Recommending web articles classification using NLP, Machine Learning

Objective:

Perform web articles classification into 3 defined categories (Engineering, Startups & Business, Product & Design). Compare the classifier accuracy with different models ranging from Naive Bayes to Support Vector Machine (SVM).

Dataset:

The dataset of this project is JSON file for a group of categorized articles as we will divide those articles into 3 groups: training data, validating data, testing data.

Data Fields:

- Title of the article.
- Body of the article.
- Category of the article (Engineering, Startups & Business, Product & Design).

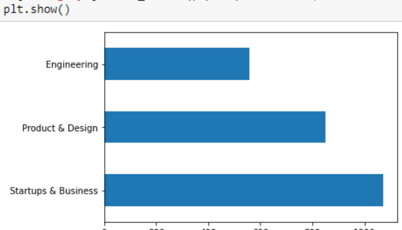
The steps to build the model are:

- Data Exploration.
- Text Preprocessing.
- Feature Extraction.
- Modeling.
- Evaluation.

<u>Data exploration:</u>

1. Importing the data and read it.
2. Explore its shape, type of categories.
3. Check if there is a null value.
4. Plot the categories distribution.

```
2481 rows × 3 columns

In [5]: df.shape
Out[5]: (2481, 3)

In [6]: df['category'].unique()
Out[6]: array(['Engineering', 'Startups & Business', 'Product & Design'],
              dtype=object)

In [7]: # Check null
        df.isnull().sum()
Out[7]: body        0
        title       0
        category    0
        dtype: int64

In [8]: df['category'].value_counts().plot(kind='barh')
        plt.show()
```



Text preprocessing:

1. Tokenization: Splitting text strings into smaller pieces, called "tokens" Tokenizing paragraphs into sentences and sentences into words is possible.
2. Convert letters into lowercase.
3. Remove stopwords.
4. Lemmatization which means conserving the root word of a word.
5. Printing the most common data in every label.

Tokenization

In [10]:
```
nltk.download('punkt')
nltk.download('wordnet')
```
```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Laptop\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Laptop\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```
Out[10]: True

In [11]:
```
df["title"]=df["title"].str.lower()
df["body"]=df["body"].str.lower()
df.head()
```
Out[11]:

	body	title	category
0	protecting netflix viewing privacy at scale\r\...	protecting netflix viewing privacy at scale	Engineering
1	introducing winston — event driven diagnostic ...	introducing winston - event driven diagnostic ...	Engineering
2	performance & usage at instagram\r\n\r\nat ins...	performance & usage at instagram	Engineering
3	the simple example of calculating and formatti...	refactoring a javascript video store	Engineering
4	billing applications have transactions that ne...	netflix billing migration to aws - part iii	Engineering

In [12]:
```
#tokenization of words
df['body'] = df.apply(lambda row: word_tokenize(row['body']), axis=1)
df.head()
```
Out[12]:

	body	title	category
0	[protecting, netflix, viewing, privacy, at, sc...	protecting netflix viewing privacy at scale	Engineering
1	[introducing, winston, —, event, driven, diagn...	introducing winston - event driven diagnostic ...	Engineering
2	[performance, &, usage, at, instagram, at, ins...	performance & usage at instagram	Engineering
3	[the, simple, example, of, calculating, and, f...	refactoring a javascript video store	Engineering
4	[billing, applications, have, transactions, th...	netflix billing migration to aws - part iii	Engineering

In [13]:
```
#only alphanumerical values
df['body'] = df['body'].apply(lambda x: [item for item in x if item.isalpha()])
```
In [14]:
```
#lemmatazing words
df['body'] = df['body'].apply(lambda x : [WordNetLemmatizer().lemmatize(y) for y in x])
```
In [25]:
```
#removing stopwords
stop = stopwords.words('english')
df['body'] =df['body'].apply(lambda x: [item for item in x if item not in stop])
df.head()
```

I divide those articles into 3 groups: training data, validating data, testing data.

Divide dataset to train,validate, test

In [168]:
```
train, validate, test = \
              np.split(df.sample(frac=1, random_state=42),
                       [int(.6*len(df)), int(.8*len(df))])
```

In [189]:
```
train.shape
```

Out[189]: (1488, 3)

In [172]:
```
validate.shape
```

Out[172]: (496, 3)

In [173]:
```
test.shape
```

Out[173]: (497, 3)

In [174]:
```
df = pd.concat([train, validate])
```

Feature Extraction:

We will implement different ideas in order to obtain relevant features from our dataset.

1. TF-IDF Vectors Word level - Now in order to assign weightage to the above feature vector, we have used Term Frequency – Inverse Document Frequency logic.
2. I used unigrams and Bigram to build feature set.

Feature Extraction

In [175]:
```
word_vectorizer_uni = TfidfVectorizer(
    sublinear_tf=True,
    strip_accents='unicode',
    analyzer='word',
    token_pattern=r'\w{1,}',
    ngram_range=(1, 2),
    max_features =7000)
```

In [178]:
```
unigramdataGet= word_vectorizer_uni.fit_transform(df['body'].astype('str'))
unigramdataGet = unigramdataGet.toarray()
vocab = word_vectorizer_uni.get_feature_names()
unigramdata=pd.DataFrame(np.round(unigramdataGet, 1), columns=vocab)
unigramdata[unigramdata>0] = 1
```

In [179]:
```
X=unigramdata
y=df['category']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=100)
```

In [180]:
```
selector = SelectKBest(chi2, k=700)
selector.fit(unigramdata, df['category'])

unigramdata = selector.transform(unigramdata)
```

Modelling:

I will use SVM and GuassianNB then I will select the best one based on its accuracy.

