Taming WOLF:

Building a More Functional and User-Friendly Framework

Casey Sader

Electrical Engineering and Computer Science
University of Kansas
June 12, 2019

Committee: Dr. Michael Branicky (chair), Dr. Suzanne Shontz, Dr. Bo Luo

A project report presented for the degree of
Masters of Computer Science

Taming WOLF:

Building a More Functional and User-Friendly Framework

Casey Sader

Abstract

Machine learning is all about automation. Many tools have been created to help data scientists automate repeated tasks and train models. These tools require varying levels of user experience to be used effectively. The "machine learning WOrk fLow management Framework" (WOLF) aims to automate the machine learning pipeline. One of its key uses is to discover which machine learning model and hyper-parameters are the best configuration for a dataset. In this project, features were explored that could be added to make WOLF behave as a full pipeline in order to be helpful for novice and experienced data scientists alike. One feature to make WOLF more accessible is a website version that can be accessed from anywhere and make using WOLF much more intuitive. To keep WOLF aligned with the most recent trends and models, the ability to train a neural network using the TensorFlow framework and Keras library were added. This project also introduced the ability to pickle and save trained models. Designing the option for using the models to make predictions within the WOLF framework on another collection of data is a fundamental side-effect of saving the models. Understanding how the model makes predictions is a beneficial component of machine learning. This project aids in that understanding by calculating and reporting the relative importance of the dataset features for the given model. Incorporating all these additions to WOLF makes it a more functional and user-friendly framework for machine learning tasks.

Introduction

Within the last few years, machine learning (ML) has become extremely popular and successful in areas such as health, computer vision, finance, speech recognition, sports, and many other fields. Because of the large amount of data in these fields, both experienced researchers in data science and novices that have only just heard about machine learning want to be able to utilize its power. Machine learning algorithms, by definition, automate the rule induction for tasks such as classification and regression that can then be used to better understand a dataset and make predictions on the future or on what actually occurred. There is a lot of work that goes into the data science pipeline besides just fitting a machine learning model and performing all steps, including choosing the best hyper-parameters for the model, can be a cumbersome task. To help with these tasks, WOLF (machine learning WOrk fLow management Framework) was originally created by Pranav Bahl under Dr. Luke Huan in December 2016. This project began on a team under Dr. Huan in 2017 to continue the work and has now become an individual project under Dr. Branicky to keep improving the framework. This project focuses on adding common features that WOLF is lacking and meeting the expected goals of users to make WOLF more user-friendly and accessible. The novice user is the main experience level kept in mind, but an experienced user could also benefit from some of the automation WOLF offers.

Objective

The goal of this project is to improve the existing WOLF framework so that a user has more tools at their exposure and freedom to do what they feel is needed to learn about their dataset. With WOLF typically having a novice in mind, this project has a focus on what WOLF was lacking that they may want. WOLF does have some limitations and the goal is to improve on some of these that a user would feel are needed in a machine learning framework. The main goal is for WOLF to be user-friendly while still allowing as much customization as the user would like to perform. The three ways this project aims to accomplish this is to create a website version of WOLF, provide access to some of the most used technologies at the moment, and provide the most information to a user as possible.

The website should have all the capabilities of the command line version of WOLF. The main improvement the website provides is in ease of selecting the configuration for a workflow through a graphical user interface. It also allows a user to create an account and have different WOLF projects associated with their account.

Any good machine learning framework is going to allow the user to train a neural network on their dataset. As TensorFlow is currently one of the most popular libraries for creating deep learning models, this project aimed for learning about the framework and implementing a model in as general a way as possible.

This project added the capability for a user to have access to the trained model so that they could use it in the future. One of the main ways that one would use a trained model is to make predictions on another similar dataset. A framework to be able to do this was also added. One of the most important tools of machine learning is understanding the importance that each feature of the dataset has in the model. Displaying this information for each model type became a key goal of the project.

In order to show that the neural network and the framework as a whole worked as intended and was feasible, datasets from the UCI Machine Learning repository (Dua and Graff, 2017) were downloaded and tested. The last goal was to improve the efficiency of WOLF and fix as many bugs as possible.

Motivation

WOLF is driven by the desire to automate any task that is repeatable and provide a framework where this automation can be fully controlled through one single interface. WOLF is an effective tool when a user first starts working on it, but some flaws will be noticed right away. While WOLF allows the selection of a dataset, model, and hyper-parameters, the information returned to the user is not expansive and does not allow for much extra work. The features that this project adds are

- The framework for a website
- A neural network model using TensorFlow
- Saving trained models
- The ability to make predictions using the model
- Feature importance values
- More datasets

A website will make WOLF more accessible and provide a more user-friendly interface for users than a configuration file. This project takes steps to put a website in place.

The reason this project adds a neural network using TensorFlow is because it is becoming an industry standard to have knowledge of it. It is important to have knowledge on it for industry, but also having the option for users to have access to its capabilities is needed today. It is most commonly used for creating Neural Networks so this project improves on the existing NN by updating it.

WOLF provides the results on how each model performed and which hyper-parameters it found to be the best for that model, but it never actually provides that model to the user. This really limits the novice user who may not know how to write the code to train the model while also alienating an experienced user who wants to use WOLF to automate tasks. This project provides the ability to save the models as there is little use in determining the best model if you do not plan on using it in the future. Furthermore, since there is now a model saved, this project also adds the ability to use the model to make predictions directly in the WOLF framework.

Training a model to make predictions is not the only use for machine learning though. A major reason for the rise of data science is the desire to understand a dataset. Being able to have a model that makes good predictions is not going to move research and understanding forward if there is not a way to interpret what the model is doing. This is why the project adds recording and displaying feature importance values. Knowing the relative importance of a feature to a model can help a researcher collect better data in the future and potentially even allow them to create a model or algorithm that explains their data without the need for "black box" machine learning anymore.

To demonstrate the success in adding to the framework, this project needs to show that it can work on benchmark data. Data was collected from the UCI Machine Learning repository (Dua and Graff, 2017). These are datasets that have been used many times and are known to provide good test cases for ML. The benchmark data can also be helpful to the novice user to provide datasets for practice with WOLF and machine learning in general.

Background

WOLF

WOLF is a framework that is composed of tasks of machine learning from preprocessing of data to selecting the best model for that data. It allows a user to control each of these steps of the machine learning pipeline however they desire for a supervised classification problem. The models WOLF supports are:

- AdaBoost
- Bernoulli Naive Bayes
- Gaussian Naive Bayes
- Decision Tree (DT)
- Logistic Regression (LR)
- Random Forest (RF)
- C-Support Vector Machine(C-SVM)
- Linear Support Vector Machine (Linear SVM)
- Nu Support Vector Machine (Nu SVM)
- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- Deep Neural Network (DNN)

WOLF splits each step of the ML pipeline into an individual task but they are usually dependent on previous tasks, e.g., all models are trained separately, but the metric calculation cannot be performed until the model training is complete. Each of these tasks are called "transactions". The WOLF configuration file is a `yaml` file that is fully customizable to choose which dataset to use, which transactions to run, and how to run these transactions. The transactions you could choose from and modify are pre-processing, splitting

data, feature extraction, feature selection, model construction, metric evaluation, and model selection. A visual representation of the workflow can be seen in Figure 1.

