Context:

Companies that provide services or products to their customers would like to know which **aspects** of their products preoccupy most of the customers. This knowledge usually helps to improve product portfolio. Are people talking the most about the **quality** of the food, the **price** of an item or the **battery life** of a computer? All those terms are what we call *aspect term*.

You are asked to build an *Aspect Term Extractor* (ATE). An ATE is a model that extracts aspect terms from a review, i.e. for each word of a review, your model should predict if the word is an aspect term or not of the reviewed product.

The input data has the following format: *review → list of aspect terms*, e.g. "The battery life is really good and its size is reasonable" → "battery life", "size". We recommend to change it and use the BIO format instead. Example:

The	battery	life	is	really	good	and	its	size	is	reasonable
O	B	I	O	O	O	O	O	B	O	O

With this format, we can see that the role of an ATE is to assign to each word one of the three possible classes:

- O = not an aspect (Outside)
- B = first word of an aspect (Beginning)
- I = second, third, ... word of an aspect (Inside)

For this assignment, there are two data sets:

- *'Laptops_Train_v2.xml'* - to be used to train your model
- *'Laptops_Test_Gold.xml'* - to be used to evaluate your model

Both data sets are available for download here: https://drive.google.com/drive/folders/1XYzYT-ZzBT58YbuZDdYl4JUMpDdG7cIc?usp=sharing

Important: Two aspects can exist in the same sentence and that one aspect term can be composed of more than one token (battery life). In such a case, it should be clear that it forms one aspect and not two aspects.

Objectives:
Please prepare the presentation with your results. In it you'll have to justify and explain:

- The feature extraction process: what is the intuition behind your method, which features are the most important, etc.?
- The algorithm you selected (we ask you to explore **at least two** different algorithms)
- The metric that you used for quantifying the performance of your model
- The performance evaluation of the algorithms you selected

Important: Your model must be **interpretable**, i.e. you are **not allowed** to use any kind of deep learning method (LSTM, CNN, …).

Some Background:

The problem:

To identify aspect of product from a review, i.e. for each word of a review, identify which terms of their products preoccupy most of the customers.

There are two approaches to prepare data:

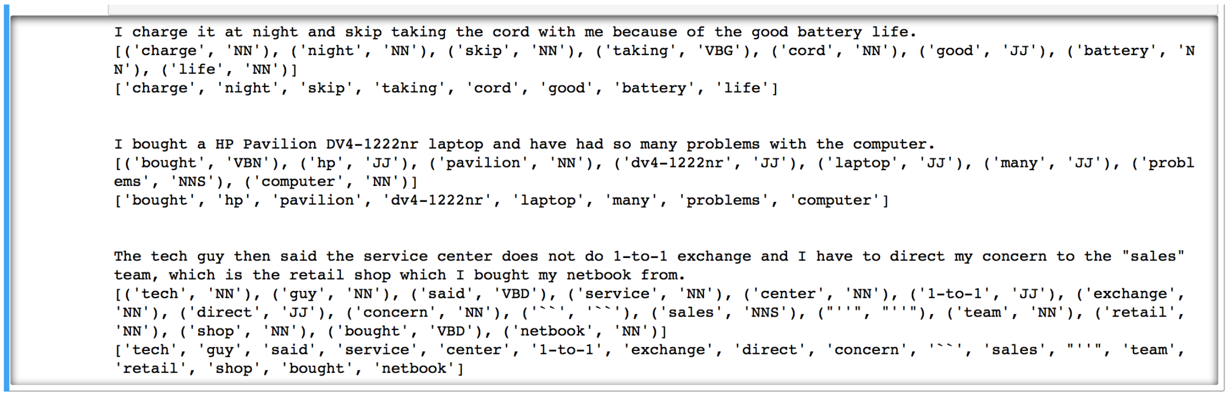
1. **Unsupervised Aspect Term Extraction** for instance: https://arxiv.org/abs/1709.05094
2. Hand labeled terms or use the insight to write code that bootstrapped the aspect terms

   PS: We are provided labeled data so we just use them in our feature set.

The data is XML and looks as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sentences>
    <sentence id="892:1">
        <text>Boot time is super fast, around anywhere from 35 seconds to 1 minute.</text>
        <aspectTerms>
            <aspectTerm term="Boot time" polarity="positive"  from="0" to="9"/>
        </aspectTerms>
    </sentence>
    <sentence id="1144:1">
        <text>tech support would not fix the problem unless I bought your plan for $150 plus.</
            text>
        <aspectTerms>
            <aspectTerm term="tech support" polarity="negative"  from="0" to="12"/>
        </aspectTerms>
    </sentence>
    <sentence id="805:2">
        <text>but in resume this computer rocks!</text>
    </sentence>
    <sentence id="359:1">
        <text>Set up was easy.</text>
        <aspectTerms>
            <aspectTerm term="Set up" polarity="positive"  from="0" to="6"/>
        </aspectTerms>
    </sentence>
```

The generation of data for features and some exploratory data analysis in attached as pdf:



