**CHICAGO SALARY PREDICTOR**

**ABSTRACT**

The Goal of the project is to build models which could predict the salary range for an employee in the City of Chicago given inputs like job description and title. Each model we tried to build had its advantage and disadvantage; our project will summarize the results of different model and their result for different scenarios. Since our system does the prediction of salary which is a continuous variable we tried an approach to convert the continuous dependent variables into categorical variables by the introduction of bins. The dataset which we took for training contained imbalanced data, so we followed three different balancing techniques to solve this problem. By solving the imbalanced property good model without error in the prediction of data was built. Accuracy cannot be considered as a good measure when the dataset is imbalanced so we used a new technique called the average accuracy for the calculation of the accuracy along with F1 score. Several parameter tuning was done to find the best parameter that fits for the problem. We compared Decision tree classification, Random forest classification and Logistic Regression model for our problem. Our results indicate that decision tree and random forest classification algorithm produce better result than the logistic regression in which Random Forest Classification produced the top result when the dataset was cleaned from its imbalanced property.

**1) DATA:**

For this project we have used two different datasets; dataset\_1 (referred to as “dataset” in the report) which is the main dataset used for building the model and other is the dataset\_2 (referred to as “Test Dataset” in the report) used for the testing purpose.

**1.1) Data Source:**

**Dataset:**

* The dataset is a listing of all current City of Chicago employees, complete with full names, Job Titles, Department, Full or Part-Time, Salary or Hourly, Typical Hours, Annual Salary, Hourly Rate. The dataset is downloaded from the city of Chicago portal [https://data.cityofchicago.org/Administration-Finance/Current-Employee-Names-Salaries-and-Position-Title/xzkq-xp2w/data]
* The size it takes on disk: is 1.78 MB which easily processed in the memory with RAM greater than 4 GB.
* This is the primary dataset from which various models are built. For the evaluation of model we have another dataset which is mentioned below.
* Once loaded for fitting the dataset contains 8 columns with 32698 records.

**Test Dataset:**

* The Test dataset contains a listing of all possible job title, description, its corresponding category and salary of the employee.
* This is used as the testing dataset for the model. Here a small modification needs to be done during the implementation of testing so that the job category and description of test dataset is replaced by its corresponding category and description of dataset. This is explained in a clear way at section 5.1.
* Once loaded for testing the dataset contains 10 columns with 36755.

**1.2) Data Format:**

* The data we have is in CSV (comma separated variables) format for both the datasets. CSV is simple to implement and parse.
* The processing can be done by almost all existing applications in a fast and easy rate. CSV is smaller in size and is considered to be standard format.
* Most of the classifiers perform well in CSV rather than JSON which does not have a structure of its own.
* The CSV format can be read using pandas and converted into data frames. The data frame structure provides easy processing of the data and provides building of the training model without difficulty.
* The column format for dataset contains ‘Name’, ‘Job Titles’, ‘Department’, ‘Full or Part-Time’, ’ Salary or Hourly’ , ‘Typical Hours’, ‘ Annual Salary’, ‘Hourly Rate’.
* The Test dataset has a column format of ‘Id’, ‘Title’, ‘FullDescription’, ‘LocationRaw’, ‘LocationNormalized’, ‘ContractType’, ‘ContractTime’, ‘Company’, ‘Category’, ’SourceName’.

**1.3) Language for coding:**

We have decided to implement the project with python as our programming language. The following table explain the packages needed and the purpose of the packages.

|  |  |
| --- | --- |
| **Package Name** | **Purpose** |
| 1) matplotlib.pyplot | To plot the result |
| 2) pandas | Package providing data structures design to work with “relational” or “labeled” data. |
| 3) LabelEncoder | Used to encode text into string |
| 4) KFold | Used to create cross validation folds |
| 5) LogisticRegression | Implements Logistic Regression model |
| 6) DecisionTreeClassifier | Implements Decision Tree Classifier model |
| 7) RandomForestClassifier | Implements Random Forest Classifier model |
| 8) mean\_absolute\_error | Calculate mean absolute error |
| 9) mean\_squared\_error | Calculate mean square error |
| 10) f1\_score | Calculate F1 score. |
| 11) scikit\_learn | Library that implements a range of machine learning techniques. |

These are some of the major ML packages used for our problem.

**2) EXPERIMENT:**

In this section we will discuss about the approaches used and how they are implemented in our project.

