Classifying Job Post Text Documents with the Malaysian Standard Classification of Occupations

Michael Sommer

Department of Information and Computer Science

University of Hawaii at Manoa

Honolulu, U.S.A

mpsommer@hawaii.edu

• Section 1 - Introduction

The prevailing goal of an emerging market is to become a developed, high income earning nation. Such goals are partially realized by analyzing trends in industries to ensure that in demand occupations are being fulfilled by the markets work force [9]. Malaysia is one such emerging market intent on reaching this goal. Understanding occupational trends and in demand skills provides Malaysia much needed insight for the development of their economy. A method for determining in demand occupations and their corresponding critical skills involves categorizing job post text documents (such as job posts from employment websites) with some predefined occupation category, and analyzing the resulting dataset. An occupation category structure developed by the Malaysian Ministry of Human Resources that can be used in categorizing job post documents is the *Malaysian Standard Classification of Occupations* (MASCO) [9]. MASCO is an occupational classification structure with numerical labels and corresponding occupation documents. Assigning MASCO labels to job post documents is an instance of the general text classification problem, which is simply the task of assigning a document to one or more classes or categories. Classifying text documents can be done manually and automatically. Common approaches for automated methods of text classification include rule based and machine learning systems. This project develops a machine learning system to classify job post text documents with MASCO labels.

Text classification in machine learning is a supervised learning process where an algorithm classifies documents according to some predefined set of classes. A common model used is the bag-of-words model [7]. This model is often used in natural language processing for things like labeling news articles from a set of predefined topics, or detecting spam email. The bag-of-words model is explored in this project with the implementation of a machine learning pipeline that assigns MASCO labels to job-post documents. The following report details the process of establishing training datasets with labels from the MASCO structure, the implementation of a machine learning pipeline, and analysis of obtained results.

Contributions

The objective of our project is to develop an application that can optimally label job post documents according to the MASCO structure. The primary factor that makes our objective challenging is the lack of a quality dataset of job post documents labeled according to the MASCO structure. Various techniques are used to overcome this challenge such as mapping the labels from the supplied dataset to MASCO labels, manually labeling a subset of the data with MASCO labels, and leveraging the text occupations from different versions of the MASCO structure to use as training data.

Paper Organization

The rest of the paper is organized as follows. Section 2 is an in-depth overview of the entire project. Section 3 introduces related work. Section 4 explains and analyzes the various approaches used to achieve our objective. Section 5 concludes the paper.

• Section 2 – Overview

Text Classification Problem

The text classification problem is a specific instance of the general document classification problem that is relative to information and computer science. The focus in computer science is an algorithmic approach to assigning a label from a set of predefined labels to a given text document. More formally, given a document $d \in D$, where $D = \{d_1, d_2, \ldots d_n\}$ is a set of n documents, and a fixed set of m classes $C = \{c_1, c_2, \ldots c_m\}$, text classification consists of mapping each d_i $(1 \leq i \leq n)$ document to a class $c \in C$. In machine learning, this is achieved through a supervised learning approach for a given classifying algorithm. The learning algorithm Γ takes a training set as input, and outputs a learned classification function γ which is used to classify documents beyond the training set. In our application, the set of classes is the MASCO 2008 unit group 4-digit labels, and the training documents are job post documents from a popular Malaysian employment website.

Malaysian Standard Classification of Occupations

MASCO is a classification structure for organizing information about labor and jobs for the country of Malaysia [9]. MASCO serves as the national benchmark for occupational classification and is customized to Malaysia's economy. MASCO was adapted from the International Standard Classification of Occupations (ISCO), which is a structure that serves the same purpose as MASCO, but at an international level. Each numerical MASCO label corresponds to a text occupation. There are several versions of MASCO but the two used in our project are MASCO 2008 and MASCO 2013.

MASCO 2008 consist of 5 levels: Major Group (1-digit), Sub-Major Group (2-digit), Minor Group (3-digit), Unit Group (4-digit), and Small Unit Group (5-digit). The number of labels for each group is 10, 44, 144, 489, and 4804 respectively [9]. An example label from the unit group is label 2632 with the corresponding occupation: "Sociologists, Anthropologists and Related Professionals." The MASCO 2008 4-digit unit group labels are used for classifying documents in this project and in future sections of this report, the term *MASCO labels* refer to the MASCO 2008 unit group.

Although MASCO 2013 labels were not explicitly used as labels for classifying documents, the corresponding occupations at the 6-digit level proved to be informative when training classifiers, therefore, these labels (and corresponding occupations) were mapped to the MASCO 2008 unit group labels. This was done by collapsing all 6-digit 2013 occupations into the corresponding 4-digit unit group, and then mapping the 2013 unit group to the 2008 unit group. An example of a finer grained occupation at the 6-digit level is 2512-14 with the corresponding occupation "java developer".

