Industry Classification Problem

<u>Overview</u>

There are various job Categories that can be found in the Real world , also under each job category there is a wide range of Job titles that can be found, So some thing important is to classify World job categories and job titles under them .

Using

Supervised learning techniques for classification problem, Used to analyze text given to classify to which industry field it follows from 4 categories (IT, Marketing, Education, Accountancy) , The classifier makes the assumption that each new complaint is assigned to one and only one category. This is multi-class text classification problem.

We used The data set provided for Job Analysis having two variables (Job title & Industry) in a csv format of more than 8,500 samples.

This data sets are imbalanced so I deal with this problem of Imbalanced data, So we will provide the model with a full insight for all the data.

Techniques used for Cleaning data :

First of all I explored the data for Some Struggles that may lead our model to fail this problems like :

1- **Null data** as some models don't work if data contain nulls, so dealing with nulls isn't a choice it is a must (In our Data set there is no null values after checking using **df.info()** command .

2- **Check Frequency** Through **df.describe()** command to check most frequent data in each column.

3- **Check for duplicates** We need to check for duplicate data as this duplicated rows are an extreme case of nonrandom sampling, and they bias our fitted model. Including them will essentially lead to the model overfitting this subset of points.

4- **EDA** to explore and visualize our data so that we can see the imbalancing in our data, Through plotting 'job title' Column with the count of each job title found, So we see that 'Accountancy' field is found (<6%) in the data after dropping duplicates, So we need to deal with this column as it may disturb our model .

5- **Text Preprocessing** is a method to clean the text data and make it ready to feed data to the model. Text data contains noise in various forms like emotions, punctuation, Stopping word ,text in a different case , So we normalized all text in lower case then deal with numbers in numerical forms through changing them to text form, after that we need some regular expressions to match only text and exclude un important things like salaries , then we need to remove punctuation signs and white spaces .

Using Model to get Predictions :

I tested many models (Logistic regression, SVM, Naïve Bayes, SGD) this are from the best known classifiers for text classification, and they deal with imbalanced data that I will show in the next section.

I used **SGD (Stochastic gradient descent)** which have a cost function to minimize and it is better version of gradient descent Model as it use only one instance of the dataset at each update not the whole dataset as GD, I used grid search for hyper parameter tuning to get the best hyper parameters for it.

Naïve bayes is a simple algorithm based on probability for each class as it deal god with multi classes problems .

Logistic Regression is a good algorithm when we have the value of the target variable is categorical in nature. Logistic regression is most commonly used when the data in question has binary output, so when it belongs to one class or another.

Support Vector Machine (SVM) is an algorithm that determines the best decision boundary between vectors that belong to a given group (or category) and vectors that do not belong to it, It is the algorithm that gives the highest accuracy among all other classifiers, I needs the data to be vectorized before entering the classifier so it can separate between the regions of classification using the best fit line or plane.

Pipelining

We made a pipeline that contain CountVectorizer and TfidfVectorizer which both are methods for converting text data into vectors
In CountVectorizer we only count the number of times a word appears in the document which results in biasing in favour of most frequent words. this ends up in ignoring rare words which could have helped is in processing our data more efficiently.

To overcome this , we use TfidfVectorizer .

In TfidfVectorizer we consider overall document weightage of a word. It helps us in dealing with most frequent words. Using it we can penalize them. TfidfVectorizer weights the word counts by a measure of how often they appear in the documents then we add our model to the pipeline.

Dealing with imbalanced data :

Imbalanced datasets are a special case for classification problem where the class distribution is not uniform among the classes as in our dataset <6% of it is in 'Accountancy' class which makes it an imbalanced dataset , this made some bias in the model to classify the new text to the classes it see, so it makes the model overfit for some classes more than others with low existence.

There are many techniques to deal with imbalanced dataset Like over sampling, Under and Class weights in the models .

For dealing with imbalanced dataset I used Class weights, So that we can specify a higher weight for the minority class. Most of the machine learning models provide the parameter called Class weights, For example Logistic Regression, What happens that the Model have a loss function which is **$-y\log(p) - (1-y)\log(1-p)$**, it gives equal weights to all classes so when we specifiy class weights as {0:1,1:20,2:15,3:0.1} so the loss needed to be minimized , what happens exactly here is that if our model gives a probability of 0.3 and we misclassify a positive example, the NewLoss acquires a value of -20log(0.3) = 10.45.

Otherwise our model gives a probability of 0.7 and we misclassify a negative example, the NewLoss acquires a value of -log(0.3) = 0.52

That means we penalize our model around twenty times more when it misclassifies a positive minority example in this case.

To get class_weights using the distribution of the y variable, we can use the compute_class_weight utility from sklearn.

We can Extend this model to give us more better performance through using ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Through **Hard Voting,** as in hard voting the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the *output class(A, A, B)*, so here the majority predicted *A* as output. Hence *A* will be the final prediction.

Or Through **Soft Voting,** as in soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class *A = (0.30, 0.47, 0.53)* and *B = (0.20, 0.32, 0.40)*. So the average for class *A is 0.4333* and *B is 0.3067*, the winner is clearly class *A* because it had the highest probability averaged by each classifier.

I used the metric from Scikit learn to get the metrics for our classifier and the confusion matrix , to visualize the predictions of our classes what each classifier predict correctly and what it failed or classify wrongly .

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while **recall** (also known as sensitivity) is the fraction of relevant instances that were retrieved. Both precision and recall are therefore based on relevance.

F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.

```
F1 = 2 * (precision * recall) / (precision + recall)
```

ROC – AUC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1.

Our predictions or model may fail because of the imbalanced data that may effect our data and model as we see as it made the model biased for classes over other classes as they see this classes many times during the training process .

Also **duplicated** data make some bias for this data over other data .

Not having enough data also effects , as training our model over small data doesn't make sense especially for time series data .

Not having the right model A model is a simplified representation of reality. These simplifications are made to discard unnecessary fluff, noise, and detail. A good model allows its users to focus on the specific aspect of reality that is important in a particular domain. For example, in a marketing application, keeping attributes such as customer email and address might be important. Whereas in a medical setting a patient's height, weight, and blood type might be more important. These simplifications are rooted in assumptions; these assumptions may hold under certain circumstances, but may not hold in others. This suggests that a model that works well under a certain scenario may not work in another.