# School Budget Line Item Classification

## A Multi-class, Multi-label Classification Problem

Final Report for Capstone Project

Springboard, DSCT

July 2018

Mark Sausville
saus@ieee.org

# School Budgets: Multi-class, multi-label classification, Capstone Final Report

## Overview of the problem

Budgets for schools and school districts are large, complex, and unwieldy. Education Resource Strategies (ERS) is a non-profit that analyzes school budgets using human expert analysts with the goal of letting districts be more effective in their spending.

The task is to correctly label every budget line item in nine different classifications with appropriate labels for each of 9 classes. These labels allow ERS to understand how schools are spending money and tailor their strategy recommendations to improve outcomes for students, teachers, and administrators.

An effective machine learning model will let human analysts complete work much faster by eliminating a substantial amount of time-consuming hand analysis and allow ERS to serve its core client base (public schools and school districts) more effectively.

The data consists of 14 columns of descriptive free-form text data and 2 numerical columns related to the nature of the expenditure.

The project is a multi-class, multi-label classification problem with challenges in text processing, feature design and feature engineering, model selection and model tuning. It also has the advantage of participation in an active competition, so models can be objectively judged for quality.

The data is hosted as part of a competition at Drivendata.org. Along with the competition documentation, DrivenData has created a tutorial demonstrating several relevant modeling techniques.

The goal of this project is to analyze the performance of the presented models, and independently produce a competitive and high-quality model.

This project breaks down into the following steps:

- Acquire the data.
- Explore the characteristics of the data.
- Analyze the structure and performance of the models that DrivenData has presented.
- Independently produce a competitive model.

## Data Acquisition

The data are available at this link.   DrivenData also provides documentation on the dataset and required submission format.

The files are:

- TrainingData.csv – 400k rows, 9 targets, 16 features
- TestData.csv – 50 rows, 16 features

The labels of the test data are withheld and used by the hosting organization to judge the competition.

## Problem Description

The goal is to fit a model that predicts for each row the most likely label for each of the 9 target columns, minimizing the log loss.  Log loss (cross-entropy) is a convenient metric for judging the competition as it provides a single number summarizing prediction quality.

In this document and the associated notebooks, I use the term 'multi-target, multi-class' in preference to the more usual 'multi-class, multi-label' because the terms 'class' and 'label' are often used interchangeably.  Multi-target-multi-class unambiguously distinguishes this kind of problem as producing multiple independent predictions, each with its own set of labels.

Please see appendix A for a description and clarification of terminology.

## Data Description

All rows have 9 target columns, each containing 1 label. Each row in the budget has 14 free-form text features and two floating point features. Any feature may be null.

### The features in the dataset

The dataset contains the following 16 feature columns:

- FTE (float) - If an employee, 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 - If this is an employee, 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 (float) - The total cost of the expenditure.

## The targets in the dataset

The dataset contains the following 9 target columns.

| Target Column | Number of labels | Sample of labels |
|---|---|---|
| Function | 37 | 'Teacher Compensation', 'NO_LABEL', 'Substitute Compensation'… |
| Object_Type | 11 | 'NO_LABEL', 'Base Salary/Compensation', 'Benefits'… |
| Operating_Status | 3 | 'PreK-12 Operating', 'Non-Operating', 'Operating, Not PreK-12' |
| Position_Type | 25 | 'Teacher', 'NO_LABEL', 'Substitute'… |
| Pre_K | 3 | ('NO_LABEL', 'Non PreK', 'PreK') |
| Reporting | 3 | 'School', 'NO_LABEL', 'Non-School' |
| Sharing | 5 | 'School Reported', 'NO_LABEL', 'School on Central Budgets'… |
| Student_Type | 9 | 'NO_LABEL', 'Unspecified', 'Special Education'… |
| Use | 8 | 'Instruction', 'NO_LABEL', 'O&M'… |

## Feature Example

The table below shows all 16 feature names and sample values for one row.

| Feature Name | Feature Value |
|---:|---:|
| Object_Description' | NaN |
| 'Text_2' | SPECIAL EDUCATION INSTRUCTION |
| 'SubFund_Description' | LOCAL |
| 'Job_Title_Description' | Teacher, Special Education |
| 'Text_3' | NaN |
| 'Text_4' | NaN |
| 'Sub_Object_Description' | NaN |
| 'Location_Description' | NaN |
| 'FTE' | 1.0 |
| 'Function_Description' | NaN |
| 'Facility_or_Department' | NaN |
| 'Position_Extra' | NaN |
| 'Total' | 67397.91883 |
| 'Program_Description' | NaN |
| 'Fund_Description' | NaN |
| 'Text_1' | NaN |

## Label Example

The table below shows the 9 target names and label value for the features above.

| Target | Label |
|---|---|
| Function | Teacher Compensation |
| Use | Instruction |
| Sharing | School Reported |
| Reporting | School |
| Student_Type | Special Education |
| Position_Type | Teacher |
| Object_Type | Base Salary/Compensation |
| Pre_K | NO_LABEL |
| Operating_Status | PreK-12 Operating |

## Submission Format

DrivenData requires that a submission be in the following format:

- 50064 rows, 104 columns of floating point probabilities
- Label columns with "TargetName__PossibleLabel", e.g. Function__Teacher_Compensation, replacing spaces in labels with underscores.