**2.1) APPROACHES:**

To do the prediction of salary we decided to use classification to classify salary based on the independent variable. Since our dataset is imbalanced we used certain approached to make that balanced. To ease description we will talk about the different approaches used for training the model and why we choose to use these approaches.

**2.1.1) DATA:**

* Our dataset comes from the city of Chicago portal on which the ‘Annual Salary’ of the employee is extracted as a dependent variable; with 2 columns ‘Job Title’ and ‘Department’ used as the independent variable.
* But the accuracy increases if we provide more independent variable to the model.

**2.1.2) PRE PROCESSING:**

* Before we begin classification we need to pre-process the data, First NaN values must be removed to make the dataset fit into the model. With NaN values in the data the model will produce error while prediction the result.
* Since we have ‘$’ sign for representation of salary we need to remove the special character for the ease classification process.
* The next major problem which arises is about the continuous value which exists in out dependent variable. Salary is always represented as a continuous variable but we tried to make better result by converting this continuous variable into a categorical variable. The method used for the process was generation of bins. Bins are nothing but a bucket which can store data for a particular range. First the data is sorted and then distributed into equal bins. By this way the continuous value is converted into a categorical variable.
* Now the process of converting the string entities in the independent variable to float is done using the help of label encoder. The label encoder distinguishes the string into unique float values which is fed into the classifier to fit the suitable model.
* At the end of the pre-processing we are left with independent variable which is converted from string to float and with bins of salaries which are our continuous dependent variables.

**2.1.3) BALANCING THE DATASET:**

Any data set that exhibits an unequal weight distribution between its classes i.e. the class weight for a particular label is high when compared to others is considered to be an imbalanced dataset. The dataset which we have is an imbalanced dataset.

* To solve this problem we will be using three different approaches; First is using the class\_weight=”balanced” thanks to the library scikit learn which provides the parameter class weight. The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \*np.bincount(y)).
* The next method is by using the under sampling technique. The approach is nothing but reduction of the class weights to the weight of the lowest label. By doing so the weight is balanced between the labels.
* The last approach is using SMOTE (Synthetic Minority Oversampling Technique). Random over sampling doesn’t produce good result. So we will use the approach of synthetically oversampling of the minority weight class. The structure of the method is that a minority sample is selected and its k nearest neighbours is selected. If we want to oversample by 200% then two samples are randomly picked from the k nearest neighbours and new samples are generated in their path. By doing so we get a more general oversample rather than random over sampling.

**2.1.4) CLASSIFICATION, REGRESSION ALGORITHM AND CROSS VALIDATION:**

* We will be using three different methods for fitting the model. The Decision tree classification, random forest classification and logistic regression which are done using the existing libraries of the python scikit learn.
* Two tree based classification and one regression algorithm are taken so that a better visualization of the result can be learnt from the result.
* Cross validation process with 10 folds is applied to get a better result than the normal test train split.

**2.2) IMPLEMENTATION:**

**2.2.1) DATA FRAME:**

* The data is read from csv format using pandas and converted into a data frame. This data frame is used for further processing. The path of the data set is mentioned as the parameter.

**2.1.2) PRE PROCESSING:**

* The removal of NaN is done by dropping the index which contains the NaN value. The special character in the salary column ‘$’ is removed by replacing this by an empty string.
* Bins are categorized into ‘low’, ‘medium’, ‘high’ and ‘very high’ with a specific range of salaries. Thus the continuous variables are converted into categorical variables and embedded into the dataframe.
* LableEncoder package is used to convert the string in the data frame into its corresponding float value.
* Now the dependent variable and independent variables are separated from the data frame.

**2.1.3) BALANCING THE DATASET:**

* The first approach for balancing the dataset is implemented by adding the parameter class\_weight=’balanced’ in the classifier and regression.

For example: classifier = RandomForestClassifier (class\_weight = 'balanced')

* “RandomUnderSampler” library is imported and the corresponding X and y are fitted into the sampler which results in the resampled values of X and y along with the sampling index.
* “SMOTE” library is imported and the X and y are fitted to the SMOTE sampler resulting in the resampled X and y. This method works well for binary classes only; so we have implemented a small trick by applying it again and again which could make it work for multiple classes.

**2.1.4) CLASSIFICATION, REGRESSION ALGORITHM AND CROSS VALIDATION:**

* We will implement the classification and regression algorithm by importing the “RandomForestClassifier”, “DecisionTreeClassifier” and “LogisticRegression” from the scikit-learn library.
* A cross validation of 10 folds are implemented by the “KFold” package with the length of y and the number of folds as its parameter.

