Machine Learning (ML) Classifier to Assist Metadata Creation

1st Hannah Collier
Environmental Science Division
Oak Ridge National Laboratory
Oak Ridge, USA
collierhr@ornl.gov

2nd Eric Enright
Environmental Science Division
Oak Ridge National Laboratory
Oak Ridge, USA
enrightew@ornl.gov

3rd Sujata Goswami
Advanced Light Source
Lawrence Berkeley National Laboratory
Berkeley, USA
sujatagoswami@lbl.gov

4th Chirag Shah
Environmental Science Division
Oak Ridge National Laboratory
Oak Ridge, USA
shahch@ornl.gov

5th Maggie Davis
Buildings and Transportation Science Div
Oak Ridge National Laboratory
Oak Ridge, USA
davismr@ornl.gov

6th Rachael Isphording
Climate Change Research Centre;
ARC Centre of Excellence for Climate Extremes
University of New South Wales
Sydney, Australia
0000-0003-4451-6204

***Abstract*—The Atmospheric Radiation Measurement (ARM) Data Center is responsible for the timely collection, archival, and curation of science data products. These products are freely available through an online data repository. Metadata creation is paramount for scientific users to find and access over seven petabytes of atmospheric science data. The hierarchical metadata structure allows users to search for information at both broad and narrow levels. This project aims to leverage 30 years' worth of manually created metadata to enable machine predictions of broad-term classifications from narrow-term descriptions. These classification predictions would assist metadata coordinators with their term selections. This paper discusses the cleaning and preprocessing of the training data, the pipeline developed to determine the best model for this task, and the creation of an API metadata classifier for ARM measurement metadata. Our results show that the Linear Support Vector Classification (LinearSVC) algorithm, along with the Term Frequency – Inverse Document Frequency (TF-IDF) vectorizer, is well-suited for our multi-class classification task. Lengthier input training data led to better results, and artificial balancing was unnecessary for this particular use case. This predictive classifier enhances efficiency in metadata creation, as well as supports greater consistency and accuracy in metadata tagging.**

***Index Terms*—ARM Data Center, Metadata, Supervised Machine Learning, TF-IDF, LinearSVC**

I. Introduction

A Machine Learning (ML) model is the output generated when you train an ML algorithm with data [1]. ML rules are based on input data as well as the answers expected from the data. Specifically for supervised ML, these inputs and outputs are labeled, and therefore can be ingested by the machine. Training a model using labeled data can help eliminate manual classification work and can help with future prediction. Supervised machine learning can be used to classify inputs into two or more classes.

Researchers have investigated the feasibility of text multi-class classification ML capabilities across several domains [2], [3], [4], [5] and [6]. Many of these publications discuss performance metrics across traditional supervised multi-class classification algorithms, such as Random Forest, Naive Bayes, K-Nearest Neighbors, and Support Vector Machines. In this paper, we explore using Linear Support Vector Classification, K-nearest Neighbors Classifier, and Multinomial Naive Bayes classifiers to support metadata tagging and search capabilities within a data repository.

For domain-specific data repositories, metadata tagging requires domain and ontology knowledge. One such domain-specific data repository is the Atmospheric Radiation Measurement (ARM) data repository. ARM is a United States Department of Energy Office of Science user facility, which is operated across nine national laboratories and has collaborations with many national and international partners. ARM has been collecting continuous observations of cloud and aerosol properties, among other atmospheric science measurements, for over 30 years. The ARM Data Center (ADC), located at Oak Ridge National Laboratory, works to ensure the timely collection, processing, and delivery of data products to the scientific community. Teams within the ADC focus on following the FAIR data principles by ensuring data is (f)indable, (a)ccessible, (i)nteroperable, and (r)eusable [7]. Efforts include providing a user-friendly experience for data searchers through thoughtful user interfaces and clearly labeled data products.

Metadata tagging is critical to how easily a user can sift through over seven petabytes (PB) of data, finding and

This manuscript has been authored by UT-Battelle LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

accessing the most appropriate atmospheric science data products within the search interface. Augmenting the human-lead process of metadata term classification using natural language processing (NLP), a subset of ML, can help with keyword accuracy and consistency, as well as workflow efficiency.

