as np

# To collapse list of lists
from pandas.core.common import flatten

# To request web pages
import requests

# To parse pages in a website and read them from HTML
from bs4 import BeautifulSoup
from lxml import html

# To use regex
import re

# To tokenize text
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

# To count word frequencies
from collections import Counter

# To import a dictionary of English words
import enchant

# To create train/test split
from sklearn.model_selection import train_test_split

# To create a neural network
from sklearn.neural_network import MLPClassifier

# To visualize confusion matrix
from sklearn.metrics import classification_report, confusion_matrix, accuracy_
score
import seaborn as sns

# To evaluate visually the model
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

# To set all text as visible
pd.set_option('display.max_colwidth', None)

# To suppress warnings
import warnings; warnings.simplefilter('ignore')
```

```
C:\Users\E0658269\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmo
dels\tools\_testing.py:19: FutureWarning: pandas.util.testing is deprecated.
Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

1. Sourcing and transforming data

Before extracting any data from the website, I need to be sure about company policies on scraping product pages. Reading articles about scraping in the internet and asking within RS itself makes me understand that there is no risk in doing so. However, it is recommended in several articles to refer to the `robots.txt` page of the website to check whether there is any restriction to be aware of.

Visualizing the URL `https://it.rs-online.com/robots.txt` , I checked that no particular warning or restriction has been issued by RS about scraping the pages, apart of course from blocking several pages with personal information from customers. I can see that pages linked with baskets, logins etc., have been correctly excluded from the list of crawlable pages.

Now that I am sure about not breaching any security policy with my analysis, I can proceed.

1.1 Creating a code to extract text

To gather data, I will scrape a portion of the Italian website of my company using the library `BeautifulSoup` . I will build this method with functions that allow for better efficiency, but also will make my code re-usable.

In [2]:
```
# Choosing a URL that contains all the products I want to examine
URL_with_links = 'https://it.rs-online.com/web/c/connettori/connettori-iec-di-alimentazione-ed-accessori/accessori-per-connettori-iec/?rpp=100'

# Specifying the root domain
domain_URL = 'https://it.rs-online.com'
```

The first function allows the extraction of all the links contained in one page. This is possible by searching the HTML code of the page for `href` elements using the `xpath` method. However, the output would include all the links to irrelevant elements - such as images, therefore I will filter the results to obtain only those links that end with a 7 digit number and a `/` , which is the typical format for our product pages.

In [3]:
```
# Creating a function that extracts all links in the products page
def links_extraction(url):
    # Creating an empty list with all links
    list_of_links = []
    # Looping through the first 10 pages in case more are present
    for page in range(10):
        # Request the webpage
        request = requests.get(URL_with_links)
        # Creating an object and exploring its content
        source = request.content
        # Reading the HTML of the source object
        page_source = html.fromstring(source)
        # Mining for all the links within the page
        list_of_links.extend(page_source.xpath('//@href'))
        # Creating a regex that only leaves us with links that end with 7 digi
ts (product numbers)
        regex = re.compile(r'^(?!.*\d{7}\/?$).*$')
        # Filtering the entire list with the regex to obtain the final list
        list_of_links_filtered = [i for i in list_of_links if not regex.match(
i)]
    return list_of_links_filtered
```

Now I will pass the function on to the URL we started with.

In [4]:
```
# Extracting the links from the URL
list_of_links = links_extraction(URL_with_links)

# Eliminating duplicates by creating a dictionary (which can't have duplicate
 keys)
list_of_links = list(dict.fromkeys(list_of_links))

#Visualizing a couple of links
list_of_links[:2]
```

Out[4]:
```
['/web/p/accessori-per-connettori-iec/5260724/',
 '/web/p/accessori-per-connettori-iec/0544128/']
```

As we can see, these links don't contain the domain. To have complete and functional links, I will create a function that prepends the domain name specified at the beginning of this section. I included the domain earlier, so that the code can run almost entirely automatically from the beginning of this notebook.

In [5]:
```
# Creating a function that prepends the domain
def prepend_domain(list, str):
    # Define the element that will be appended
    str += '%s'
    # Define the operation to be done in each element of the list
    list = [str % i for i in list]
    return list
```

In [6]:
```
# Using the prepending function
final_URL_list = prepend_domain(list_of_links, domain_URL)

# #Visualizing a few complete links
final_URL_list[:2]
```

Out[6]:
```
['https://it.rs-online.com/web/p/accessori-per-connettori-iec/5260724/',
 'https://it.rs-online.com/web/p/accessori-per-connettori-iec/0544128/']
```

At this point, I need to create a function that will analyze and scrape each link in the `final_URL_list` that we just created.

