O*NET and ORS Work Task Classification Report

Author: Rebecca Hu

Last Updated August 1, 2019

Introduction

The Occupational Information Network (O*NET) database is a rich source for information like occupational requirements and worker characteristics. We hope to use the data provided by O*NET to train a machine learning model to classify worker tasks to General Work Activities (GWA). For example, a task might be "*Recommend locations for new facilities or oversee the remodeling or renovating of current facilities.*" and the corresponding GWA label would be "*Making Decisions and Solving Problems*". Our goal is to apply this trained model to the Occupational Requirement Survey (ORS) data. ORS data contains a more robust source of work tasks, but contains messy, unstructured text and does not contain GWA labels for work tasks. Classifying the tasks in the ORS data would then allow us to explore questions like what kinds of jobs are at risk of being automated or moved overseas. This report contains a detailed account of the steps taken in the classification process. The notebooks containing all relevant code are referenced throughout.

Outline

1. O*NET Task Classification
    a. Data Compilation
    b. Exploratory Data Analysis
    c. Pre-Processing
    d. Model
    e. Results
    f. Discussion
2. ORS Task Classification
    a. Exploratory Data Analysis
    b. Pre-processing, Model Selection, and Prediction
    c. Results
    d. Discussion
3. Concerns

O*NET Task Classification

Data Compilation

(See *merge_task_gwa.ipynb* for code)

This section walks through the process of obtaining the data from the O*NET website and merging select files together to create a dataset that contains all of the information we need for our analysis, and no unnecessary fields. The details of this process are specified in the *merge_task_gwa.ipynb* notebook, but here we will outline the steps. First we downloaded all relevant task and work activity data from the O*NET website, This resulted in four different .csv files, all of which had pieces of the information we were looking for. Our goal here was to create a file that contained every work task and every associated GWA with each task. For example, one file contained the task text, but only contained each task's associated Detailed Work Activity ID number, whereas another file would only contain information about the GWA, but not include anything else. We could merge information from each of the four files to eventually create one that only contained the fields we are interested in. The final file, named *onet_tasks_gwas.csv*, is the one we will be using in this report.

Exploratory Data Analysis

(See *onet_eda.ipynb* for code)

The data we are using here is from the Occupational Information Network (O*NET) database of standardized occupation-specific descriptors. The specific file we are working with is a CSV with one column containing the "Task" and another containing that task's assigned "GWA" or General Work Activities (See *Figure 1*).

Task	GWA
Review and analyze legislation, laws, or publi...	Analyzing Data or Information
Review and analyze legislation, laws, or publi...	Provide Consultation and Advice to Others
Direct or coordinate an organization's financi...	Guiding, Directing, and Motivating Subordinates
Confer with board members, organization offici...	Communicating with Supervisors, Peers, or Subo...
Analyze operations to evaluate performance of ...	Analyzing Data or Information
Direct, plan, or implement policies, objective...	Making Decisions and Solving Problems
Direct, plan, or implement policies, objective...	Developing Objectives and Strategies
Direct, plan, or implement policies, objective...	Guiding, Directing, and Motivating Subordinates
Prepare budgets for approval, including those ...	Guiding, Directing, and Motivating Subordinates
Direct or coordinate activities of businesses ...	Guiding, Directing, and Motivating Subordinates

Figure 1: A table illustrating the contents of the CSV. The "Task" column contains a piece of work specific to an occupation, the "GWA" column contains a more generalized classification of the task. A task could be labeled with more than one GWA as seen

in rows 0 and 1.

After removing duplicate rows, the data contains 20223 rows, with 17397 unique tasks and 37 unique GWA. The O*NET website describes 41 unique GWA in the documentation, so there are 4 GWA that have not been assigned to any of the tasks in our data.

The distribution of GWA among tasks are uneven. According to *Figure 2*, the difference between the number of tasks labeled with "*Handling and Moving Objects*" versus the number of tasks labeled with "*Coaching and Developing Others*" is over 2000. Due to how unbalanced the distribution of labels is within the data, we may need to oversample the underrepresented labels and undersample the overrepresented labels when training our prediction model. We might also consider consolidating labels, especially given the similarity between labels like "*Training and Teaching Others*" and "*Coaching and Developing Others*".

