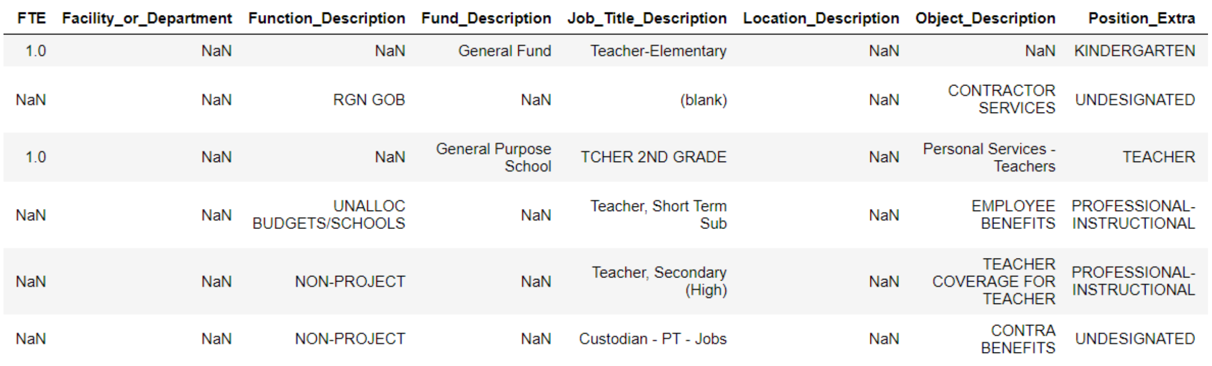
I. Definition

Project Overview

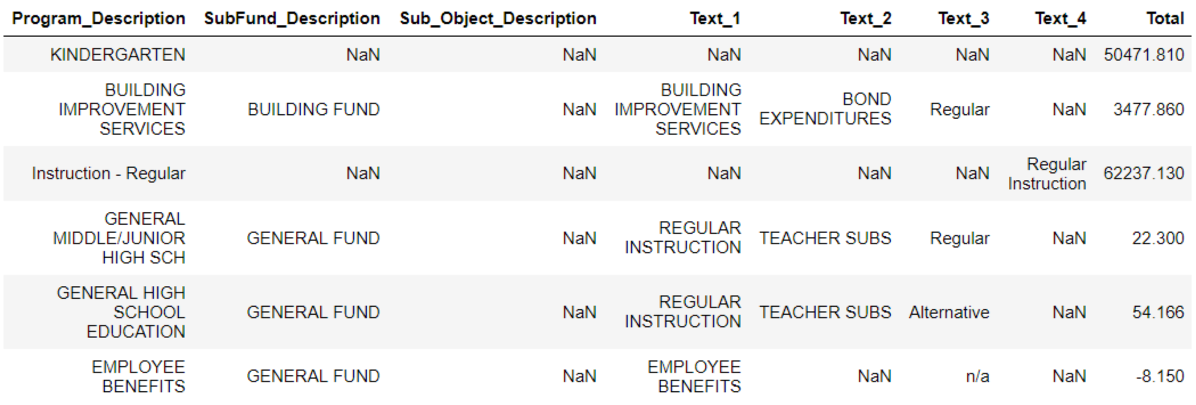
As the cost of education per student is increasing in US still it lags behind the global competitors in educational outputs. For the last many years, there is almost no correlation between dollars spending per student and their performance. This may be because the money has been spent on the areas which does not contribute or add to the student achievement and performance. This problem can be solved by finding a better way to distribute the resources for schools and districts and clarifying their spending habits. And finally implementing strategies to use money on education in a more effectively and efficient way.

This problem has been actively handled by ERS, a non-profit consulting firm established in 2004. Since 2004, They have been working with more than 20 school systems across US, including 16 of the 100 largest urban districts, on topics such as teacher compensation and career path, funding equity, school design, central office support, and budget development. In the last few years they have partnered with states as well. So, in this project I will use the dataset provided by ERS to create a strategy on how to better spend the money to improve the education system. This problem is a part of Machine Learning competition hosted by Driven Data. Dataset can be downloaded from here.

The dataset contains 400277 observations and 25 variables, of which 16 variables are predictors and 9 variables are target (each containing different number of classes). The aim of this project is to predict the class probabilities of each classes of those 9 target variables. Almost all the variables are textual in nature except FTE and Total which are numerical variables. Below is a screenshot of some of the predictor variables



FTE	Facility_or_Department	Function_Description	Fund_Description	Job_Title_Description	Location_Description	Object_Description	Position_Extra
1.0	NaN	NaN	General Fund	Teacher-Elementary	NaN	NaN	KINDERGARTEN
NaN	NaN	RGN GOB	NaN	(blank)	NaN	CONTRACTOR SERVICES	UNDESIGNATED
1.0	NaN	NaN	General Purpose School	TCHER 2ND GRADE	NaN	Personal Services - Teachers	TEACHER
NaN	NaN	UNALLOC BUDGETS/SCHOOLS	NaN	Teacher, Short Term Sub	NaN	EMPLOYEE BENEFITS	PROFESSIONAL-INSTRUCTIONAL
NaN	NaN	NON-PROJECT	NaN	Teacher, Secondary (High)	NaN	TEACHER COVERAGE FOR TEACHER	PROFESSIONAL-INSTRUCTIONAL
NaN	NaN	NON-PROJECT	NaN	Custodian - PT - Jobs	NaN	CONTRA BENEFITS	UNDESIGNATED