II. Goals

The goal of this project was to classify metadata terms for atmospheric science datasets using ML. In this paper, we discuss the process of reviewing and cleaning the training dataset, along with its challenges and necessity. Further, we present the pipeline that we implemented to test various algorithms for text classification.

In Section 3, we describe ARM metadata, workflows, and terminologies. In Section 4, we explain the data preprocessing steps, and in Section 5 we present the three models tested and the pipeline created to test the various models. Each algorithm was tested with two separate training datasets and with and without artificial data balancing. We then discuss the results in Section 6, followed by discussions.

III. ARM Metadata workflow and structure

Metadata, or "data about data", plays an important role in how data products can be found within a data repository. Users can search for ARM data through the Data Discovery interface (https://adc.arm.gov/discovery/#/), and other helpful information can be found through the ARM website (arm.gov). At the ADC, metadata specialists tag data products with information such as the location where data were collected, the type of instrument used, the primary variables collected, and the level of data processing. Created metadata are stored in a PostgreSQL database, which is then called for web page content and Data Discovery search filters.

For routine data collected from ARM instruments, certain metadata codes are combined into a naming convention called a datastream. ARM datastreams include information on the location, the level of data processing, and the instrument. Each component of the datastream, including site code, facility code, instrument code, and data level, can be searched for within Data Discovery. A datastream makes up the first part of a data filename, followed by the date and time stamps (Fig. 1).

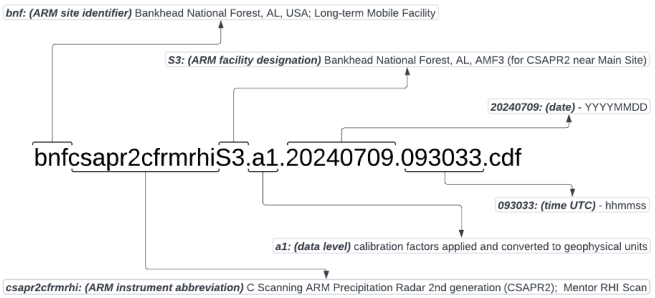

Fig. 1. Datastream naming process.

The instrument code within a datastream is the lowest level metadata contained in a hierarchical structure for instrument categorization. An instrument code is categorized into a broader instrument class, and then an even broader instrument category. This hierarchy is important in allowing different levels of search filtering within Data Discovery.

Measurement metadata are also configured in a hierarchical structure. ARM data files are typically stored in Network Common Data Form (NetCDF) format. NetCDF is a community standard format for creating, accessing, and sharing array-oriented scientific data. ARM NetCDF files have embedded metadata in the file's header, including the collected measurement variables and their descriptions. Primary measurement status is assigned to certain variables by instrument mentors as the measurements that are the most scientifically relevant for a data product. These are the measurements that are available for search through the Data Discovery interface.

Currently, metadata specialists review each primary measurement variable and its description to manually classify it into a wider measurement class, called Primary Measurement Type (PMT) (Fig. 2). PMTs are classified into a larger umbrella term called a measurement category. The hierarchical structure of variables assigned to PMT classifications is the basis of this project. With over 30 years of ARM data collection, a robust dataset of such classifications exists. Over 62,000 variable/PMT pairs have been created and associated with ARM data products, and these pairs are maintained in a PostgreSQL database. Using past variable descriptions and their PMT classifications as a training dataset, we aimed to train a model to predict PMT classifications for variable descriptions.

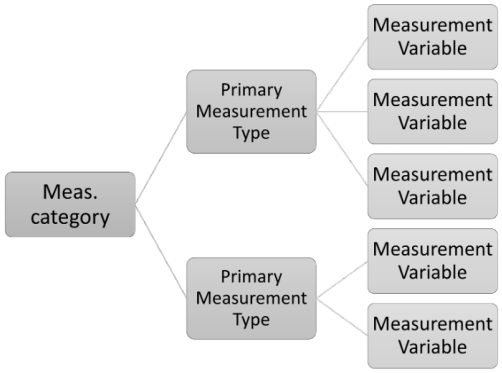

Fig. 2. Primary Measurement Type classification.

Importantly, the PMT classifications will always go through a manual review. While we anticipate using this model within our workflow to cut down on research time and effort, a metadata specialist will review the top model-selected PMTs for each measurement variable description and select the best fit. In this way, ML will be used to augment the workflow, providing guidance in consistency, rather than replacing domain expert knowledge.