## Multi-target, multi-class classification and the tutorial’s approach

### Classification

The DrivenData tutorial’s approach to classification is to one-hot encode all targets and use sklearn.multiclass’s OneVsRestClassifier to drive a logistic regression classifier across all 104 resulting binary columns. This classifier can then produce probability predictions in the correct format, treating the problem as 104 separate binary classification problems.

Sklearn classifiers, in general cannot deal with multi-target, multi-class problems (with some exceptions). While logistic regression will properly classify in single-target, multi-class problems, it will not work in the more general setting.

It should be noted that some Sklearn classifiers can natively deal with multi-target, multi-class tasks (notably RandomForestClassifier). These classifiers, however produce prediction outputs in a different format (see first_models.ipynb, mod2 for details).

The choice of OneVsRestClassifier has several ramifications. First, Sklearn does not provide generally applicable metrics for multi-target, multi-class problems. For the competition, DrivenData uses (and provides) a specific metric that calculates the average log loss across all targets.

Second, standard classification metrics (such as accuracy, precision, recall, F1, etc.) require label predictions to work correctly. The predictions from the OneVsRestClassifier() are in the wrong format for these metrics, as the classifier produces 104 different and independent binary predictions for each possible label. For the standard metrics to work correctly, the probabilities should be normalized such that within a set of columns corresponding to one of the nine targets, the sum of probabilities is 1. In addition, to produce correct predictions the highest probability within the normalized columns should be selected.

For this project, we built a context-specific tool (see flat-to-labels.ipynb) that correctly restructures the classifier output (104 columns of probabilities) to 9 columns of labels (matching the input target structure) to be able to apply the standard Sklearn classification quality metrics for each target.

### Feature Engineering

In their tutorial, DrivenData takes the approach of merging all text components within a row into a single string. That text data is then converted to word vectors by either CountVectorizer or HashingVectorizer.

The word vectors are then appended to the numeric data and this combined data becomes the input for classification.

In essence, the labels for a row are predicted based on the count of words (tokens) occurring within that row.  In later models, interaction terms are added that provide an indication of  certain words being used in combination.

## The models in the tutorial

The DrivenData tutorial presents five different models.

### Mod0

A rudimentary model that uses only the numerical data, ignoring all the text data.  This has the advantage of producing a working model that produces output in the correct syntax, with correct semantics.  This model is then used to predict probabilities on the holdout data that can be used for submission to the competition.   The framework illustrated in this model is carried forward throughout the tutorial.

### Mod1

This model adds text processing to the workflow, illustrating several important features:

- ***Pipeline*** and related tools that allow different feature types to be processed separately and recombined before classification.
- ***CountVectorizer*** is used to create sparse word-count vectors for input to the classifier with default settings.

The pipeline structure in this model is retained throughout the rest of the tutorial, facilitating model changes and enhancements.

### Mod2

RandomForestClassifier replaces the OneVsRestClassifier demonstrating the flexibility provided by the pipeline.

### Mod3

This model changes the configuration of the CountVectorizer by adding bigrams and modifying the tokenization pattern.

### Mod4

The final model introduces the HashingVectorizer for reduction of memory usage.  The feature set is then reduced with SelectKBest and a custom tool (provided by DrivenData) adds interaction features.  Essentially, the set of all combinations of  the k best selected features is produced and the product of each pair of features is calculated.  This provides an indicator matrix that shows when tokens appear together in the data.  These new features are added to the feature set.  This is an implementation of a technique used in a previous winning entry.

## Initial Findings

In the first phase of this project we did the following:

- Acquired the data.
- Performed an exploratory data analysis (see EDA_ddbpfe.ipynb).
- Split the data into train and test sets
- Coded all 5 models. (see first_models.ipynb)
- Fitted all models and made predictions for holdout set (see first_models.ipynb)
- Submitted predictions to the competition.
- Coded tools to correctly normalize and restructure model output so that the detailed performance of each model can be observed.
- Measured performance of all models with standard metrics on all 9 targets in the dataset.