Program_Description	SubFund_Description	Sub_Object_Description	Text_1	Text_2	Text_3	Text_4	Total
KINDERGARTEN	NaN	NaN	NaN	NaN	NaN	NaN	50471.810
BUILDING IMPROVEMENT SERVICES	BUILDING FUND	NaN	BUILDING IMPROVEMENT SERVICES	BOND EXPENDITURES	Regular	NaN	3477.860
Instruction - Regular	NaN	NaN	NaN	NaN	NaN	Regular Instruction	62237.130
GENERAL MIDDLE/JUNIOR HIGH SCH	GENERAL FUND	NaN	REGULAR INSTRUCTION	TEACHER SUBS	Regular	NaN	22.300
GENERAL HIGH SCHOOL EDUCATION	GENERAL FUND	NaN	REGULAR INSTRUCTION	TEACHER SUBS	Alternative	NaN	54.166
EMPLOYEE BENEFITS	GENERAL FUND	NaN	EMPLOYEE BENEFITS	NaN	n/a	NaN	-8.150

Here is the data dictionary of the problem dataset-

FTE: The percentage of full-time that the employee works

Facility_or_Department: If expenditure is tied to a department/facility, that department/facility.

Function_Description: A description of the function the expenditure was serving.

Fund_Description: A description of the source of the funds.

Job_Title_Description: A description of that employee's job title.

Location_Description: A description of where the funds were spent.

Object_Description: A description of what the funds were used for.

Position_Extra: Any extra information about the position that we have.

Program_Description: A description of the program that the funds were used for.

SubFund_Description: More Detail on **Fund_Description**

Sub_Object_Description: More Detail on **Object_Description**

Text_1: Any additional text supplied by the district.

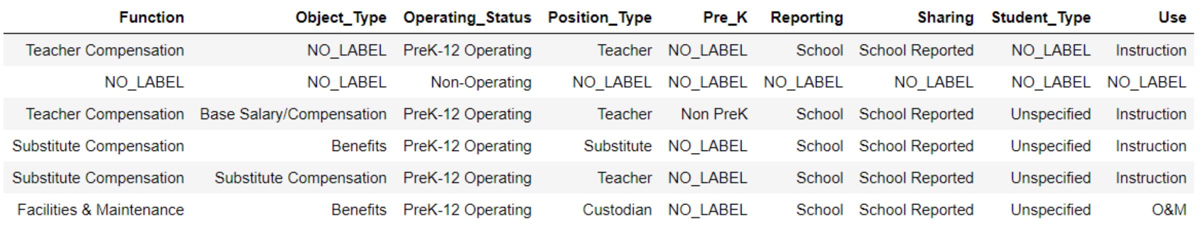
Text_2: Any additional text supplied by the district.

Text_3: Any additional text supplied by the district.

Text_4: Any additional text supplied by the district.

Total: The total cost of the expenditure.

Similarly, below is the screenshot for the target variables



Function	Object_Type	Operating_Status	Position_Type	Pre_K	Reporting	Sharing	Student_Type	Use
Teacher Compensation	NO_LABEL	PreK-12 Operating	Teacher	NO_LABEL	School	School Reported	NO_LABEL	Instruction
NO_LABEL	NO_LABEL	Non-Operating	NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL
Teacher Compensation	Base Salary/Compensation	PreK-12 Operating	Teacher	Non PreK	School	School Reported	Unspecified	Instruction
Substitute Compensation	Benefits	PreK-12 Operating	Substitute	NO_LABEL	School	School Reported	Unspecified	Instruction
Substitute Compensation	Substitute Compensation	PreK-12 Operating	Teacher	NO_LABEL	School	School Reported	Unspecified	Instruction
Facilities & Maintenance	Benefits	PreK-12 Operating	Custodian	NO_LABEL	School	School Reported	Unspecified	O&M

