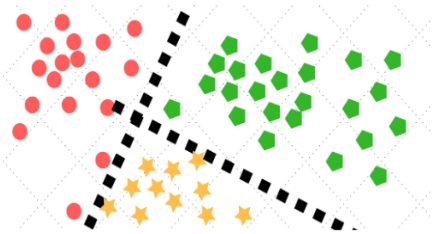
TEXT CLASSIFICATION USING SVM



What is text classification?

Given a truck-load of textual data, it is a huge task to analyse what is inside it owing to its lack of structure. If machines can help us to automate this mechanical process of grouping the text, it is indeed substantial!

Text classification is a way of identifying the category in which the contents of a text belong. Depending on the scenario, this classification can be binary (positive/negative, spam/non-spam) or categorical (politics, technology business, fashion, sports etc).

Classification of textual data is a means to clean it, organize it, make it user-friendly and to make sense out of the unstructured data. It marks its applications in the field of spam detection, sentiment analysis, tagging, language detection and a lot more.

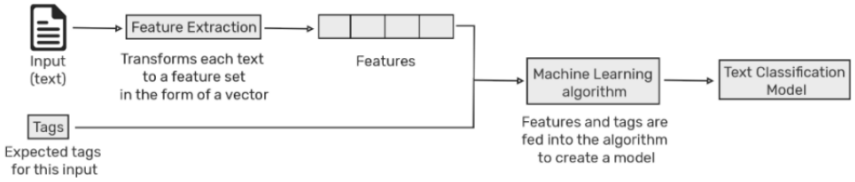
How is it done?

There are numerous machine learning algorithms like Naive Bayes, Random Forest, Support Vector Machines, Neural Models which make use of the training data to arrive at the conclusive category for the new, previously unseen text data.

Steps:

1. Data download.
2. Data pre-processing: xml-parsing, lower-case conversion, punctuation and special character removals, stopwords removal, lemmatization etc, all depending upon the requirement!
3. Word-to-vector Conversion: Machine do not understand text, so we vectorize every word into numerics and feed the vectors thus formed in the machine learning model.
4. Implementing the algorithm.

Here in this article, we will discuss one of these algorithms, Support Vector Machines from scratch.



What is Support Vector Machine (SVM) ?

Support Vector Machines or SVM as we call them, are based on the concept of 'lines' or 'hyperplanes' dividing the space of vectorized data into two or multiple subspaces.

In layman terms, SVM first analyses the given points in the n-dimensional space, then figures out the line/hyperplane that separates the points. Then, points belonging to one category falls onto one side of the line/hyperplane and points of another category falls exactly into its opposite side. SVM model, once trained over the training data, puts the test point in the vector space and analyses its respective position with the line/hyperplane and hence decides its category!

A linear classifier has the form:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

In 3D it is plane, while in nD we call it a hyperplane. But data points are rarely linearly separable or they are so intricately mixed that they are not even separable. In that complex case, say data points are not separable in x-y plane, then we add another dimension z, and plot the 2D space points into 3D use a hyperplane to separate and transform that hyperplane back to the original 2D space, thereby getting a separation in the original space. We call these transformations as 'Kernels'.

As a matter of fact, there can be multiple hyperplanes separating the data-points. Which one should be chosen as the decision boundary? The one that maximizes the smallest distance between the data points of both the classes seems to be a natural choice, providing better margins. Support vectors are the sample data points which lie clode to the decision boundary.

The loss function, which is a function of data point, prediction class and actual label, is actually a measure of the penaly for wrong prediction. SVM uses Hinge loss as the loss function, given as below:

$$l(f(x_i|\theta), y_i) = \max(0, 1 - f(x_i|\theta)y_i)$$

It tells us how better we are doing with respect to one training example. When we sum over all the training examples, what we get is the cost function.

The optimization problem finally looks like:

$$\arg\min_{\mathbf{w}} R(\mathbf{w}) + C\sum_{i=1}^{N} L(y_i, \mathbf{w}^\mathsf{T}\mathbf{x}_i)$$

where R(w) is the regularization function used to tackle overfiiting, C is the scalar constant, and L is the loss fuction.