Provided Dataset

The dataset for this project is from a well-known Malaysian employment website. The raw data was collected over a period of six months (January to June) in 2016 and consists of 115,367 samples. Each sample consists of a unique identifier, a job title, sub specialization, listed role, ISCO 2008 code, and confidence score. The unique identifier is for indexing and is not used in this project. The job title is the title as it appears on the website. The sub specialization corresponds to the industry category for the job, and the listed role is the specific role of the job. The ISCO 2008 label and confidence score were appended to each job post document and were generated from an online application CASCOT. An example of a sample from the dataset is: *Senior Graphics Designer, Arts/Creative Design Jobs, Graphic Designer, 2166, 67,* where *Senior Graphics Designer* is the job title, *Arts/Creative Design Jobs* is the sub specialization, *Graphic Designer* is the listed role, *2166* is the ISCO label, and *67* is the confidence score.

CASCOT

CASCOT is an acronym standing for Computer Assisted Structured Coding Tool. CASCOT is an application that labels job title text documents to various occupational structures. Labels in our dataset come from CASCOT International, a version of the application that is capable of labeling job title documents with the ISCO structure. According to the CASCOT website, "The performance of CASCOT has been compared to a selection of high quality manually coded data. The overall results show that 80% of records receive a score greater than 40 and of these, 80% are matched to manually coded data." [6]

Training sets from mapping ISCO labels to MASCO

Because the provided dataset contained ISCO labels and our objective was to assign text documents with MASCO labels. We needed a training set for our machine learning algorithms that contained MASCO labels. To obtain such a training set, we converted the provided ISCO labels to MASCO labels by first manually mapping the ISCO labels to the appropriate MASCO labels and then creating a python script that converted all the provided ISCO labels to the corresponding MASCO labels in the map. Two training sets were created with this method. Set 1 utilizes all 115,367 samples regardless of the CASCOT confidence score. 349 unique MASCO labels are present in set 1. Set 2 contains only the samples that have a confidence score of 80% or greater. This reduced the size of the dataset to 2096 samples with 131 unique MASCO labels.

A problem with mapping ISCO labels to MASCO labels is that there are 53 less ISCO labels than MASCO. Figuring out which ISCO label maps to multiple MASCO labels and then resolving samples that have these ISCO labels requires manual inspection that was impractical for our project. Also, as explained in the background section on CASCOT, the accuracy of CASCOT is 64% on high quality manually coded data. This is not very promising and less than two percent of the provided ISCO labels have a confidence score greater than or equal to 80. Other problems include the fact that we do not know the underlying algorithms for CASCOT, and CASCOT was not trained with our dataset. Because of these reasons, the two training sets created from the mapping are suspect, never the less we use them as input into our machine learning pipeline and analyze and contrast the results with other training sets.

Creation of ground truth training set

Due to the lack of confidence in the two training sets from mapping ISCO labels to MASCO labels, it was necessary to construct a ground truth training set that was manually labeled with MASCO labels. This was done by extracting a random subset of the provided dataset and manually assigning a MASCO label to each sample. For each sample, the job title, sub specialization, and listed role was taken into consideration when labeling. In total, 994 samples were labeled with MASCO labels and 143 unique MASCO labels are present. In this report, the term *ground truth* refers to this manually labeled set of data.

MASCO occupation variations

To increase the size of the ground truth training set and ensure all MASCO labels were represented at least once in the training data, five MASCO occupation variation datasets were constructed with some combination of MASCO 2008 unit group occupations, MASCO 2008 unit group job descriptions, and MASCO 2013 6-digit occupations. Variation 1 only contains the MASCO 2008 unit group occupations for a given label. Variation 2 contains the MASCO 2008 unit group occupations as well as their corresponding descriptions (provided text that described the occupation) for a label. Variation 3 contains the MASCO 2013 6-digit occupations mapped to a MASCO 2008 labels. Variation 4 contains the MASCO 2008 unit group occupations as well as the MASCO 2013 6-digit occupations mapped to MASCO 2008 labels. Variation 5 contains the MASCO 2008 unit group occupations with the corresponding descriptions and the MASCO 2013 6-digit occupations mapped to MASCO 2008 labels. Variation 5 therefore, contains the most robust text document (bag of words) corresponding to a given MASCO label. All the variations consist of 489 samples and each sample represents one unique MASCO label. These variations are referred to as *MASCO occupation variations* in this report.

Machine Learning Pipeline

To train machine learning classifiers with our various training sets, a machine learning pipeline was constructed that preprocessed the job documents, extracted features, trained classifiers, and optimized the hyper parameters for each classifier. The pipeline was developed using Python 3.5 and scikit-learn.

Step 1 - Data preprocessing

Step one in our machine learning pipeline involves normalizing the data. The normalization performed on the training sets consists of removing all non-alphabetic characters, converting each character to lowercase, and removing all redundant whitespace in the document. Besides this, techniques were developed for spelling corrections, changing American English spelling to British English, compiling a stop word list specific to Malaysian job titles, expanding acronyms, and lemmatizing all the tokens in each text document.

Correcting Misspelled Words

Using the Wordnet synset from the NLTK corpus, misspelled words in the dataset were extracted and manually corrected. Wordnet is a lexical database for the English language, and synset provides a list of synonyms for a given word. To extract misspelled words from the ground truth, each job title in the dataset was traversed and a list of all word tokens that did not have a synonym was compiled. Words that were spelled correctly but had no synonyms were left untouched, incorrectly spelled words were corrected, and a separate list of acronyms was extracted. In total 737 misspelled words were corrected.