IV. Methods

The ADC maintains metadata tags and associations in a PostgreSQL database. This database was queried to create a training set and test set for testing the ML algorithms. The effectiveness of supervised machine learning depends on the quality and quantity of the model's training data. This step takes a thorough understanding of the data. Initial ML model testing identified the need for focused cleaning of the training data set.

A. Mitigating Inconsistent Classification with Data Cleaning

There were 198 distinct measurement variable descriptions found that had been assigned to 2 or 3 different PMTs. During the course of ARM data collection and metadata creation, new PMTs have been created as measurements and semantics evolved. While we try to ensure that previously assigned measurement variables are updated to be classified with the most accurate PMT, this is hard to ensure with the wealth of historic metadata. Cleaning the training data became an auditing task which involved updating some variables' PMT assignments to ensure the best possible consistency and accuracy.

B. Testing Augmented Input Data to Solve Further Inconsistent Classification

Some measurement variable descriptions are vague and logically can be classified within more than one PMT. A PMT is assigned to a variable within a specific datastream, so the instrument information is always known during this step. In order to augment the input data with as much information as possible, we included the instrument name with the variable description within the training dataset. Table I provides examples of the base training data with only the variable description as input (Var Dataset) and the augmented training data (Var-Inst Dataset) which contains the variable description and instrument name.

TABLE I
Examples of the base training data

Training Data	***Input***	***PMT Name (training output)***
Var Dataset	Hemispheric broadband irradiance, at end of shadowband sweep, broadband channel	irradswbbtotdn
Var-Inst Dataset	Hemispheric broadband irradiance, at end of shadowband sweep, broadband channel. Portable Radiation Package: Fast Rotating Shadowband Radiometer full resolution 6-s sampling	irradswbbtotdn

We were unsure whether this additional text would benefit or hinder the model training, so we tested both training datasets in the model selection process described in the next section. These tests helped us decide which training dataset was the best for this task. Results showed that the input data including the instrument name improved model performance (Section 6).

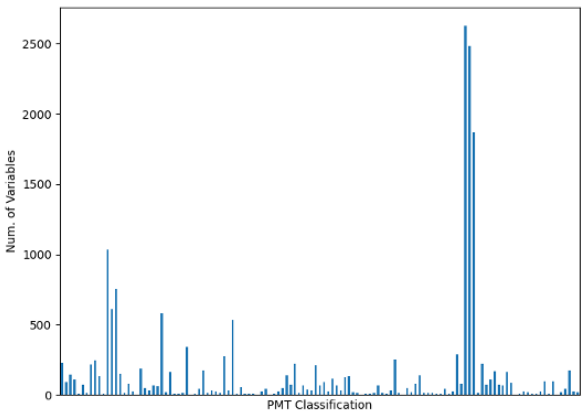

Fig. 3. Imbalanced PMT.

C. Testing Artificial Balancing to Mitigate Imbalanced Data

The training dataset is imbalanced. Overall our training data contains 127 distinct PMTs. There are some PMTs which contain thousands of unique variable descriptions (majority classes), while many others contain five or less (minority classes) (Fig. 3). This imbalance is cause for concern, as the model's predictions may be biased towards the majority classes.

There are a number of methods available to mitigate imbalanced training data. Oversampling increases the number of samples in the minority class in order to balance the distribution samples among classes in the training data. Conversely, undersampling reduces the number of samples in the majority class in order to provide balance. These methods can be used individually or combined in order to artificially balance training data.

It was unclear how the imbalanced data would affect our use case. We intended that the model would always provide the user with five PMT classification suggestions for a variable description, which will then go through a manual selection and approval process. To determine if balancing was needed to ensure that the correct PMT was in the top five recommended classifications, we used SciKit Learns' Synthetic Minority Over-sampling Technique (SMOTE). SMOTE is an oversampling method, which creates synthesized examples for minority classes, without simply duplicating samples. Section 6 provides results showing how this method was not needed for our use case.

