1 Monolingual embedding:

1.1 Word2Vec :

In this section there was no big difficulty filling the needed snippets of the code. The only slight detail that is worth mentioning here is that we discard from the sentences the words that do not appear in the lookup dictionary of the Word2Vec embedding (which is similar to assigning them to a null vector).

```
Loaded 50000 pretrained word vectors
cat tree 0.2644975466165475
cat dog 0.7078641298542562
cat pet 0.6753313359976381
Paris France 0.6892958925806542
Paris Germany 0.4051242286737548
Paris baguette 0.29399958277802224
Paris donut -0.006588507552348005
['cats', 'kitty', 'kitten', 'feline', 'dog']
['dogs', 'puppy', 'pup', 'canine', 'pet']
['dog', 'cats', 'puppies', 'Dogs', 'pets']
['France', 'Parisian', 'Marseille', 'Brussels', 'Strasbourg']
['Austria', 'Europe', 'Berlin', 'Hamburg', 'Bavaria']
```

Figure 1: Results

We get to see experimentally that the pair of words that do not have something in common get to have a cosine similarity (between their embeddings) that is pretty low : For instance Paris / Donut and Paris/France.

1.2 BoV :

Now we move on the the sentence level, by comparing the similarity metrics using the mean or the idf weighted mean of the word vectors composing the sentence. (Figure 2)

I would argue that the results we got are not very good in either cases (using mean of weighted means of word vectors). Indeed, if I would pick the most similar sentence to "1 smiling african american boy" I would pick neither of the resulted sentences. As for the "christmas presents" sentence, I would say that the most similar sentence if the one ranked third in our case (without counting the self similarity).

To sum up, I would conclude by saying that this approach of sentence comparison using independent word vectors and aggregating them is not an appropriate method since we discard the relationship between the words and thus lose a lot of information in the process.

2 Multilingual word embeddings :

The goal is to solve this optimisation problem :

$$W^* = \text{argmin}_{W \in \mathcal{O}_d(\mathcal{R})} \|WX - Y\|_F$$

Which is the same as solving this problem :

$$W^* = \text{argmin}_{W \in \mathcal{O}_d(\mathcal{R})} \|WX - Y\|_F^2$$

$$\begin{aligned}
\|WX - Y\|_F^2 &= \text{Tr}\left((WX - Y)(WX - Y)^T\right) \\
&= \text{Tr}\left(WXX^TW^T\right) + \text{Tr}\left(YY^T\right) - 2 < WX, Y >_F \\
&= \text{Tr}\left(W^TWXX^T\right) + \text{Tr}\left(YY^T\right) - 2 < WX, Y >_F \\
&= \|X\|_F^2 + \|Y\|_F^2 - 2 < WX, Y >_F
\end{aligned}$$

Using the facts that $W^TW = I$ and the permutation property of the trace operator.

So

```
Loaded 100000 pretrained word vectors

        --------------------------

        Average of word embeddings

        --------------------------
1 man singing and 1 man playing a saxophone in a concert .
10 people venture out to go crosscountry skiing .
0.7065220648251476

        ---------------------------------------
1 smiling african american boy .
1) 1 smiling african american boy .
2) 2 woman dancing while pointing .
3) 5 women and 1 man are smiling for the camera .
4) a small boy following 4 geese .
5) 2 female babies eating chips .
6) a young boy and 2 girls open christmas presents .

        ---------------------------------------

        idf weighted average of word embeddings

        ---------------------------------------
1 man singing and 1 man playing a saxophone in a concert .
10 people venture out to go crosscountry skiing .
0.45131736734105854

        ---------------------------------------
a young boy and 2 girls open christmas presents .
1) a young boy and 2 girls open christmas presents .
2) two solders are carrying multiple christmas presents .
3) it 's christmas , a family posing by their tree and presents .
4) one boy in a red shirt with two girls all sitting on the floor opening up presents , with another person and a christma
e in the background .
5) a group of people are holding presents in a room with a christmas tree .
6) three small children open christmas presents underneath a christmas tree .
```

Figure 2: Results

$$W^* = \text{argmin}_{W \in \mathcal{O}_d(\mathcal{R})} \|WX - Y\|_F^2 = \text{argmax}_{W \in \mathcal{O}_d(\mathcal{R})} \ < WX, Y >_F \ = \text{argmax}_{W \in \mathcal{O}_d(\mathcal{R})} \ < WX, Y >_F$$

Using the SVD decomposition : $YX^T = U\Sigma V^T$ we get that :

$$\begin{aligned} < WX, Y >_F &= \text{Tr}\left(YX^TW^T\right) \\ &= \text{Tr}\left(U\Sigma V^TW^T\right) \\ &=< U\sqrt{\Sigma}, \sqrt{\Sigma}V^TW^T >_F \end{aligned}$$

Using the Cauchy Schwarz inequality and the fact that U, V and W are othogonal matrices (which norm is the Identity matrix) :

$$< WX, Y >_F \leq ||U\sqrt{\Sigma}||_F||\sqrt{\Sigma}V^TW^T||_F \leq ||\sqrt{\Sigma}||_F||\sqrt{\Sigma}||_F = \text{Tr}(\Sigma)$$

We have equality in the previous inequality (and thus we reach the maximum bound) if :

$$Tr(U\Sigma V^TW^T) = Tr(\Sigma)$$

$$Tr(\Sigma V^TW^TU) = Tr(\Sigma)$$

$$\Sigma V^TW^TU = \Sigma$$

$$\Sigma V^TW^TU = \Sigma$$

$$W = UV^T$$

Those are satisfying results.

```
There are 14361 common words that are used as anchors in the two language spaces
----------
fr: "chat"
en: "cat"
en: "kitten"
en: "kitty"
----------
fr: "chien"
en: "dog"
en: "cat"
en: "pet"
----------
fr: "voiture"
en: "car"
en: "vehicle"
en: "automobile"
----------
fr: "zut"
en: "oops"
en: "Ah"
en: "ah"
```

Figure 3: Results

3 Sentence classification with BoW :

All the results I got are in the table below.