This function creates a data frame with the **ID** of each product, the **title** that the customers can see in each page, and the **text** included in the page. Since the text is sometimes created collating different files, I will scrape the entire `productDetailsContainer` section of the HTML code, to capture all text in one go.

In [7]:
```
# Creating a scraping function for the necessary text
def scrape_each_page(url):
        # Creating empty objects that will contain the information needed
        product_id = []
        product_title = []
        product_description = []
        # Request to get the URL specified
        page = requests.get(url)
        # Parsing the content of the HTML
        soup = BeautifulSoup(page.content, 'html.parser')
        page_source = html.fromstring(page.content)
        # The ID is taken by the URL, being always in the last position
        prod_id = url.split("/",-1)[-2]
        product_id.append(prod_id)
        # Title and descriptions are obtained with anchors to different sectio
ns of the HTML
        title = soup.find('div', class_='[redacted]')
        title_text = title.h1.text
        product_title.append(title_text)
        description = soup.find('div', class_='[redacted]')
        description_text = description.text
        product_description.append(description_text)
        # Creating an object that contains all the information needed
        data = [product_id,product_title, product_description]
        # Transforming the object in a Data Frame, and transposing it to make
 it more clear
        corpus = pd.DataFrame(data).T
        # Renaming the column appropriately
        corpus.rename(columns={0:'ID', 1:'Title', 2:'Text'}, inplace=True)
        # Getting the final form of the data frame
        return corpus
```

All data scraped will be added to a database called `final_corpus` .

In [8]:
```
# Creating an empty data frame to contain the corpus
final_corpus = pd.DataFrame()

# Iterating over each entry in the list to concatenate the text
for i in final_URL_list[:]:
    final_corpus = pd.concat([scrape_each_page(i), final_corpus])

# Visualizing a couple of rows
final_corpus.head(2)
```

Out[8]:

	ID	Title	Text
0	8688743	\r\n HN 14.43b cable guard\r\n	\nDettagli prodotto\n\n\n\nProtezione cavo Schurter\n\r\n \r\n Gamma professionale di protezioni per cavo Schurter disponibili in numerose dimensioni. \r\n \r\n \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
0	8688706	\r\n HN 14.109c cable guard\r\n	\nDettagli prodotto\n\n\n\nProtezione cavo Schurter\n\r\n \r\n Gamma professionale di protezioni per cavo Schurter disponibili in numerose dimensioni. \r\n \r\n \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n

It is clear that the text obtained needs some cleaning up. I will do so using **regex**. Also, I will transform all text to lower case, in order not to introduce bias in the frequency of words during my analysis.

In [9]:
```
# Cleaning the dataframe using regex
final_corpus[['Title', 'Text']] = final_corpus[['Title', 'Text']].replace([r
'[^a-zA-Z]+', # numerical and special char
                                                                            r
'(^| ).(( ).)*( |$)', # all single letters
                                                                            r'
 +'], # trailing spaces
                                                                           " ",
regex=True)

# Applying lower case to all text
final_corpus['Title'] = final_corpus['Title'].str.lower()
final_corpus['Text'] = final_corpus['Text'].str.lower()

# Having a look at the result
final_corpus.head(3)
```

Out[9]:

	ID	Title	Text
0	8688743	hn cable guard	dettagli prodotto protezione cavo schurter gamma professionale di protezioni per cavo schurter disponibili in numerose dimensioni
0	8688706	hn cable guard	dettagli prodotto protezione cavo schurter gamma professionale di protezioni per cavo schurter disponibili in numerose dimensioni
0	8767928	insulating boot cg kg	dettagli prodotto accessori iec schurter gamma professionale di accessori iec schurter gli accessori comprendono un pressacavo con serracavo progettato per prevenire gli scollegamenti accidentali del connettore maschio avvitamento dalla parte anteriore

Finally, I will create a dataset that lists all pieces of text referring to the same product in one single column. This will simplify my analysis. I want to keep titles and text description separated because they are included in different sections in our CMS, and it is much more efficient to know exactly what to amend rather than having to figure it out at the end.

Also, in this last bit of code, I introduce the appropriate language label for this particular project.