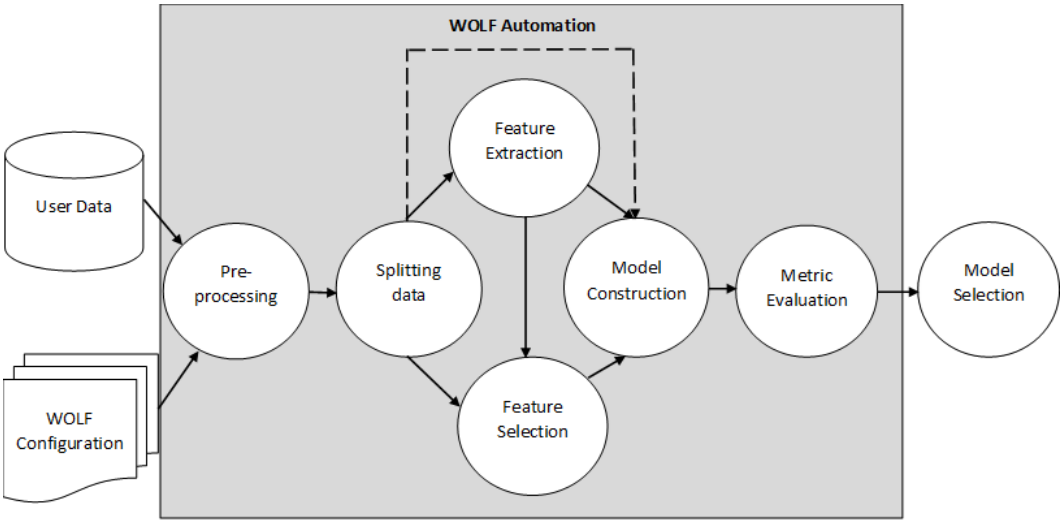

Figure 1: Original WOLF architecture (Bahl, 2016)

Not all of the transactions are mandatory. The dotted line signals that a user can skip feature extraction and/or feature selection if they want. A user also does not have do any pre-processing if their data is already in the correct format. All other transactions are required for WOLF to work though. Each transaction performs a vital step in the pipeline.

- **Pre-Processing:** Performs any step that would be needed to make the dataset complete for WOLF. Some of these actions would include replacing any missing values with a user specified value or scaling numerical values from a provided ratio. An important step in pre-processing is to encode all categorical features and labels into a numerical value.

- **Splitting Data:** Splits the data into train and test files. There are two values the user provides: number of folds and number of repetitions. The number of times is the number of train and test files to create. This allows cross-validation to be performed. The repetitions value is the number of times to perform the split. The splits are performed on a randomized order of the dataset.

- **Feature Extraction:** Uses the train and test files as input and attempts to perform dimensionality reduction. This could lead to new features being created that combine multiple features in the dataset. (Optional)

- **Feature Selection:** This step selects features that are too noisy to be meaningful or will be redundant in the model and removes them. It

then creates new train and test files with the selected features removed. (Optional)

- **Model Construction:** This is the transaction where most of the work is done. For each machine learning model type and each combination of hyper-parameters that are to be tested on, a model is trained on every train/test set combination. This will output the hyper-parameters used along with the predictions that were made from the model.

- **Model Evaluation:** After each model is trained, different metrics are calculated based on the true values and the predictions. These metrics can include accuracy, precision, area under the curve receiver operating characteristics (ROC-AUC), Matthew's correlation coefficient (MCC), F1-Score, and the capability for more. These are the metrics that will be used for choosing the best model.

- **Model Selection:** This transaction takes the model metrics as input and determines the be best model and hyper-parameter combination. All of the metrics are written to a results excel file and sorted by a chosen metric. In this file the best configuration is also given.

To keep track of the data as it flows through the pipeline, a non-relational database, MongoDB, is used. Every configuration and parameter is stored and an id value for the run is used to keep track of each WOLF configuration run. There are seven databases and each holds collections to store the values. These databases mainly follow the format of the transactions. The databases are Pre-Processing, Split Data, Files, Feature Extraction, Feature Selection, Algorithm, and Result. Each value that would be in a collection stored in the database can be seen in Figure 2.

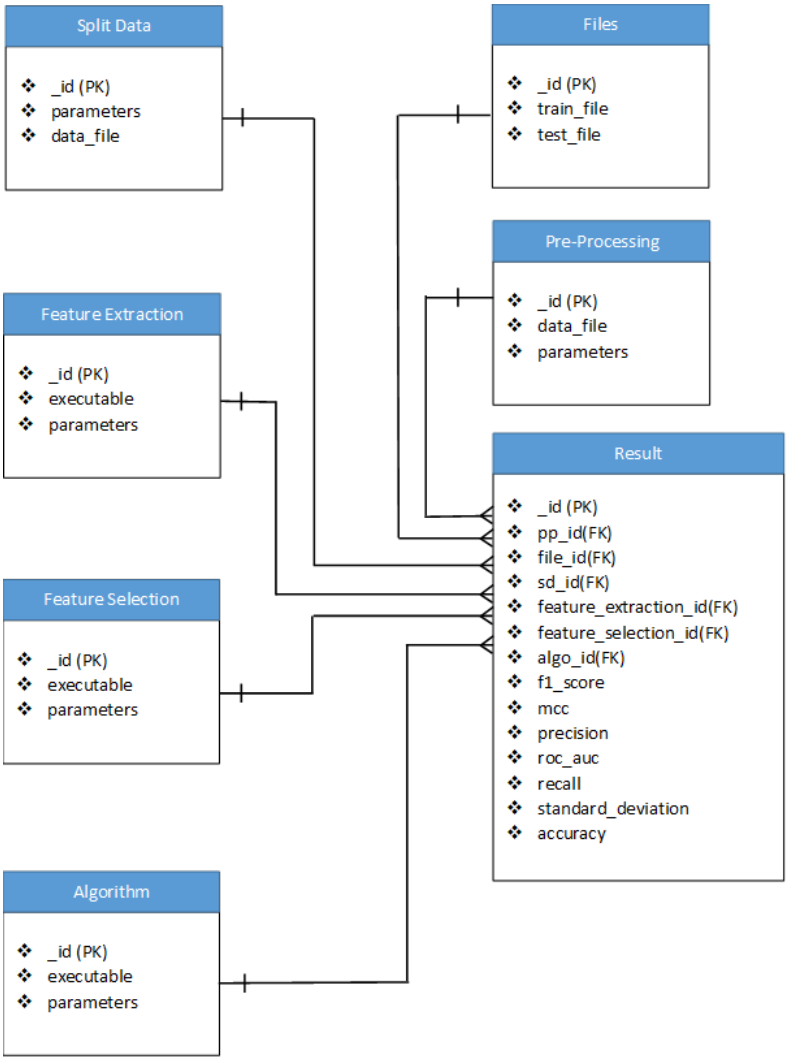

Figure 2: Original WOLF MongoDB (Bahl, 2016)