### Detailed Results for all models presented in the tutorial

Model 0

| metric scores on test set<br>Mod0 | | competition score | log loss | accuracy | F1 | ROC AUC |
|---|---|---|---|---|---|---|
| **All targets** | **aggregate** | 1.3314 | **1.3557** | **0.5599** | **0.1532** | **0.5645** |
| Scores on individual targets | Function | | 2.5994 | 0.2760 | 0.0312 | 0.5665 |
| | Use | | 1.7791 | 0.4253 | 0.1183 | 0.6974 |
| | Sharing | | 0.4372 | 0.8588 | 0.3083 | 0.5666 |
| | Reporting | | 2.0767 | 0.3729 | 0.0512 | 0.6146 |
| | Student_Type | | 0.6171 | 0.7663 | 0.2892 | 0.5726 |
| | Position_Type | | 0.8652 | 0.6410 | 0.2611 | 0.5592 |
| | Object_Type | | 1.1127 | 0.6337 | 0.1554 | 0.5064 |
| | Pre_K | | 1.2525 | 0.5570 | 0.0798 | 0.5218 |
| | Operating_Status | | 1.4616 | 0.5082 | 0.0845 | 0.4752 |

Model 1

| metric scores on test set<br>Mod1 | | competition score | log loss | accuracy | F1 | ROC AUC |
|---|---|---|---|---|---|---|
| **All targets** | **aggregate** | 0.7456 | **0.5117** | **0.8619** | **0.7496** | **0.9653** |
| Scores on individual targets | Function | | 0.9227 | 0.7976 | 0.6864 | 0.9812 |
| | Use | | 0.5640 | 0.8730 | 0.8010 | 0.9849 |
| | Sharing | | 0.1992 | 0.9357 | 0.7968 | 0.9560 |
| | Reporting | | 0.6877 | 0.8419 | 0.8097 | 0.9866 |
| | Student_Type | | 0.1632 | 0.9529 | 0.8644 | 0.9785 |
| | Position_Type | | 0.4119 | 0.8651 | 0.8225 | 0.9439 |
| | Object_Type | | 0.5941 | 0.8146 | 0.6243 | 0.9214 |
| | Pre_K | | 0.4566 | 0.8729 | 0.7000 | 0.9713 |
| | Operating_Status | | 0.6062 | 0.8035 | 0.6411 | 0.9634 |

Model 2

| metric scores on test set<br>Mod2 | | competition score | log loss | accuracy | F1 | ROC AUC |
|---|---|---|---|---|---|---|
| **All targets** | **aggregate** | 1.6941 | **0.2876** | **0.9757** | **0.9330** | **0.9838** |
| Scores on individual targets | Function | | 0.6479 | 0.9531 | 0.8630 | 0.9696 |
| | Use | | 0.2160 | 0.9806 | 0.9560 | 0.9946 |
| | Sharing | | 0.1798 | 0.9844 | 0.9210 | 0.9703 |
| | Reporting | | 0.2440 | 0.9781 | 0.9472 | 0.9906 |
| | Student_Type | | 0.1070 | 0.9897 | 0.9674 | 0.9896 |
| | Position_Type | | 0.2620 | 0.9778 | 0.9687 | 0.9908 |
| | Object_Type | | 0.3349 | 0.9705 | 0.9486 | 0.9897 |
| | Pre_K | | 0.2324 | 0.9789 | 0.9412 | 0.9829 |
| | Operating_Status | | 0.3645 | 0.9680 | 0.8837 | 0.9765 |

Model 3

| metric scores on test set<br>Mod3 | | competition score | log loss | accuracy | F1 | ROC AUC |
|---|---|---|---|---|---|---|
| **All targets** | **aggregate** | 0.8174 | **0.3572** | **0.8891** | **0.7733** | **0.9759** |
| Scores on individual targets | Function | | 0.6513 | 0.8183 | 0.6428 | 0.9796 |
| | Use | | 0.3326 | 0.9023 | 0.7703 | 0.9875 |
| | Sharing | | 0.1464 | 0.9517 | 0.8456 | 0.9742 |
| | Reporting | | 0.4625 | 0.8657 | 0.7683 | 0.9863 |
| | Student_Type | | 0.0975 | 0.9679 | 0.9073 | 0.9885 |
| | Position_Type | | 0.3173 | 0.8848 | 0.8449 | 0.9628 |
| | Object_Type | | 0.4188 | 0.8617 | 0.7455 | 0.9587 |
| | Pre_K | | 0.3194 | 0.8984 | 0.7692 | 0.9803 |
| | Operating_Status | | 0.4689 | 0.8511 | 0.6660 | 0.9656 |