**2.3) EVALUATION OF RESULT:**

* As professor mentioned in our report 2 comments accuracy cannot be considered as a good measure. So I started working on research papers to find a different measure along with MSE and MAE which could be considered as a best measure for the calculation of accuracy.
* F-measure and AUC assume that there is an active class: F-measure ignores true negatives as it is the harmonic mean of precision and recall. AUC ignores true negatives and false negatives, so evaluating the model based on the accuracy cannot always be correct.
* I implemented a new measure called average accuracy. Usually it is used in authentication systems under the form of Half Total Error Rate. For E.g. In [2] they provided a statistical test for that.
* MSE and MAE are implemented by using the inbuilt packages and the average accuracy which evaluate based on the formula 1/2\*((TP/TP+FN) + (TN/TN+FP)).

**3) EXPERIMENT AND RESULT:**

**3.1.1) DATASET:**

The dataset is read by the method as mentioned in the previous block and get the resultant data frame. The data frame is of size (32063, 4) and contains all the values of the data set.

**3.1.2) PRE PROCESSING:**

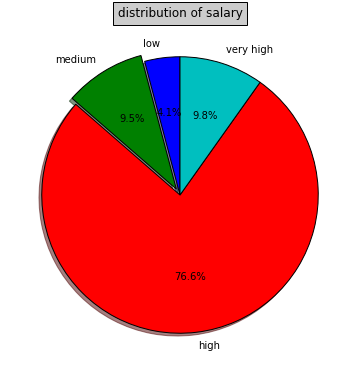
When the ‘NaN’ values are removed we have removed only one record from the data frame. So the new updated size is (32062, 4). This is because we weren’t left with much ‘NaN’ values if there are many ’NaN’ values in the data set then the size reduces considerably.

The next is the replacement of ‘$’ by ‘ ’ . Example: “$99936.00” becomes “99936.00”.

**3.1.3) BIN CREATION:**

When the creation of bins is implemented in our dataset we get the following resultant table.

|  |  |
| --- | --- |
| **Label** | **Weight** |
| high | 24551 |
| very high | 3151 |
| medium | 3039 |
| low | 1321 |



* The above table states that there are four bins created with labels and their corresponding weights. The pie chat provides more details about imbalanced data's.
* The Salary Labels and the bins are embedded into our data frame thus making our data frame of size (32062, 6). Notice that the column count is increased from 4 to 6.

**3.1.4) LABEL ENCODER:**

The text must be represented into its corresponding numeric format as we have discussed in our approach. Once this approach is implemented each text in the column is converted into its corresponding numeric entity. The following table give a sample encoding about how 5 of our dataset is encoded by numeric entity. In our experiment all the records are encoded for understanding we have explained with 5 records.

Table before encoding:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Position Title | Department | Annual Salary | Salary Labels | Salary Bins |
| WATER RATE TAKER | WATER MGMNT | 90744.0 | high | (55000, 105000] |
| POLICE OFFICER | POLICE | 84450.0 | high | (55000, 105000] |
| POLICE OFFICER | POLICE | 84450.0 | high | (55000, 105000] |

Table after encoding:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Position Title | Department | Annual Salary | Salary Labels | Salary Bins |
| 1083 | 34 | 692 | 0 | 3 |
| 760 | 27 | 621 | 0 | 3 |
| 760 | 27 | 621 | 0 | 3 |

We can note that similar texts are encoded with similar numeric values.

**3.1.5) IMBALANCED DATA:**

* The pie chart in 3.1.3 gives us a clear picture how the dataset which we are having is imbalanced. The above result show that high is of weight “24551” (76%) and low is of weight “1321” (4%).
* So the chances of prediction of low is comparatively less when compared to that of high. This states the inclination of the classifier towards the prediction of high.
* This problem occurred when I tried to fit the dataset initially into the Logistic regression without any balancing algorithm.

The confusion matrix appeared as follows:

[[6149 0 0 0] [327 0 0 0] [755 0 0 0] [785 0 0 0]]

* This shows that three classes have not been predicted by the classifier which makes the model fail. This problem is solved by using three different methods.

**3.1.6) CLASS WEIGHT PARAMETER:**

The scikit learn library from python comes for our saviour in this case. If the parameter class\_weight is set to balance the library will automatically adjust the proportional to the class frequency. The results for the various balancing are discussed in the analysis section.