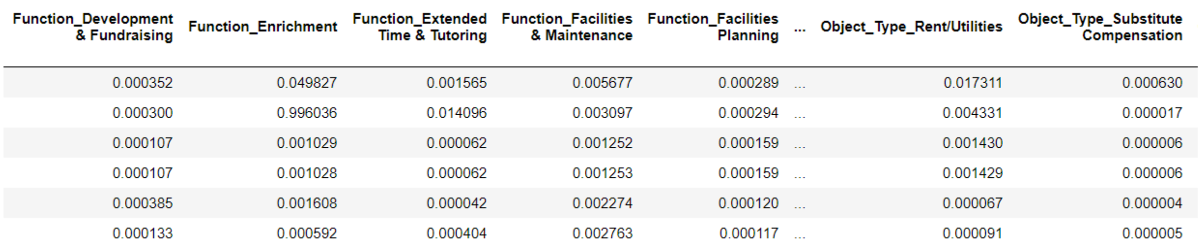
Where the target variable **Function** has 37 different classes, **Object_Type** has 11 different classes, **Operating_Status** has 3 different classes, **Position_Type** has 25 different classes, **Pre_K** has 3 different classes, **Reporting** has 3 different classes, **Sharing** has 5 different classes, **Student_Type** has 9 different classes and the variable **Use** has 8 different classes. To know more about classes in different target variables click here. There is a hierarchical relationship for these labels. If a line is marked as Non-Operating in the **Operating_Status** category, then all of the other labels should be marked as NO_LABEL since ERS does not analyse and compare non-operating budget items. So, our aim is to calculate probabilities for each of these classes (a total of 37 + 11 + 3 + 25 + 3 + 3 + 5 + 9 + 8 = 104 classes) of these 9 target variables.

Problem Statement

Budgets for schools and school districts are huge, complex, and unwieldy. It's not easy task to find out where and how schools are using their resources. ERS is a non-profit organization which tackles just this task with the goal of letting districts be smarter, more strategic, and more effective in their spending. So, the problem statement in this project **to solve a multi-class-multi-label classification problem with the goal of attaching canonical labels to the free form text in budget line items** (I have already explained the predictor and target variables in details in the above section). These labels let ERS understand how schools are spending money and tailor their strategy recommendations to improve outcomes for students, teachers, and administrators.

To solve this problem, I will use different classification techniques like Logistic Regression, Decision Trees and Random Forest classifiers etc along with Natural Language Processing techniques like Token Vectorization, n-grams and calculate the probabilities for each class of those target variables so that we can predict most likely classes for each target variables given the predictor variables.

The solution to this problem will look like below



Function_Development & Fundraising	Function_Enrichment	Function_Extended Time & Tutoring	Function_Facilities & Maintenance	Function_Facilities Planning	...	Object_Type_Rent/Utilities	Object_Type_Substitute Compensation
0.000352	0.049827	0.001565	0.005677	0.000289	...	0.017311	0.000630
0.000300	0.996036	0.014096	0.003097	0.000294	...	0.004331	0.000017
0.000107	0.001029	0.000062	0.001252	0.000159	...	0.001430	0.000006
0.000107	0.001028	0.000062	0.001253	0.000159	...	0.001429	0.000006
0.000385	0.001608	0.000042	0.002274	0.000120	...	0.000067	0.000004
0.000133	0.000592	0.000404	0.002763	0.000117	...	0.000091	0.000005

Where the model predicts the class probabilities for each target variable for different observations of the test data and then we can recommend the most likely class from each target variable based on the predictor variable. In simpler words, given the predictor variables like FTE, Function_Description, Fund_Description etc we can recommend the most likely label for the target variables for which the spending strategy will work most effectively.

Metrics

The evaluation metric that I will use in this project is the log loss. This is the loss function used in (multinomial) logistic regression and it is also used in other ML algorithms such as neural networks, defined as the negative log-likelihood of the true labels given a probabilistic classifier's predictions. The log loss is only defined for two or more labels. For a single sample with true label yt in {0,1} and estimated probability yp that yt = 1, the log loss is-

$$-\log P(yt|yp) = -(yt \log(yp) + (1 - yt) \log(1 - yp))$$

So, if I need to create a benchmark just by random guessing then I will give equal probabilities for each label within 9 different categories and that would give me

- Label probabilities for variable Function = 1/37 = 0.027
- Label probabilities for variable Object_Type = 1/11 = 0.091
- Label probabilities for variable Operation_Status = 1/3 = 0.33
- Label probabilities for variable Position_Type = 1/25 = 0.04
- Label probabilities for variable Pre_K = 1/3 = 0.33
- Label probabilities for variable Reporting = 1/3 = 0.33
- Label probabilities for variable Sharing = 1/5 = 0.20
- Label probabilities for variable Student_Type = 1/9 = 0.11
- Label probabilities for variable Use = 1/8 = 0.125

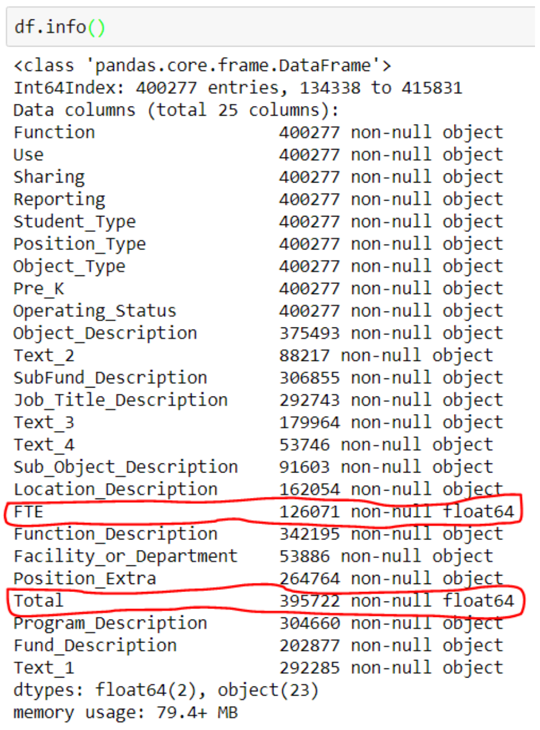
Using these equal probabilities for all the labels across 9 different categories we will get the log loss as 2.0455 (which is my benchmark model) and my aim will be reduce the log loss in comparison to the benchmark log loss by creating an improved model.