For the improvement part of the question, first I thought of looking at the preprocessing of the sentences before using their respective embeddings :

- Setting the sentences to lowercase : I observed that words (Example : Cat and cat) were assigned 2 different embedding. So in order to get a coherent encoding I put all the sentences to lowercase.
- Removing special characters : I saw that ponctuations and some other weird combination of characters had embeddings assigned to them too. So removing them was a good thing to start with.
- Removing stopwords : words like ”and”, ”the”, ”are” were very common among the sentences and by removing them I hoped that there will only remain keywords and important words for the classification.

But the problems I faced were that for some sentences there was no other word remaining : ”this is no “ waterboy ! ””. Indeed in this sentence, only the term **waterboy** remains after the preprocessing. Thus I simplified the preprocessing by leaving the special punctuation (”?”, ”!”,...).

Secondly, I thought of improving the classifier by using aggregation. Indeed I performed hard voting between a Linear SVM, GradientBoosted tree and a logistic regression.

I trained the ensemble classifier and observed a performance similar to the one where I am using the untreated sentences with a logistic regression classifier only. But when I tried my ensemble classifier on the original sentences, I saw an increase of the dev dataset accuracy (without any fine tuning on the hyper parameters of the classifiers in the committee so there is room for increase in performance).

Data set	LR unp	LR unp wgt	Vote unp
Tr	45.65	41.35	**54.43**
Dev	40.96	37.14	**41.14**

Accuracies - unp : Unprocessed - wgt : weighted - LR : Logistic Regression

4 Deep Learning models for classification :

4.1 Base provided LSTM :

I first use the provided model : Randomly initialized ”look-up table” as an Embedding layer that feeds its embedded 32 dimensional representation of the words to a RNN using a single 64 hidden layers LSTM. After that we feed the encoded sequences to a fully connected + a sigmoid function (which is equivalent to performing a Logistic Regression) over the 5 classes to predict. The results are displayer in Figure 4.

The loss optimized in the process is :

$$CE = -\sum_{x \in D} p(x) \log(q(x))$$

Which tries to approach the probability p (over the real classes, whichi is a one hot vector) by a probability q coming from the softmax.

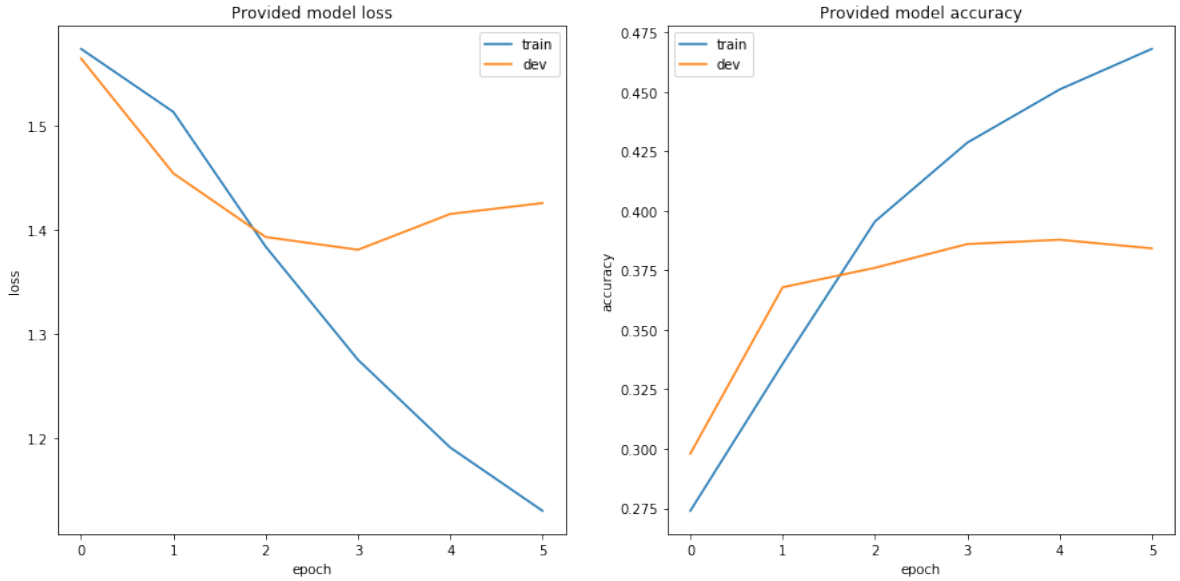

Figure 4: Provided model performance

We see that this simple model overfits the training data after the third epoch. A solution would be to perform early stopping or to add dropout layers.

To producte the test preditions, I concatenate the training data and dev data and early stop the RNN at the 3rd epoch.

4.2 Improved model :

I thought of using BERT model at first, we got discourages when I saw the amount of modifications I had to do to the code in order for it to work... Then I thought of ELMO embeddings (since out task is sentiment analysis and some commentaries are written in a complex way and each word means something in the context -sometimes even in a sarcastic way, which makes the task hard if we embed each word to a unique embedding regardless of the context-) but I did not manage to make it work as well (some problems with tensorflow hub and my laptop...) I opted finally for a 2 layers Bidirectional LSTM with GLoVE pretrained word embeddings that I finetune on this dataset early stopped at the 8th epoch and using a smaller batch size of 32.

The previous proposed single LSTM architecture performed 38.87% accuracy on the dev set whereas mine did 43.60% on the same set.