**3.1.7) UNDERSAMPLING:**

When the under sampling is implemented the weights are converted as follows:

Original dataset shape Counter before under sampling: ({0: 24551, 3: 3151, 2: 3039, 1: 1321})

Original dataset shape Counter after under sampling: ({0: 1321, 1: 1321, 2: 1321, 3: 1321})

**3.1.8) SMOTE:**

SMOTE when applied into our data frame considers k (5 in our case) nearest neighbours (in feature space) of a minority sample to create a synthetic data point by taking the vector between one of those k neighbours, and the current data point.

Multiply this vector by a random number x which lies between 0, and 1. Add this to the current data point to create the new, synthetic data point. This is subjected to produce more general result than the random oversampling technique.

When the trick we discussed earlier is applied we get the iteration as follows;

Original dataset shape Counter ({0: 24551, 3: 3151, 2: 3039, 1: 1321})

Resampled dataset shape Counter ({0: 24551, 1: 24551, 3: 3151, 2: 3039})

Original dataset shape Counter ({0: 24551, 3: 3151, 2: 3039, 1: 1321})

Resampled dataset shape Counter ({0: 24551, 1: 24551, 3: 3151, 2: 3039})

Original dataset shape Counter ({0: 24551, 1: 24551, 2: 24551, 3: 3151})

Resampled dataset shape Counter ({0: 24551, 1: 24551, 2: 24551, 3: 24551})

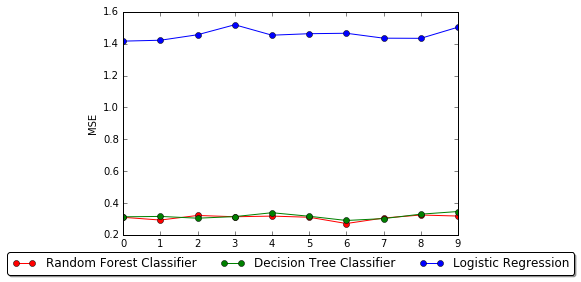
**3.1.9) FITTING THE MODEL AND CROSS VALIDATION:**

As discussed earlier the three models were trained and tested with the Test Dataset and their results are evaluated in the next section.

**3.2) EVALUATION OF RESULT:**

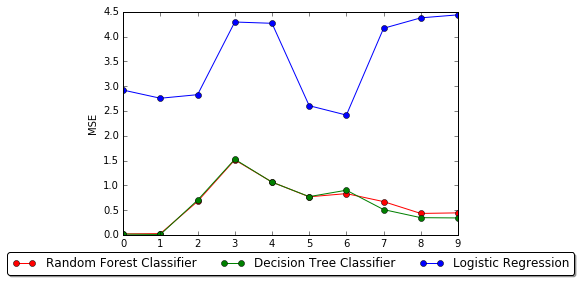
1. **Class\_weight accuracy and MSE:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **MSE** |
| DecisionTree Classifier | 0.830 | 0.889 | 0.887 | 0.559 |
| Logistic Regression | 0.735 | 0.541 | 0.732 | 1.491 |
| RandomForest Claasifier | 0.856 | 0.854 | 0.851 | 0.781 |



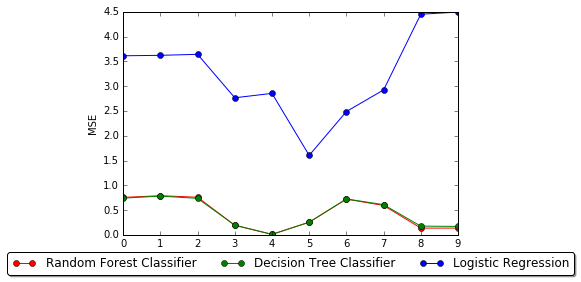
1. **Under-sampling accuracy and MSE:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **MSE** |
| DecisionTree Classifier | 0.799 | 0.841 | 0.887 | 0.632 |
| Logistic Regression | 0.715 | 0.490 | 0.732 | 1.512 |
| RandomForest Claasifier | 0.832 | 0.826 | 0.851 | 0.786 |