```
I charge it at night and skip taking the cord with me because of the good battery life.
[('charge', 'NN'), ('night', 'NN'), ('skip', 'NN'), ('taking', 'VBG'), ('cord', 'NN'), ('good', 'JJ'), ('battery', 'N
N'), ('life', 'NN')]
['charge', 'night', 'skip', 'taking', 'cord', 'good', 'battery', 'life']


I bought a HP Pavilion DV4-1222nr laptop and have had so many problems with the computer.
[('bought', 'VBN'), ('hp', 'JJ'), ('pavilion', 'NN'), ('dv4-1222nr', 'JJ'), ('laptop', 'JJ'), ('many', 'JJ'), ('probl
ems', 'NNS'), ('computer', 'NN')]
['bought', 'hp', 'pavilion', 'dv4-1222nr', 'laptop', 'many', 'problems', 'computer']


The tech guy then said the service center does not do 1-to-1 exchange and I have to direct my concern to the "sales"
team, which is the retail shop which I bought my netbook from.
[('tech', 'NN'), ('guy', 'NN'), ('said', 'VBD'), ('service', 'NN'), ('center', 'NN'), ('1-to-1', 'JJ'), ('exchange',
'NN'), ('direct', 'JJ'), ('concern', 'NN'), ('``', '``'), ('sales', 'NNS'), ("''", "''"), ('team', 'NN'), ('retail',
'NN'), ('shop', 'NN'), ('bought', 'VBD'), ('netbook', 'NN')]
['tech', 'guy', 'said', 'service', 'center', '1-to-1', 'exchange', 'direct', 'concern', '``', 'sales', "''", 'team',
'retail', 'shop', 'bought', 'netbook']
```

Some terminologies:

The problem we are trying to solve can be broadly classified as Named Entity recognition

Definition:

Named Entities provides critical information for many NLP applications. Named Entity recognition and classification (NERC) in text is recognized as one of the important sub-tasks of Information Extraction (IE).

Theory:

The overwhelming amount of unstructured text data available today from traditional media sources as well as newer ones, like social media, provides a rich source of information if the data can be structured. Named Entity Extraction forms a core subtask to build knowledge from semi-structured and unstructured text sources. Some of the first researchers working to extract information from unstructured texts recognized the importance of “units of information” like names (such as person, organization, and location names) and numeric expressions (such as time, date, money, and percent expressions). They coined the term “Named Entity” in 1996 to represent these.

Considering recent increases in computing power and decreases in the costs of data storage, data scientists and developers can build large knowledge bases that contain millions of entities and hundreds of millions of facts about them. These knowledge bases are key contributors to intelligent computer behavior. Not surprisingly, Named Entity Extraction operates at the core of several popular technologies such as smart assistants (Siri, Google Now), machine reading, and deep interpretation of natural language.

Available TOOLS:

There are several ways to accomplish given task and following are three popular, open source NERC tools. The tools are NLTK, Stanford NER, and Polyglot. A brief description of each follows.

- NLTK has a chunk package that uses NLTK's recommended named entity chunker to chunk the given list of tagged tokens. A string is tokenized and tagged with parts of speech (POS) tags. The NLTK chunker then identifies non-overlapping groups and assigns them to an entity class. Or we can feed it to classifier and let Classification Algorithm handle it for us.
- Stanford's Named Entity Recognizer, often called Stanford NER, is a Java implementation of linear chain Conditional Random Field (CRF) sequence models functioning as a Named Entity Recognizer. Named Entity Recognition (NER) labels sequences of words in a text that are the names of things, such as person and company names, or gene and protein names.
- Polyglot is a natural language pipeline that supports massive multilingual (i.e. language) applications. It supports tokenization in 165 languages, language detection in 196 languages, named entity recognition in 40 languages, part of speech tagging in 16 languages, sentiment analysis in 136 languages, word embeddings in 137 languages, morphological analysis in 135 languages, and transliteration in 69 languages. It is a powerhouse tool for natural language processing.

The toolset for POC used is NLTK and SCIKIT

NLTK Basics:

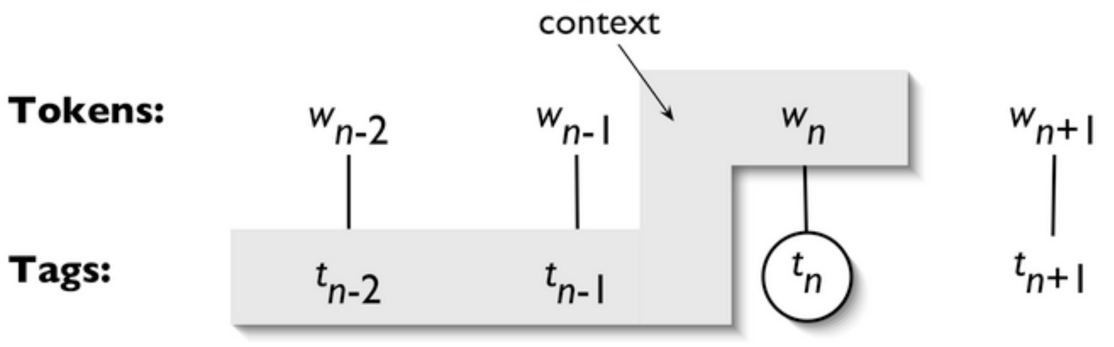
Tagger:
A part-of-speech tagger, or **POS-tagger**, processes a sequence of words, and attaches a part of speech tag to each word (don't forget to import nltk):

```
>>> text = word_tokenize("And now for something completely different
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

General N-Gram Tagging

When we perform a language processing task based on unigrams, we are using one item of context. In the case of tagging, we only consider the current token, in isolation from any larger context. Given such a model, the best we can do is tag each word with its *a priori* most likely tag. This means we would tag a word such as *wind* with the same tag, regardless of whether it appears in the context *the wind* or *to wind*.