Acronym Expansion

The extracted acronym list previously mentioned consists of 945 entries. 296 acronyms relevant to job postings were successfully expanded manually. An example is CCNA, which expands to Cisco Certified Network Associate.

Converting American English Spellings to British English Spellings

MASCO uses the British spellings of English words while the job titles often use the American spelling. Words like *fibre* and *jewellery* (the British equivalent of *fiber* and *jewelry* in American English) are used in MASCO labels. To convert American English spelled words that appear as the English spelling in the MASCO labels, a list of British spelled words from the MASCO labels corresponding occupations was compiled and English spelled words in a job title that matched a word in the list were converted to the British spelling.

Stop Words

Stop words are words that contain no important significance in classifying a text document and are commonly filtered out in preprocessing. Stop words were filtered out of our training sets from a stop word list containing common English words like *the* and *an* as well as Malaysian locations like *Kuala Lumpur*.

Lemmatization

A significant step in preprocessing that improved the training data in our application comes from Lemmatizing. Lemmatization groups inflectional forms into a common base. With lemmatization words like *management* and *manager* are reduced to *manage*. The lemmatization used in our application comes from the NLTK corpus reader Wordnet. In general, lemmatization increased the accuracy of classifiers by 2%.

The effects of these preprocessing techniques were tested using our ground truth training set. Classifiers were first trained on a ground truth that was only minimally preprocessed (characters converted to lowercase, non-alphabetical characters removed). This served as a baseline to compare other preprocessing techniques. Then classifiers were trained with versions of the ground truth that only corrected spelling, only expanded acronyms, only removed stop words, and only lemmatized the words. Table 1 displays results of training classifiers with these isolated preprocessing techniques. Each column displays the accuracy of a given preprocessing technique. Accuracy in this table is ensemble accuracy (average of the accuracy from the five classifiers). When comparing each technique to the baseline minimal preprocessing, the table shows no significant gains in accuracy except for lemmatization. This is believed to be because of the small size of the training set (994 samples).

	Minimal Preprocessing	Corrected Spelling	Acronyms Expanded	Removal of Stop Words	Lemmatized words
Ensemble Accuracy	0.636	0.636	0.635	0.637	0.659

Table 1*: The columns represent the different data preprocessing techniques in isolation. The row displays the resulting accuracy for each technique. All techniques used minimal preprocessing (characters to lowercase, etc.) therefore, the Minimal Preprocessing column is the benchmark to compare against all others. Ensemble accuracy is the average accuracy between the five text classifiers.*

Step 2 - Vectorization and feature extraction

Step two in our pipeline consists of vectorization and feature extraction to form a numerical representation of the text documents. In the bag-of-words approach, vectorization must occur to prepare data as input for machine learning classifiers. Vectorization is the process of turning text documents into feature vectors. This process begins when a text document is split (by whitespace, tabs, etc.) and each word is represented as an integer term. The way the terms are stored in a feature vector determines how the underlying text document is represented, and is referred to as the feature selection. One feature selection representation is done by counting the occurrences of each term for a text document and storing these term frequencies (TF) in a feature vector. A corpus of documents can then be treated as a matrix where each row is a feature vector and each column represents a count of a terms. The draw back to using feature vectors that contain term frequencies is that all terms are considered equally important in the feature vector, when in fact, certain terms contribute very little to the relevance of a given text document. These are terms that occur often in the dataset like “the”. To combat this issue, the mechanism known as TF-IDF is used. TF-IDF stands for term frequency multiplied by inverse document frequency. Inverse document frequency (IDF) is the log of the division of the total number of documents in a collection by the number of documents in a collection that contain the term t. The IDF acts as a weight on a TF and resolves the above issue by scaling down frequently occurring terms.

The vectorization and feature extraction process in our pipeline was handled with CountVectorizer and TfidfTransformer from the sklearn.feature_extraction.text package. CountVectorizer builds a feature vector of term frequencies and supports n-grams of words. A grid search of ranges for n-grams resulted in the use of unigrams and bigrams for our project. This means that feature vectors included term frequencies for unigrams as well as bigrams. To combat the issues previously mentioned with term frequencies, TfidfTransformer was used to compute the TF-IDF representation of the feature vector. TF-IDF was the feature that represented a text document in our training sets. The parameters passed to TfidfTransformer are *norm*, *use_idf*, *smooth_idf*, and *sublinear_tf*. *norm* allows for different ways to normalize the term vectors, either L1, L2, or none. *use_idf* is a Boolean that enables inverse document reweighting. *smooth_idf* is a Boolean that when set to true, prevents zero divisions by adding 1 to each term (LaPlace smoothing). *sublinear_tf* is a Boolean that if set to true, applies sublinear tf scaling (replace tf with 1 + log(tf)). For our application, grid search resulted in setting the *norm* to L2, *use_idf* and *smooth_idf* to true and *sublinear_tf* to false. The use of bi-grams with TF-IDF features resulted in a general increase in accuracy of about 1.7% compared to only using unigrams with our various training sets.

Step 3 – Training text classifiers

Step three in our pipeline consists of splitting the data into training and test sets, training the machine learning classifiers, and computing the accuracy of each classifier.