In [10]:

```
# Creating datasets with only appropriate columns
subsets = [final_corpus[['ID', 'Title']], final_corpus[['ID', 'Text']]]

# Concatenating the two subsets
IT_corpus = pd.concat(subsets, join='outer', sort=False)

# Creating a column with all text together
IT_corpus['Title/Text'] = IT_corpus['Title'].fillna(IT_corpus['Text'])

# Leaving only the relevant columns
IT_corpus = IT_corpus[['ID', 'Title/Text']]

# Exploring the dataset to get an idea of how much data we have
IT_corpus.info()

# Creating a copy of the excel file for labelling (see section 1.2)
IT_corpus.to_excel('IT_corpus.xlsx')

#Visualizing a few lines
IT_corpus.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 148 entries, 0 to 0
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   ID          148 non-null    object
 1   Title/Text  148 non-null    object
dtypes: object(2)
memory usage: 3.5+ KB
```

Out[10]:

	ID	Title/Text
0	8688743	hn cable guard
0	8688706	hn cable guard
0	8767928	insulating boot cg kg
0	8688762	cover accessoryboot
0	1902964	cord retaining kit flat

This concludes the automatic scraping of the website for a single language, which covers the first part of my **Proposed solution**.

1.2 Creating the final dataset for modelling

In order to correctly train the data (and as a trial for the reusability of my previous code), the exact same dataframe of section **1.2** has been obtained for the English counterpart of these pages, scraping the following URL and domain:

```
https://uk.rs-online.com/web/c/connectors/mains-iec-connectors-accessories/iec-conn
ector-accessories/?rpp=100
https://uk.rs-online.com
```

Also, I manually labelled both files with a `1` for a row that contains English text, and `0` for a row with Italian text, so that both corpora are ready for modelling.

In [11]:
```
# Importing the English dataset
EN_corpus_labelled = pd.read_excel('EN_corpus_labelled.xlsx')

# Exploring a sample
EN_corpus_labelled.sample(2)
```

Out[11]:

	ID	Title/Text	Language
71	2110913	bulgin insulation boot	1
64	1739943	schurter insulation boot	1

In [12]:
```
# Importing the labelled file for the Italian text
IT_corpus_labelled = pd.read_excel('IT_corpus_labelled.xlsx')

# Exploring a sample
IT_corpus_labelled.sample(2)
```

Out[12]:

	ID	Title/Text	Language
146	544128	dettagli prodotto calotta isolante per connettori fe ina aschio polarizzati protezione in pvc sta pata progettata per isolare proteggere le connessioni posteriori di connettori aschio di rete connettori fe ina di rete colore nero tensione di funziona ento ax test flash kv attenzione connettori di ali entazione rs seguenti sono adatti per il collega ento diretto all ali entazione di rete eno che non venga indicato altri enti bisogna fare attenzione nello scegliere un prodotto per il collega ento alla rete elettrica	0
96	8688756	dettagli prodotto protezione cavo schurter ga professionale di protezioni per cavo schurter disponibili in nu erose di ensioni	0

Finally, I obtain a single dataframe containing both English and Italian. I will then use this for modelling in **section 3**. This dataframe will have of course many ID repeated, but this is necessary to identify the products later on.

In [13]:
```
# Merging the two datasets with the 'outer' method
EN_IT_corpus = pd.merge(IT_corpus_labelled, EN_corpus_labelled, how='outer')

# Obtaining some information about the dataset
EN_IT_corpus.info()

# Exploring a sample
EN_IT_corpus.sample(2)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 300 entries, 0 to 299
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   ID          300 non-null    int64 
 1   Title/Text  300 non-null    object
 2   Language    300 non-null    int64 
dtypes: int64(2), object(1)
memory usage: 9.4+ KB
```

Out[13]:

	ID	Title/Text	Language
288	1739943	product details schurter appliance inlet protection cover fro schurter this connector cap is for use with the appliance inlet and aintains an ip ip level of environ ental protection when the connector is un ated	1
6	1902981	cover thermoplastic accessory boot	1

2. Exploring the English dataset

Before modelling, I would like to explore the English dataset. I want to check that the text used is compatible with the taxonomical position of the product family within the website structure, paying particular attention to the most important words used throughout the descriptions.

To do so, I will first extrapolate the cleaned English text from our final dataset, then tokenize all words in it.