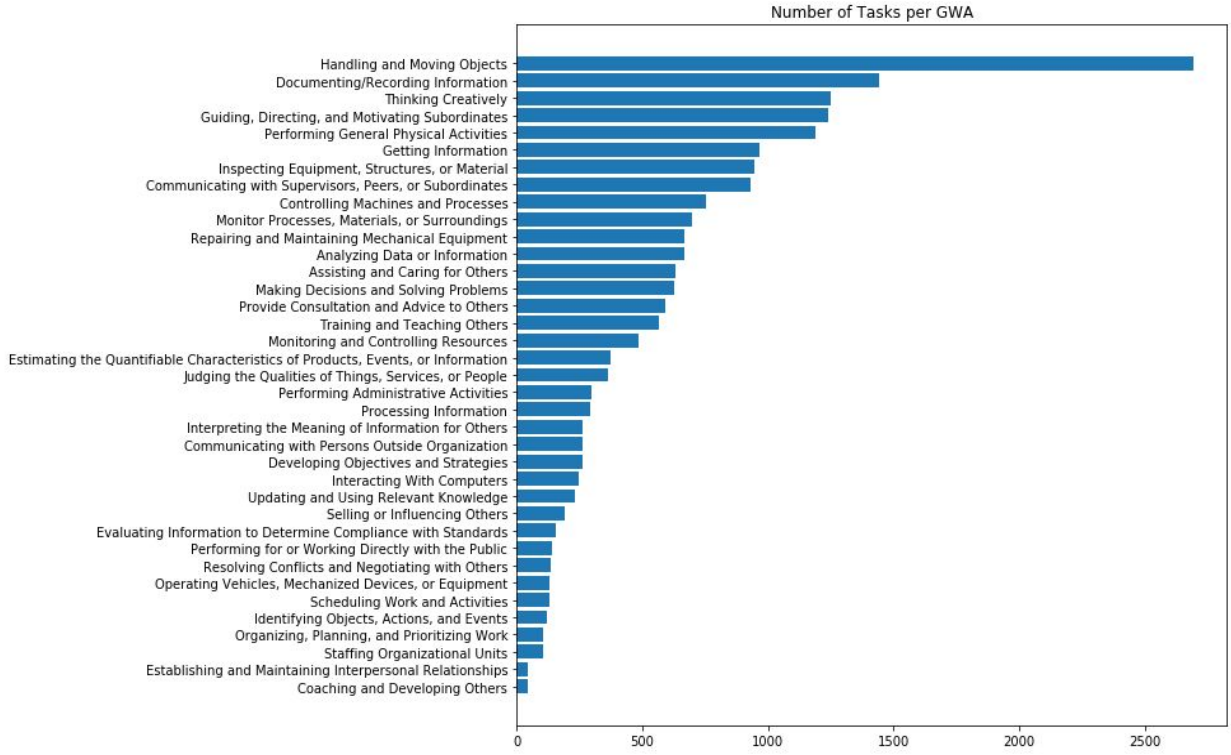

Figure 2: The bar chart illustrates the uneven distribution of GWA labels within the data.

We previously saw that the number of rows did not equal the number of unique tasks; there were more rows of data than there were unique tasks. After removing duplicate rows, that is, rows where both the value in the "Task" column and the value in the "GWA" column are the same, the same inequality holds. This means some tasks have been assigned multiple GWA. This is also evident in *Figure 1*, a task like "*Review and analyze legislation, laws, or public policy and recommend changes to promote or support interests of the general population or special groups.*" involves both "*Analyzing Data or Information*" and "*Provide Consultation and Advice to Others*". *Figure 3* displays the distribution of GWA per Task. Most

tasks have only been assigned to one GWA, but over 2000 tasks have two GWA, and a small proportion have three.

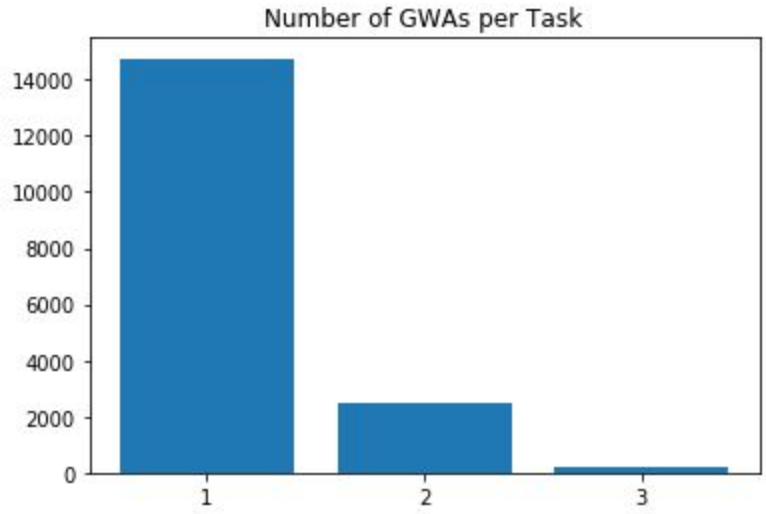

Figure 3: Most tasks only have one GWA, but over 2000 have two, and some have three.

