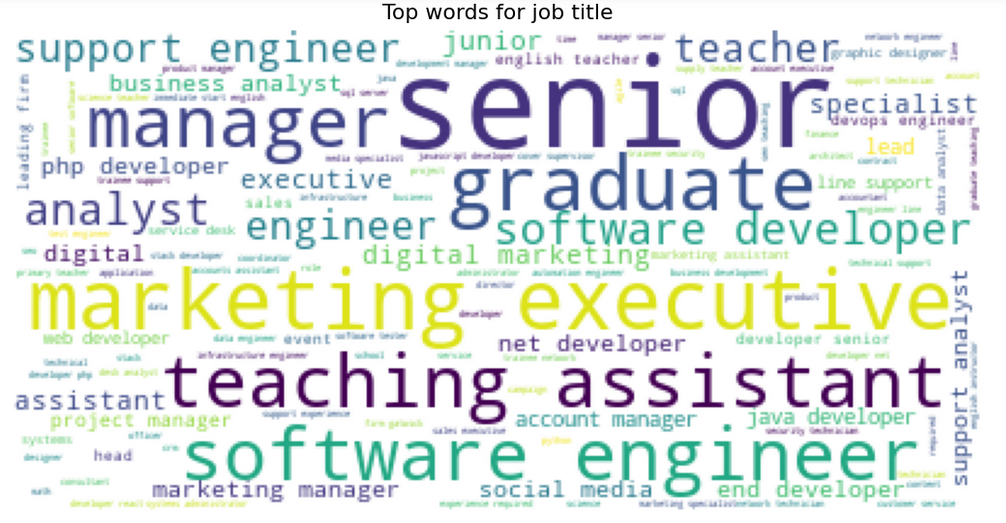
**Job titles and industries**

**Classification Task**

1. **Which techniques you have used while cleaning the data if you have cleaned it?**
   1. as the data is tabular shape there is no null values
   2. removing duplicate row make the model give 90% accuracy and that was lower than the current one 95% so I kept it, typically a good idea to remove duplicate data points so the model can better generalize to the full dataset, But to address what you might be seeing in competitions, it might be that your model is focusing on the duplicates as "more important" and it happens to be that the competition test data has more of those data points that were repeated.
   3. There is a 3 stage to clean each job title
      1. Removing STOPWORDS and words lower than 3 characters
      2. Some redundant words like (country, London …etc.)
      3. Found some numbers express the salary removed it to

There is no need for lemmatization and stemming because we don’t not care about the meaning of sentences but the similarity at this point

Here is the word cloud after cleaning



1. **Train & Test dataset**

Splitting our data 3 categories Train, Validation, Testing and apply the following techniques on them fitting on train, transform on validation and test

70% train

15% validation

15% test

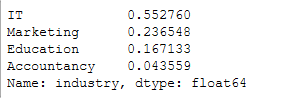
(Note after reaching good accuracy removed the validation set and the new shape of dataset for 80% train and 20% test).

1. **Label encoder**

Apply on target to change it form text shape no class numbers.