The metric log loss is helpful for evaluating models which have multi class multi label target variables as we cannot simply use accuracy metric to find out the predictive power of the model. The lower the value of the metric log loss, more is the predictive power of the model.

II. Analysis

Data Exploration

If we see the info about the dataset as below, we can see that only the columns **FTE** and



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 400277 entries, 134338 to 415831
Data columns (total 25 columns):
Function                  400277 non-null object
Use                       400277 non-null object
Sharing                   400277 non-null object
Reporting                 400277 non-null object
Student_Type              400277 non-null object
Position_Type             400277 non-null object
Object_Type               400277 non-null object
Pre_K                     400277 non-null object
Operating_Status          400277 non-null object
Object_Description        375493 non-null object
Text_2                    88217 non-null object
SubFund_Description       306855 non-null object
Job_Title_Description     292743 non-null object
Text_3                    179964 non-null object
Text_4                    53746 non-null object
Sub_Object_Description    91603 non-null object
Location_Description      162054 non-null object
FTE                       126071 non-null float64
Function_Description      342195 non-null object
Facility_or_Department    53886 non-null object
Position_Extra            264764 non-null object
Total                     395722 non-null float64
Program_Description       304660 non-null object
Fund_Description          202877 non-null object
Text_1                    292285 non-null object
dtypes: float64(2), object(23)
memory usage: 79.4+ MB
```

Total are of type float, the remaining columns which are text columns are stored as objects, we need to convert to strings/text later on for proper analysis. Here is a sample of how the data looks like

Location_Description	FTE	Function_Description	Facility_or_Department	Position_Extra	Total	Program_Description	Fund_Description	Text_1
NaN	1.0	NaN	NaN	KINDERGARTEN	50471.81	KINDERGARTEN	General Fund	NaN
NaN	NaN	RGN GOB	NaN	UNDESIGNATED	3477.86	BUILDING IMPROVEMENT SERVICES	NaN	BUILDING IMPROVEMENT SERVICES
NaN	1.0	NaN	NaN	TEACHER	62237.13	Instruction - Regular	General Purpose School	NaN

And the target variables

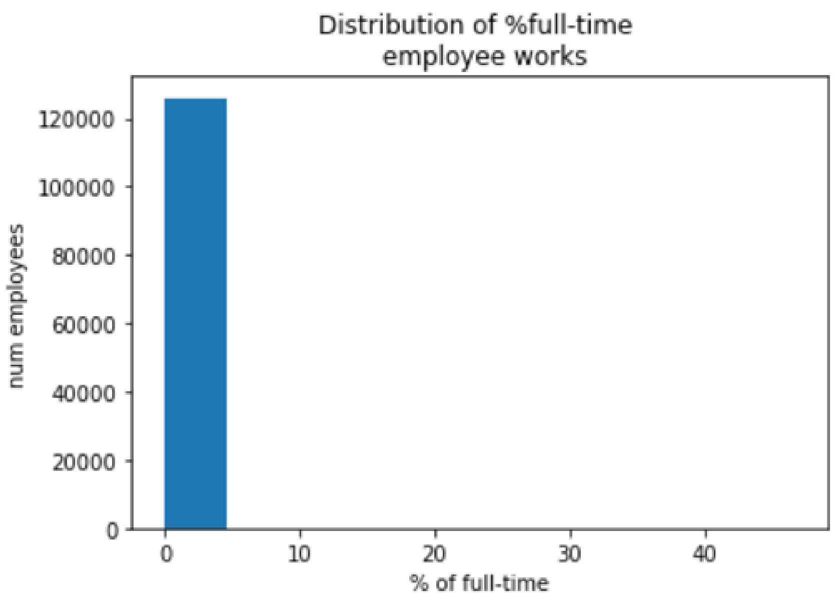
Function	Use	Sharing	Reporting	Student_Type	Position_Type	Object_Type	Pre_K	Operating_Status
Teacher Compensation	Instruction	School Reported	School	NO_LABEL	Teacher	NO_LABEL	NO_LABEL	PreK-12 Operating
NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL	NO_LABEL	Non-Operating
Teacher Compensation	Instruction	School Reported	School	Unspecified	Teacher	Base Salary/Compensation	Non PreK	PreK-12 Operating

As we can see there are lot of missing values in the data which are stored as NaN. Below is the summary statistics of the numeric columns

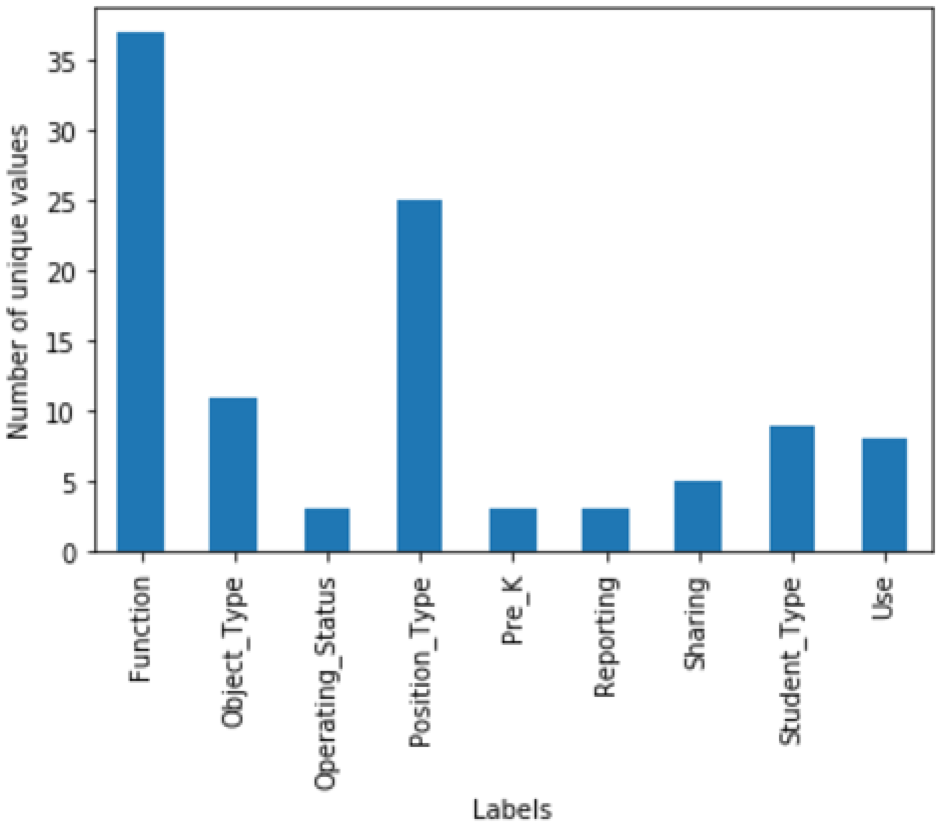
```
                 FTE         Total
count  126071.000000  3.957220e+05
mean        0.426794  1.310586e+04
std         0.573576  3.682254e+05
min        -0.087551 -8.746631e+07
25%         0.000792  7.379770e+01
50%         0.130927  4.612300e+02
75%         1.000000  3.652662e+03
max        46.800000  1.297000e+08
```

Exploratory Visualization

Below plot shows the histogram of employee works as a percentage of full time and the distribution shows it all. The distribution is right skewed and there are very few employees who works more than 5% of full time.



The following plot shows the number of unique labels of each target variables, highest being Function (with 37 unique labels) followed by Position_Type (25 unique labels)



Algorithms and Techniques

One of the problem with this project is that we don't have only one target variables unlike a conventional machine learning problem usually have. In this case, we have nine (9) target variables with each having a different number of classes. So, we cannot use a simple train-test split to split our data into training and testing sets. Also, as we have more than 100 labels, some labels might not get represented in the training data if we use normal train test split from sklearn module. So, I have used custom made function to split our data where all classes in target variables are represented at least some predefined times.

Another technique, that I will use in this project is OneVsRestClassifier which fits one classifier per class. For each classifier, the class is fitted against all the other classes. This approach improves the computational efficiency, and also improves the interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy for multiclass classification and is a fair default choice.

As I have already explained that 23 variables out of a total 25 variables in this dataset are textual in nature as shown below. So, it makes sense to use Natural Language Processing techniques like Tokenization or bag of words to quantify the words so that we can fed the numerical values into the machine learning algorithm.