Although the majority of tasks only have one GWA assignment, many tasks have two or three which means our classification problem is both multiclass and multi-label. It is multiclass because we have more than 2 labels, we have 37, and it is multi-label because a task could be assigned more than one label.

Pre-Processing

(See *onet_eda.ipynb* for code)

Before we apply a model to our text data, we must first transform the data to a form that is readable by a machine. This means finding a numeric, quantifiable representation of the input text. To convert the texts to their numeric representations we will use term frequency-inverse document frequency (TF-IDF).
TF-IDF is a statistic that helps us represent how important a word is to a document within a corpus. In our case, we can think of each task as a document and the full collection of tasks is our corpus.

To calculate the TF-IDF for a term i in document j:

$$w_{i,j} = tf_{i,j} \times log(\frac{N}{df_i})$$

$tf_{i,j}$ = number of occurrences of term i in document j
df_i = number of documents containing i
N = total number of documents

However before we even calculate our TF-IDF matrix, we also have to deal with things like capitalization, punctuation, stop words, n-grams, and stemming. We want all of our terms to be in the same case, “Analyze” and “analyze” should be treated as the same word. We also want to get rid of any punctuation. Our input data contain periods and commas, since these do not contribute any meaning to the tasks, we

can remove them. Similarly to punctuation, our text also includes words like “and”, “to”, and “or”. These words also do not contribute much to the meaning of the tasks so we will also remove them.

Next we want to consider the use of n-grams. If we only consider each term to be a single word, we may lose the importance of context and the meaning of phrases. For example, within the phrases “train employees” and “train conductor”, the word “train” has a different meaning which could be lost if we broke the phrases up into single words. Using bigrams (two words) could allow the model to differentiate between assigning “Training and Teaching Others” and “Operating Vehicles, Mechanized Devices, or Equipment”. Last, we want to think about words that appear in various tenses. For example, consider the words “schedule”, “scheduled”, and “scheduling”. We want our model to assign the same TF-IDF value to all of these words, but how can we get it to recognize they are essentially the same word? We can stem the words to a root form so that the model will recognize all the different forms of “schedule” as “schedul” (Note: stemming may not result in a real word). After applying the use of n-grams, the number of words and phrases in our “word bank” multiplies, so stemming also helps reduce some of these unique words.

All of this sounds like a lot but luckily, the TfidfVectorizer() method from the Python package scikit-learn can handle most of these operations for us with just a few additional method arguments (See *Figure 4*).

```
#Remove punctuation & stopwords, use stemming to remove inflections, calculate TF-IDF matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer("english", ignore_stopwords=True)
analyzer = TfidfVectorizer(lowercase = True, stop_words = 'english', ngram_range = (1, 3)).build_analyzer()

def stemmed_words(doc):
    return (stemmer.stem(w) for w in analyzer(doc))

#example of a TF-IDF matrix for 3 document & stemmed words
stem_vectorizer = TfidfVectorizer(analyzer=stemmed_words)
preproc_example = stem_vectorizer.fit_transform(['Schedule Work Activities for Workers',
                                'Scheduled Events to Help Students',
                                'Scheduling Tasks for Employees',
                                'Helping Students Schedule Classes'
                                  ])
print('Stemmed words and n-grams: \n', stem_vectorizer.get_feature_names())
print('\nTF-IDF Matrix: \n', preproc_example.todense())
```

```
Stemmed words and n-grams:
 ['activ', 'activities work', 'class', 'employe', 'event', 'events help', 'events help stud', 'help', 'help stud', 'helping st
ud', 'helping students schedul', 'schedul', 'schedule class', 'schedule work', 'schedule work act', 'scheduled ev', 'scheduled
events help', 'scheduling task', 'scheduling tasks employe', 'student', 'students schedul', 'students schedule class', 'task',
'tasks employe', 'work', 'work act', 'work activities work', 'worker']

TF-IDF Matrix:
 [[0.34768534 0.34768534 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.18143663
  0.         0.34768534 0.34768534 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.34768534 0.34768534 0.34768534 0.34768534]
 [0.         0.         0.         0.         0.36477154 0.36477154
  0.36477154 0.28759007 0.36477154 0.         0.         0.19035292
  0.         0.         0.         0.36477154 0.36477154 0.
  0.         0.28759007 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.43551105 0.         0.
  0.         0.         0.         0.         0.         0.22726773
  0.         0.         0.         0.         0.         0.43551105
  0.43551105 0.         0.         0.         0.43551105 0.43551105
  0.         0.         0.         0.        ]
 [0.         0.         0.36477154 0.         0.         0.
  0.         0.28759007 0.         0.36477154 0.36477154 0.19035292
  0.36477154 0.         0.         0.         0.         0.
  0.         0.28759007 0.36477154 0.36477154 0.         0.
  0.         0.         0.         0.         0.        ]]
```

Figure 4: After applying all the pre-processing transformations to the text, our new features include stemmed words and n-grams (in this case trigrams). Then, we can use TfidfVectorizer() to calculate a TF-IDF matrix for each word/ n-gram.