1. **How do you deal with (Imbalance learning)?**

Using class weight as we can see

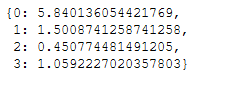


Almost 55% of data only come from IT

So we balance it and assign for every class a weight come from

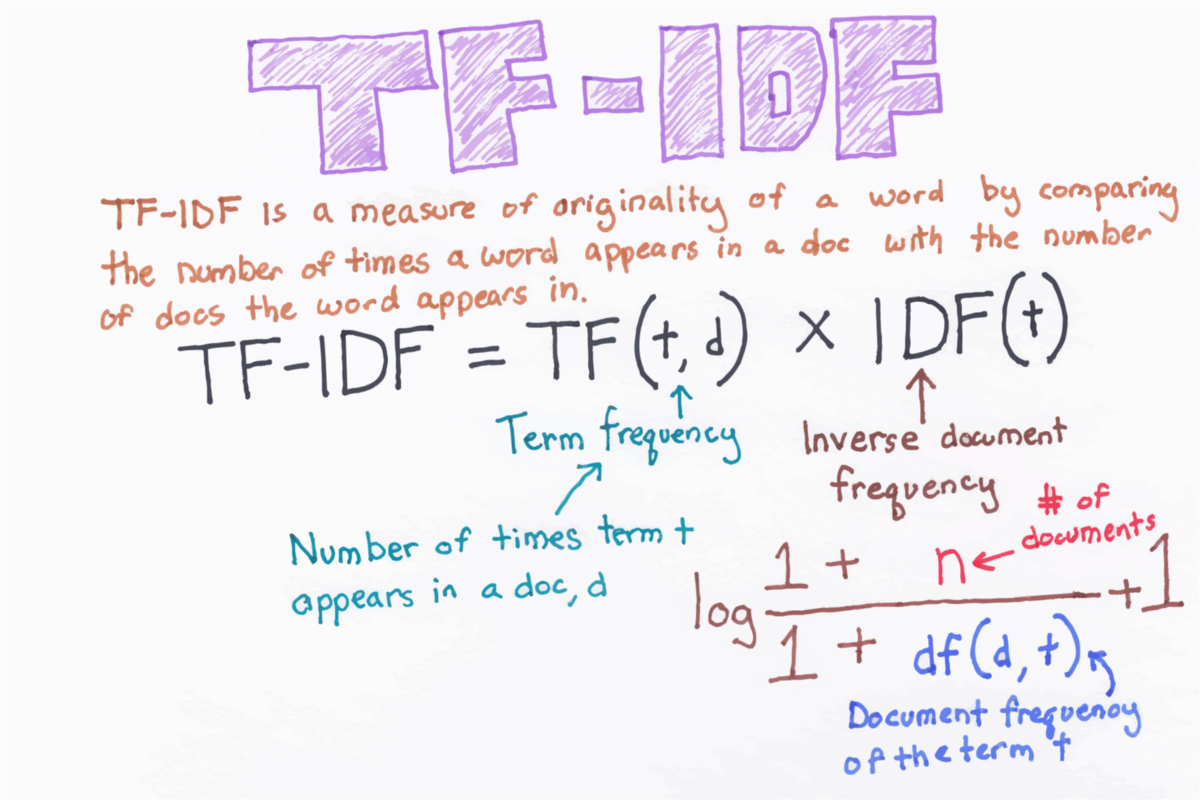
n\_samples / (n\_classes \* np.bincount(y))

And we got this

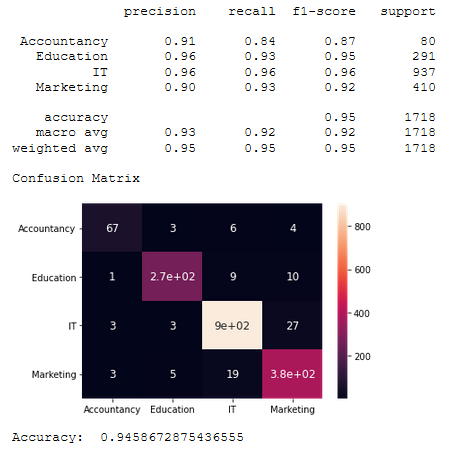


1. **Preprocessing phase?**
   1. As we are going to use some NLP techniques we need to change this text format to some sort of numbers format so from this stage applied 2 classifier svm, lstm dl some common preprocessing techniques common between them.
2. **Models**

* **SMV support vector machine**
  + **Preprocessing** 
    - CountVectorizer to form a frequency dictionary for every word in our training set.
    - tfidf\_transformer feeding the trainingset vectorized one to the tf-idf transformer



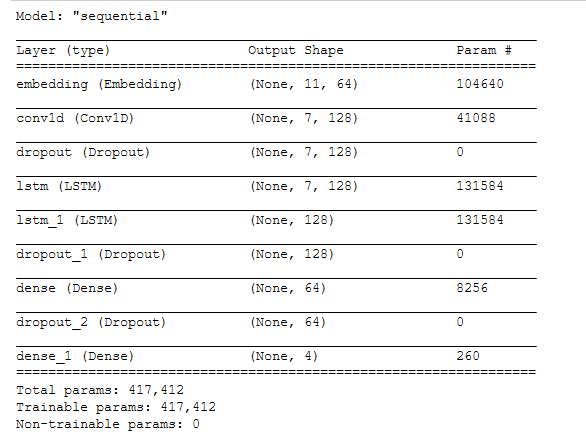
* + - Where ft is the raw count of a term t in a document, the number of times that term t occurs in document d.
    - And idf inverse document frequency is a measure of how much information the word provides, if it's common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient).
  + Already Combine the 2 preprocessing function in one pipeline to be used in testing on flask app.
  + Now we are ready to run the svm with shape (6868, 1634) sample x exist of word weight
  + Why have you chosen this classifier?
    - Svm can be extended with kernel trick to work in high dim so it can easy find nonlinear relation, can handle the C parms to overcome the over fit.
  + How do you evaluate your model?