```
Function                                        Student Transportation
Use                                                                O&M
Sharing                                                Shared Services
Reporting                                                   Non-School
Student_Type                                               Unspecified
Position_Type                                                    Other
Object_Type                               Other Compensation/Stipend
Pre_K                                                        Non PreK
Operating_Status                                     PreK-12 Operating
Object_Description        Extra Duty Pay/Overtime For Support Personnel
Text_2                                                             NaN
SubFund_Description                                         Operations
Job_Title_Description                    TRANSPORTATION,BUS DR., RADIO
Text_3                                                             NaN
Text_4                                     transportation - Second Runs
Sub_Object_Description    Extra Duty Pay/Overtime For Support Personnel
Location_Description                                       Unallocated
FTE                                                                NaN
Function_Description                                    Transportation
Facility_or_Department                       Transportation Department
Position_Extra                                              BUS DRIVER
Total                                                          1752.45
Program_Description                                      Undistributed
Fund_Description                                  General Operating Fund
Text_1                                                   EXTENDED DAYS
```

So, the aim here to combine all the variables which contains the text data and then create a bag of words. This can be achieved by converting the text data in each row of the data frame into a single string. CountVectorizer function sklearn feature extraction text module has been used to create the bag words. And to combine all the text data of each observation into a single string and then passing it to CountVectorizer has been done by writing a custom function. Also, the missing values if any (written as NaN) has been replaced by empty strings before creating the bag of words.

Benchmark

There is no benchmark model available as of now. This problem is a part of Machine Learning Competition that I am participating in. So, if I need to create a benchmark just by random guessing then I will give equal probabilities for each label within 9 different categories and that would give me

- Label probabilities for variable **Function =** 1/37 = 0.027
- Label probabilities for variable **Object_Type =** 1/11 = 0.091
- Label probabilities for variable **Operation_Status =** 1/3 = 0.33
- Label probabilities for variable **Position_Type =** 1/25 = 0.04
- Label probabilities for variable **Pre_K =** 1/3 = 0.33

- Label probabilities for variable **Reporting =** 1/3 = 0.33
- Label probabilities for variable **Sharing =** 1/5 = 0.20
- Label probabilities for variable **Student_Type =** 1/9 = 0.11
- Label probabilities for variable **Use =** 1/8 = 0.125

Using these equal probabilities for all the labels across 9 different categories we will get the log loss as **2.0455** (which is my benchmark model) and my aim will be reduce the log loss in comparison to the benchmark log loss by creating an improved model.

III. Methodology

Data Preprocessing

In this project, we have used pre-processing steps mainly during train-test split of the dataset and during creation of the bag-of-words representation of all the text data of predictor variable.

Since we have nine (9) target variable and 104 target classes, it may happen that training data set might not contain some target labels. And if this happens our model will not generalize to unseen data in future. So, we have created a custom function to split the data in a way such that a pre-specified minimum number of observations containing each target labels gets represented in the training data.

Also, another feature transformations that we have done is creating the bag of words representation. For this we have combined all the variables which were textual in nature and combined them into one string per observation and then we have performed the tokenization process to create tokens which in turn creates the bag of words representation of the whole dataset.