Figure 3 shows an example of a configuration file that uses all transactions and trains on three model types with varying hyper-parameter possibilities. There are two options when providing values for a hyper-parameter: "single" and "collection". If "single" is chosen, one value is provided. If "collection" is given, either "list" or "range" must chosen. A list is then provided and if "list" then those are the values tested; if "range" the values tested are all values between the first two numbers with a step size of the third.

```
cluster_config:
    email: "default_wolf_email@email.com"

preProcessing:
    executable: "PreProcessing.py"
    missing_value: 0
    label_encoding: True

datasplit :
    executable : "Splitingdata.py"
    no_of_files : 1
    data_file : "diabetes.arff"
    output_folder : "output"
    parameters :
    - ["collection","list",-k,[3,5]]
    - ["collection","list",-r,[10,15,20]]

feature_extraction:
    algo1 :
        executable : "PCA.py"
        parameters :
            - ["single",-n,8]
            - ["single",-c,True]
            - ["single",-w,False]
    no_of_files : 1

feature_selection:
    algo1 :
        executable : "SVM_RFE.py"
        parameters :
            - ["single",-n,5]
            - ["single",-s,1]
    no_of_files : 1

algorithm :
    algo1 :
        executable : "RandomForest.py"
        parameters :
            - ["collection","range",-d,[5,10,1]]
            - ["collection","range",-t,[100,200,50]]
    algo2 :
        executable : "LinearDiscriminantAnalysis.py"
        parameters :
            - ["collection","list",-s,['svd', 'lsqr', 'eigen']]
            - ["single",-v,"auto"]
            - ["single",-t,0.0001]
    algo3 :
        executable: "DecisionTree.py"
    no_of_files : 1

metric_calculation :
    no_of_files : 1
    executable : "MetricCollection.py"
```

Figure 3: WOLF configuration file example

Other Tools

One popular tool that WOLF could be compared to and was an influence is Auto-WEKA (Thornton et al., 2013). It was developed in 2013 at the University of British Columbia. It is a downloadable tool that allows classification

algorithms to be searched and for hyper-parameter optimization to be performed. It runs on 39 different classification algorithms. Tree-based Bayesian optimization is used to search through the hyper-parameter space as opposed to the grid search that WOLF provides.

Another important tool is Michelangelo (Hermann and Balso, 2019). It was created by Uber to, like WOLF, perform the machine learning workflow from beginning to end. The six steps in the workflow that Michelangelo follows are "managing data, training models, evaluating models, deploying models, making predictions, and monitoring predictions". It was made with the massive scale that Uber needs in mind and the goal of standardizing each step of the pipeline within the company. Thus, there is a shared feature store within the tool that teams can use for others to utilize in their work. Training the model returns many values on runs, including the best hyper-parameters, importance of features, and visualization of different metric scores. Deploying models is performed in a standardized way that meets the needs of the Uber model framework. Once deployed, predictions can be made on both data loaded in for testing and on real world data that can be monitored for performance. Michelangelo can be used both online and offline.

Methodology

Before this project began, the WOLF framework was largely in place, so modifications could readily be made. The first few months of work were spent going through the WOLF code to understand the workflow and be able to add in new features. When this project began, Sohaib Kiani was working on adding a neural network model that used Caffe. My role involved helping to read in and use the hyper-parameters the user provided that were to be tested. To make changes in WOLF, it is required to learn about the complete workflow to ensure understanding of what each file in the WOLF framework does and how data is passed amongst them. Work on new features could begin once the framework was understood.

The website was planned and worked on in a team setting. Goals on what would be desired in the website and how it should generally be set up were discussed. The main goal was for the version of WOLF created by Bahl (Bahl, 2016) to have every feature in the website. To be able to select every model and hyper-parameter combination, a database needed to be set up and populated with initial values. It also needed to store information on users and all runs of WOLF. The database was my contribution.

Once the work became just this project, existing tools and experiences about machine learning were analyzed to determine what would be wanted out of WOLF that was not there already. The first task was in determining if there was a better way to implement the neural network. The documentation for Caffe (Jia et al., 2014), PyTorch (Paszke, 2016) and TensorFlow (Google Brain, 2015) were studied. It was decided that the best framework to use would be TensorFlow due to its popularity and the ability to use Keras. The reason for Keras is the capability to make a dynamic model without having to reinvent the wheel.

Once all the models were integrated in that were wanted, the project moved on to saving trained models. For this, it is most common to save the models as a pickle file and the user can load that up in another file if they wish to use it. Because of how WOLF trains models, there are going to be multiple models trained for each data split. All of these models are saved into a folder specific to the model type and hyper-parameters and the results excel file provides a file path to the best model for those hyper-parameters based on a chosen metric.

After models were successfully saved, another transaction type was created

that allows for choosing some data and a model to use and then WOLF outputs predicted values to a file. This is done by reading in the data and ensuring it is in the same format as the data that trained the model. Then the model is loaded using pickle and a function is called to predict the label. The predictions are then written to a file that the user will have access to.

Calculating feature importance was done in a similar way as saving a model. Each model has its own way of determining the best features, but they all had to be put in a standardized output. For some models, there is a `feature_importances_` attribute in `scikit-learn` (Pedregosa et al., 2011) that provides a value on each parameter. For other models a `coef_` attribute is used and then the values one is interested in are taken out. For the Neural Network, ELI5 (Korobov and Lopuhin, 2016) can be used to determine the feature importances by replacing the values of a dataset feature with random values and seeing how it affects the model accuracy. This is helpful because instead of having to take out a feature and then retrain, we can keep the same model and just make predictions to see how each feature affects the model output.

I then took benchmark datasets and ran them in WOLF to both show my understanding and allow proof of WOLF's ability for a conference (Li et al., 2017). The datasets are listed in the Presentation of Work section and the results of these runs can be seen in the Results section.

Finally, some practical aspects were also explored, such as obtaining a reservation on the cluster, using different priorities, selecting processor types (CPU and GPU), choosing how many of each processor would be used, and monitoring the order of transactions.

Presentation of Work

Website

In its previous version, the accessibility of WOLF was limited to only ITTC members. Creating a website solution would allow anyone interested in running ML models easy access to WOLF. The goals in creating the website were to:

- implement all of the features of the command line version
- keep the runs of all users separate
- make WOLF more user-friendly and intuitive

Much of the work for the website was planning out the goals and what a user should expect. Deciding on the file structure, a project timeline, and the visual layout were important in getting the work started. Once this began, Lei Wang worked on all of the user interface (UI) components and matching the layout for the goals that were put in place. My work on this involved making sure the initial setup of the SQL database for storing all runs of WOLF on the website was ready. The database also stored all model and dataset types and needed to be created in such a way that a new model or dataset could be added seamlessly and appear in website drop-down menus. For the runs, all users needed to have independent views. Each user should only be allowed to view their own runs and models. This was performed by having "projects" that could be made public or private. The database is setup in a way that the owner of the project would also have the ability to add other collaborators to the project and allow them to make changes and perform their own runs.

Other work on the website included working on collecting model run data and putting it into tables for the 2017 IEEE International Conference on Big Data, at which other members of the team presented. A portion of the slides for the presentation (Li et al., 2017) were created using this data. Another portion of slides on related works, such as Auto-Weka and Michelangelo, were also created. All this information was also included in an unpublished report (Kiani et al., 2017).

Neural Network

When this project began, deep learning was possible in WOLF with a Neural Network created using Caffe. Caffe is a framework developed by Berkeley AI Research for deep learning (Jia et al., 2014). It supports both CPU and GPU construction. In the past it had been a popular framework to use for creating neural networks, but in recent times it has been slowly falling out of use as data scientists are turning towards other libraries (Hale, 2018). From the same site, it can be seen that the most popular frameworks are currently TensorFlow and PyTorch. Both of these are extremely popular in industry at this time, with TensorFlow showing no signs of stopping and PyTorch quickly rising.