An **n-gram tagger** is a generalization of a unigram tagger whose context is the current word together with the part-of-speech tags of the *n*-1 preceding tokens, as shown in figure The tag to be chosen, t_n, is circled, and the context is shaded in grey. In the example of an n-gram tagger shown in figure , we have *n*=3; that is, we consider the tags of the two preceding words in addition to the current word. An n-gram tagger picks the tag that is most likely in the given context.



Note

A 1-gram tagger is another term for a unigram tagger: i.e., the context used to tag a token is just the text of the token itself. 2-gram taggers are also called *bigram taggers*, and 3-gram taggers are called *trigram taggers*.

Classification Algorithms:

Task: **Document/Text classification**

Document/Text classification is one of the important and typical task in *supervised* machine learning (ML). Assigning categories to documents, which can be a web page, library book, media articles, gallery etc. has many applications like e.g. spam filtering, email routing, sentiment analysis etc. There are various algorithms which can be used for text classification.

Example: **Naive Bayes, Decision Tree, Maxent , Support Vector Machines (SVM) , Grid Search etc**

For POC there following were explored: **Naive Bayes**, **Decision Tree, Maxent.**

What is the Naive Bayes Classifier?

The Naive Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem with strong and naïve independence assumptions. It is one of the most basic text classification techniques with various applications in email spam detection, personal email sorting, document categorization, sexually explicit content detection, language detection and sentiment detection. Despite the naïve design and oversimplified assumptions that this technique uses, Naive Bayes performs well in many complex real-world problems.

Even though it is often outperformed by other techniques such as boosted trees, random forests, Max Entropy, Support Vector Machines etc, Naive Bayes classifier is very efficient since it is less computationally intensive (in both CPU and memory) and it requires a small amount of training data. Moreover, the training time with Naive Bayes is significantly smaller as opposed to alternative methods.

Naive Bayes classifier is superior in terms of CPU and memory consumption as shown by Huang, J. (2003), and in several cases its performance is very close to more complicated and slower techniques.

When to use the Naive Bayes Text Classifier?

You can use Naive Bayes when you have limited resources in terms of CPU and Memory. Moreover when the training time is a crucial factor, Naive Bayes comes handy since it can be trained very quickly. Indeed Naive Bayes is usually outperformed by other classifiers, but not always! Make sure you test it before you exclude it from your research. Keep in mind that the Naive Bayes classifier is used as a baseline in many researches.

What is the Max Entropy Classifier?

The Max Entropy classifier is a probabilistic classifier which belongs to the class of exponential models. Unlike the Naive Bayes classifier, the Max Entropy does not assume that the features are conditionally independent of each other. The MaxEnt is based on the Principle of Maximum Entropy and from all the models that fit our training data, selects the one which has the largest entropy. The Max Entropy classifier can be used to solve a large variety of text classification problems such as language detection, topic classification, sentiment analysis and more.

When to use the MaxEnt Text Classifier?

Due to the minimum assumptions that the Maximum Entropy classifier makes, we regularly use it when we don’t know anything about the prior distributions and when it is unsafe to make any such assumptions. Moreover Maximum Entropy classifier is used when we can’t assume the conditional independence of the features. This is particularly true in Text Classification problems where our features are usually words which obviously are not independent. The Max Entropy requires more time to train comparing to Naive Bayes, primarily due to the optimization problem that needs to be solved in order to estimate the parameters of the model. Nevertheless, after computing these parameters, the method provides robust results and it is competitive in terms of CPU and memory consumption.

The code for above three is in GIT repo and I personally think it could be done in much better way but due to limited time constraints the POC is minimalistic

The metrics can be calculated by Precision and Recall Ideally:

There are four ways of being right or wrong:

- TN / True Negative: case was negative and predicted negative
- TP / True Positive: case was positive and predicted positive
- FN / False Negative: case was positive but predicted negative
- FP / False Positive: case was negative but predicted positive

For POC I have just calculated the accuracy of classifier:

1. accuracy NaiveBayesClassifier= 0.855523535062
2. accuracy DecisionTreeClassifier= 0.882612872238
3. accuracy MaxentClassifier= 0.871277617675

Things that can be done better:

1. Using a better tagging and chunking method.
2. Evaluating several other algorithms
3. Cleaning data in a better way.
4. Exploring unsupervised aspect term methodologies to improve feature set.
5. Improvising in hyper parameters of algorithms used
6. Evaluation of metrics using Precision and Recall