Model

Now that we have completed some preliminary data investigation and prepared our data to be inserted into a model, we can think about what strategy would best address our multiclass, multi-label classification task.

Algorithm Benchmarking

(See *onet_algorithm_benchmarking.ipynb* for code)

Through benchmarking, we previously found the logistic regression model to be a reliable baseline model with an average F1-score of 51% and accuracies that were consistent across training and testing (69% and 60%, respectively). Logistic Regression can be thought of as an extension of linear regression. In linear regression, a relationship between a dependent variable and one or more explanatory variables is modelled using linear predictor functions, the parameters of which are estimated from the data.With logistic regression, we use the same principles from linear regression but incorporate a logistic function to make the outputs discrete. Other models, like Multilayer Perceptron Classifier and Gradient Boosting would perform better on the training data but failed to deliver consistent results on the test sets. The Linear Support Vector Classifier also performed well, but had an overfitting problem as well. If we can manipulate the parameters of these algorithms to prevent overfitting, they may all be promising options. *Table 1* below summarizes the results of the model benchmarking.

Algorithm Name	Training Accuracy	Test Accuracy	F1 Score
Logistic Regression	0.688	0.602	0.515
Multinomial Naive Bayes	0.512	0.441	0.253
Linear SVC (Support Vector Classifier)	0.771	0.6	0.518
Gradient Boosting	0.765	0.525	0.431
ExtraTrees (Extremely Randomized Trees)	0.879	0.484	0.411
Random Forest	0.879	0.522	0.425
Multi-layer Perceptron Classifier	0.879	0.486	0.439
XGBoost (Extreme Gradient Boosting)	0.643	0.551	0.472

Table 1: Eight different machine learning algorithms were benchmarked and it was determined that a simple logistic regression was the model to continue using due to its consistent performance across the training data and test data.

Label Consolidation

(See *gwa_consolidation.ipynb* for code)

One of the major concerns with this specific task of classifying tasks into GWA is that there are numerous labels, many of which are quite similar. For example, “Handling and Moving Objects” and “General Physical Activities” overlap in many aspects and as such, the logistic regression model often mistakenly classifies these two labels. Combining GWA is one strategy of alleviating this issue. We consolidated the 41 initial GWA into 21 by combining similar labels into one and relabeled the tasks with one of the 21 new labels. After running the logistic regression model on these relabeled, the results of the predictions did not change dramatically, as is summarized in Table 2 below. Additionally due to the subjectivity of how the new labels were decided, we will not continue using this method until a more systematic way of consolidating labels can be determined. Also, the new labels were influenced by which labels were most often mistaken by the logistic regression, and making changes to the model based on the results of the model creates a dangerously biased feedback loop which should be actively avoided.

	Training Accuracy	**Test Accuracy**
41 Unique Labels	0.688	0.602
21 Unique Labels (Consolidated)	0.683	0.558

Table 2: After comparing the results of the logistic regression model before and after label consolidation, we can see very little difference with consolidation. Accuracy actually decreased slightly in this instance.

Inserting Multi-label Data into a Machine Learning Model

(See *onet_logit_model.ipynb* for code)

A challenge I discovered when working with the multi-labelled O*NET tasks was that I had to develop a method to input tasks that had more than one label into the classification models. Typically, one might use sklearn’s *train_test_split()* method to create a training set and test set. This function will (usually) randomly assign the majority of your data to be used for training and a smaller chunk to be used for testing. However, consider the format our data is originally in and consider the task “*Review and analyze legislation, laws, or public policy and recommend changes to promote or support interests of the general population or special groups.”* which is labeled with both *“Analyzing Data or Information”* and *“Provide Consultation and Advice to Others”* (see *Figure 1*). If we used *train_test_split()* on this dataframe, we could end the task labeled as “*Analyzing Data or Information*” in our training set, but the same task labeled *“Provide Consultation and Advice to Others”* in our test set. This would be a problem for our model because theoretically it would see the task, and classify it as *“Analyzing Data or Information”.* Then the prediction for the task in the test set would be incorrect, and the model wouldn’t learn that “*Provide Consultation and Advice to Others*” is also a true label for the task. To address this issue, we will reformat our original dataframe so that each task has each of its labels in one row, producing an “easy-to-read” version of the data (See *Figure 5*). Then we will one-hot encode the dataframe so that the new format has

one row for each task, and one column for each GWA, with the values in the table being a 1 if the task has that particular label, and 0 otherwise (See *Figure 6*). Note: the functions for reformatting the data can be found in the *task_classification_helper_ functions.py* file. The one-hot encoded version of our data addressed our multi-label problem and also helps us with the one-vs-rest technique we will use in the next section.