**c) SMOTE accuracy and MSE:**

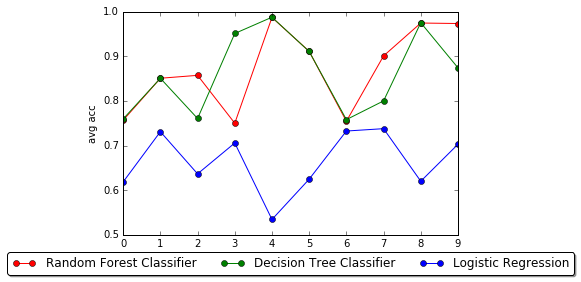
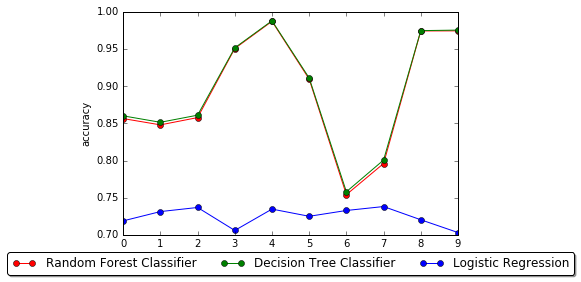
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** | **MSE** |
| DecisionTree Classifier | 0.862 | 0.912 | 0.912 | 0.502 |
| Logistic Regression | 0.765 | 0.586 | 0.765 | 1.304 |
| RandomForest Claasifier | 0.909 | 0.888 | 0.862 | 0.740 |



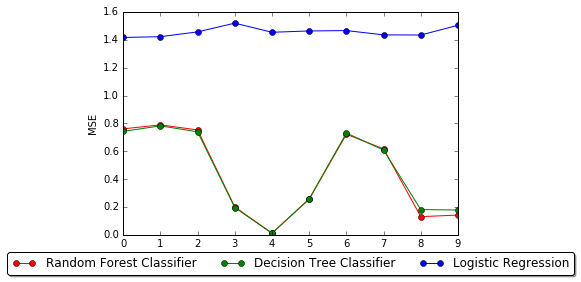
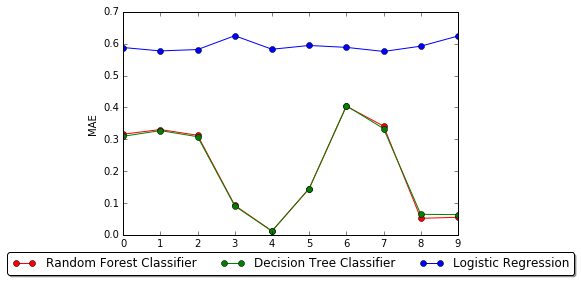
The best evaluation measure observed during testing is tabulated below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Mean Absolute Error** | **Mean Squared Error** | **Average Accuracy** |
| Decision Tree | 0.887876641579 | 0.214541086881 | 0.461206261117 | 0.8973 |
| Random Forest | 0.890992715638 | 0.208471883994 | 0.44580924931 | 0.9052 |
| Logistic Regression | 0.724907967035 | 0.510450042902 | 1.27241434782 | 0.7031 |

Note that the average accuracy is almost equal to 90% for Random Forest and Decision tree. The detailed analysis of our report is given in the following section 4 point 1.

The above graph shows the accuracy and average accuracy of the table. Their Mean Absolute Error and Mean Square Error are added evaluation techniques which are shown below.

**3.3) PARAMETER TUNING:**

Parameter tuning helps in determining the best possible parameters that could be applied for a classifier to give accurate results. We have used three classifiers for modelling our dataset and predicting the results. We have tried various combinations for each of the classifier and summarized the result below. This is an additional measure to state that Random Forest produces the best result for our problem.

Parameter Description:

min\_samples\_split – The minimum number of samples required to split an internal node:

criterion - The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Note: this

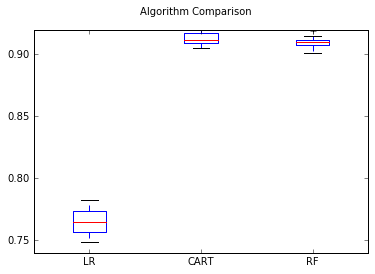
parameter is tree-specific.

max-iter - Maximum number of iterations taken for the solvers to converge. Default : 100

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Classifiers** | **min\_samples\_split** | **max\_iter** | **criterion** | **Accuracy** | **Precision** | **Recall** | **MSE** |
| Random Forest | 20 | - | entropy | 0.910 | 0.913 | 0.912 | 0.478 |
| Logistic Regression | - | 100 | - | 0.777 | 0.789 | 0.778 | 1.231 |
| Decision Tree | 20 | - | entropy | 0.891 | 0.887 | 0.851 | 0.509 |