Creating training and test sets

Each of the datasets used to train machine learning classifiers in our project were divided into a training set of 90% of the data and a test set with the remaining 10%. The 90-10 split allows for maximal training data as well as an appropriately sized test set. One caveat is that when MASCO occupations were appended to a given input, they were only used in the training set, resulting in an increase of 489 MASCO samples for that training set.

Training classifiers

Each classifier was trained using K-folds cross validation. K was set to ten, so each classifier was trained 10 times where the test set changed by a K-fold (10% of the data excluding MASCO occupations) each time. The resulting accuracy for all 10 iterations was then averaged to obtain the cross validated accuracy for a classifier. Table 2 illustrates this process:

← **Dataset** →

Iteration1	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Iteration2	🟦	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Iteration3	🟦	🟦	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Iteration4	🟦	🟦	🟦	🟥	🟦	🟦	🟦	🟦	🟦	🟦
Iteration5	🟦	🟦	🟦	🟦	🟥	🟦	🟦	🟦	🟦	🟦
Iteration6	🟦	🟦	🟦	🟦	🟦	🟥	🟦	🟦	🟦	🟦
Iteration7	🟦	🟦	🟦	🟦	🟦	🟦	🟥	🟦	🟦	🟦
Iteration8	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟥	🟦	🟦
Iteration9	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟥	🟦
Iteratino10	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟥

🟦 = training set 🟥 = test set

Table 2: This table represents the division of the training and test sets for each of the learning iterations. Blue cells represent the 90% training set for a given classifier training iteration. Red cells represent the 10% test set.

Classifiers used in machine learning pipeline

The classifiers trained in our machine learning pipeline were Multinomial Naïve Bayes, k-Nearest Neighbor, Linear Support Vector Classifier, Random Forest, and Stochastic Gradient Descent. These classifiers have been heavily researched and have a proven track record of working with well with text data [2][4][7][8][10].

Multinomial Naïve Bayes (MNB) is a variation of Naïve Bayes that assumes a multinomial distribution of the feature set. MNB is a probabilistic classifier and is commonly used as a baseline in text classification. MNB estimates the conditional probability of a term, given a class, as the relative frequency of term t in documents belonging to class c: $P(t|c) = \frac{T_{ct}+1}{(\sum_{t\in V} T_{ct})+B}$, where T_{ct} is the number of occurrences of t in the training documents from class c and $B = |V|$ is the number of terms in the vocabulary for a class [8]. This equation also uses Laplace smoothing by adding 1 to each count which ensures all conditional probabilities will not equal zero. Our pipeline uses the multinomial naïve Bayes classifier from the sklearn.naive_bayes package. The two parameters that were passed to the classifier were the *alpha* and *fit_prior*. The *alpha* parameter is a smoothing parameter that when set to 1 applies Laplace smoothing and when set to 0 applies no smoothing. The *fit_prior* parameter is a Boolean that if true, will learn class prior probabilities, and if false will assume a uniform prior. Setting *alpha* to 0.01 and *fit_prior* to false produced the best results for our application.

K-Nearest-Neighbors (KNN) is a classifier that finds a label with some similarity calculation, such as Euclidean or cosine distance, for a given feature vector and all other training document vectors [7]. The label that corresponds to the majority of the user defined K neighbors is the label that is assigned to the text document in question. KNN does not rely on prior probabilities and is computationally efficient. Our pipeline uses the KNeighborsClassifier from the sklearn.neighbors package. There were four parameters that were optimized for this classifier: *algorithm*, *metric*, *n_neighbors*, and *weights*. The *algorithm* parameter specifies what type of algorithm is used to compute the nearest neighbors. For our datasets, *brute* was the best parameter for *algorithm*, which is simply a brute force search. The *metric* parameter is the distance metric used in calculating nearest neighbors, which was the Euclidean distance in our case. *N_neighbors* was set to 2 for our classifier, this is the number of neighbors to consider when assigning labels. The *weights* parameter is the weight function used in prediction. In our case, *weights* was set to *distance*, which weights points by the inverse of their distance. This means that closer neighbors of a point will have more influence than ones that are farther away.

Linear Support Vector Classifier (LinearSVC) is implemented in terms of liblinear as opposed to libsvm. Liblinear does not support the "kernel trick" and is optimized for linear classification and supports L1 and L2 loss. LinearSVC handles multiclass labeling with a one vs rest strategy and has been proven to perform well with document classification problems and is effective in high dimensional space [4]. It works by finding the optimal separating hyperplane between classes from training data where the n features of the feature vector are treated as an n-dimensional point. We utilized LinearSVC from the sklearn.svm package. Although we attempted to optimize the parameters for LinearSVC using grid search, the best parameters proved to be the default ones.

Random Forest (RF) in an ensemble method for classification that constructs decision trees with various sub-samples of training data and uses averaging to improve the predictive accuracy and control over-fitting [10]. The output from RF is the class that is the mode of the outputs from the individual decision trees. We use the RandomForestClassifier from the sklearn.ensemble package. The parameters that we optimized with grid search include *bootstrap*, *max_features*, and *n_estimators*. *Bootstrap* is a Boolean that was set to false for our classifier which means bootstrap samples were not used when building trees. The *max_features* parameter was set to log2 which meaning the number of features considered when looking for the best split was log base 2 of the number of features. The *n_estimators* parameter is the number of trees in the forest. This is set to 128 in our pipeline.