	GWA 1	GWA 2	GWA 3
Review and analyze legislation, laws, or public policy and recommend changes to promote or support interests of the general population or special groups.	Analyzing Data or Information	Provide Consultation and Advice to Others	None
Direct or coordinate an organization's financial or budget activities to fund operations, maximize investments, or increase efficiency.	Guiding, Directing, and Motivating Subordinates	None	None
Confer with board members, organization officials, or staff members to discuss issues, coordinate activities, or resolve problems.	Communicating with Supervisors, Peers, or Subo...	None	None
Analyze operations to evaluate performance of a company or its staff in meeting objectives or to determine areas of potential cost reduction, program improvement, or policy change.	Analyzing Data or Information	None	None
Direct, plan, or implement policies, objectives, or activities of organizations or businesses to ensure continuing operations, to maximize returns on investments, or to increase productivity.	Making Decisions and Solving Problems	Developing Objectives and Strategies	Guiding, Directing, and Motivating Subordinates

Figure 5: This dataframe is the result of reformatting the original O*NET data into a table with one row per task and each of that tasks GWAs in the same row.

	Analyzing Data or Information	Provide Consultation and Advice to Others	Guiding, Directing, and Motivating Subordinates	Communicating with Supervisors, Peers, or Subordinates	Making Decisions and Solving Problems	Developing Objectives and Strategies	Resolving Conflicts and Negotiating with Others	Documenting/Recording Information	Communicating with Persons Outside Organization
Review and analyze legislation, laws, or public policy and recommend changes to promote or support interests of the general population or special groups.	1	1	0	0	0	0	0	0	0
Direct or coordinate an organization's financial or budget activities to fund operations, maximize investments, or increase efficiency.	0	0	1	0	0	0	0	0	0
Confer with board members, organization officials, or staff members to discuss issues, coordinate activities, or resolve problems.	0	0	0	1	0	0	0	0	0

Figure 6: This dataframe is result of one-hot encoding the “easy-to-read” one. This format of the data will allow us to use the one-vs-rest approach and solve our multi-label problem.

One-vs-rest

(See *onet_one_vs_rest.ipynb* for code)

One-vs-rest is a commonly used strategy for multiclass and multi-label classification problems, it involves training a single classifier for each class where all the tasks labeled with that class would be positively labeled and all tasks labeled with any other label would be negatively labeled. Each classifier for each class is trained, then to predict the GWA of an unseen task, we take the results of the classifiers that predict a positive label with the highest probability. This model is arguably the most straightforward so we first explore using the one-vs-rest approach.

We will begin by testing the one-vs-rest approach on various classes with different sizes to determine if this strategy is viable to apply to the entire classification task.

For multiple classes, we will start by separating out one particular class from the rest of the tasks with other labels. Then we'll undersample the "rest" group and over-represent the target class in the training data to create a more balanced dataset. Then we will train the model as a binary class problem rather than a multiclass problem, labeling a 1 if the task belongs to the targeted class, 0 otherwise.

After evaluating metrics from a few targeted classes, we see that generally using this approach on individual classes yields low error rates and high F1 scores. Given the results of these tests, we can presume one-vs-rest will continue to be an effective strategy for our final model. The results of a one-vs-rest approach for the GWA "Handling and Moving Objects" is demonstrated below in Figure 7.

```
# baseline preprocessing + sampling
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support

hamo_sample = df[df['GWA'] == 'Handling and Moving Objects'].sample(2000)
nonhamo_df = df[df['GWA'] != 'Handling and Moving Objects'].sample(3000)
nonhamo_df['GWA'] = ['Not Handling and Moving Objects'] * 3000
equal_hamo_data = pd.concat([hamo_sample, nonhamo_df])
X_train, X_test, y_train, y_test = train_test_split(equal_hamo_data['Task'],
                                                    equal_hamo_data['GWA'],
                                                    test_size = 0.10,
                                                    shuffle = True)
```

```
# fit the model and make predictions
logit_pipe.fit(X_train, y_train)

train_predicted = logit_pipe.predict(X_train)
test_predicted = logit_pipe.predict(X_test)

train_p, train_r, train_f1, train_s = precision_recall_fscore_support(y_train, train_predicted, labels = y_train.unique())
```

```
print('Training Error: ', round(sum(train_predicted != y_train)/len(y_train), 3))
print('Precision: ', train_p)
print('Recall: ', train_r)
print('F1-Score: ', train_f1)
print('**********************************************************')
print('Test Error: ', round(sum(test_predicted != y_test)/len(y_test), 3))
test_p, test_r, test_f1, test_s = precision_recall_fscore_support(y_test, test_predicted, labels = y_train.unique())
print('Precision: ', test_p)
print('Recall: ', test_r)
print('F1-Score: ', test_f1)
#results = pd.DataFrame({'Task': X_test, 'Actual': y_test, 'Predicted': test_predicted})
#results[results['Actual'] != results['Predicted']].head(10)
```

```
Training Error:  0.056
Precision:  [0.95390898 0.93754487]
Recall:  [0.90381426 0.97064288]
F1-Score:  [0.9281862  0.95380683]
**********************************************************
Test Error:  0.162
Precision:  [0.78350515 0.87254902]
Recall:  [0.79581152 0.86407767]
F1-Score:  [0.78961039 0.86829268]
```