PyTorch (Paszke, 2016) is a framework built using Torch, which is a tensor library for GPUs. It is extremely helpful for creating reusable neural networks. The way that it runs is using "reverse-mode auto-differentiation," which allows for changes to be made to the model with little overhead. PyTorch is meant to be easy to learn and has a strong community contributing to the open-source software. TensorFlow was created by Google as a way to perform computations on CPUs or GPUs (Google Brain, 2015). TensorFlow takes advantage of data flow graphs of computations and multi-dimensional arrays called tensors. Like PyTorch and Caffe, it can be used for end-to-end machine learning workflow creation. With the large market share that TensorFlow has, it is apparent that learning how to use it is valuable for working in industry. It would also be useful for a user to be able to train a neural network using TensorFlow.

With TensorFlow decided on, the best ways to integrate it into WOLF were explored. Keras appeared to be the best tool to use to implement a TensorFlow neural network. Keras (Chollet, 2015) is an API built to run on top of TensorFlow. Essentially it is a version of TensorFlow built for high level creation of neural networks. It is still the second most popular framework to build neural networks behind TensorFlow itself. Besides being extremely popular and manageable to understand, Keras also felt like Scikit-Learn (Pedregosa et al., 2011) in the way the function calls were set up, making it an easy transition in terms of syntax and fitting the calls into the WOLF model template style. To help write the code, a tutorial (Brownlee, 2016) was used.

Just like the Caffe version, the neural network models using TensorFlow and Keras allow for a selection of a variety of hyper-parameters. These hyper-parameters are:

- the number of layers and nodes in each layer as a python list
- activation function
- input dropout
- hidden dropout

- learning rate
- number of epochs
- batch size

Saved Model and Predictions

One major feature that WOLF was lacking was the ability to return a trained model to the user. As it stood in its previous form, there was only the ability to learn which hyper-parameters were the best for the dataset. When WOLF trains a model, it trains on each train/test data split pair. That means that a new model is created for each pair. When each model is trained, it is then saved to a file using pickle. For each hyper-parameter set, a folder is made with the same number of models as the dataset splits. The hyper-parameters with the best metrics allow for selection of the best of these models. The full path to the best pickled model is provided to the user in the results. If the user chooses to select a different model based on a different metric or a combination of multiple metrics, all models are still saved and one can match up the metric file with a model file to choose a different pickled file path than the results file provided.

Once a user chooses a model that they like, WOLF now allows a user to edit the configuration file to make predictions on a dataset. WOLF will read in the dataset, confirm it is in the correct form or try to format it if not, and then load the pickled model. Using this model, `predict` can be called and the predictions can be written to a file that a user can download and use how they wish.

Feature Importance

Once one has a trained model, it can be used to understand more about the dataset. A key area of model understanding is knowing the importance of each feature in the dataset. These values are the relative importance each feature of the dataset has on the prediction outcome of a model. That means these feature importance values are calculated from the model itself instead of from the dataset alone as feature extraction and feature selection do. If the model is good, these importances should line up with the overall importance of the feature in the dataset and real world.

To calculate these importances, some models inherently have feature importance in the way the model works. These models, which have a built in attribute in `sklearn` called `feature_importances_`, are random forest, decision trees, and adaboost classifier. Other models required more work to calculate. For most others except for neural networks, the coefficients matrix could be used in the calculations. The coefficients matrix is an attribute,

`coef_`, of many algorithms in the `sklearn` library. The values in the matrix are positive if there is a correlation between the feature and the label being true and are negative if there is a correlation between the feature and the label being false. The further the value is from 0, the greater the correlation. To match the format of the models with the `feature_importances_` attribute, `coef_` values need to be normalized between 0 and 1. The closer to 1 a value is, the more important the feature is. To normalize the values, the following equation is used, where Z is the calculated importance value matrix and C is the coefficient matrix:

$$Z_i = \frac{|C_i|}{\sum_j |C_j|}$$

For the neural network using Keras, there is not a built in way to measure feature importance. Because of the complexity of neural networks, they are inherently not transparent or interpretable. There is much research being done in this area (Lipton, 2016), but in the meantime, a simpler, more brute force way of finding feature importance must be done. A common way to do this is to perform a "leave one out" strategy (Feng, Chen, and Xu, 2013). Because we do not want to train the model again for each feature, we instead can simulate taking out a feature by replacing all of the values with random values and making predictions on the training set. A package called ELI5 (Korobov and Lopuhin, 2016) was used for this.

Once the feature importances are calculated, they are sorted and written to a file. Just like saving a model, this is done for each hyper-parameter combination and each train-test pair. All of the feature importance values are stored in a single file and the relative feature importances of the best configuration can be viewed in the main results file.

Datasets

For showcasing the capability of WOLF at finding a model with optimal hyper-parameters, many datasets were downloaded from the UCI Machine Learning repository (Dua and Graff, 2017) that houses datasets for tasks of binary classification, multiclass classification, regression, and more. These datasets had to be downloaded and converted to `arff` format while also confirming that the features and classes would work in WOLF or be able to pass through the pre-processing transaction to then work in WOLF. For binary and multiclass classification the datasets currently available are:

- banknote
- breast cancer
- climate
- congress

- credit card
- dermatology (multiclass)
- diabetes
- fertility
- iris (multiclass)
- leaf (multiclass)
- pageblocks (multiclass)
- sonarmines
- spam
- transfusion
- vertebral
- voice rehab
- whitewine (multiclass)
- wholesale

The results of running WOLF can be seen in the results section.

New Architecture

After the implementation of these new features, the architecture of WOLF is changed from Figure 1 to that in Figure 4. As can be seen in the graphic in green, saving models using pickle, feature importance, data prediction, and a neural network have been added. The website has also been added that incorporates every feature of WOLF. The prediction task is connected to the data by a dotted line as it is optional. It also is not connected to any of the other tasks, except for pickling a model. But this pickling must be done on a previous run of WOLF, otherwise the user will not know which model to tell WOLF to use.