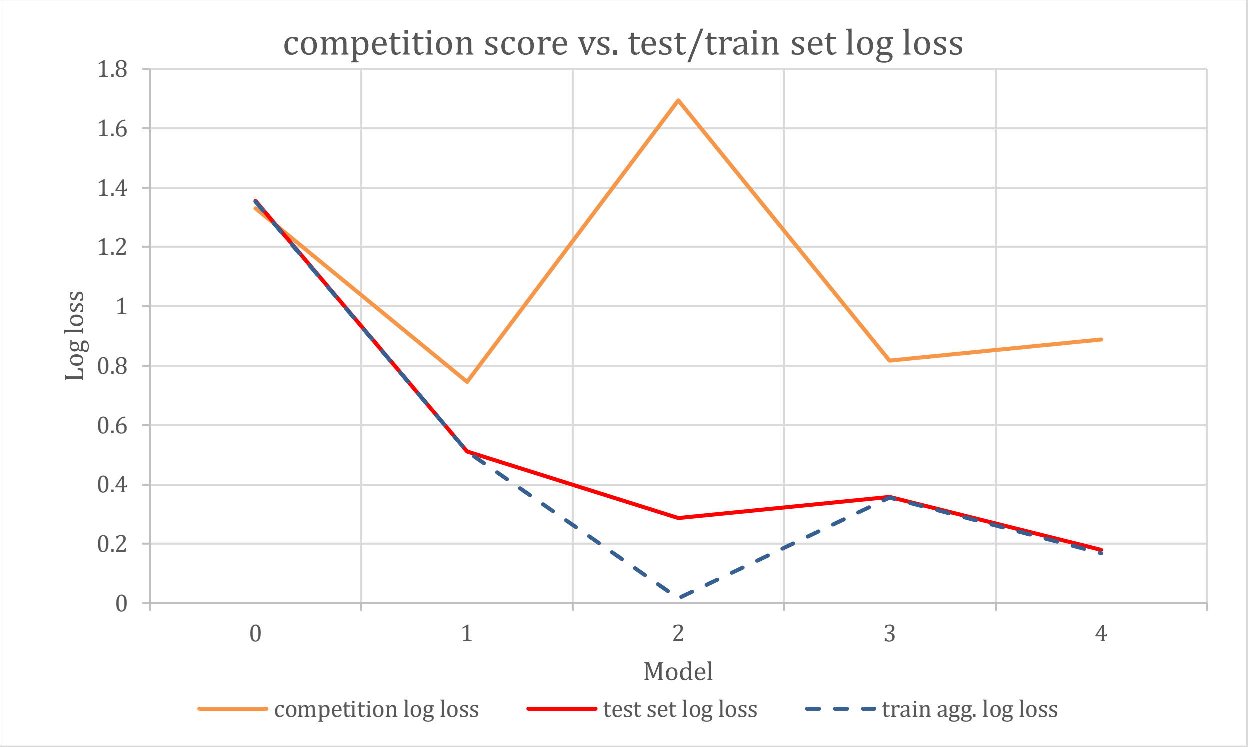
Model 4

| metric scores on test set<br>Mod4 | | competition score | log loss | accuracy | F1 | ROC AUC |
|---|---|---|---|---|---|---|
| **All targets** | **aggregate** | 0.8893 | **0.1796** | **0.9449** | **0.8664** | **0.9933** |
| Scores on individual targets | Function | | 0.3502 | 0.9007 | 0.7602 | 0.9921 |
| | Use | | 0.1521 | 0.9550 | 0.8925 | 0.9970 |
| | Sharing | | 0.0767 | 0.9740 | 0.9060 | 0.9926 |
| | Reporting | | 0.2502 | 0.9264 | 0.8595 | 0.9955 |
| | Student_Type | | 0.0533 | 0.9821 | 0.9497 | 0.9966 |
| | Position_Type | | 0.1386 | 0.9550 | 0.9377 | 0.9921 |
| | Object_Type | | 0.1929 | 0.9401 | 0.8870 | 0.9907 |
| | Pre_K | | 0.1544 | 0.9491 | 0.8395 | 0.9942 |
| | Operating_Status | | 0.2482 | 0.9221 | 0.7657 | 0.9891 |

## Issues raised by first phase

There are some questions raised by the next phase.

- While the metrics scores from train/test on the dataset generally improve from the simpler to more complex models, the competition scores decline. For example, the RandomForestClassifier has aggregate log loss of 0.288 on the test set as measured by the supplied metric and my independent calculation of aggregate log loss but scores the worst of all models submitted to the competition.



- The models from the tutorial make multiple changes at once, so it is difficult to attribute an improvement to one feature if more than one element of feature processing or modeling is changed between models.

## The investigation

To pursue these questions, we rebuilt the models adding one change at a time and scoring the models (with several metrics). In this way we discovered what changes in feature processing and classification were responsible for enhancing or degrading model performance and their relative strength or weakness.

The competition provides a somewhat challenging setting in that the labels of the holdout set are not available. As a result, while it is possible to split the training data into train and test partitions, the only way to ultimately judge competition performance is to submit a prediction for the holdout set to DrivenData. After submission, DrivenData calculates the log loss on the held-out labels. They limit submissions to 3/day, so it is not possible to use standard automated techniques for tuning hyperparameters since these require the availability of the entire data in labeled format.

An additional challenge for automation is inherent in the dataset. Several of the targets are characterized by significant label imbalance. As a result, it is necessary to take care when splitting the data into train and test partitions so that all the labels are sufficiently represented in both splits, so the classifier can be exposed to them. For this reason, DrivenData provides a tool that will produce such splits (multilabel_train_test_split).

This function is used in the tutorial for producing appropriate train/test splits to demonstrate the modeling techniques and we continued with the same approach, submitting predictions at various progress points.