Figure 7: After using sampling methods to create a balanced dataset of tasks that are labelled “Handling and Moving Objects” and tasks not labelled “Handling and Moving Objects”, we then separate the dataset into a training set and test set, and proceed to train a basic logistic model on the now binary task.

Final Model

(See *onet_logit_model.ipynb* for code)

To create the final model, we will combine the logistic regression algorithm with the one-vs-rest strategy to predict multiple labels from multiple classes for each task. The process involves first separating our data into two sets, a training set and test set. Then, for each unique GWA, we will treat that particular GWA as the target class, implement our sampling technique to produce a balanced dataset, and then train a one-vs-rest logistic regression model on this balanced dataset and save the trained model. After we have done this for each unique GWA, we will use each trained model to predict binary labels (either a 1 or a 0) on the test set. This combination of multiple trained models (one for each unique GWA), serves as the final model. For every individual model that predicts a 1, the final model will predict that target class as a GWA for that task. This allows the model to predict both multiple classes and multiple labels for each task.

```python
def train_classifier(gwa, data):
    '''
    oversamples target gwa, undersamples rest, then fits models based on balanced dataset

    Parameters:
    -----------
    gwa: string
        the name of the target gwa
    data: pd.DataFrame
        a dataframe containing the one-hot encoded training data (index holds task, one column per GWA, column headers are GWA)

    Returns: sklearn.pipeline.Pipeline
    -----------
    '''
    target_size = str(data[data[gwa] == 1].shape[0])

    if int(target_size) < 100: #If the class is small (less than 100)
        pos_sample_size = int(target_size)
        neg_sample_size = int(target_size) + 100

    pos_sample_size = int(target_size[0] + ('0' * (len(target_size) - 1)))
    neg_sample_size = pos_sample_size + int('1' +'0' * (len(target_size) - 1))

    #Oversample target class and undersample rest
    pos_sample = data[data[gwa] == 1][[gwa]].sample(n = pos_sample_size)
    neg_sample = data[data[gwa] == 0][[gwa]].sample(n = neg_sample_size)

    balanced = pd.concat([pos_sample, neg_sample]).sample(frac=1)

    return lr_pipe.fit(balanced.index, balanced[gwa])
```

```python
# Train, save, and predict on training and test data for each the 41 classifiers (Really 37)
from sklearn.externals import joblib

train_preds = {}
test_preds = {}
for count, gwa in enumerate(df.GWA.unique()):
    model = train_classifier(gwa, train)
    fn = './trained_classifiers/' + str(gwa).replace(' ', '_').replace('/', '_').replace(',', '') +'.pkl'
    joblib.dump(model, filename = (fn))  #<--- uncomment this line to save the models
    train_preds[gwa] = model.predict(train.index)
    test_preds[gwa] = model.predict(test.index)
```

Figure 8: train_classifer() is a method that takes a GWA and the training data, then oversamples the GWA class and undersamples the rest of the tasks to create a balanced dataset. It returns a fitted sklearn pipeline object.

Results

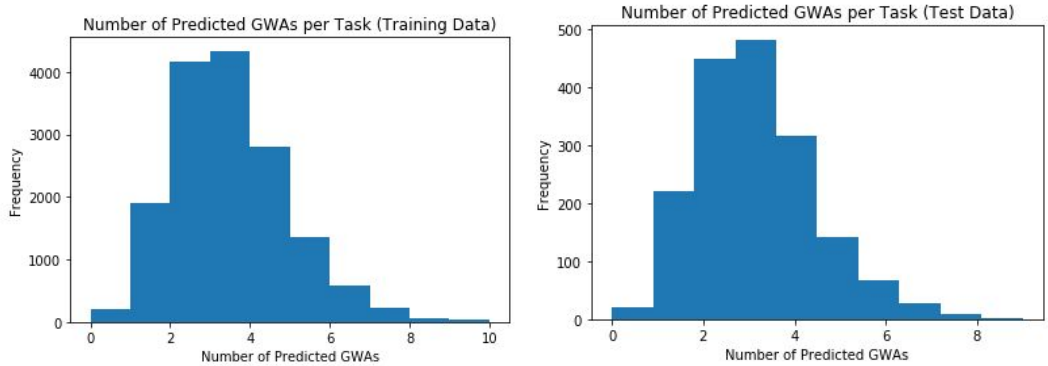

Figure 9 and 10: On average the final model predicts 3 or 4 GWA per task on the training data and the test data.