D. Preprocessing - Natural Language Toolkit

Once the model input was determined and the training data was created through PostgreSQL queries, the pipeline shown in Fig. 4 commenced. To begin this pipeline, the training data (either Var Dataset or Var-Inst Dataset) were imported into the Python script using the Pandas library, creating a dataframe, and then preprocessed. Preprocessing included dropping any

duplicates from the training data. As the metadata stored in the database included every variable/PMT pair associated with various data products, there were many duplicates.

For Var Dataset, removing duplicate pairs reduced the training data to 4,383 variable descriptions and their PMT classifications. For Var-Inst Dataset, 17,786 variable and instrument descriptions with PMT classifications remained. Then, the Natural Language Toolkit (NLTK) library was used to remove stopwords, which are common language words like "the", "and" and "is". Since these words typically do not convey important semantic information, it is common to remove these words with NLP.

E. Preprocessing - Text Vectorization

Term Frequency – Inverse Document Frequency (TF-IDF) was used to vectorize the text into numerical data. This process measures the importance of a term based on how frequently it is used in a corpus. A term that is used often in one input description, but infrequently in the other input descriptions, will be weighted high. TF-IDF is calculated by multiplying the term frequency (TF) by the inverse document frequency (IDF).

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) * \text{idf}(t, D) \tag{1}$$

The term frequency is the number of times a term (t) occurs within a document (d):

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t'd}} \tag{2}$$

IDF is the logarithm of the total number of documents (N) divided by the number of documents that contain the term:

$$\text{idf}(t, D) = log \frac{N}{|\{d \in D : t \in d\}|} \tag{3}$$

V. Model Selection

After the data review and preprocessing steps, we tested several algorithms. Selecting a ML algorithm for a certain task can take lots of research, tests, and trial and error. In order to test multiple algorithms, we used the Python package, Scikit Learn [8]. The Scikit Learn package documentation provides a helpful flowchart to help choose the appropriate estimator for a project, based on different data availability and objectives [9]. Using this flowchart, we decided to test the following three algorithms in more detail:

- Linear Support Vector Classification
- K-nearest Neighbors Classifier
- Multinomial Naive Bayes

The next three subsections provide general information on each of these classification algorithms.

A. LinearSVC

Linear Support Vector Classification (LinearSVC) is a type of support vector machine (SVM) that creates a decision boundary, also known as a hyperplane. This hyperplane is used to classify data points. In SVMs, the margin refers to the distance between the hyperplane and the closest data points. SVMs aim to find the hyperplane that maximizes the margin, which helps improve the separation between different classes. LinearSVC focuses on creating a linear hyperplane, making it more efficient for certain data. Additionally, LinearSVC supports multi-class classification, making it a well-suited option for our use case.

B. KNeighborsClassifier

The K-Nearest Neighbors (KNN) classifier uses an algorithm called KNN. This algorithm identified the k nearest data points to a new, unclassified point based on a chosen distance metric. In our case, we used the Euclidean distance. During classification, the KNN algorithm considers the k nearest neighbors of the unclassified point. Within Scikit Learn, k is an adjustable hyperparameter which determines the number of neighbors to examine. The algorithm assigns the class label that is most frequent among these k neighbors.

C. MultinomialNB

Multinomial Naive Bayes (MNB) is based on Bayes' Theorem and works by calculating probabilities based on the distributions of text within the training data. These probabilities are calculated for each class, and ultimately, the class with the highest probability score is chosen as the predicted class. MNB is often significantly faster than other models.

D. Methods

In addition to algorithm selection, different hyperparameters were tested for the TF-IDF vectorizer, both training datasets were tested for each algorithm, and the SMOTE balancing method was also tried for each algorithm. To do this, a pipeline (Fig. 4) was created to facilitate selecting the following for our use case:

- Best algorithm and hyperparameters
- Best TF-IDF hyperparameters
- Best training dataset
- Whether or not to use SMOTE balancing

For each run, Var Dataset or Var-Inst Dataset was selected, we determined whether or not to include SMOTE balancing, and we selected an algorithm to test. We then split the selected dataset into 80% training, and 20% validation. The validation data was set aside to collect metrics for each of the runs. With the 80% training data, we train the algorithm and various parameters using the GridSearchCV method in Scikit Learn. This allowed us to test many different hyperparameters in one run. Parameters were provided in dictionaries for both the TF-IDF vectorizer and for the algorithm. While GridSearchCV is computationally expensive, we had access to high performance computing resources, which allowed for a shorter computing time.