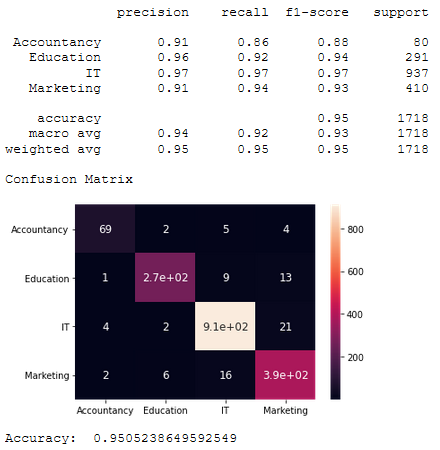
* + Seem the model still cant classify Accountancy well, 80 sample for is small for good high result like rest.
  + Save the model for use for flask api
* **LSTM Long short term memory deep learning** 
  + Almost the label encoder + will encode the label in one hot encode using utils\_category
  + Preprocessing here will be
    - Tokenize every sample to list of words now we have a list of words index for every sample row
    - Pad every list with post of 0 and the max len of each list will be the max list len of samples
  + Model
    - First layer of our lstm will be the embedding layer where
    - The Embedding layer is defined as the first hidden layer of a network, specify 3 arguments:
      * input\_dim: This is the size of the vocabulary in the text data, and get it from the tokenizer vocabulary len.
      * output\_dim: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word, assumed it equal 32.
      * input\_length: This is the length of input sequences, as you would define for any input layer of a Keras model, we already defined by seq padding and it equal 11

e = Embedding(# train samples, 32, 11)

* now we use some conv1 layer with 128 filer and activation relu
* 2 LSTM layer with ability to return sequence but why LSTM?
  + LSTM networks are an extension of recurrent neural networks (RNNs) mainly introduced to handle situations where RNNs fail. Talking about RNN, it is a network that works on the present input by taking into consideration the previous output (feedback) and storing in its memory for a short period of time (short-term memory).
* Use 3 dropout layers with different percent to overcome overfitting
* Then a Dense layer with soft max activation function.



* Using adam optimizer
* Early stopping to overcome to overfitting
* Checkpoint to save best val accuracy
* Fit the model with 100 epoch and batch size 16
* And the accuracy for this model on test set 95%

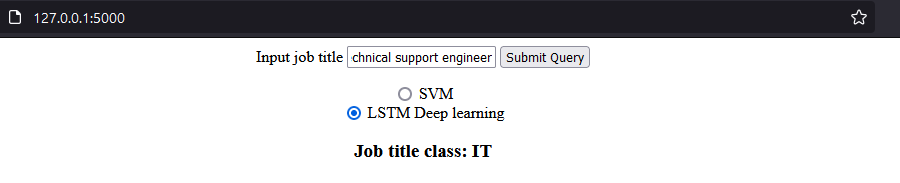


* We can see a little improvement from smv in the accountancy class.

**All Models and pre processing + label encoder saved using joblib, keras model save, npy for later use in flask api**

**Flask**

Was a simple index template where you can set the job title and get the class for it



The models need to be loaded before the flask server run and then pass it to the flask app in config dictionary to work with it.

**SO YOU NEED TO RUN IT FROM MAIN NOT FORM APP**

1. **How can you extend the model to have better performance?**

* If we have a more samples for accountancy we can improve our models
* Taught about up sampling using smote, but I assumed it will add more noise and will lead to more miss classification
* May use more increase our reversed word to remove non useful words on cleaning phase stage 2