Details of the model

1. I chose a multi-layer perceptron neural network for this task as it allows me to model complex data and predict probabilities of multiple classes. The architecture consists of a dense layer with 512 neurons, a relu activation function, dropout layer of 50%, another dense layer with all categories and a final softmax layer to indicate probabilities of a given class. The summary of a model is shown below:

_____

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	2048512
activation_1 (Activation)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1419)	727947
activation_2 (Activation)	(None, 1419)	0

Total params: 2,776,459
Trainable params: 2,776,459
Non-trainable params: 0

_____

The model is implemented in Keras as I'm most comfortable with this deep learning framework. Apart from this model I also used Random Forest, XGBoost or LSTM with embeddings but the performance of the current model was the highest. Also, the speed of training and prediction was lower compared to LSTM.

2. I performed the following pre-processing steps
    - I removed the stop words based on the English dictionary.
    - Used the counter to get the counts of individual words.
    - Transformed the target column with labels into categorical columns.
    - Split the dataset into train and test sets with 20% for the test set.
    - Fitted the Keras tokenizer on the training set. Ideally the num_words parameter would equal to the length of the calculated counter but due my laptop not having enough RAM I had to limit it. One the tokenizer was fitted it was saved to be later used for predictions.
    - Transformed the training and test feature series to lists.
    - Used the tokenizer to transform the series above to sequences. Later they were converted to feature matrices and term frequency–inverse document frequency values were assigned.
3. I used the accuracy and loss for monitoring performance. I used a validation split of 10% and monitored the performance at each epoch. The performance at each epoch and on the test set is shown below:

```
Epoch 1/3
1366/1366 [==============================] - 128s 94ms/step - loss: 3.4144 - accuracy: 0.4402 - val_loss: 2.3238 - val_accuracy: 0.5324
Epoch 2/3
1366/1366 [==============================] - 131s 96ms/step - loss: 2.2369 - accuracy: 0.5380 - val_loss: 2.1913 - val_accuracy: 0.5448
Epoch 3/3
1366/1366 [==============================] - 130s 95ms/step - loss: 2.0265 - accuracy: 0.5633 - val_loss: 2.1636 - val_accuracy: 0.5522
380/380 [==============================] - 19s 51ms/step - loss: 2.1707 - accuracy: 0.5510
Test loss: 2.1707165241241455
Test accuracy: 0.5509990453720093
```

There wasn't much performance improvement after 3 epochs, so I settled for using 3 epochs. When the predictions were made, I also looked at random samples of predictions to see if their predicted categories are like the sentences in the test set. For example, "trailer homes uk" was predicted to be in category 915. Similar queries were in the training set.

	Text	category_id
155580	mobile homes for sale in watford	915
107339	mobile homes for sale in greece	915
570227	mobile homes for sale in nottingham	915
384482	portable toilet cabin	915
33588	les pierres couchées	915
...	...	...
334217	park homes for sale in cheltenham	915
540854	mobile homes for sale berkshire	915
400339	mobile homes off site for sale uk	915
325626	mobile homes for sale in thanet	915

4. The training code takes 9.5 mins to train on 606824 samples. The prediction code takes around 50 seconds to run on 67424 testing samples. There's a large memory overhead at the training stage where the feature sequences are transformed into matrices using the tokenizer. Reducing the number of num_words in the tokenizer or training it on a machine with more RAM resolves this issue.
5. Weaknesses:
    a. The model's accuracy is around 55%. However when inspecting the predictions at random most test sentences get allocated to relevant groups. There are a few cases of incorrect labels in the training data. For example, "nike cortez", "swiss watches" and "blowfish" are in the same category in the training set.
    b. There's a large memory overhead at the training stage described above.
    c. The current architecture may be too simplistic to model the data well.
    d. There may not be enough data to predict all the cases accurately.
    e. The model may not be as fast as some other non-deep learning models like decision trees
    f. The number of samples for each category is not the same therefore the model may struggle to predict categories with low sample size

Future improvements:

- Trying out different network architectures
- Using pre-pre-trained word embeddings
- Using transfer learning and pre trained models for the task
- Randomly oversampling the test data for categories with low sample sizes
- Training the model with more data