```
In [181]: svc=LinearSVC(C=1, max_iter=500)
          svc= svc.fit(X_train , y_train)

          y_pred = svc.predict(X_test)
          dm=svc.score(X_test, y_test)
          print('Accuracy score= {:.2f}'.format(svc.score(X_test, y_test)))
```

```
Accuracy score= 0.87
```

```
In [182]: nab=GaussianNB(var_smoothing=1e-08)
          nab= nab.fit(X_train , y_train)

          y_pred1 = nab.predict(X_test)
          nb=nab.score(X_test, y_test)
          print('Accuracy score= {:.2f}'.format(nab.score(X_test, y_test)))
```

```
Accuracy score= 0.70
```

Compare the accuracy:

Based on the accuracy percentage, SVM_classifier is the fastest and most accurate classifier.

Compare Accuracy

```
In [183]: from sklearn.metrics import accuracy_score, f1_score
```

```
In [184]: def eval_model(y,y_predicted):
              print("F1 score of the model")
              print(f1_score(y,y_pred,average='micro'))
              print("Accuracy of the model")
              print(accuracy_score(y,y_pred))
              print("Accuracy of the model in percentage")
              print(round(accuracy_score(y,y_pred)*100,3),"%")
```

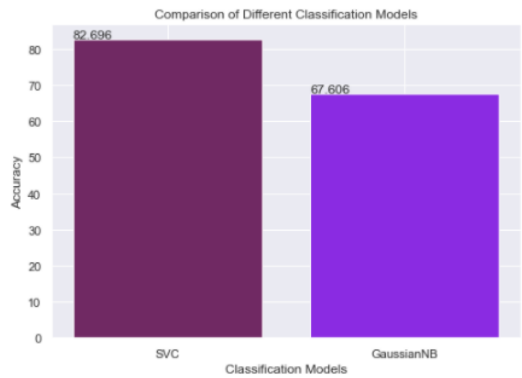
```
In [185]: eval_model(y_test,y_pred)
          a=round(accuracy_score(y_test,y_pred)*100,3)
```

```
F1 score of the model
0.8664987405541562
Accuracy of the model
0.8664987405541562
Accuracy of the model in percentage
86.65 %
```

```
In [186]: eval_model(y_test,y_pred1)
          b=round(accuracy_score(y_test,y_pred1)*100,3)
```

```
F1 score of the model
0.8664987405541562
Accuracy of the model
0.8664987405541562
Accuracy of the model in percentage
86.65 %
```

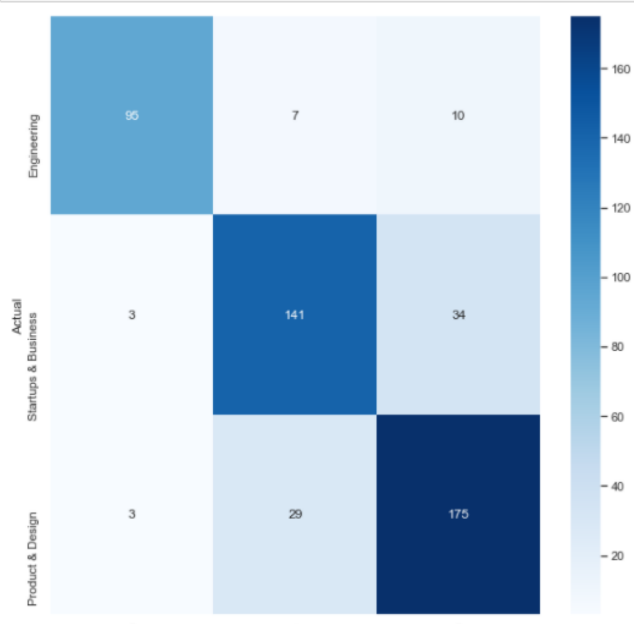
In [17]:
```python
sns.set()
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
Models = ['SVC', 'GaussianNB']
Accuracy=[a,b]
ax.bar(Models,Accuracy,color=['#702963','#8a2be2']);
for p in ax.patches:
     ax.annotate(str(p.get_height()), (p.get_x() * 1.005, p.get_height() * 1.005))
plt.title('Comparison of Different Classification Models');
plt.ylabel('Accuracy');
plt.xlabel('Classification Models');

plt.show();
```



Confusion Matrix:

Cofusion matrix

In [27]:
```python
from sklearn.metrics import confusion_matrix
Cofusion_mat = confusion_matrix(y_test, y_pred1)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(conf_mat, annot=True, fmt='d',
            yticklabels=['Engineering', 'Startups & Business', 'Product & Design'], cmap="Blues")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Notes:

1. In Tf-idf vectorization, when changing the max_features to a highest value in the most common words, it increase the accuracy of the model.
2. I updated the stopwords based on the most common words as I found some words that could be considered as a noise.
3. In Feature Extractions, I tried unigram only, bigram only and both. I found that both unigram and bigram give better accuracy.