Stochastic Gradient Descent implements regularized linear models such as support vector machines with stochastic gradient descent learning. This iteratively minimizes an objective function [2]. This classifier works well with sparse matrix data such as text documents. We use SGDClassifier from the sklearn.linear_model package. The only parameter that differed from the default parameters for SGD was the *alpha* parameter. *Alpha* is the constant that is multiplied by the regularization term, and it was set to 0.001.

Step 4 – Optimizing hyper parameters

Step four in our pipeline involves optimizing the hyper parameters for each classifier. For a machine learning classifier, hyper parameters are parameters that cannot be directly learned from the data. An example of a hyper parameter is the *n_estimators* parameter discussed in the RF classifier section. The range of values for a hyper parameter can be quite large, and there can be several hyper parameters per classifier. This means that finding the optimal combination of hyper parameters can be a difficult task. This task is often treated as a search problem and a common technique used to optimize hyper parameters is grid search. Grid search is an exhaustive search over a grid of parameters for a classifier. The output is the optimal value for the specified hyper parameters on a given classifier model. For our pipeline, we use GridSearchCV from the sklearn.grid_search package to obtain optimal parameters for CountVectorizer, TfidfTransformer, and each previously discussed classifier. For each dataset used in our pipeline, four iterations of grid search are performed. The search is performed on the training data excluding any MASCO occupations. The iteration that yielded the best accuracy for a classifier is the one used.

Computing Accuracy

In our work, accuracy for a given machine learning classifier is defined as the number of predictions that were correct on a test set when training a classifier divided by the total number of predictions made for the test set.

• Section 3 - Related Work

Text classification is well-researched in areas such as natural language processing and information retrieval. Various methods of text classification have been established over time and the appropriateness of a given method is highly dependent on the application and desired results.

[5] discusses their existing proximity based, multi class, and multi label approach to job title classification at CareerBuilder.com, and provides enhancements in the form of a hierarchal method of classification using coarse and fine grained cascading architecture utilizing support vector machine and K nearest neighbor classifiers. There are similarities between the approach we use in our machine learning pipeline and the current system in [5]. There current system is a semi-supervised approach that uses multi-class proximity based classifier. Although our approach is not semi-supervised, we do use proximity based classification with our KNN and SVM algorithms. [5] also employs unigram, bigrams, and TF-IDF in their feature selection just as we do. [5] uses SVM as a coarse-grained classifier in their hierarchal approach. The version of SVM they use is the same as ours. They implement SVM from LIBLINEAR with the one-vs-all strategy, the L2 loss function, and the Crammer Singer method. [5] differs from our approach

by hierarchically classifying documents in an iterative process where granularity increases with each iteration.

[3] utilizes three separate methods for classifying free text radiology reports with clinical codes. The first developed method is a natural language processing pipeline that trains a classification algorithm from extracted features. The second method is rule based and compares the overlap between the reports and the code descriptions to assign a label to a document. The last method is a specialized system aimed at the most common codes and mimics the guidelines a human medical coder might follow. Our approach is most like the first method developed by [3]. Just as we do, [3] preprocesses documents by converting all characters to lowercase and removing all non-alphabetical characters. They did use n-grams with feature selection, but they do not specify how large n becomes. The algorithm [3] uses for classification is a k-best version of the MIRA algorithm. MIRA has a multi class version as well as a simplified binary version that does not require solving a quadratic programming problem. This binary version can also be extended to multiclass with a one-vs-all strategy, similar to the version of LinearSVC we use. Unfortunately, the authors in [3] do not indicate what version of MIRA they use.

[1] performs a case study for different methods of classifying job title documents with ISTAT, an Italian structure for labeling occupations. The three methods they study are an explicit rule based approach, a machine learning approach, and an LDA based approach. Our work is most like the work done in [1], specifically the machine learning approach they study. They essentially take the same steps in their pipeline as we do, but the specifics of each step are different. Where we use lemmatization for preprocessing documents, [1] stems words. We use TF-IDF to represent a text document where [1] uses just TF. [1] uses sci-kit learn package, specifically the LinearSVC and Perceptron classifiers. Although we use LinearSVC, we also use four other machine learning classifiers for a more well-rounded approach to classifying. [1] also indicates they use Grid Search like us, but they do not explain the specifics of how they use the tool.

• Section 4 – Approaches and results

This section explains the various approaches implemented for the purposes of achieving our objective. The accuracy and effectiveness of each approach is analyzed and discussed. Such approaches include: 1) implementing a nearest neighbor algorithm that compares a given sample job document to all MASCO occupations, and computes the most similar match, 2) mapping the ISCO labels in the dataset to MASCO labels and using the results as training/test data for text classifiers, and 3) combining MASCO occupation variations with the ground truth and inputting it into the pipeline.