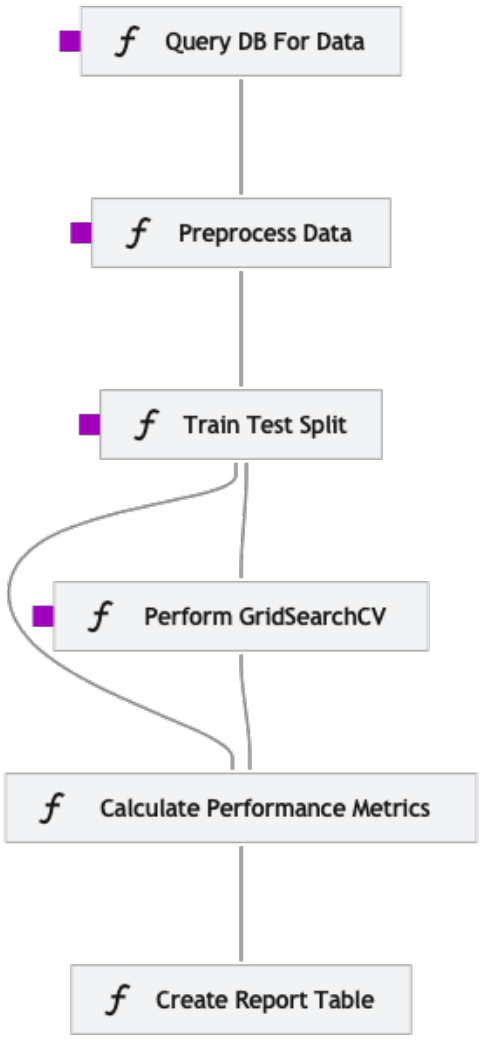

Fig. 4. Algorithm selection pipeline generated using Kedro [10].

GridSearchCV utilizes the k-fold cross validation method. In the k-fold cross validation method, the training data is split into smaller sets, or folds. During each step of k-fold, one fold is used for testing data, while the remaining folds are used for training data. After the data is split the model is trained.

This process is repeated so that the number of steps equals the number of folds; we used five folds. This repetition ensures that the model is not overfitting the training data and is able to predict classification for new input data. At the end of each step a cross validation score is calculated. These scores are then averaged to create a mean cross validation score.

E. Metrics

After GridSearchCV ran for a particular algorithm, dataset and balancing method, we accessed different attributes like the best hyperparameters, the best performing model that was trained during k-fold, and the mean cross validated score. With the 20% validation data, we determined additional performance metrics. We used the best model determined by the GridSearchCV process to predict classifications for the validation data input. Using the Scikit Learn classification_report method, the predicted classes were compared to the true classifications provided in the validation data. This method provided us with metric calculations for precision, recall, and F1-score. The metrics were all weighted based on the amount of instances per class. Precision and recall are calculated in equation 4 and 5, where TP is true positive, TN is true negative, FP is false positive, and FN is false negative. Lastly, the F1-score is the harmonic mean of precision and recall.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1 = 2 \cdot \left(\frac{(precision \cdot recall)}{(precision + recall)} \right) \tag{6}$$

Additionally, to ensure the model was working for our use case, we investigated the model's misclassifications. Ideally, even if the sample is misclassified with the model's top prediction, the correct label should be in the top five recommended classifications to satisfy our use case.

The top 5 recommended classifications correspond to the highest probabilities obtained using CalibratedClassifierCV. During training, this method performs cross-validation by creating k folds. It splits the training data into validation and training subsets for each fold. In each fold, a copy of the chosen classifier (the base estimator) is trained on the training subset. Then, the validation subset is used with the newly trained model to obtain un-calibrated probabilities. These un-calibrated probabilities might not directly reflect the true class probabilities. To address this, CalibratedClassifierCV employs a sigmoid regression model to adjust the raw predictions closer to the actual labels. The sigmoid model uses the sigmoid parametric sigmoid equation [11]:

$$p(y_i = 1|f_i) = 1/1 + exp(Af_i + B) \tag{7}$$

In this equation, y_i is the actual label for the sample i, and f_i is the output of the base estimator classifier for the sample i. The values for A and B are calculated when fitting the regressor. This process transforms the uncalibrated probabilities into calibrated probabilities. Finally, CalibratedClassifierCV calculates an average of the calibrated probabilities from all folds to provide the final set of recommended classifications.