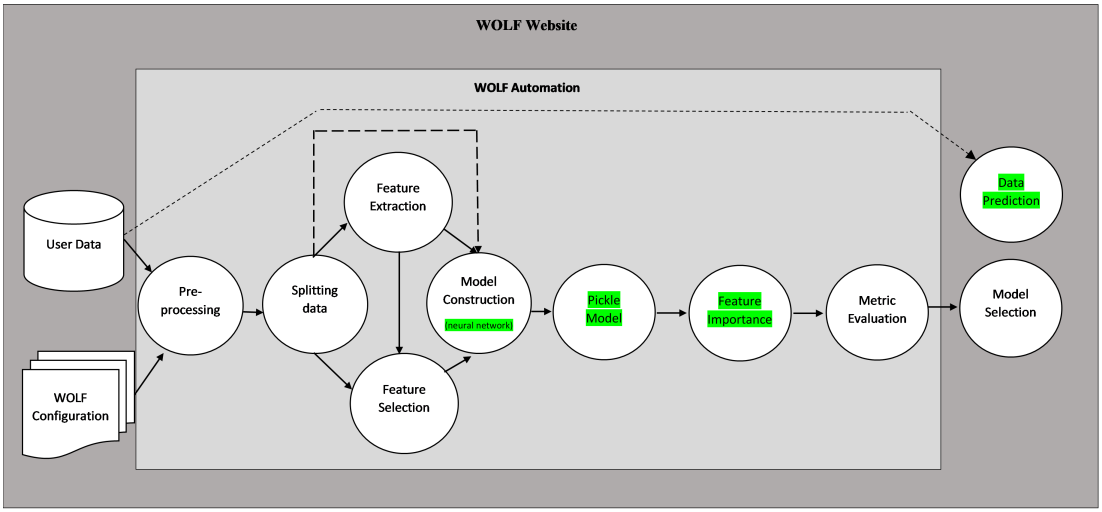

Figure 4: New WOLF architecture

The MongoDB architecture has also changed slightly. Everything is the same as in Figure 2 except for the Results database. The results database has gained two fields: `model_path` and `feature_importance_path`. These are included so they can be passed to the metric collection and database results file and used appropriately for writing to the results excel file. The change can be seen in Figure 5.

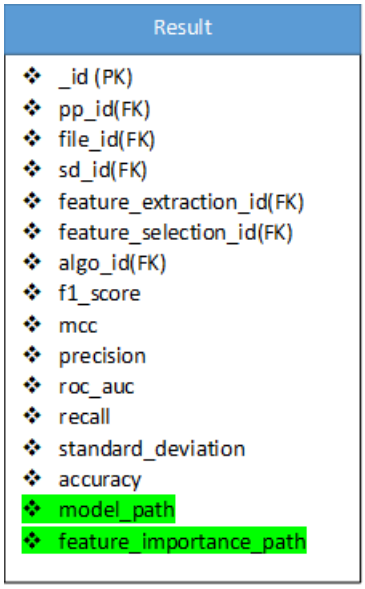

Figure 5: New WOLF results database

Results

In this section, the results of the main areas this project focused on will be presented. These areas were: a website version of WOLF, a neural network model using TensorFlow, saved models, predictions from models, feature importance, and more benchmark datasets.

Website

After setting up and logging into an account on the website, users will be brought to the home screen where information about all activities and components can be seen (Figure 6).

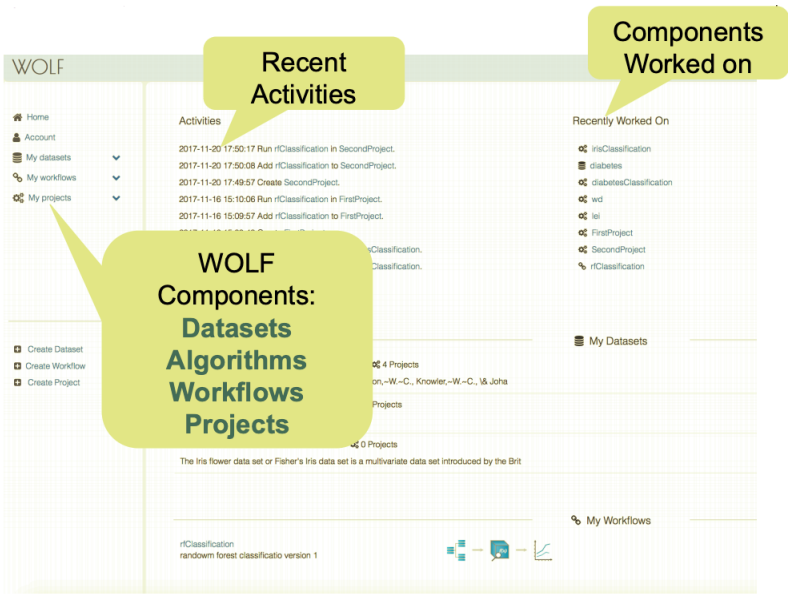

Figure 6: WOLF homepage (Li et al., 2017)

If a user wants to create a workflow, they are asked to give it a name and a description as well as provide the parameters for each transaction. Figure 7 shows this, along with the data split transaction.

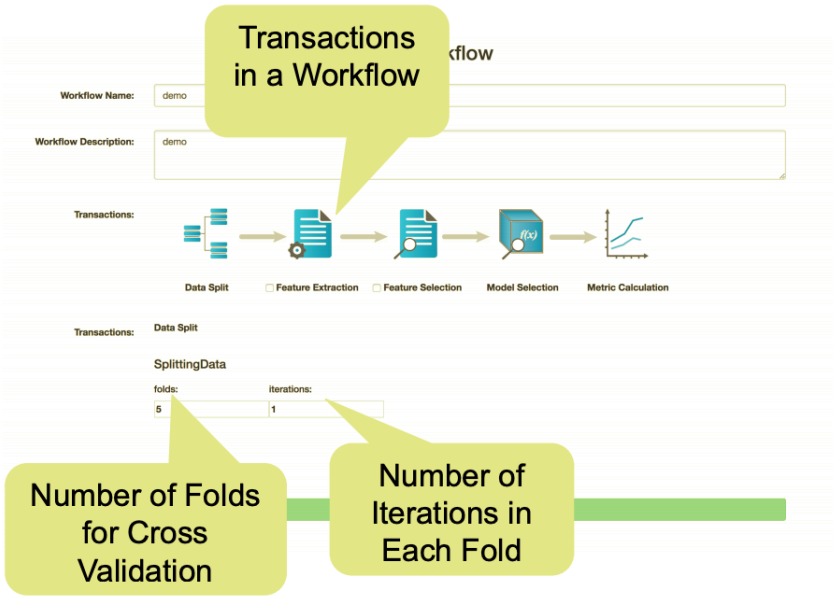

Figure 7: WOLF transactions (Li et al., 2017)

The model selection transaction (Figure 8) shows a list of all the possible models to pick from. When the user selects one, it moves to the list on the right and also adds a section below where the hyper-parameters to be tested can be chosen.

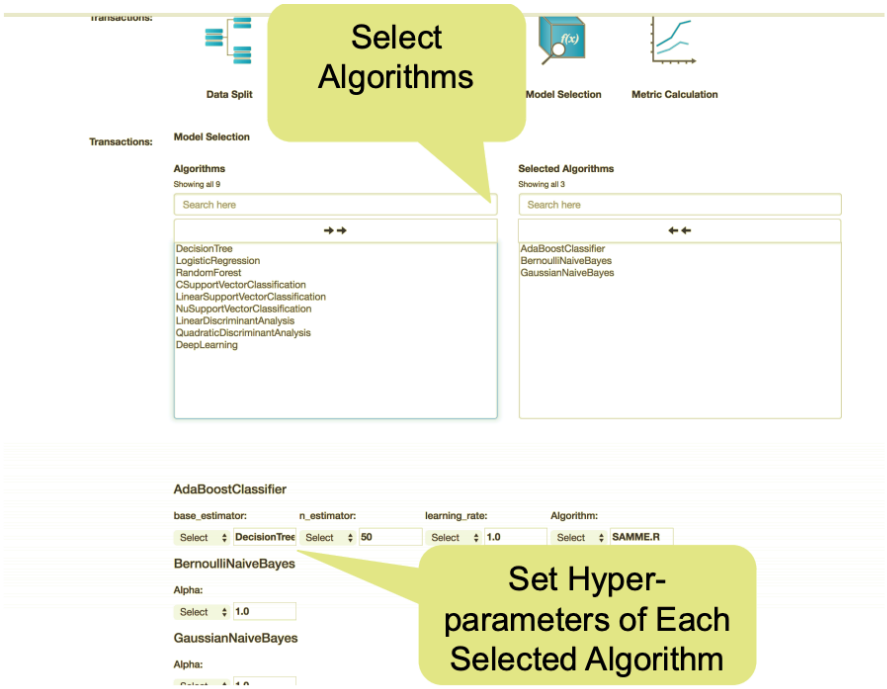

Figure 8: WOLF models selection (Li et al., 2017)

Once the run of the workflow is completed, the details of the run are shown and a results file can be chosen. This is the one part of the website that currently does not work. But Figure 9 shows how it should look when back in a working state. Some of the information shown includes when the job was submitted, when it was completed, the current state of the run, and, most importantly, a link to download the results file.