## The Basic Models

The table below shows the models and scores with one change at a time for all the models in the notebook one_at_a_time_part_1.ipynb.

| model | comp. score | agg. log loss | agg. F1 score | comment |
|---:|---:|---:|---:|---:|
| **mod0** | **1.3314** | **1.356** | **0.441** | **numerical features only** |
| mod0_1 | | 1.323 | 0.441 | same as mod0, but use standard scaler before prediction |
| mod0_1a | | 1.295 | 0.454 | scaling + convert total to absolute value |
| mod0_2 | | 1.362 | 0.406 | same as mod0 but use standard scaler and default imputer before prediction |
| **mod1** | **0.7546** | **0.512** | **0.853** | **pipeline, numerical features and text features (fillna with empty string; combine all text columns within row; default count vectorizer)** |
| mod1_1 | | 0.094 | 0.974 | same as mod1 but ignore numerical data |
| **mod1_1_1** | **0.6827** | **0.094** | **0.974** | same as mod1_1; work around n_jobs=-1 bug for faster fit |

As is apparent from scores above, removing the numerical features was a significant step forward. DrivenData's models all include both numerical variables with minimal preprocessing (mean imputation and scaling in some instances). Without the numerical variables, test log loss dropped from 0.512 to 0.094 and F1 went from 0.853 to 0.974, exceptional scores. Unfortunately, while the competition scores improved, they did not track the test score, with the score dropping from 0.7546 to 0.6827, only a modest improvement.

Overfitting was not considered at this point because train and test scores were tracking very well in both mod1 and mod1_1 (e.g.log loss on training set: 0.0874, log loss on test set: 0.0940).

From this model forward, we no longer used the numerical features because they had been shown to be strongly detrimental to model classification performance.

## Adding bigrams and feature selection

The next steps were developed in the notebook, one_at_a_time_part3.ipynb. The table below shows the test log loss, F1 and competition scores where submitted.

| model | comp. score | agg. log loss | agg. F1 score | comment |
|---:|---:|---:|---:|---:|
| mod3_1 | 0.6599 | 0.0573 | 0.982 | text features only, add bigrams |
| mod3_1a | 0.7531 | 0.0593 | 0.982 | text features only, add bigrams, add scaler |
| mod3_2 | | 0.305 | 0.900 | same, but reduce to 300 features using SelectKBest |
| mod3_2a | | 0.0798 | 0.976 | same, but reduce to 3000 features using SelectKBest |
| mod3_2b | | 0.0589 | 0.982 | same, but reduce to 15000 features using SelectKBest |
| mod3_3 | | 0.0580 | 0.982 | same, but reduce to 15000 features using SelectFromModel |
| | | | | |

Mod3_1 adds bigrams to the features. There is a substantial improvement in quality both in the test set and on the competition score with the addition of bigrams. It should be noted that adding bigrams expands the feature set from ~3000 features to ~30,000 features as common words are preceded or followed by many different words and each such pair is a new bigram. Models including the bigrams are consequently slower than those without.

Adding a scaler into the pipeline worsened both the test and competition scores.

We also see that feature selection did not improve the test scores. 300, 3,000, 15,000 features were selected, and models run and measured. Since there was no improvement in test score, these models were not submitted to the competition.

## The most predictive features

The 10 most important features selected by SelectFromModel(RandomForestClassifier()) in order were: `['employee', 'general fund', 'disadvantaged children', 'targeted', 'teacher', 'general', 'services', 'school', 'regular', 'personal']`. The top 10, 100, 1000, and 10000 features account for 10%, 38%, 81% and 99% of feature importance.

It should be noted however that these apply to the overall classification, not particular targets. This may illustrate a disadvantage of the OneVsRest approach to this problem. One might arrive at quite different feature importances if each target were classified by its own classifier. We can see that there is a slight degradation in performance even when reducing features to the best 15000, which should capture 99.8% of feature prediction power.

### Adding Feature Interactions

In Mod4.ipynb we explore adding feature interactions as in the tutorial. The provided function, SparseInteractions, computes all combinations of its input features and outputs a single array with the product of each combination appended with its input.

Here we see the real motivation for reducing the feature set to a smaller set of important variables – the number of combinations of 2 features is proportional to the square of the number of features.

With a reduction to 300 features and their interactions, we have approximately 45,000 new features. As a result, the classification takes somewhat longer than the previous models).

The test scores and the competition score for this model are not as good as previous models, so we ran a test increasing the feature selection to 1600 rather than 300. This was an overnight calculation, with results that still fell short of earlier, simpler models.

| model | comp. score | agg. log loss | agg. F1 score | comment |
|---:|---:|---:|---:|---:|
| Mod4 | 0.8893 | 0.1796 | 0.866 | 300 text features with interactions |
| Mod4_1600 | 0.7774 | 0.1489 | 0.978 | 1600 text features with interactions (slow) |
| twoC_besties | 0.7373 | 0.0565 | 0.976 | 200 best features interactions plus all original features |
| Mod400 | 0.7307 | 0.0577 | 0.976 | 400 best features interactions plus all original features |

In the prior experiments we had noticed that at least half of the features contribute to classification quality (evidenced by degradation in metrics when using 50% of the features). This motivated building a complimentary feature interaction pipeline that would pass through all original features as well as the interactions of a selection of important features (development in my_pipe.ipynb).