This pipeline was run 12 times. Each of the three algorithms were tested with both datasets, and each dataset was tested with and without SMOTE balancing. Therefore, four runs were completed for each algorithm. From the metrics discussed in the next section, we selected the best performing training dataset, model, hyperparameters, and balancing method.

TABLE II
BEST SCORES FOR EACH OF THE PIPELINE RUNS

Model	Training Dataset	Balancing	Mean Cross Validated Score	Recall	Precision	Accuracy	F1-score	Correct PMT in Top 5
KNC	Var	none	75.20%	78.67%	81.44%	78.67%	78.48%	92.89%
KNC	Var	smote	75.70%	80.89%	85.11%	80.89%	81.50%	91.10%
KNC	Var-Inst	none	82%	86.00%	86.92%	86.00%	85.67%	97.48%
KNC	Var-Inst	smote	83.80%	87.20%	88.78%	87.20%	87.48%	93.76%
LSVC	Var	none	85%	88.89%	91.81%	88.89%	89.38%	97.78
LSVC	Var	smote	85.30%	87.11%	88.50%	87.11%	86.53%	96%
LSVC	Var-Inst	none	96.10%	98.58%	98.95%	98.58%	98.70%	99.63%
LSVC	Var-Inst	smote	96.50%	98.47%	98.90%	98.47%	98.61%	99.12%
MNB	Var	none	65.20%	80.44%	85.13%	80.44%	80.39%	91.56%
MNB	Var	smote	76.10%	79.11%	84.62%	79.11%	79.92%	93.78%
MNB	Var-Inst	none	74.50%	81.29%	82.97%	81.29%	80.70%	95.08%
MNB	Var-Inst	smote	78.60%	83.26%	89.80%	83.26%	85.42%	92.56%

VI. RESULTS OF CLASSIFICATION

Table II shows the best scores for each of the pipeline runs. The Var-Inst Dataset, which included measurement and instrument descriptions (Table I) scored better than the Var Dataset for each of the algorithms. This is likely due to the larger amount of unique input and output pairs. The results may have also benefited from the additional content provided in the input data.

The ideal percentage for our metrics is 100%. With F1-score, recall, precision, mean cross validation score, and accuracy results over 90%, LinearSVC demonstrated incredibly promising results, and we selected it as the algorithm for this project. While using SMOTE to artificially balance the training data slightly benefited the MNB and KNN algorithms, it showed very little improvement for the LinearSVC algorithm, if any. In fact, recall, precision, accuracy, F1-score, and the amount of correct PMTs in the top 5 results were slightly better without SMOTE balancing for the highest performing pipeline run (using LinearSVC and Var-Inst Dataset).

Based on the results of the GridSearchCV, the best performing hyperparameters selected for the LinearSVC algorithm and TF-IDF text vectorizer include:

```
tfidf_params = {
    "max_df": 0.4,
    "min_df": 1,
    "ngram_range": (1,2)
}

LSVC_params = {
    "C": 100,
    "loss": "squared_hinge",
    "max_iter": 5000,
    "solver": "lbfgs"
}
```

A. Testing

After selecting the best hyperparameters for LinearSVC and TF-IDF, we split our data into 95% training data and 5% testing data to train and test the selected model and parameters for the LinearSVC algorithm, with Var-Inst Dataset and without SMOTE balancing. This strategy allows us to see how the model functions on a larger portion of training data to test our selection. After training, we test the results with the remaining 5% of the data, calculating the same metrics that were presented from the GridSearchCV pipeline runs (Table III). The mean cross validated score is not included since it was only calculated during the k-fold cross validation method, which was not used in this final testing. With metrics over 98%, the testing step corroborates our previous results.

B. Implementation

Using the best performing parameters, a model was then trained on the full Var-Inst Dataset, using the TF-IDF vectorizer and LinearSVC algorithm. This final model was saved to a pickle (.pkl) file so it can be called into other scripts.