**4) ANALYSIS OF RESULT:**

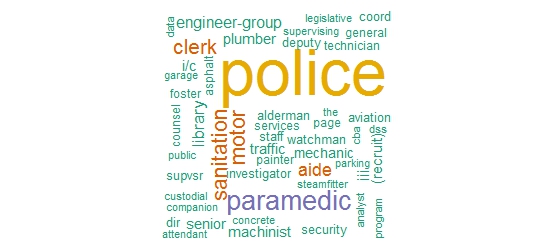
* Form the table in section 3.2 we can infer that the Random Forest Classifier works well. We know that Random forests is an ensemble learning method for classification that operate by constructing multitude of decision trees at training time and outputting the class that is the mode of the classes (in case of classification)of the individual trees as suggested by [3]. This is the reason why it fits best for our problem than the other two. The research papers [4], [5] may also give a brief idea why the random forest works best.
* Logistic regression works well for binary dependent classes; we cannot expect it to win as we have 4 bins in our dependent classes. However logistic regression can be made to produce 90% accuracy by using multinomial logistic regressor which brings non linearity for multiple classes.
* The table also suggests that SMOTE results are the best for balancing.Since it syntactically oversamples there by leading to a more generalized result rather than the random over or under sampling. Section 2.1.4 (point 3) adds clarity to the statement.
* We also tried another algorithm which could suit best for the SMOTE process. It’s the k-NN classification algorithm which produced almost 95% accuracy. Since we are using the above three models, I have not included it in our table. And the reason is pretty open since K-NN classifies object by a majority vote of its neighbours.



From the above box plot we can conclude that random forest classifier produces the best result when compared to the other two.

**5) ANALYSIS OF THE ERROR:**

* The major error which we faced during the problem was the clustering phase. As our problem states we have two datasets one for training and other for testing purpose. The model which is built using the training dataset is to be tested with the data’s in the test dataset.
* What we thought is that we will take the full description and category from the test dataset and the position title and department from the Training dataset convert that into a corpus of words do the pre procession and get the word cloud. Then form a cluster with that and get the similar department and title which could be used as a common independent variable for both the dataset.
* The result word cloud appeared as follows, which shows the major clustering of words in our corpus



* On the half way of our project we found that clustering doesn’t word for the above problem and we were moving on the wrong path so we stopped the approach by clustering and it made no sense for establishing a bridge, or to put it in a better way I found a better method which could bring the connection in an easy way.

**5.1) SOLUTION FOR THE ERROR:**

* There was another way to bring a common bridge between the two dataset. The approach was to find the cosine similarity between two words.

cos(x, y) =x\*y/sqrt(x^2)\* sqrt(y^2)

* As per the process I generated two list of words; one with the department of training dataset and other with the category of the test dataset. The list was made into a set so redundant words were removed.
* Now each word was converted into its corresponding word2Vec and the lists are made to run on a loop calculation its cosine similarity. The words which showed high cosine similarity were taken as a match.
* In a similar way the job description and the position title was done and their cosine similarity were taken. Kindly note that the process of binning the continuous dependent variable(salary) must be implemented in our Test Data set for getting the result.

The following table are some of the recorded cosine similarity (we can also consider Title for X from Test Dataset):

|  |  |  |  |
| --- | --- | --- | --- |
|  | X | Y | Cosine Similarity |
| X = Job Description  Y = Position Title | Water Resources Consultant, Salary: up to \*\*\*\*K (Plus Benefits) ……… | WATER RATE TAKER | 0.59484006339964346 |
| X = Category  Y = Department | Travel Jobs | STREETS & SAN | 0.75822390441428572 |
| X = Category  Y = Department | Hospitality & Catering Jobs | HEALTH | 0.50413037 |
| X = Job Description  Y = Position Title | Engineering Manager Location: Crawley Salary: \*\*\*\*\*\*\*\* Engineering Manager JOB ….. | SUPERVISING VENTILATION AND FURNACE INSPECTOR | 0.88670415 |

**6) CONCLUSION:**

We have generated a model which could take in job description and department as input and predict the salary for the corresponding texts. For this we have used Binning and Grouping for converting continuous class variables to categorical. This was done so that the used could get a clear view about his salary range rather than some inaccurate running number.

By analysis of different classification algorithms for real-time dataset and applying performance measures we could get the best model which could predict the salary for different inputs.

We also learnt how to use different approaches for balancing the dataset and how SMOTE helps us to get a more generalized dataset in terms of balancing.

Calculate accuracy for the imbalanced data sets takes a different approach in our problem by which we could give a justified measure for accuracy.

Calculation the cosine similarity of the terms we can create relation between Description and department.