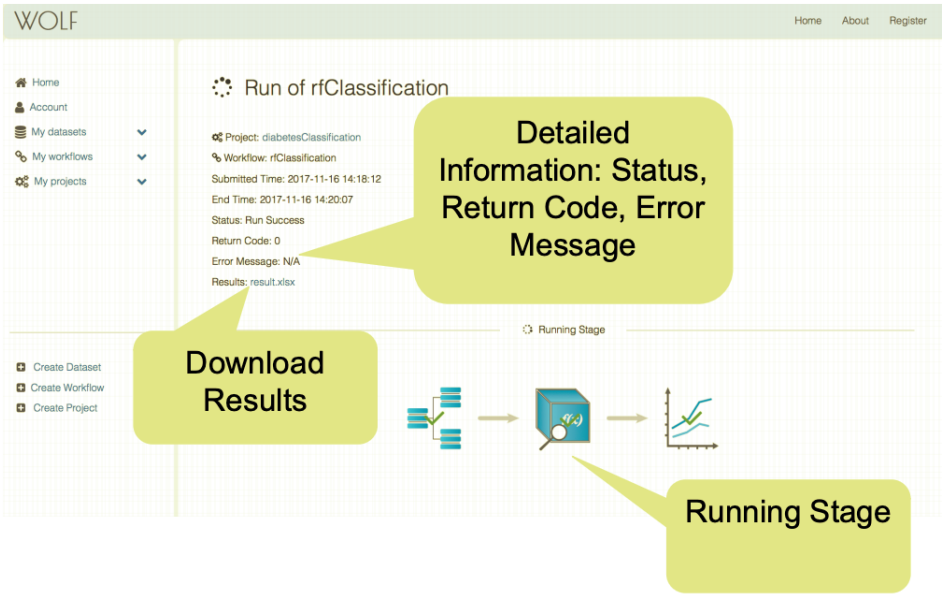

Figure 9: WOLF run details (Li et al., 2017)

Neural Network

The neural network was implemented simply and successfully. Keras was helpful in this and allowed for looping through the number of layers needed and adding each with the provided number of nodes. Then a sequential model is compiled with all of the hyper-parameters. Table 1 shows all of the hyper-parameters supported and their default values if the user does not provide one. TensorFlow for GPUs is used to run on the ITTC cluster to have faster train times than running on the CPUs. At this time, only one GPU can be used or else it becomes difficult to be assigned nodes and the job becomes stuck waiting in the queue.

Parameter flag	Description	Default value
-a	activation function	“relu”
-l	layers	[100,100,100]
-d	input layer dropout	0
-h	hidden layer dropout	0.5
-e	epochs	10
-b	batch size	None
-r	learning rate	0.001

Table 1: TensorFlow neural network hyper-parameters

Saved Model and Predictions

Saving each model to a pickle file creates a different folder of models for each hyper-parameter and model type combination. For each data split, there is one model in the folder. Figure 10 shows the file hierarchy and the list of files for a run with 5 folds.

```
[[csader@login1 Splittingdata1]$ ls
DecisionTree_result   splitdatafiles.yaml  test_2.csv  test_3.h5   test_5.csv   train_1.h5   train_3.csv  train_4.h5
RandomForest_result1  test_1.csv          test_2.h5   test_4.csv  test_5.h5    train_2.csv  train_3.h5   train_5.csv
RandomForest_result2  test_1.h5           test_3.csv  test_4.h5   train_1.csv  train_2.h5   train_4.csv  train_5.h5
[[csader@login1 Splittingdata1]$ cd RandomForest_result1
[[csader@login1 RandomForest_result1]$ ls
FeatureImportance  metric2.yaml  metric4.yaml  models       result2.csv  result4.csv  results.yaml
metric1.yaml       metric3.yaml  metric5.yaml  result1.csv  result3.csv  result5.csv
[[csader@login1 RandomForest_result1]$ cd models/
[[csader@login1 models]$ ls
RandomForestModel1.pkl  RandomForestModel2.pkl  RandomForestModel3.pkl  RandomForestModel4.pkl  RandomForestModel5.pkl
```

Figure 10: WOLF saved models

The best model is selected based on the highest metric (ROC-AUC by default) for the chosen model type and hyper-parameter configuration. This best model will have its file path given in the “best configuration” tab of the results file that can be seen in Figure 11, which was trained on the diabetes dataset.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1		accuracy	algo_name	algo_param	f1_score	mcc	precision	recall	roc_auc	model path									
2	0	0.7629081	RandomForest	{'c': 'gini', 'd': '10', 'm': '0', 'l': '1', 'p': '1e-7', 's': '2', 't': '50', 'w': 'False'}	0.6306859	0.4634382	0.69188302	0.58287212	0.7211361	/WOLF_CL/output/Splittingdata1/RandomForest_result1/models/RandomForestModel1.pkl									
3																			
4		name	importance																
5		plas	0.368909207																
6		mass	0.147325008																
7		age	0.130517471																
8		pedi	0.094459587																
9		insu	0.076882703																
10		preg	0.074408122																
11		skin	0.056018316																
12		pres	0.051479585																
13																			

Figure 11: WOLF results file

For making predictions, these will be written to a file called `predictions.csv`, which will have as many rows as there were data points to predict. There will only be one column in the file for the class which is to be predicted, but if multi-label classification is added to WOLF, then there will be a column for each label to be predicted.

Feature Importance

In calculating feature importance, just like saving models, there will be a folder of feature importances for each model type and hyper-parameter combination. And also just like the models, there is a different file for each data split as each trained model will have different feature importance values. This file hierarchy for a 5 folds data split can be seen in Figure 12. The feature importance values for training on the diabetes dataset can be seen in Figure 11. The features are sorted by importance.

```
[[csader@login1 Splittingdata1]$ ls
DecisionTree_result   splitdatafiles.yaml  test_2.csv  test_3.h5   test_5.csv   train_1.h5   train_3.csv  train_4.h5
RandomForest_result1  test_1.csv          test_2.h5   test_4.csv  test_5.h5    train_2.csv  train_3.h5   train_5.csv
RandomForest_result2  test_1.h5           test_3.csv  test_4.h5   train_1.csv  train_2.h5   train_4.csv  train_5.h5
[[csader@login1 Splittingdata1]$ cd RandomForest_result1/
[[csader@login1 RandomForest_result1]$ ls
FeatureImportance  metric2.yaml  metric4.yaml  models       result2.csv  result4.csv  results.yaml
metric1.yaml       metric3.yaml  metric5.yaml  result1.csv  result3.csv  result5.csv
[[csader@login1 RandomForest_result1]$ cd FeatureImportance/
[[csader@login1 FeatureImportance]$ ls
RandomForestFI1.csv  RandomForestFI2.csv  RandomForestFI3.csv  RandomForestFI4.csv  RandomForestFI5.csv
```

Figure 12: WOLF feature importance files

Datasets

WOLF was run on all of the benchmark datasets to test the effectiveness of the workflows. The results for the binary classification datasets can be seen in Table 2. All scores are the ROC-AUC values that have been calculated for each model type and each dataset.