An API was developed using FastAPI, which is a Python framework for building APIs. The API works by a user sending a request to the API with a datastream as the input parameter. The API then queries the database to retrieve the associated variable descriptions and instrument name for that datastream. These variable descriptions are concatenated with the instrument code name and are fed into the model. The model outputs the top 5 recommended PMTs for each input, along with the calibrated classifier probability value for each recommendation (Fig. 5).

C. Next Steps

We intend to implement this API within our internal metadata creation tool which connects to the database. Once this is done, the metadata specialist can make a selection out of the PMTs provided for the official PMT classification. As new variable/PMT pairings are created, so too are more training data. A new model will be retrained on ARM metadata routinely, likely on a monthly basis, and the new model will be saved to be used within the API. As the metadata specialists review the PMT selections, they will ensure the model is functioning well, noting any issues or errors. Ultimately, the model should become even better over time.

Upon successful implementation of this model into our classification workflow, additional models may be trained using TF-IDF and the LinearSVC algorithm to classify other hierarchical metadata terms.

TABLE III
GRIDSEARCHCV PIPELINE RUNS

Model	Training Dataset	Balancing	Recall	Precision	Accuracy	F1-score	Correct PMT in Top 5
LSVC	Var-Inst	None	98.28%	98.30%	98.28%	98.22%	99.70%

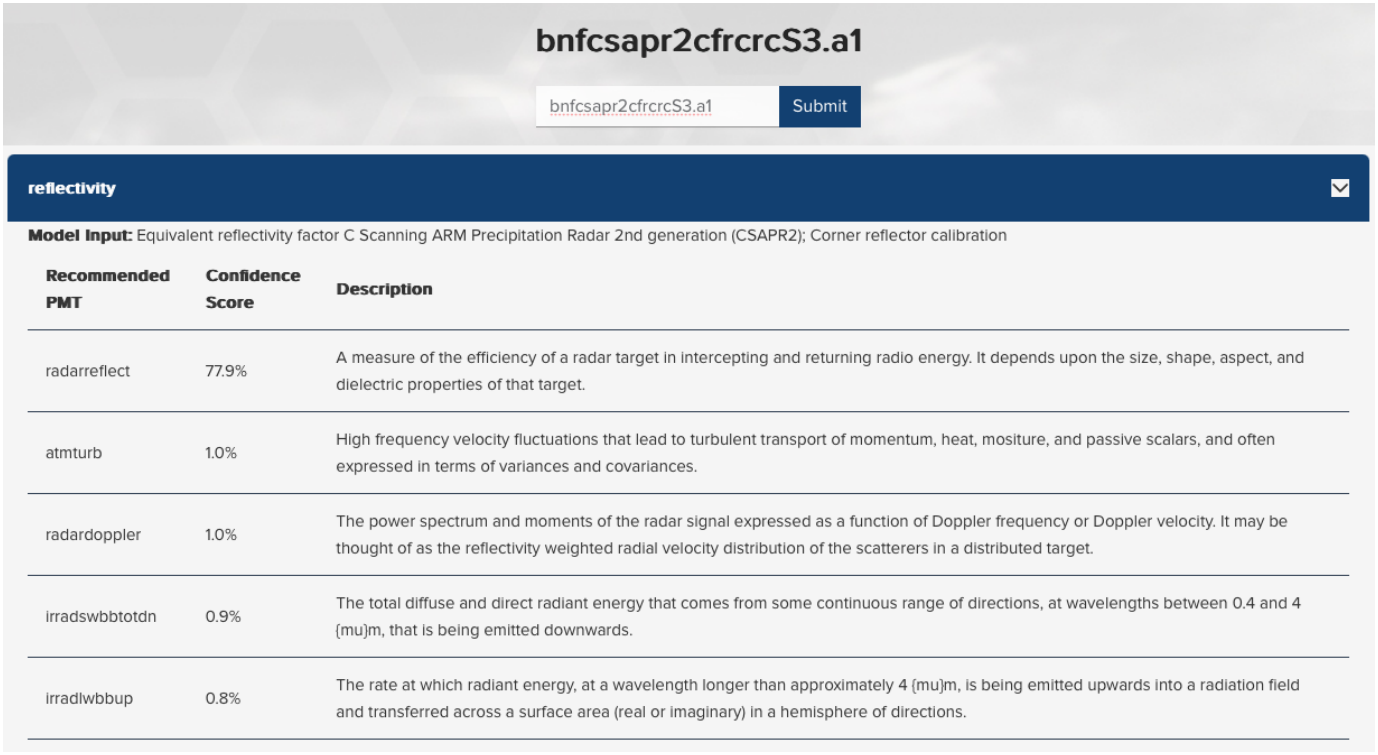

Fig. 5. Recommended PMT results with an API.