Approach 1 – Nearest Neighbors

Because the provided dataset was not labeled with MASCO labels, an initial nearest neighbors approach was implemented to assign MASCO labels to job titles using the Jaccard similarity coefficient. In this approach, each sample from the dataset was treated as a set. The set was lemmatized and sorted alphabetically. The Jaccard similarity coefficient was then computed between the sample set and each stemmed and sorted MASCO occupation. The MASCO label that corresponded to the MASCO occupation with the highest Jaccard index was assigned to each job sample. This approach resulted in 448 unique MASCO labels represented in the output. A problem with this approach is that 41, 435 samples of the 115,367 total samples had multiple MASCO labels with the same similarity scores. Manual inspection of the multiple MASCO labels is required resolve which of the multiple labels is the correct one. This was infeasible given our timeframe. Another issue we had with this approach is its inability to recognize that a job title such as *java engineer* should be assigned the MASCO label *2512-software developer*, not *2142-civil engineer*. This approach was incapable of semantically assigning labels to job post documents. The issues with this initial approach revealed the need for a training set labeled with MASCO labels. This is what led to mapping the ISCO labels to MASCO labels and building the machine learning pipeline.

Approach 2 – Training classifiers with training sets developed by mapping ISCO labels to MASCO labels

Approach 2 involves using the two training sets derived from mapping the ISCO labels to MASCO labels. The results of the two sets are displayed in table 3 and figure 1. The ensemble score for each row in the table is the average of all the classifiers. Set 1 has an ensemble score of 79% while set 2 has an ensemble score of 94%. It is not surprising that Set 2 performs much better, as these are the samples with the high confidence score. This also shows that CASCOT is most likely using a similar approach to the machine learning pipeline we developed. Although the above table and chart shows high prediction accuracy, the MASCO labels that result from the mapping are suspect due to the issues discussed in the overview section. This leads to a lack of trust in the two training sets and underlines the need for a training set with MASCO labels we are confident are labeled correctly. This is what led to constructing the ground truth and MASCO occupation variations for use in the machine learning pipeline.

	MultiNB	KNN	RF	SGD	LinearSVC	Ensemble
Set 1	0.70	0.85	0.88	0.70	0.86	0.79
Set 2	0.939	0.938	0.940	0.938	0.944	0.940

***Table 3**: Set 1 contains all MASCO mapped labels, regardless of confidence score. Set 2 contains only MASCO mapped labels that have a confidence score >= 80. Each column corresponds to a given classifier. The ensemble column contains the average accuracy of the five classifiers.*

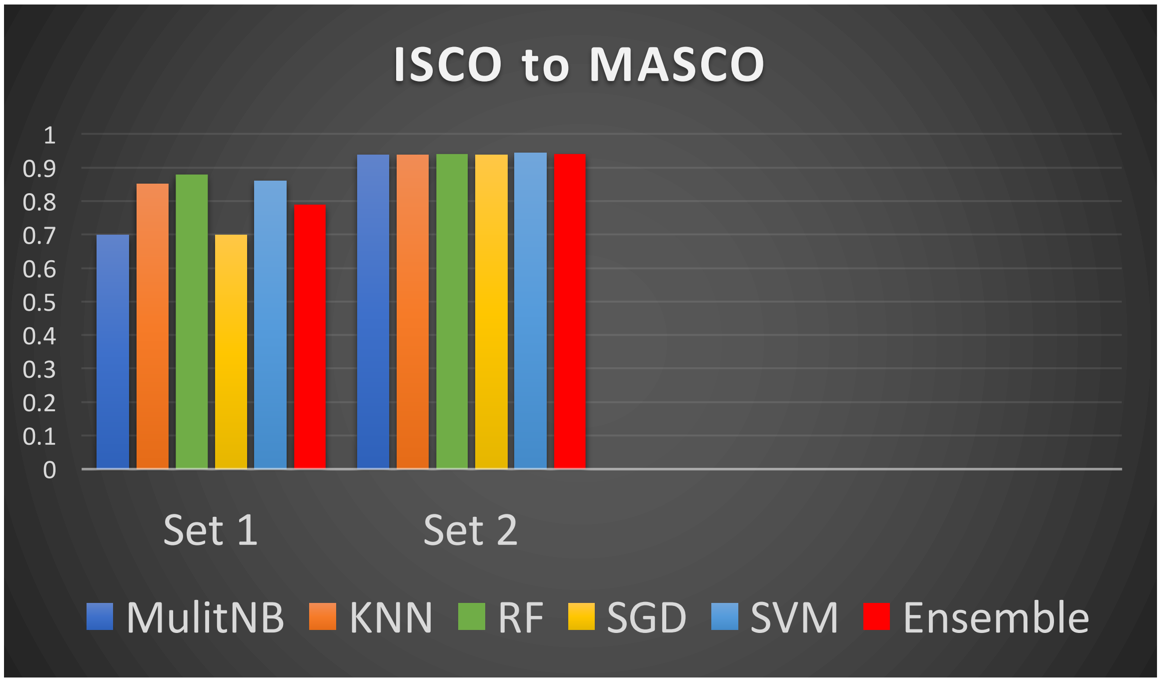

Figure 1: Set 1 contains all MASCO mapped labels, regardless of confidence score. Set 2 contains only MASCO mapped labels that have a confidence score >= 80. Each bar corresponds to a given classifier. The ensemble bar contains the average accuracy of the five classifiers.