Another pre-processing step used in imputation of missing values. I have used Imputer class from sklearn pre-processing module. The default Imputer function imputes the missing values in a column by mean of the variable, which is what I have used in this problem. And we have only two numerical variables in our dataset and in the data exploration section I have created the histogram of these two features and we can observe that there is no abnormality in the data which needs to be addressed.

Implementation

In the implementation step of this project, we have used pipeline class from sklearn module. Pipeline sequentially apply a list of transforms and a final estimator. The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters. For this, it enables setting parameters of the various steps using their names and the parameter name separated by a ‘__’. A step’s estimator may be replaced entirely by setting the parameter with its name to another estimator, or a transformer removed by setting to None.

But as we have both kinds of data viz. numeric and text, so we cannot use both of them together in a single pipeline. Therefore, we have used FunctionTransformer class from sklearn pre-processing step. A FunctionTransformer forwards its input arguments to a user-defined function or function object and returns the result of this function. This is useful for stateless transformations such as taking the log of frequencies, doing custom scaling, etc. So, in this project we have used two FunctionTransformers one to extract numerical variables from the dataset and another one to extract text variables from the dataset. And after extracting the desired variables and doing the pre-processing tasks of imputing missing values in numerical variables and tokenizing the text variables we can use the FeatureUnion from sklearn pipeline module to merge the both numeric and text FunctionTransformers and then apply the machine learning algorithm on the FeatureUnion.

Following are the different steps that have been taken to complete this project

1. One-hot-encoding of the target labels (implemented using pd.get_dummies() function)
2. Multi label train-test splitting of the dataset (implemented by creating a custom-made function multilabel_sample_dataframe()to split the data so that target class labels are represented at least some predefined times, as mentioned in the section Algorithms and Techniques)
3. Creating FunctionTransformer to extract numerical and text data (implemented using FunctionTransformer() function from sklearn pre-processing module)
4. Using FeatureUnion to do pre-processing steps (like imputing, tokenization etc) separately for numeric and text data within the nested pipelines (implemented using FeatureUnion() function from sklearn pipeline module)
5. Used NLP technique ngram_range inside the pipeline to select unigram and bigram tokens only (implemented using CountVectorizer() function from sklearn feature extraction text module)
6. As we are not using the whole dataset for model building due computation limitation on my local machine. I have taken a sample of the dataset to remove the curse of dimensionality I have performed a dimension reduction technique. (Implemented by using SelectKBest() function from sklearn feature selection module)
7. Then we scaled the relevant features after dimension reduction (Implemented using MaxAbsScaler() function from sklearn pre-processing module)
8. Using a classification machine learning algorithm to the data tried to predict the target class probabilities (used Logistic Regression and Random Forest algorithms)

The only complications that required changes was to split the dataset to include all the 104 target classes into training and test datasets. So, achieve this we cannot simply use the train_test_split function from sklearn model selection module. Hence, I have used custom-made function achieve the desired split in the data. Remaining all the steps have been carried out by using standard sklearn functions

Refinement

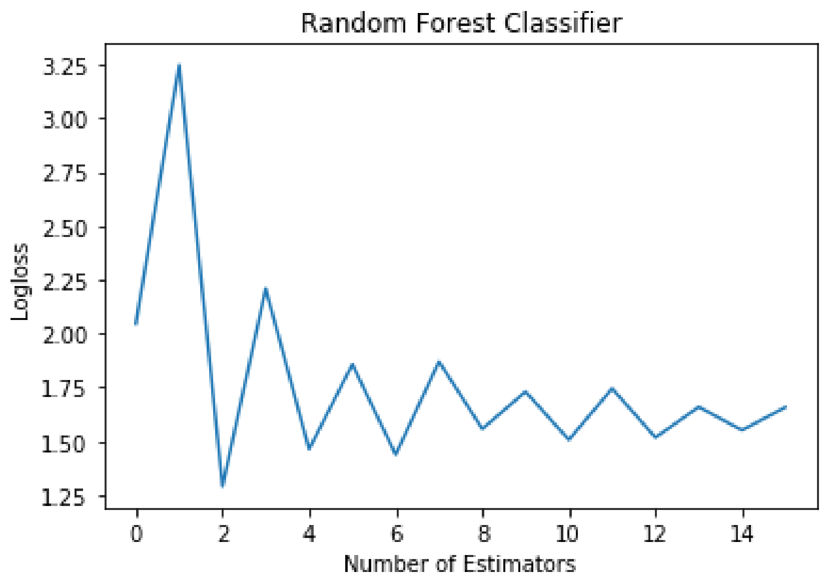
Initially I have executed a Logistic Regression on the training data after completing all the pre-processing steps as mentioned in the above section. Then I have used Random Forest Classifier with default parameters, as we can see from the below table the log loss metric for the test data has improved from earlier Logistic Regression Classifier.

Algorithm	Parameters	Logloss	Accuracy
Logistic Regression Classifier	Default	1.9721	65.78%