We used that tool to construct two models which used the interactions of the 200 and 400 best features and all the original tokens and bigrams. That model had better competition score and log loss than the DrivenData feature interaction model (see mod_best_plus.ipynb).

### The missing piece

At this point, we realized that something was missing. We had several classifiers with excellent test scores that scored well in the competition (top 10) but had not made significant progress even after including feature interactions, a key ingredient in a winning classification strategy according to the tutorial. The best models clearly had enough variance to support the dataset as evidenced by extremely good metrics on the test sets.

The usual indicator of overfitting (good performance on training set and poor performance on test set) was not present. But we reasoned that the models must be overfit (*with respect to the holdout set*) because excellent test performance was not generalizing to the holdout set.

We then began to regularize the most effective models. It should be noted that this is not the standard practice. In most cases, regularization (and other hyper-parameters) are tuned by partitioning into training, validation and test sets, all of which are drawn from the same probability distribution. The model is then tuned with train and validation sets (often using cross-validation within the training set, with the validation set used to optimize regularization). Finally, generalization performance is confirmed by the test set.

Regularization was added to the best performing models using a manual logarithmic search using the scores from submissions as a metric. Values were found that optimized competition performance.

Our best performing model in the competition was mod3_1 with regularization. Mod3_1 is a simple model, using only the text data and its bigrams. The competition score for the orginal unregularized model was 0.6599. With the best regularization, the competition score dropped to 0.5228. This score was 4th best in the competition.

| **metric on test set**<br>**Mod 3_1_reg** | | **competition score** | **log loss** | **accuracy** | **F1** |
|---|---|---|---|---|---|
| **All targets** | **aggregate** | **0.5228** | **0.106** | **0.974** | **0.974** |
| Scores on individual targets | Function | | 0.221 | 0.951 | 0.95 |
| | Use | | 0.082 | 0.982 | 0.982 |
| | Sharing | | 0.054 | 0.984 | 0.984 |
| | Reporting | | 0.107 | 0.980 | 0.979 |
| | Student_Type | | 0.039 | 0.989 | 0.988 |
| | Position_Type | | 0.086 | 0.979 | 0.978 |
| | Object_Type | | 0.126 | 0.968 | 0.968 |
| | Pre_K | | 0.094 | 0.974 | 0.974 |
| | Operating_Status | | 0.143 | 0.964 | 0.963 |

## Recommendations to the client

The best scoring classifier we produced has worst case accuracy (and F1 score) of 95% on the test set and is sufficient to solve the business problem.  It's not possible to say with certainty what accuracy will be on new budgets but given the difference in log loss between the test and holdout sets, it is likely that accuracy will still be very good.  Our experience with a range of models tells us that better than 95% average accuracy/F1 can be expected from models with log loss of 0.52 (our best model's score on the holdout set).

Across all the models, the lowest accuracy was on the target, 'Function'.  This target has 37 possible labels, some of which are very rare.  Human analysts should pay close attention to this target to validate the model's choice of label.

To further focus human analysts, the confusion matrices for each target from the classification can be examined to get a better idea which labels are most likely to be labeled incorrectly.   The figure below shows a normalized confusion matrix for the target 'Function'.  The darker entries off the diagonal show where significant misclassification has occurred.