Approach 3 - Training classifiers with the ground truth dataset and MASCO occupation variations.

The third approach is to combine the ground truth with the MASCO occupation variations and use them as input into the machine learning pipeline. The results of this approach are displayed in table 4 and figure 2. In this approach, the ground truth is combined with each of the five variations of the MASCO occupations resulting in 6 training sets total (five sets with a MASCO occupation variation and one without). The best training set in this approach is the ground truth with the MASCO occupation variation five. This is not surprising as variation five contained the most robust job documents consisting of a job title, a corresponding job description, and the fine grained 2013 occupations mapped to 2008 labels. Adding variation five increased the accuracy of all classifiers by an average of 2.5%. The best performing classifiers are SGD and LinearSVC. Although the accuracy is lower than the previous approach, we are more confident in these results as they use a training set with MASCO labels that we trust. We are confident that our machine learning pipeline is optimized and the lower accuracy in this approach results from the smaller training set sizes.

	Ground Truth	**Ground Truth MASCO Variation 1**	**Ground Truth MASCO Variation 2**	**Ground Truth MASCO Variation 3**	**Ground Truth MASCO Variation 4**	**Ground Truth MASCO Variation 5**
NB	0.601	0.614	0.619	0.620	0.626	0.629
KNN	0.577	0.596	0.604	0.606	0.608	0.610
RF	0.604	0.623	0.630	0.630	0.631	0.633
SGD	0.668	0.675	0.680	0.681	0.686	**0.689**
LinearSVC	0.667	0.672	0.678	0.679	0.684	0.687
Ensemble	0.623	0.636	0.642	0.643	0.647	0.650

Table 4: Each row shows the accuracy for a given classifier with the corresponding training set. The ensemble row contains the average accuracy of the five classifiers.

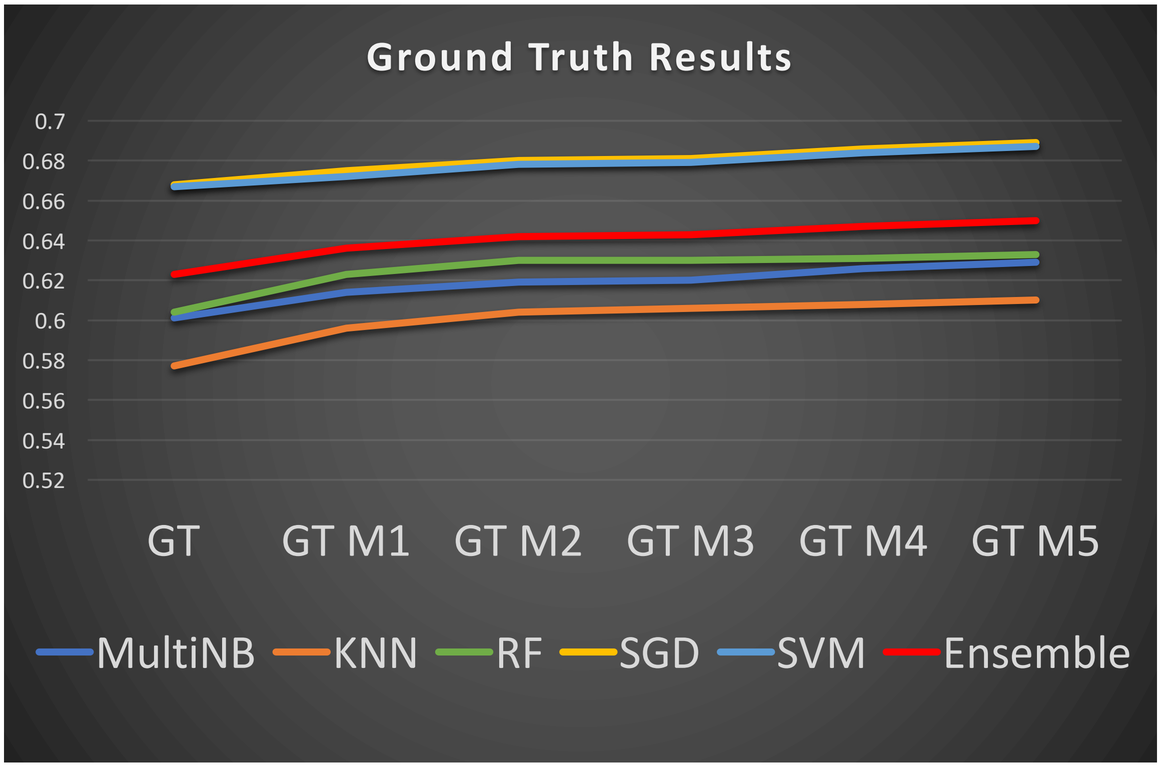

Figure 2: Each line shows the accuracy for a given classifier with the corresponding training set. GT = ground truth, and M1-M5 represent the five MASCO occupation variations. The ensemble line shows the average accuracy of the five classifiers.

Use of the ground truth with MASCO occupation variation five as input to the machine learning pipeline. Comparing the use of the manually assigned labels and the ISCO to MASCO mapped labels.