Dataset Name	RF	DT	Lin SVM	Log Reg	BNB	LDA	Ada Boost	NN - Caffe	NN - TF
Bank note auth.	0.9923	0.9810	0.9889	0.9896	0.8419	0.9786	0.996	0.9998	0.9999
Blood Transfusion	0.6279	0.5852	0.5323	0.5495	0.4993	0.5419	0.6181	0.5003	0.5427
Climate Sim. Crashes	0.5501	0.6527	0.7941	0.5773	0.5000	0.7158	0.7535	0.6581	0.7570
Sonar, Mines/Rocks	0.8167	0.6919	0.7692	0.7507	0.5051	0.7358	0.7954	0.6100	0.7662
Default of credit card	0.6540	0.6081	0.5217	0.4999	0.6731	0.6127	0.6388	0.5000	0.5000
Fertility	0.5531	0.4964	0.4960	0.4988	0.5000	0.4878	0.5375	0.5223	0.5333
Voice Rehabilitation	0.7831	0.7351	0.5058	0.5529	0.6854	0.7256	0.7916	0.6344	0.4934
Pima Indians Diabetes	0.7216	0.6412	0.5597	0.7151	0.5035	0.7253	0.7131	0.6864	0.7324
Spambase	0.9323	0.9025	0.8252	0.9215	0.8736	0.8699	0.9345	0.6706	0.8887
Vertebral Column	0.8058	0.7592	0.7261	0.8095	0.6438	0.8017	0.7917	0.8101	0.7619
Wholesale customers	0.9053	0.8520	0.6939	0.8765	0.5000	0.7748	0.8800	0.5000	0.5348

Table 2: WOLF benchmark data ROC_AUC

Future Work

When making any tool, there are always going to be features one wishes they had time to implement. Here are some that have already been started or that have been talked about as something that is needed.

Website

Work on a website version of WOLF has begun and is nearly deployable. It currently allows the loading of a dataset, selection of which transactions to run, and selection of the hyper-parameters to test on. It currently only fails in signaling that the run is complete and providing the results back to the user, which was not a part of the website I was assigned to work on. However, the results file is created and can be found.

There is also the possibility of adding more website-specific features such as visualization or the ability to work on a workflow as a group. The current website and SQL database were created with the intention of WOLF being like a social network for running machine learning models. If this is still the goal, then the current setup could be used and completed. If this is not the case, then they could be used as a reference for a brand new project.

My recommendation would be to use the UI as a visual reference either way and switch from the existing PHP code to JavaScript for its popularity and security. Tracking the run and updating the database and website also needs to be looked into as that is the main problem right now. The cause is that there is something wrong with the watcher files stored in a jar file. The files in here are Java binaries, and we do not have the source code. As for the SQL database, it is a good database to start with for functionality, but more security features should be looked into, including adding increased security through Linux permissions.

Feature importance

After completion of this project, for feature importance WOLF lists the features and a number in a sorted format to show the order of the relative importance of the features to the selected model. This is better than nothing, but should be improved upon. Visualizations would be one way to do this.

Another way to improve would involve looking into more ways to calculate feature importance in the dataset, e.g., explained variance (Gron, 2017).

ELI5 was used for the neural network, but it does have the capability to run on Scikit-Learn models. Looking into if it is better to use ELI5 instead of, or in addition to, the existing `sklearn` methods would be a beneficial exploratory task. It would provide more information to the user and multiple perspectives that could confirm the idea of which features are the most important.

Local runs

Currently to run WOLF, a user must have an ITTC account and then run on the ITTC cluster by submitting the job through the login servers. This is fine for KU users, but requires an internet connection, space on the cluster to be requested, and alienates non-KU users. A working website would remove the need to be affiliated with KU, but still does not solve the need for internet or the potential for a long wait on the cluster. Having a version that can be downloaded and run locally using both a user's CPU and GPU would be an optimal solution to this problem.

To make WOLF open source and reproducible, the first thing that would need to be changed is the submission scripts which have the templates in `Wolf.py` for the cluster. These would need to be changed to run on a local machine. I have not attempted this too deeply, but one idea is that the cluster configuration section before the python calls could be removed and it might work by just creating scripts that make `python` calls. This would need to be tested for Windows, Mac, and Linux. Chances are this might work for Mac and Linux, but not Windows. It would also need to be looked into if the dependencies that `Wolf.py` sets up for the submission script running order is consistent with a standard terminal or only the cluster. There would also be a need to setup a WOLF MongoDB environment for the user and have all of the necessary packages installed using `pip`. A script could be created to do both of these tasks. It may also be necessary to create separate versions of WOLF for different operating systems. Another area that could be beneficial would be to create an executable that could be downloaded and run by a user of WOLF. This could either be an executable that installs everything for them, or could even be used to run WOLF for users that do not care about editing or knowing about anything except the configuration files.

Another option that could be looked into is using Docker (Docker Inc., 2019). It is an open-source software tool that allows for applications to be built in Docker containers and shared as one package. It can then be run on any Linux machine making Docker similar to running WOLF in a virtual machine. There are also versions of Docker for creating containers for Windows and Mac.

Image data

One of the most popular uses of machine learning is image recognition, but WOLF currently does not support this. One of the main reasons is the need for a dataset to be uploaded as an `arff` file that WOLF then uses to split and create `csv` files. Having a way to signify that the data will be images and handle that appropriately would be important to the future success of WOLF.

This could just entail having the feature values in the `arff` file be the names of each image, but this would need to be tested thoroughly and my suspicions are that WOLF would not support this technique, especially for calculating feature importances. Alternatively, my recommendation would be to create a key in the `yaml` file which can be chosen for image data upon which a different method for pre-processing and datasplitting will be performed. This could either involve creating features from each pixel in the image or, maybe best case, finding a way to use the file names of the images in the `csv` files from the data splits. If each pixel value is used to create a feature instead of using the filename, I assume it will be required that the images in the dataset are the exact same dimensions.

Python 3

Support for Python 2.7 will end on January 1, 2020 and because of this, any improvements to WOLF should be made in Python 3 and all existing code should be ported over to Python 3. It can be assumed that when Python 2.7 support ends, Scikit-Learn will also stop any support for new features for the 2.7 version. There are already warning messages that some functions will be depreciated in future versions of Scikit-Learn. It is not a priority for WOLF in its current state and the code will still work, but with any improvements, this should be done.