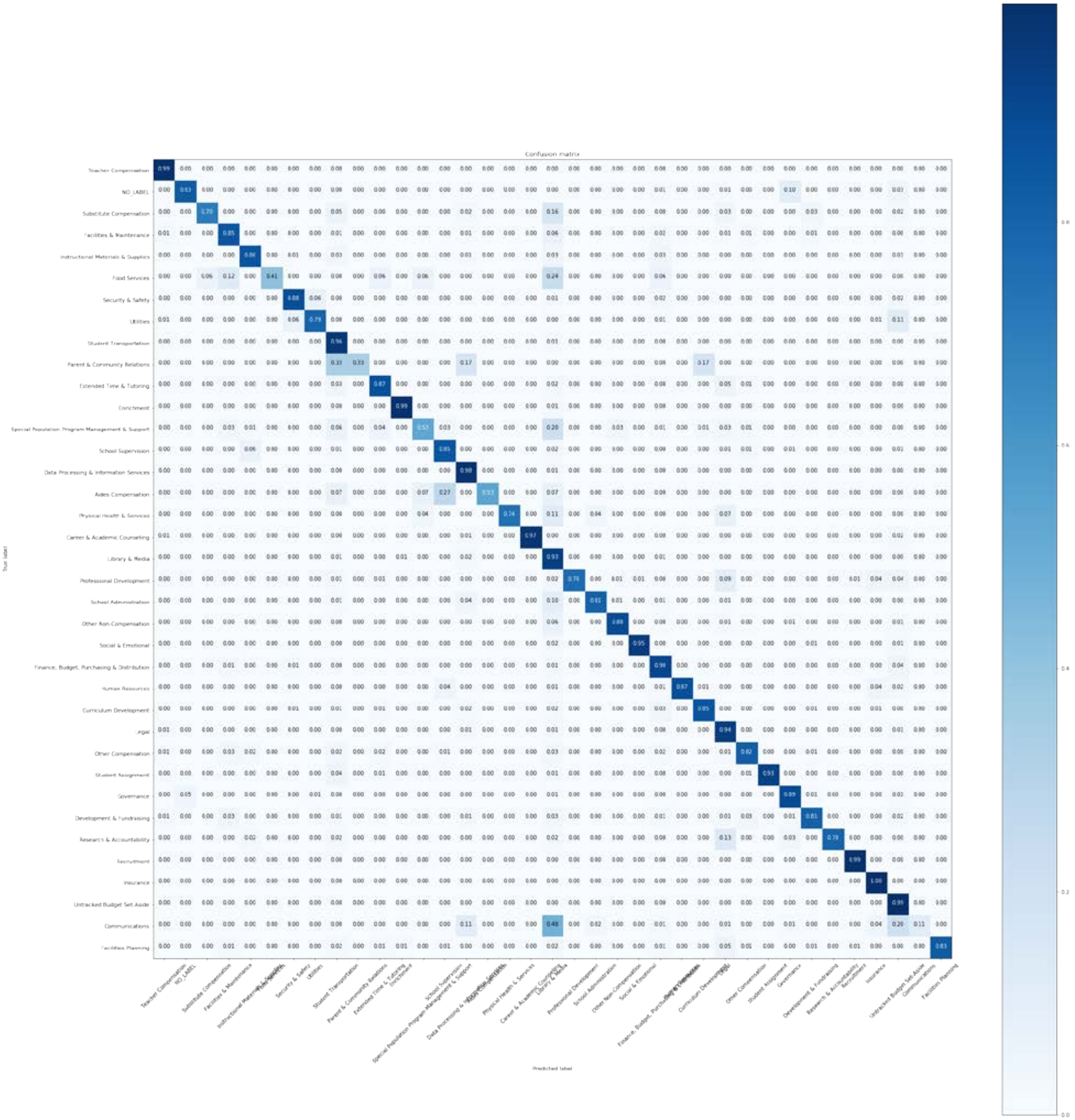


*Figure 1 Confusion matrix for target 'Function'*

## Future Work

There are a number of aspects of this project that could be enhanced. First, it is more appropriate to think of the models from the DrivenData tutorial as a sketch, rather than an implementation of the best scoring model (https://github.com/drivendata/boxplots-for-education-1st-place).

Having examined that code, there are several significant differences:

- Python only, no sklearn, no numpy. Pypy for execution speed.
- Stochastic Gradient Descent with multiple passes over the data, adaptive learning rate and early stopping.
- Feature Engineering
    a. Much richer feature set (subsumes DrivenData approach).
    b. Interaction only for hand selected features, rather than all interactions of automatically selected features.
    c. Hashing of all features.

In particular, the early stopping aspect of the SGD provides regularization on-the-fly within the original classification for better generalization to unseen data (of perhaps different distribution).

