Journal of Artificial Intelligence Research ? (YYYY) 1-?? Submitted M/YY; published M/YY

Standard Machine Learning Language: A Language Agnostic Framework for Streamlining the Development of Machine Learning Pipelines

Kelechi Ikegwu IKEGWU2@ILLINOIS.EDU
226 Astronomy Building, MC-124
1002 W. Green St.
Urbana, IL 61801

Micheal Hao ???@ILLINOIS.EDU
???

Robert Brunner BIGDOG@ILLINOIS.EDU
226 Astronomy Building, MC-221
1002 W. Green St.
Urbana, IL 61801

Abstract

Standard Machine Learning Language (SML) is a language agnostic framework that integrates a query-like language to simplify the development of machine learning pipelines. Emphasis was placed on ease of use and abstracting the complexities of machine learning from the end user encouraging it's use in professional and academic settings for a variety of disciplines. SML's architecture is discussed, followed by multiple interfaces that one could use to interact with SML. We then apply SML to a few research problems and compare the complexities of SML queries and traditional code used to solve problems. Lastly we perform a case study on SML. The source code and documentation for SML is open sourced and publicly available on github [1].

1. Introduction

Machine Learning has simplified the process of solving a vast amount problems in a variety of fields by learning from data. In most cases machine learning has become more attractive than manually creating programs to solve these same issues. However there's a multitude of nuisances involved when developing machine learning pipelines (Domingos, 2012). If these nuisances are not taken into consideration one may not receive satisfactory results. A novice with domain knowledge utilizing machine learning to solve a particular problem may not want or have the time to deal with these complexities. To combat these issues we introduce Standard Machine Learning Language (SML).

The overall objective of the SML is to provide a level of abstraction which simplifies the development process of machine learning pipelines. Consequently this enables students, researchers and industry professionals without a background in machine learning to solve problems. We developed a query like language to which serves as an abstraction from writing actual machine learning code see Figure 1 for an example. In the subsequent sections related works are discussed followed by defining the grammar used to create queries for SML. The

1. https://github.com/lcdm-uiuc/sml

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND CLASSIFY
(predictors = [1,2,3,4], label = 5, algorithm = svm)
```

Figure 1: Example of a SML Query performing classification.

architecture of SML is described, lastly SML is applied to use-cases to demonstrate how it reduces the complexity of solving problems that utilize machine learning .

2. Related Works

TPOT (?) is a tool implemented in python that creates and optimizes machine learning pipelines using genetic programming. Given cleaned data, TPOT performs feature selection, preprocessing , and construction. Given the task (classification, regression, or clustering) it uses the best features to determine the most optimal model to use. Lastly, it performs optimization on parameters for the selected model. What differentiates SML from TPOT is that in addition to feature, model, and parameter selection/optimization a framework is in place to apply these models to different datasets and construct visualizations for different metrics with each algorithm.

3. Grammar

The SML language is a domain specific language with grammar implemented in Bakus-Naur form (BNF). Each expression has a rule and can be expanded into other terms. Figure 1 is an example of how one would perform classification on a dataset using SML. The query in Figure 1 reads in a dataset, performs a 80/20 split of training and testing data respectively, and performs classification on the 5th column of the hypothetical dataset using columns 1,2,3,4 as predictors. In the subsequent subsections SML's grammar in BNF form is defined in addition to the keywords.

3.1 Grammar Structure

In this subsection we define the grammar of SML in terms of BNF. A query can be defined by a delimited list of actions where the delimiter is an AND statement; with BNF syntax this is defined as:

$$< Query >::=< Action > | < Action > AND < Query > \tag{1}$$

An action in (1) follows one of the following structures defined in (2) where a keyword is required followed by an *Argument* and/or *OptionList*.

$$\begin{aligned} < Action >::=& < Keyword > \ < Argument > \\ | & < Keyword > \ < Argument > (< OptionList >) \\ | & < Keyword > (< OptionList >) \end{aligned} \tag{2}$$

A keyword is a predefined term associating an Action with a particular string. An Argument generally is a single string surrounded by quotes that specifies a path to a file.

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND CLASSIFY
(predictors = [1,2,3,4], label = 5, algorithm = svm)

<Keyword> <Argument>
AND <Keyword> (<OptionList>) AND <Keyword>
(<OptionList>)
```

Figure 2: Transcribing Figure 1 into BNF form. Where the top query was defined in Figure 1 and the bottom statement is in BNF format.

