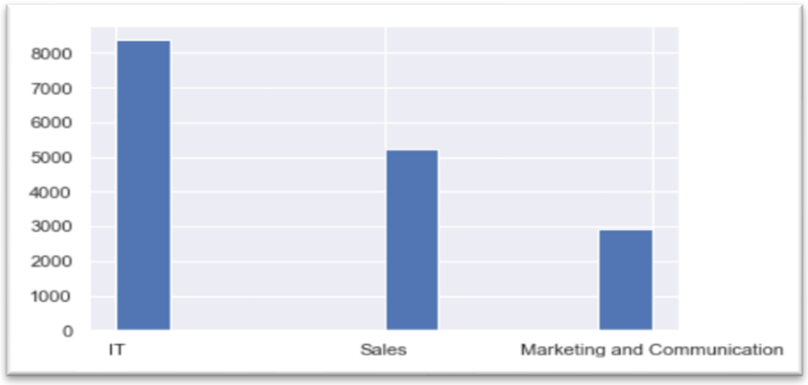
Job Title Classification

Balance the dataset

The data looks imbalanced as the three categories from the Category column of the data are not equally distributed like:
IT - 8377
Sales - 5227
Marketing and Communication – 2919



There are several common ways to deal with imbalanced datasets. However, we can use a method called **stratification** or **stratified sampling (**stratified train-test split**)**. This technique removes the proportions variance, meaning it maintains the same class distribution in each subset of the complete training dataset while training, which gives more stability during the training.

This can be achieved by setting the "stratify" argument on the call to train_test_split() and setting it to the "y" variable containing the target variable from the dataset.

Note - We can also use stratified k-fold cross-validation which is effectively same but it is computationally expensive.

Feature Extraction

After text pre-processing, we need to perform feature engineering and data preparation when using machine learning model to train the data. We can use **Word2Vec** Feature Extraction method to give a dense vector for each word.

Word2Vec capture the semantic meaning of words. It is a technique to find continuous embeddings for words. It learns from reading massive amounts of text and memorizing which words tend to appear in similar contexts. Hence, it will **encounter words that we have not seen in our training set before**.

We can train **word2vec** using **gensim.models** python package.

Also I have referred to one paper [1] where they have concluded that word2vec is the better method for feature extraction.

Disadvantage –

- As the word and vector is one to one relationship, so polysemous problem cannot be solved.
- Although we are converting the text data into English language while cleaning the data, we have one more disadvantage - scaling to new languages requires new embedding matrices and does not allow for parameter sharing, meaning cross-lingual use of the same model isn't an option.

We can also use **SentenceTransformers** to compute sentence embeddings for more than 100 languages. Instead of dealing with individual words, we could work directly with individual sentences. The framework is based on PyTorch and Transformers and offers a large collection of pre-trained models tuned for various tasks. We can use ` paraphrase-MiniLM-L6-v2 ` model for our task as it supports multilingual.
Sentence embeddings will also be able to encounter the new words which are not seen in our training dataset.
If we use contextualized sentence embedding, there is no need for tokenization, translation and spell check while doing data cleaning.
Recommended Python 3.6 or higher, and at least PyTorch 1.6.0.

Encode the class labels

Label Encoding refers to converting the text labels into numeric form so as to convert it into the machine-readable form.

Since all the categories has the same weight and there is no priority within them, we can use **One Hot Encoder** from **sklearn.preprocessing** library of python.

Select the model

When working on a supervised machine learning problem with a given data set, we can use effective text classification models and compare the models we trained in order to choose the most accurate one for our problem. As it depends on what kind of data we are handling, We can't tell which algorithm will perform best before experimenting them.

After reading various papers [4] , I have selected few models for our text classification problem given below:

- **Linear SVM**
- **Multiclass Logistic Regression**

When using SVM model we don't need to encode the labels as SVM has in-built method to do it. So, we can directly pass the categorical label.

Evaluate the model

We can estimate the time taken by every model while training the data and compare the time taken by each model by plotting a bar plot of x=techniques and y=training time to understand which models works fast.

To evaluate the performance of every model we can use **Confusion Matrix** to calculate the **accuracy, F1 score** of every model.

We can use **sklearn.metrics** python package to implement the confusion matrix.

References

[1]http://www.ijaerd.com/papers/finished_papers/A_review_of_feature_extraction_methods_for_text_classification-IJAERDV05I0489982.pdf

[2] https://sci2lab.github.io/ml_tutorial/multiclass_classification/#Training-Time-Comparison

[3] https://machinelearningmastery.com/cross-validation-for-imbalanced-classification/

[4] https://www.webology.org/2015/v12n2/a139.pdf