In [14]:
```
# English only dataset
cleaned_en = EN_IT_corpus[EN_IT_corpus['Language']==1]

# Creating a column with tokens
cleaned_en['Tokens'] = cleaned_en['Title/Text'].apply(word_tokenize)

# Visualizing a sample
cleaned_en.sample(2)
```

Out[14]:

	ID	Title/Text	Language	Tokens
241	8692799	product details schurter iec accessories professional of schurter iec accessories accessories include the cord retaining strain relief which is pri arily designed to protect any accidental disconnections fro the plug and fuse holders for use with the series of power entry odules screw on fro front side	1	[product, details, schurter, iec, accessories, professional, of, schurter, iec, accessories, accessories, include, the, cord, retaining, strain, relief, which, is, pri, arily, designed, to, protect, any, accidental, disconnections, fro, the, plug, and, fuse, holders, for, use, with, the, series, of, power, entry, odules, screw, on, fro, front, side]
188	8688740	schurter insulation boot	1	[schurter, insulation, boot]

Next, I will remove **stopwords** to get a cleaner dataset. I don't perform this transformation on the entire corpus because, as we will see in section 3, stopwords will be important for my model.

In [15]:
```
# Creating an object with english stopwords
en_stopwords = stopwords.words('english')

# Adding a few words to be removed
en_stopwords.extend(["schurter", "fro", "ther"])

# Creating a function that iterate over a column of text
def stop_remove(word_list):
    return [word for word in word_list if word not in en_stopwords]

# Deploying the function on the Tokens column
cleaned_en['Tokens'] = cleaned_en['Tokens'].apply(stop_remove)

# Visualizing a sample
cleaned_en.sample(2)
```

Out[15]:

	ID	Title/Text	Language	Tokens
197	1739263	schurter retaining clip	1	[retaining, clip]
218	801932	bulgin insulation boot	1	[bulgin, insulation, boot]

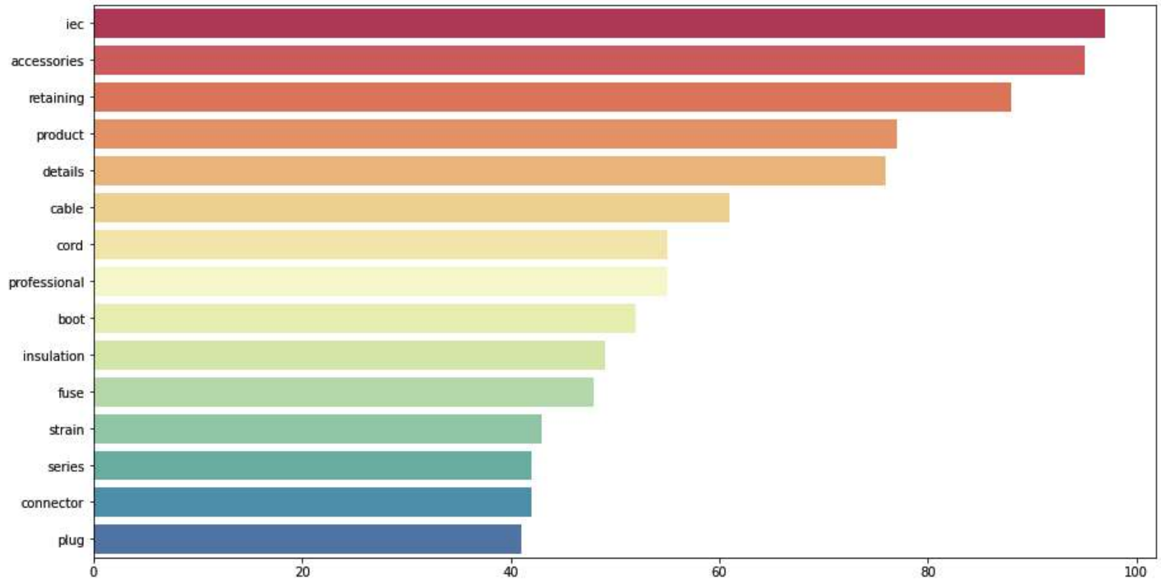
Now let us proceed to visualize the top 15 words used in the corpus.

In [16]:
```
# First a list of all tokens is built and all tokens are counted
tokens_counted = Counter(list(flatten(cleaned_en['Tokens'].to_list())))

# Creating a descending series according to frequency
top_15 = pd.Series(tokens_counted).sort_values(ascending=False)[:15]

# Plotting the first 15 most frequent words
plt.figure(figsize=(15, 8))
sns.barplot(x=top_15,y=top_15.index, palette='Spectral')
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29220282748>
```



Since we are exploring a section of the website called **IEC Connector Accessories**, this list seems quite spot on as it is.

3. Modelling a Neural Network

The model I want to apply is a simple Neural Network that would output a binary classification of `1` for English and `0` for non-English text.