Below is the python implementation of SVM using scikit learn library

Importing the libraries

In [ ]:
```
import csv
import re
import xml.etree.ElementTree as ET
import os
import pandas as pd
import string
import nltk
from string import punctuation
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
from sklearn import metrics
```

In [5]:
```
data = r'C:\Users\Aditi Sethia\HTML\training\pan12.xml'
```

Parsing the xml file

In [7]:
```
tree = ET.parse(data)
myroot = tree.getroot()
```

In [10]:
```
def getvalueofnode(node):
    return node.text if node is not None else None
def rem_num(text):
    output = re.sub('[0-9]+', '', text)
    return output
def use(x):
    o = 0
    for i in x:
        if i in lis:
            o = o + 1
            return 1
    if o == 0:
        return 0

def rem_stopword(text):
    tokens = text.split()
    result = [i for i in tokens if not i in stop_words]
    return (' '.join(result))
```

Text-preprocessing

In [14]:
```
def text_preprocessing(df):
    df["text"] = df['text'].apply(lambda x: x.lower())
    df["text"] = df['text'].apply(lambda x: ''.join(c for c in x if c not in punctuation))
    df['text'] = df['text'].apply(rem_num)
text_preprocessing(df = df)
```

In [17]:
```
file1 = open(r"\training\tcorpus.txt","r+")
lis = file1.readlines()
lis = [i[:-1] for i in lis]
```

In [20]:
```
def work(df):
    df['Predator_Present'] = df['Users'].apply(use)
work(df = df)
```

In [22]:
```
from sklearn.utils import resample
```

In [31]:
```
df_majority = df[df['Predator_Present'] == 0]
df_minority = df[df['Predator_Present'] == 1]

# Downsample majority class
df_majority_downsampled = resample(df_majority,
                                   replace=False,    # sample without replacement
                                   n_samples=4000,     # to match minority class
                                   random_state=123) # reproducible results

# Combine minority class with downsampled majority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])
```

In [33]:
```
df_downsampled.tail()
```

Out[33]:

	Users	Conversation_ID	text	Predator_Present
66688	[fac3a2081264f1dbb943eaf7165d8fc3]	cfac30bed30bc0ed991787bdc42486b4	sorry about that big lighting strike knocked ...	1
66821	[c62283536cf6261e5ffbcb323c8a2571, adb9f962493...	1c7e6180909437ab3ab8191443e09e2e	just wanted to let you know im canceling my i...	1
66880	[3e97c68b68f9aa0fb7d705a65c6a8443]	52a4e73525be80549ed86d6bb1458804	hello how was your sking	1
66906	[e4c7c376bbd07aeb4a59684a2b94a664]	a0b239c1a240bfe148928ddda8485f83	sorry i had to go so soon ttyl	1
66926	[74bfc043bd5ce9c17b37ffae6e0ba2fa, 8cd850ea421...	4ed6b02ae537fdfd6078597b706292a8	hay remember me oh yeah hi how could i forget...	1

Converting the data-set into vectors (Tf-idf)

In [43]:
```
vectorizer = TfidfVectorizer(stop_words = 'english')
X_vect = vectorizer.fit_transform(corpus_DS).toarray()
X_vect.shape
```

Out[43]:
```
(6048, 33425)
```

Splitting the data into training and testing samples

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(X_vect, y, test_size = 0.2, random_state = 0)
```

Fitting the model

In [ ]:
```
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

Results

In [47]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
print(metrics.f1_score(y_test, y_pred, average='macro'))
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       800
           1       0.96      0.79      0.87       410

    accuracy                           0.92      1210
   macro avg       0.93      0.89      0.90      1210
weighted avg       0.92      0.92      0.92      1210

0.9029627633638719
```

References:

1. https://monkeylearn.com/text-classification/
2. https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34