Something that becomes evident after prediction is that our multi-label classifier overassigns labels. In our training data, we saw that the large majority of tasks only had one label (*Figure 3*). However, the multi-label classifier, on average, assigns three labels to each task (*Figure 9*, *Figure 10*). This effect is also evident in the calculated recall metric. Both the training and test predictions resulted in relatively high recall scores, 0.86 and 0.77, respectively, but low precision scores, 0.33 and 0.29.

	Precision	**Recall**	**F1**
Training Set	0.33481233016304346	0.8663004669046965	0.48296559432849995
Test Set	0.2979047619047619	0.7750247770069376	0.43037974683544306

Table 3: Quantitative metrics evaluating the performance of the final model on the training set and test set.

Limiting Prediction Frequency

(See *onet_logit_model.ipynb* for code)

Given the nature of the model to over-predict, we tested some techniques to limit the number of labels predicted by the final model. We tried two techniques: 1.) only keeping predictions whose prediction probability are greater than a certain threshold value 2.) keeping the three predictions with the highest probability predictions. We found that both techniques resulted in a dramatic decrease in recall scores, even with conservative parameters. Because of this, we needed to take a step back to reexamine the overarching goal of our O*NET classification task. For our purposes, it would be more beneficial to have more “false positives” than to have too many “false negatives”, especially since we have determined that many of our “false positives” are still good labels for the data. Thus we abandoned trying to limit the number of GWA predicted for each task to favor recall over precision due to the nature of our project.

Discussion

Our end result is a model that is relatively good at not making false negative predictions; it seldom misses assigning a GWA. The model does this at the expense of making many false positive predictions, however if you take a closer look at these false positives, we see that they are actually sound predictions given that the “ground truth” GWA assigned actually overlap in their meanings. For example, in Figure 9 below, the task “Monitor clients’ progress to determine whether changes in rehabilitation plans are needed” is correctly predicted to be “Monitor Processes, Materials, or Surroundings”, but the other labels that are predicted like “Making Decisions and Problem Solving” are also good labels for that task. Because of this, it is difficult to grasp exactly how well the model is doing with quantitative metrics, but we can presume it is better than the calculated precision metrics imply. Now that we have a model that can predict GWA for the O*NET data, we will attempt to use this model to label Occupational Requirements Survey (ORS) task data, which lacks labels, with GWA.

Task	Actual Labels			Predicted Label(s)					
Develop instructional materials and conduct in-service and community-based educational programs.	Training and Teaching Others	0	0	Provide Consultation and Advice to Others	Thinking Creatively	Training and Teaching Others	0	0	0
Monitor clients' progress to determine whether changes in rehabilitation plans are needed.	Monitor Processes, Materials, or Surroundings	0	0	Analyzing Data or Information	Communicating with Supervisors, Peers, or Subo...	Making Decisions and Solving Problems	Judging the Qualities of Things, Services, or ...	Getting Information	Monitor Processes, Materials, or Surroundings
Maintain or arrange for maintenance of fitness equipment or facilities.	Repairing and Maintaining Mechanical Equipment	0	0	Scheduling Work and Activities	Monitoring and Controlling Resources	Repairing and Maintaining Mechanical Equipment	0	0	0
Skim or pour dross, slag, or impurities from molten metal, using ladles, rakes, hoes, spatulas, or spoons.	Handling and Moving Objects	0	0	Handling and Moving Objects	0	0	0	0	0
Compound or process ingredients or dyes, according to formulas.	Performing General Physical Activities	0	0	0	0	0	0	0	0
Observe yard traffic to determine tracks available to accommodate inbound and outbound traffic.	Monitor Processes, Materials, or Surroundings	0	0	Getting Information	Monitor Processes, Materials, or Surroundings	0	0	0	0

Figure 11: Quantitative metrics tell us the final model is not performing well with respect to precision, but looking at the predicted labels, it can be argued that many of the predicted labels also fit the description of the task.