Random Forest Classifier	Default	1.7329	59.41%
Random Forest Classifier	n_estimators = 2	1.6574	62.39%

As I have explained for a process for improvement I have changed my classifier from Logistic Regression to Random Forest as the later algorithm provided lower log loss than the former. Then after getting the improved algorithm I have tried to tune the hyperparameter n_estimators to find the optimum result. But doing so it also decreased the accuracy of the model.

I have also added two other refinements in the model. I have added interaction effect of the tokens on the model. Interaction effect means what is the effect of the predictors on the target when more than one token appears together. I have restricted myself to use interaction effect of two terms only due to computational complexity and also it is a very rare phenomenon to have appear more than two random tokens together in the dataset. Another refinement I have added is feature hashing. Feature hashing is used to increase memory efficiency and thereby reduces the computational cost. Hashing function takes tokens as an input and outputs a hashing value. This can be implemented by using HashingVectorizer function from sklearn feature extraction text module. Below are the results after adding these two refinements in the model.

Algorithm	Parameters	Logloss	Accuracy
Logistic Regression Classifier	Default	1.5706	70.91%
Random Forest Classifier	Default	1.5062	64.21%
Random Forest Classifier	n_estimators = 2	1.3187	68.63%

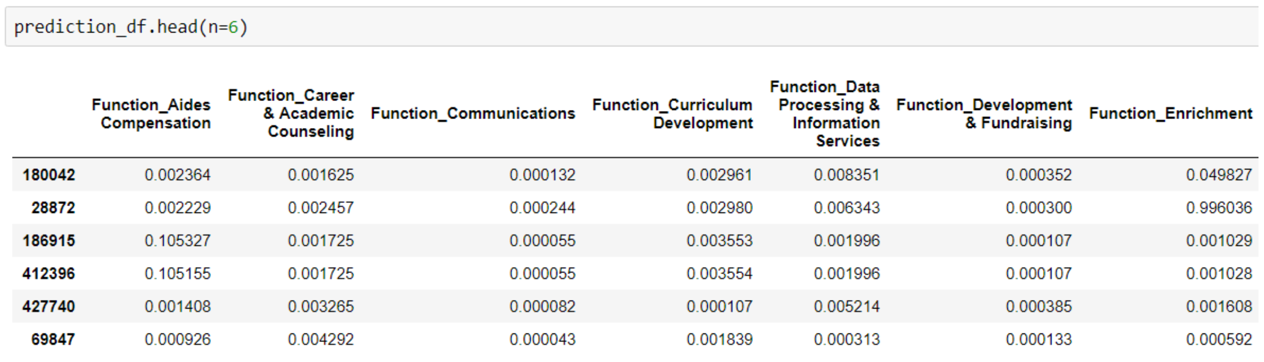


IV. Results

Model Evaluation and Validation

Now, as our aim is to predict the probabilities of the class labels for the 9 target variables so that we can get most likely classes of each target variables. We have used predict_proba() function to get those probabilities.

Also, to check whether the model generalizes to unseen data, we have a holdout data and I executed on the holdout dataset. Below is a screenshot of the final output of the model on the holdout dataset.



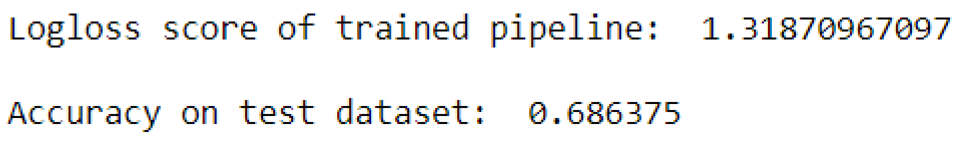
```
prediction_df.head(n=6)
```

	Function_Aides Compensation	Function_Career & Academic Counseling	Function_Communications	Function_Curriculum Development	Function_Data Processing & Information Services	Function_Development & Fundraising	Function_Enrichment
180042	0.002364	0.001625	0.000132	0.002961	0.008351	0.000352	0.049827
28872	0.002229	0.002457	0.000244	0.002980	0.006343	0.000300	0.996036
186915	0.105327	0.001725	0.000055	0.003553	0.001996	0.000107	0.001029
412396	0.105155	0.001725	0.000055	0.003554	0.001996	0.000107	0.001028
427740	0.001408	0.003265	0.000082	0.000107	0.005214	0.000385	0.001608
69847	0.000926	0.004292	0.000043	0.001839	0.000313	0.000133	0.000592

The above screenshot is only a part of the output result. As we have 104 target labels, and it is not possible to show the probabilities for all the labels together. But to get the idea, take the second observation from the above output. For the target variable Function, we have predicted the probability of occurrences for all the labels. And out of all the seven (7) different labels of Function, the "enrichment" label is most likely to occur for the second observation with a probability of 99.6%. So, we can say that the final model aligns with the solution expectations and also generalizes well to the unseen data.

To check the robustness of the model, we can change the input text values of the predictor variable and check the model performance. As out of the 25 predictor variables, 23 variables are textual in nature and then we have combined all the text data to create one single string and tokenized the whole string. So, by changing some input texts in training data would not affect the model performance much. So, we can say that the model built is robust and not affected by any small perturbations in the training data.

We can definitely trust the result found from the model. To prove this statement, we can use the metric "accuracy" from the sklearn metrics module. We can calculate the accuracy of the unseen test dataset using score() of the classifier. Below screenshot shows the accuracy of the model (Random Forest Classifier). But I have chosen the Logistic Regression as my final model. Although log loss of Random Forest is much lower than Logistic Regression but it has more accuracy than the Random Forest Model. And also the log loss for Logistic Regression model is lower than my benchmark log loss of 2.0455.