I am aware that the volume of data I have for this project is limited and a neural network might seem inappropriate at first glance. However, I would like to create a tool that can cope with thousands of pages at a time. At the time of writing, our website contains roughly 1.5 million, therefore I am aiming for higher efficiency in building a tool that can cope with these numbers.

The very first step will consist in creating 4 columns with scores, as follows:

- `En` : for words included in the English dictionary. Each word will receive a score of 1.
- `En stopwords` : for words included in the list of English stopwords. Each stopword will receive a score of 2 (since these drammatically improve the likelihood of the text being in English).
- `It` : for words **not** included in the English dictionary. Each word will receive a score of 1. For the porpouse of this project, I will assume that every word that is not recognized as an English word is included in the *other* language.
- `IT stopwords` : for words included in the list of Italian stopwords. Each stopword will receive a score of 2 (since these drammatically improve the likelihood of the text being in Italian).

Finally, each cell will be passed through the Neural Network, that will classify our text and the results will be evaluated based on accuracy and number of **False Negatives** (see *Assumptions* for more information on this).

In [17]:
```
# First I will create a copy of the dataset to apply the model
EN_IT_corpus_model = EN_IT_corpus

# Creating a function that assigns the scores to each word
def function_model(column):
    # Creating the 4 columns
    EN_IT_corpus_model['en stopwords'] = 0
    EN_IT_corpus_model['en'] = 0
    EN_IT_corpus_model['it stopwords'] = 0
    EN_IT_corpus_model['it'] = 0
    # Iterating the function over each row
    for index, sentence in enumerate(column):
        # Split the text
        words = sentence.split()
        # Assining a dictionary of English to an object
        en_dictionary = enchant.Dict("en_GB")
        # Iterating a logic check on each word
        for word in words:
            if en_dictionary.check(word) == True:
                if True:
                    if word in stopwords.words('english'):
                        EN_IT_corpus_model.at[index, 'en stopwords'] += 2
                    else:
                        EN_IT_corpus_model.at[index,'en'] += 1
            else:
                if word in stopwords.words('italian'):
                    EN_IT_corpus_model.at[index,'it stopwords'] += 2
                else:
                    EN_IT_corpus_model.at[index,'it'] += 1

# Applying the function to our corpus
function_model(EN_IT_corpus_model["Title/Text"])

# Visualizing a sample to check implementation
EN_IT_corpus_model.sample(2)
```

Out[17]:

	ID	Title/Text	Language	en stopwords	en	it stopwords	it
293	2110907	product details iec connector insulating boots co prehensive offering of iec connector insulating boots offered in variety of size designed to assist in preventing accidental shorting of the contacts at the rear of an iec connector and also offering dust and dirt ingress protection thickness oulded boots to protect the rear connections of iec connectorstype fits stock nos and type fits stock nos and the discontinued and type fits stock nos and type fits stock nos and	1	44	48	0	7
76	8767953	dettagli prodotto accessori iec schurter ga professionale di accessori iec schurter gli accessori co prendono un pressacavo con serracavo progettato per prevenire gli scollega enti accidentali del connettore aschio avvita ento dalla parte anteriore	0	0	5	8	25

We can create now a train/test split in the dataset, on which to apply the model.

In [30]:
```
# Defining X and y
X = EN_IT_corpus_model[['en stopwords', 'en', 'it stopwords', 'it']]
y = EN_IT_corpus_model[['Language']]

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, ra
ndom_state = 2017)

# Visualizing sizes for the subsets
print('Train Size', X_train.shape, y_train.shape)
print('Test Size', X_test.shape, y_test.shape)
```

```
Train Size (225, 4) (225, 1)
Test Size (75, 4) (75, 1)
```

For this very simple neural network, I will be using the `MLPClassifier()` from the `sklearn` library. To choose the hidden layers of the network, I followed the simple rule of thumb of having one more layer than the number of classes in the output.

As for the number of neurons, I have tried many different combinations until I achieved a low number of False Negatives and an accuracy score that seemed not to be overfitted.