This section compares the difference between training classifiers with the same ground truth text occupations but with different labels. Two versions of the ground truth and MASCO occupation variation 5 were inputted into the pipeline. Version 1 consisted of MASCO labels that were manually assigned and version 2 consisted of the labels that were mapped from ISCO to MASCO. Table 5 displays the results. The version with the manually assigned MASCO labels performs better by approximately 6%. Also, the two versions agreed upon a MASCO label for a text document only 30% of the time. This illustrates the importance of training sets labeled with high confidence and clearly shows that approach 3 is the best approach for reaching our objective.

	MultiNB	KNN	RF	SGD	LinearSVC	Ensemble
Manually assigned MASCO labels	0.629	0.610	0.633	0.687	0.689	0.650
Labels mapped from ISCO to MASCO	0.592	0.591	0.593	0.591	0.597	0.593

Table 5*: Displays the accuracy of training sets with the same occupation texts and different labels. The ensemble column contains the average accuracy of the five classifiers.*

• Section 5 - Conclusions

In this paper, we discuss three approaches to reach our objective of classifying job post documents with labels from the 2008 Malaysian Standard Classification of Occupations. Each approach attempts various methods to overcome the challenge of working with a dataset with no MASCO labels. Approach 1 uses a nearest neighbor approach to assign MASCO labels to job documents. Issues with approach 1 is that it does not handle semantic features from samples in the dataset, and a significant amount of manual inspection is required to resolve cases of multiple labels that are all the nearest neighbor for a sample. Approach 2 maps ISCO labels to MASCO labels and inputs two resulting training sets into our developed machine learning pipeline for training classifiers. Although approach 2 does not suffer from the same issues as approach 1, underlying problems with the provided ISCO labels lead to a lack of confidence in their correctness. Therefore, even though the classifier accuracy results from approach 2 are quite high, they cannot be trusted. Approach 3 involves developing a training set for our pipeline by manually labeling a subset of our provided dataset with MASCO labels. Approach 3 does not face the same issues as approach 1 or 2 and we our most confident in the manually assigned labels with this approach. We believe that our machine learning pipeline is optimized for the provided dataset, and that given time to manually label approximately 3000 samples, approach 3 would obtain classification accuracy above 90%.

Provided more time, we would have done a couple of things to try to produce more training data with MASCO labels. The first simple things would be to manually label more samples. We learned this is the most trustworthy and best way to get quality samples, unfortunately it is the most labor intensive way as well. The second thing to try would be to manually inspect the training set that was developed from mapping ISCO labels to MASCO. Specifically, the labels with a confidence score of 80% or higher. These samples appeared to be correct but rigorous inspection is required to confirm our observation. This would be a faster way to add more than 2000 samples to the ground truth. One caveat with this approach is that significantly less MASCO labels appeared in this training set. This could potentially skew the distribution of MASCO labels present in the ground truth.

Multiple insights were made through the development of our machine learning pipeline. One such insight involves data preprocessing. Although data preprocessing had limited effects on our ground truth training set, we believe this to be a result of the small size of the set. We are confident techniques like correcting misspelled words and removing stop words would produce noticeable gains with a data set of several thousand samples or more. Lemmatization was the one preprocessing technique that revealed itself to be a useful tool for training sets of any size. Support vector machines proved to be the most useful classifier in our pipeline. A likely reason for this is because SVM's perform well even with a small amount of labeled training samples, which was the case for our ground truth. The most significant and fundamental insight we take away from this project is that a machine learning application is only as good as the data it learns from. In text classification, to be confident in classifier output, it is imperative there is trust in the training set, specifically that the appropriate labels are assigned to text documents.

Bibliography

[1] Amato, Flora, et al. "Challenge: Processing web texts for classifying job offers." *Semantic Computing (ICSC), 2015 IEEE International Conference on*. IEEE, 2015.

[2] Bottou, Léon. "Large-scale machine learning with stochastic gradient descent." *Proceedings of COMPSTAT'2010*. Physica-Verlag HD, 2010. 177-186.

[3] Crammer, Koby, et al. "Automatic code assignment to medical text." *Proceedings of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing*. Association for Computational Linguistics, 2007.

[4] Fan, Rong-En, et al. "LIBLINEAR: A library for large linear classification." *Journal of machine learning research* 9.Aug (2008): 1871-1874.

[5] Javed, Faizan, et al. "Towards a job title classification system." *arXiv preprint arXiv:1606.00917* (2016).

[6] Jones, R., and P. Elias. CASCOT: Computer-Assisted Structured Coding Tool. Coventry, UK: Institute for Employment Research, University of Warwick, http://www2.warwick.ac.uk/fac/soc/ier/software/cascot/details. (2004)

[7] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Vol. 1. No. 1. Cambridge: Cambridge university press, 2008.

[8] McCallum, Andrew, and Kamal Nigam. "A comparison of event models for naive bayes text classification." *AAAI-98 workshop on learning for text categorization*. Vol. 752. 1998.

[9] Ministry of Human Resources Malaysia, "Malaysian Standard Classification of Occupations 2008 Third Edition", 2010

[10] Svetnik, Vladimir, et al. "Random forest: a classification and regression tool for compound classification and QSAR modeling." *Journal of chemical information and computer sciences* 43.6 (2003): 1947-1958.