ORS Task Classification

Exploratory Data Analysis

Data Cleaning Process

(Results can be found in *ors_data_cleaned.xlsx*)

After an initial review of the ORS data, it was determined before any analysis could begin, the data would need to be cleaned. The occupational task descriptions contained unnecessary, repetitive strings like “Critical Job Function: “ or notes by the survey conductor, as well as numbers, punctuation, and misspellings. Segments of the text that contained no useful information for the model were parsed out and removed by hand. Numbers, punctuation marks, and stops words (words including “the”, “and”, “or”) were left in because they could be easily dealt with python packages in the pre-processing step.

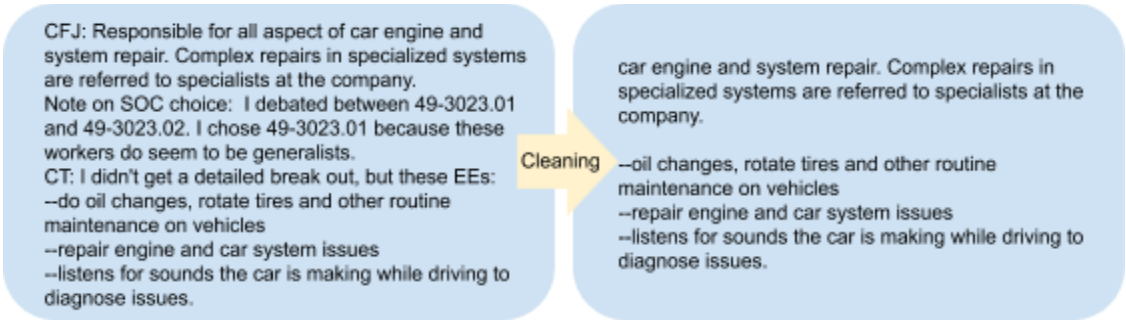

Figure 1: Unnecessary strings like “CFJ:” (Critical Job Function), “CT: “ (Critical Task), or notes taken by the survey conductor were removed.

Pre-processing, Model Selection, and Prediction

(See *ors_classification.ipynb* for code)

The ORS data will be preprocessed the same way the O*NET data was preprocessed before being fed into the model. The model used will be the final model that was pre-trained on O*NET tasks. The preprocessing step involves using sklearn’s TfidfVectorizer to calculate a TF-IDF matrix for n-grams in the text, removing stop words and non-alphabetical words, and using nltk’s SnowballStemmer to stem the words. The final model was built from the one-vs-all framework of combining multiple models that were all individually trained logistic regression classifiers. Prediction just involved passing the new ORS data into the pre-trained, pickled models.

Results

The ORS data contains no pre-defined GWA labels so it is difficult to quantify how well the model is performing on the ORS data with a single metric. One way we can gauge how the model performing is by examining the SOC codes associated with each task. However because there are many unique SOC codes and we have to check each code by hand, this still isn’t ideal. When we examine the data, we see

similar results as with the O*NET data. Typically the model will predict labels that is reasonable, but it might also predict additional ones that are inaccurate.

Discussion

In comparing the results of the model on the O*NET versus the ORS data, it appears the model tends to predict more unrelated labels on the ORS data but this is expected. The O*NET tasks were straightforward sentences that followed one particular format. The ORS tasks were large blocks of raw text that contained multiple tasks that were written in different styles with typos. The ORS tasks contained a wider range of occupations, so the model struggled when it came across words it had not encountered before in the O*NET dataset. The greatest improvement that could be made to the model would involve training the model on a more robust O*NET dataset that contained more of the domain specific words that appear in ORS. We used the 709 version of the ORS data, however we also tried running the model with the 721 ORS data which I believe helped with typos and not having large blocks of text. The 721 ORS data and the O*NET data are more similar, but the O*NET data still lacks all the domain-specific text needed to predict well on the 721 ORS data, and the ORS task data is still written too generally.

Concerns

O*NET Labels

The O*NET data does not appear to be labeled perfectly. Sometimes they are labeled in a way that I would disagree with completely, other times tasks might be labeled with one label that doesn't entirely encompass the full meaning of the task. The GWA that each task belongs is mildly subjective to begin with (i.e. "thinking creatively"), also each GWA is not mutually exclusive (as we know since it is a multi-class problem), but I think each task should have been labeled with more GWAs to encompass the entire task

ORS versus O*NET

The ORS and O*NET data are quite different, this could violate the assumption that the data the model is trained on and the data the model is tested on are drawn from the same Identically Independent Distribution (IID assumption). The way the O*NET data is written is in a particular style, whereas the ORS data is formatted differently and written with different verb tenses.

Missing ORS Labels

It was very difficult to gauge how well the model performed on the ORS data with no predefined labels. Unfortunately this was just the nature of the task.