```
Logloss score of trained pipeline:  1.31870967097

Accuracy on test dataset:  0.686375
```

Justification

The benchmark model has a log loss value of 2.0455 and accuracy of 0% and the log loss value of the final model is reduced to 1.9721 with an accuracy of 70.91%. So, we can say that the final results are found stronger than the benchmark result reported earlier.

The final solution used in this problem contains the following steps

1. One-hot-encoding of the target labels

2. Multi label train-test splitting of the dataset

3. Creating FunctionTransformer to extract numerical and text data

4. Using FeatureUnion to do pre-processing steps (like imputing, tokenization etc) separately for numeric and text data within the nested pipelines

5. Used Hashing technique to reduce computational cost

6. Used NLP technique ngram_range inside the pipeline to select unigram and bigram tokens only

7. Using Dimension Reduction technique.

8. Added feature interaction effect to the model

8. Scaled input data using MaxAbsScaler()

9. Used Random Forest Classifier with parameter tuning

And we can conclude that the final solution obtained is significant to have solved the problem. As we have reduced the log loss by 37% and increased the accuracy from 0% to 67.44%

V. Conclusion

Free-Form Visualization

I have included visualization of target label classes, log loss Vs number of estimators and few others. And also explained and analysed the visualizations in details.

Reflection

I have thoroughly summarized the entire process and steps used to solve this problem in the above section (1-9 steps). One of the interesting aspect of this project I would say is the adding interaction effect between the features in the model. After we have added the interaction features the model log loss improved significantly. And the difficult aspect of the project is to split the dataset such that a pre-specified minimum number of observations containing each target labels gets represented in the training data. The final model and solution fit my expectations for the problem, and this model can be generalized to solve these types of problems.

Improvement

Some of the possible improvements are listed below

1. Increasing the value for min_count parameter in the multilable_train_test_split() function might improve the result (as it will increase the occurrences of the target label classes in the training data)
2. Increasing the number of n_gram range (e.g. Include trigram tokens) might improve the result
3. Use of ensemble techniques like bagging and boosting might improve the result as well
4. As I have used only sample of the dataset due to computational complexity, so taking the whole dataset might have improved the model performance
5. Use of neural networks or deep learning techniques might also improve the model performance.