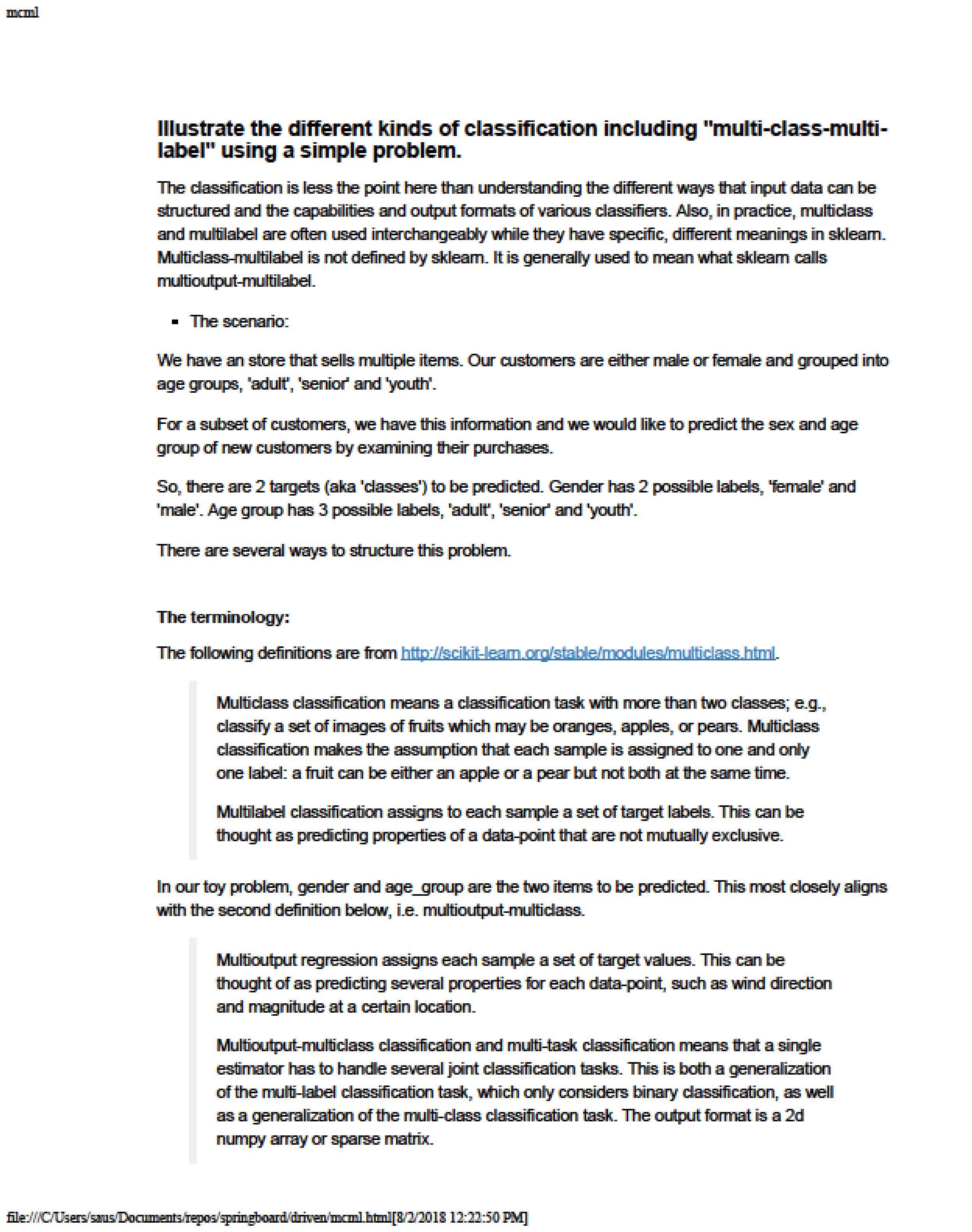
Further, while we investigated stemming and lemmatization and discovered that standard techniques were not helpful, a dictionary approach collapsing similar words and common misspellings into a single feature may be useful. This would reduce the number of features and enhance their predictive value.

It would be interesting to apply different classifiers and assess performance. We did implement a RandomForest model. While this model had excellent label predictions, its probability predictions caused the model to have high log loss in the competition. Log loss severely penalizes high-probability, incorrect predictions. Random Forest was apparently far too sure of its predictions to score highly in the competition.

Other ensemble techniques could be useful as well. We did some rudimentary testing on averaged RandomForest and logistic regression models with encouraging results. However time didn’t permit close tuning of these models.

# Appendix A.  Classification types and input and output formats

Note: this document is available as an notebook at https://github.com/leonkato/springboard/tree/master/driven/docs/mcml.ipynb



## Illustrate the different kinds of classification including "multi-class-multi-label" using a simple problem.

The classification is less the point here than understanding the different ways that input data can be structured and the capabilities and output formats of various classifiers. Also, in practice, multiclass and multilabel are often used interchangeably while they have specific, different meanings in sklearn. Multiclass-multilabel is not defined by sklearn. It is generally used to mean what sklearn calls multioutput-multilabel.

- The scenario:

We have an store that sells multiple items. Our customers are either male or female and grouped into age groups, 'adult', 'senior' and 'youth'.

For a subset of customers, we have this information and we would like to predict the sex and age group of new customers by examining their purchases.

So, there are 2 targets (aka 'classes') to be predicted. Gender has 2 possible labels, 'female' and 'male'. Age group has 3 possible labels, 'adult', 'senior' and 'youth'.

There are several ways to structure this problem.

**The terminology:**

The following definitions are from http://scikit-learn.org/stable/modules/multiclass.html.

> Multiclass classification means a classification task with more than two classes; e.g., classify a set of images of fruits which may be oranges, apples, or pears. Multiclass classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time.
>
> Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive.

In our toy problem, gender and age_group are the two items to be predicted. This most closely aligns with the second definition below, i.e. multioutput-multiclass.

> Multioutput regression assigns each sample a set of target values. This can be thought of as predicting several properties for each data-point, such as wind direction and magnitude at a certain location.
>
> Multioutput-multiclass classification and multi-task classification means that a single estimator has to handle several joint classification tasks. This is both a generalization of the multi-label classification task, which only considers binary classification, as well as a generalization of the multi-class classification task. The output format is a 2d numpy array or sparse matrix.