VII. DISCUSSIONS

Our results indicate that using TF-IDF and the Linear SVC algorithm provides promising results for this multiclass text classification task. Other studies have found similar results [3], [5], [4], although none with this large number of classes, and not for the purpose of augmenting a human workflow. We did not investigate the use of large language models (LLMs) within this research. However, others have investigated these methods for text classification [12], [13], [14], and [15]. While these methods have promising results, finding a LLM best suited for a certain domain is critical [12], and traditional supervised classifiers are comparable and use less computational power [13].

This study highlights the importance of reviewing and cleaning training data through each step of the model selection process, especially for such specialized ML applications. Additionally, it is important to consider specific use cases when reviewing a model's metrics. As the final PMT review and selection will still be done by a metadata specialist, this diminishes the need for the model's top classification prediction to be perfect. However, the need for the correct PMT to be in the top five results is a high priority metric.

Implementing this model into our metadata creation workflow helps cut down on time and effort and helps guide us towards better consistency. However, by maintaining the human element in this workflow, we can monitor the model's performance and ensure the accuracy needed for a seamless data search experience for the ARM data repository's users.

ACKNOWLEDGMENT

This research was supported by the Atmospheric Radiation Measurement (ARM) user facility, a U.S. Department of Energy (DOE) Office of Science user facility managed by the Office of Biological and Environmental Research Program.

REFERENCES

[1] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.
[2] B. A. Xavier and P.-H. Chen, "Natural language processing for imaging protocol assignment: machine learning for multiclass classification of abdominal ct protocols using indication text data," *Journal of Digital Imaging*, vol. 35, no. 5, pp. 1120–1130, 2022.
[3] C. Gong, G. Deng, H. Shan, and C. Deng, "Predictive classification model based on vespa comprehensive detection system," in *Journal of Physics: Conference Series*, vol. 1982, no. 1. IOP Publishing, 2021, p. 012080.
[4] N. Kalcheva, M. Karova, and I. Penev, "Comparison of the accuracy of svm kemel functions in text classification," in *2020 International Conference on Biomedical Innovations and Applications (BIA)*. IEEE, 2020, pp. 141–145.

[5] G. Leonard, F. Sisnadi, N. V. Wardhana, M. A. A. Al-Ghofari, and A. S. Girsang, "News classification based on news headline using svc classifier," in *2022 16th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*. IEEE, 2022, pp. 1–4.

[6] M. E. Phillips and J. Chen, "Machine learning for name type classification in library metadata," *Proceedings of the Association for Information Science and Technology*, vol. 54, no. 1, pp. 773–774, 2017.

[7] M. D. Wilkinson, M. Dumontier, I. J. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 160 018–160 018, 2016.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] S. Alam, N. L. Chan, L. Couto, Y. p. Dada, I. Danov, and . (list all authors), "Kedro," 2024. [Online]. Available: https://github.com/kedro-org/kedro

[11] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[12] E. Lee, C. Lee, and S. Ahn, "Comparative study of multiclass text classification in research proposals using pretrained language models," *Applied Sciences*, vol. 12, no. 9, p. 4522, 2022.

[13] K. Shyrokykh, M. Girnyk, and L. Dellmuth, "Short text classification with machine learning in the social sciences: The case of climate change on twitter," *Plos one*, vol. 18, no. 9, p. e0290762, 2023.

[14] A. Moreo, A. Esuli, and F. Sebastiani, "Word-class embeddings for multiclass text classification," *Data Mining and Knowledge Discovery*, vol. 35, pp. 911–963, 2021.

[15] K. Ameri, M. Hempel, H. Sharif, J. Lopez Jr, and K. Perumalla, "Cybert: Cybersecurity claim classification by fine-tuning the bert language model," *Journal of cybersecurity and privacy*, vol. 1, no. 4, pp. 615–637, 2021.