The changes needed would mainly be syntax changes and any function calls that have changed in the module version updates. It would also be needed to make sure that python 3, the Scikit-Learn module, and the TensorFlow for gpu module is updated on the ITTC cluster. Finally, all `python` calls in the submission script templates found in `Wolf.py` would need to be changed to `python3` or however the cluster support chooses to have the call at that time.

More Model Types

WOLF currently contains some of the most popular machine learning models today, but it is not a complete list, even for classification. Adding any more would be a bonus. There is also a whole area of work that needs to be added for regression. Some of the Scikit-Learn models have the ability to perform either classification or regression, such as linear regression, logistic regression, and

random forest, but most that are currently in WOLF are only setup to perform classification. There is a metrics python file, `MetricCollection_Regression.py`, that has been started for regression as well, but it is not in a state to be put in fully to WOLF yet.

GPU Reservation

In the current state of the ITTC cluster, there are a small number of GPUs. This makes it difficult to run WOLF using neural networks if the cluster is even remotely busy. One potential fix would be to gain access to a reservation of GPUs that only WOLF can access. Another potential fix would be to have more GPUs be added to the cluster.

Conclusion

In summary, the foundation for a website has been set up and is nearly ready for public use. There has also been a successful implementation of a neural network model that runs on TensorFlow and Keras. Trained models are now saved and able to be used in making predictions within the WOLF framework. The importance of features in relation to the models are also saved and displayed to a user of WOLF to help with interpreting the models. Some extra datasets are available for users to practice on and learn about optimizing WOLF. Also, optimizations have been made to the workflow on the cluster to decrease the time to wait for nodes to run on and increase the efficiency of running on nodes.

This project was a great way to learn about the machine learning pipeline and how to write code that meets industry standards. It is important to create software that allows for new pieces to be easily added and the files in WOLF are purposely created in such a way that a new model or other transaction in the pipeline can be added by following the template that all the existing files follow. This project can be a guide to help future students who would like to further improve WOLF.

Bibliography

[1] Pranav Bahl. *WOLF: machine learning WOrk fLow management Framework*. 2016.

[2] Jason Brownlee. *Develop Your First Neural Network in Python With Keras Step-By-Step*. 2016. URL: https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/.

[3] François Chollet et al. *Keras*. https://keras.io. 2015.

[4] Docker Inc. *Docker Documentation*. 2019. URL: https://docs.docker.com/.

[5] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[6] Dingcheng Feng, Feng Chen, and Wenli Xu. "Efficient leave-one-out strategy for supervised feature selection". In: *Tsinghua Science and Technology* 18.6 (2013), pp. 629–635.

[7] Google Brain. *TensorFlow GitHub*. 2015. URL: https://github.com/tensorflow/tensorflow.

[8] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017, p. 214. ISBN: 1491962291, 9781491962299.

[9] Jeff Hale. *Deep Learning Framework Power Scores 2018*. 2018. URL: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a.

[10] Jeremy Hermann and Mike Del Balso. *Meet Michelangelo: Uber's Machine Learning Platform*. 2019. URL: https://eng.uber.com/michelangelo/.

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[12] Sohaib Kiani, Xiaoli Li, Pranav Bahl, Casey Sader, and Jun Huan. *WOLF: Machine Learning Workflow Management Framework*. 2017.

[13] Mikhail Korobov and Konstantin Lopuhin. *Eli5*. 2016. URL: https://eli5.readthedocs.io/en/latest/overview.html.

[14] Xiaoli Li, Sohaib Kiani, Pranav Bahl, Casey Sader, and Jun Huan. *WOLF: Machine Learning WOrkfLow Management Framework*. Boston, MA, 2017. URL: http://cci.drexel.edu/bigdata/bigdata2017/files/Tutorial3.pdf.

[15] Zachary Chase Lipton. "The Mythos of Model Interpretability". In: *CoRR* abs/1606.03490 (2016). arXiv: 1606.03490. URL: http://arxiv.org/abs/1606.03490.

[16] Adam Paszke. *PyTorch GitHub*. 2016. URL: https://github.com/pytorch/pytorch.

[17] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, and et. al. "Scikit-learn: Machine Learning in Python". In: *The Journal of Machine Learning Research*. 2011, pp. 2825–2830. URL: https://scikit-learn.org/.

[18] Chris Thornton, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms". In: *Proc. of KDD-2013*. 2013, pp. 847–855. URL: https://www.cs.ubc.ca/labs/beta/Projects/autoweka/papers/autoweka.pdf.

Appendix

The following are two examples of configuration files for WOLF. The first is the new transaction for making predictions. It shows the required values of the prediction executable script `MakePredictions.py`, the data file to predict with, the desired name of the prediction output (`-o`), and the file path to the pickled model (`-m`). Then there are two optional parameters for the label to predict (`-l`) and whether to compute metrics (`-s`, currently only accuracy) on the predictions. The label name must be given if it is included in the dataset and the metrics can only be calculated if a label name is provided. The second configuration file shows examples of some transactions. These are `datasplit`, `feature_extraction`, `feature_selection`, `algorithm`, and `metric_collection`. Many examples of algorithms and some possible hyper-parameter combinations are also shown.

```
cluster_config:
    email: "c455s614@ku.edu"

model_prediction :
    executable : "MakePrediction.py"
    no_of_files : 1
    data_file : "diabetes.arff"
    parameters :
    - ["single",-o,"predictions.csv"]
    - ["single",-m,"models/RandomForestModel1.pkl"]
    - ["single",-l,'class']
    - ["single",-s,'True']
```

Figure 13: WOLF configuration file for predictions

```
cluster_config:
    email: "c455s614@ku.edu"

datasplit:
    executable: "Splittingdata.py"
    no_of_files: 1
    data_file: "diabetes.arff"
    output_folder: "output"
    parameters:
    - ["single",-k, 5]
    - ["single",-r, 1]
    - ["single",-l,"class"]

feature_extraction:
    algo1 :
        executable : "PCA.py"
        parameters :
            - ["single",-n,8]
            - ["single",-c,True]
            - ["single",-w,False]
    no_of_files : 1

feature_selection:
    algo1 :
        executable : "SVM_RFE.py"
        parameters :
            - ["single",-n,5]
            - ["single",-s,1]
    no_of_files : 1

algorithm :
    algo1 :
        executable : "BernoulliNaiveBayes.py"
        parameters :
            - ["collection","list",-a,[0,1.0]]
    algo2 :
        executable : "DecisionTree.py"
        parameters :
            - ["collection","list",-d,[4,8]]
            - ["collection","list",-p,['True', 'False']]
    algo3 :
        executable : "LogisticRegression.py"
        parameters :
            - ["collection","list",-p,['l1', 'l2']]
    algo4 :
        executable : "RandomForest.py"
        parameters :
            - ["collection","range",-d,[5,10,1]]
            - ["collection","range",-t,[100,200,50]]
    algo5 :
        executable : "CSupportVectorClassification.py"
        parameters :
            - ["collection","list",-k,['rbf','linear']]
    algo6 :
        executable : "LinearDiscriminantAnalysis.py"
        parameters :
            - ["collection","list",-s,['svd', 'lsqr', 'eigen']]
            - ["single",-v,"auto"]
            - ["single",-n,1]
            - ["single",-t,0.0001]
    no_of_files : 1

metric_calculation :
    no_of_files : 1
    executable : "MetricCollection.py"
    output_folder : "output"
```

Figure 14: WOLF configuration file from presentation