In [49]:
```
# Creating the algorithm shell
model = MLPClassifier(hidden_layer_sizes=(6,5,3),max_iter=100)

# Fit the data in the algorithm
model.fit(X_train, y_train)
```

Out[49]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(6, 5, 3), learning_rate='constant',
              learning_rate_init=0.001, max_iter=100, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

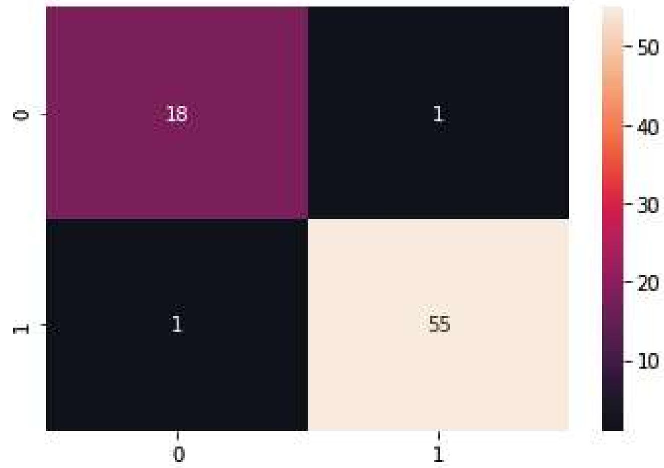
3.2. Evaluating the model

To answer the hypothesis laid out at the beginning, I will explore the number of False Negatives by visualizing the confusion matrix.

In [50]:
```
# Calculating the predictions for the model based on the test subset
predictions = model.predict(X_test)

# Visualizing the confusion matrix for the model
sns.heatmap(confusion_matrix(y_test,predictions), annot=True)
```

Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x2921f6b1ba8>



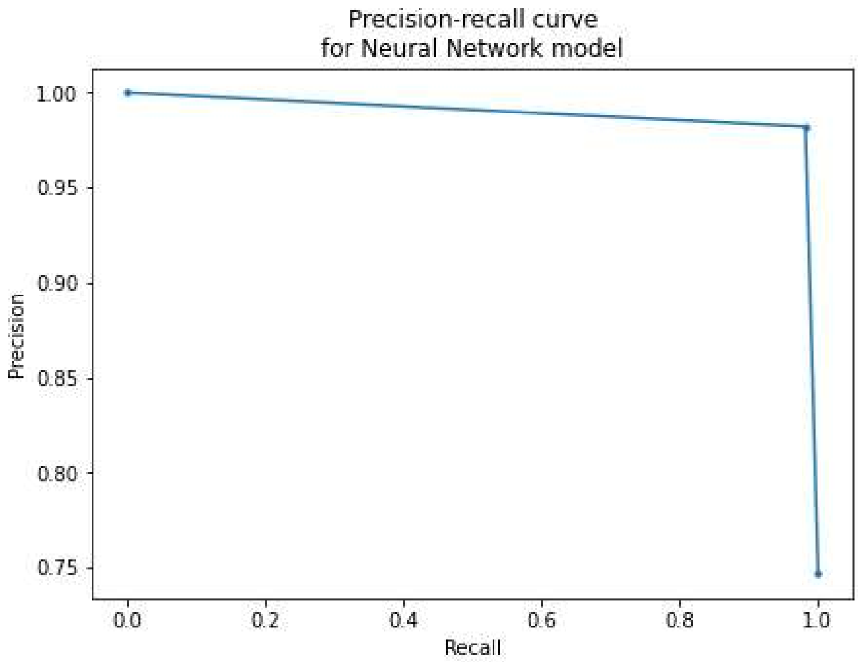
In [51]:
```
# Printing accuracy score, on which to base our judgement of the model
print('Accuracy of the model: {}%'.format(accuracy_score(y_test, predictions)*
100))
```

```
Accuracy of the model: 97.33333333333334%
```

To further evaluate the model, I want to visualize a **Precision-Recall** curve. Precision is used to evaluate how many times a model correctly evaluated something as *positive* out of all times that the label *positive* was assigned to the observations; whereas *recall* will tell us how many *positives* the model has correctly found out.

In [52]:
```
# Creating a precision-recall curve
NN_precision, NN_recall, _ = precision_recall_curve(y_test, predictions)

# Plotting the results
plt.figure(figsize=(7,5))
plt.plot(NN_recall, NN_precision, marker='.', label='Neural Network')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-recall curve\nfor Neural Network model')
plt.show()
```



4. Conclusions

I successfully created both a tool to scrape the website and a model to predict whether the language present in a list of URLs is English or not, as requested in the initial brief.

Given the limited dataset, I would consider this model to be a good fit for my initial hypothesis, since it could achieve more than 85% accuracy in the predictions while maintaining small the number of **False Negatives**. Although the accuracy obtained can be judged as bordering overfitting, it is my opinion that a bigger corpus would lead to a different answer.

The model is now ready to be deployed on different datasets and with minimal changes to the code, to help my colleagues identifying opportunity to improve the text in the website.