Lastly, an *Argument* can have a multitude of options (3) where an *Option* consist of an *OptionName* with either an *OptionValue* or *OptionValueList*. An *OptionName*, and *OptionValue* consist of a single string, an *OptionList* (4) consist of a comma delimited list of options and an *OptionValueList* (5) consist of a comma delimited list of *OptionValues*.

$$\begin{aligned} < Option > ::= & < OptionName > \ = \ < OptionValue > \\ & | < OptionName > \ = [< OptionValueList >] \end{aligned} \tag{3}$$

$$< OptionList > ::= < Option > \ | < Option > \ , \ < OptionList > \tag{4}$$

$$\begin{aligned} < OptionValueList > ::= & < OptionValue > \\ & | < OptionValue > \ , \ < OptionValueList > \end{aligned} \tag{5}$$

To put the grammar into perspective the example query in Figure 1 has been transcribed into BNF format and can be found in Figure **??**. In the next subsection the functions for all *Keywords* of SML are explained.

3.2 Keywords

In this subsection we define all of the available *Keywords* that exist in SML. Currently they're 8 keywords in SML [2].

3.2.1 Reading Datasets

When reading data from SML one must use the *READ Keyword* followed by an *Argument* containing a path to the dataset. *READ* also accepts an *OptionList*. The first query in Figure 3 consist of only a *Keyword* and *Argument*. The second query includes an *OptionList* in addition to reading data from the specified path, one specifies that the dataset is separated with semicolons and does not include a header row.

2. Detailed Documentation is publicly available on github: https://github.com/lcdm-uiuc/sml/tree/master/dataflows

```
READ "/path/to/dataset"
READ "/path/to/dataset" (sep = ",", header=None)
```

Figure 3: Example using the *READ* keyword in SML.

```
READ "/path/to/data" (separator = ",", header = None)
AND REPLACE (missing = "NaN",  strategy = "mode")
```

Figure 4: Example using the *REPLACE Keyword* in SML.

3.2.2 Cleaning Data

When NaNs, NAs and/or other troublesome values are present in dataset we clean these values in SML by using the *REPLACE Keyword*. In Figure 4 *REPLACE Keyword* replaces any instance of "NaN" with the mode of that column.

3.2.3 Partitioning Datasets

For majority of the situations in machine Learning it's often useful to split a dataset into training and testing datasets. This can be achieved in SML by using the *SPLIT* keyword. In Figure 5 an example of SML performing a 80/20 split by utilizing the *SPLIT* keyword.

3.2.4 Using Classification Algorithms

To use a classification algorithm in SML one would use the *CLASSIFY* keyword. SML has the following classification algorithms implemented: Support Vector Machines, Naive Bayes, Random Forest, Logistic Regression, and K-Nearest Neighbors. Figure 6 demonstrates how to use the *CLASSIFY* keyword in a query.

3.2.5 Using Clustering Algorithms

For clustering algorithms can be invoked by using the *CLUSTER* keyword. SML currently has K-Means Clustering implemented. Figure 7 demonstrates how to use the *CLUSTER* keyword in a query.

3.2.6 Using Regression Algorithms

Regression algorithms use the *REGRESS* keyword. SML currently has the following regression algorithms implemented:Simple Linear Regression, Ridge Regression, Lasso Regression, and Elastic Net Regression. Figure 8 demonstrates how to use the *REGRESS* keyword in a query.

```
READ "/path/to/data" (separator = ",", header = None) AND
SPLIT (train = 0.8, test = 0.2)
```

Figure 5: Example using the *SPLIT* keyword in SML.

```
READ "/path/to/data" (separator = ",", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND CLASSIFY
(predictors = [1,2,3,4], label = 5, algorithm = svm)
```

Figure 6: Example using the *CLASSIFY* keyword in SML.

```
READ "/path/to/data" (separator = ",", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND CLUSTER
(predictors = [1,2,3,4,5,6,7], algorithm = kmeans)
```

Figure 7: Example using the *CLUSTER* keyword in SML.

3.2.7 Saving/Loading Models

It's possible to save models and reuse them later. To save a model in SML one would use the *SAVE* keyword in a query. To load an existing model from SML one would use the *LOAD* keyword in a query. Figure 9 show how to save and load a model using SML.

3.2.8 Visualizing Datasets and Metrics of Algorithms

When using SML it's possible to visualize datasets or metrics of algorithms (such as learning curves, or ROC curves). To do this the *PLOT* keyword must be specified in a query. Figure 10 shows can example of how to use the *PLOT* keyword in a query.

4. SML's Architecture

With SML's grammar defined we have enough information to dive into SML's architecture. When SML is given a string, it is passed to the parser. The high level implementation of the grammar is then used to parse through the string to determine the actions to perform. The actions are stored in a dictionary and given to one of the following phases of SML: Model Phase, Apply Phase, or Metrics Phase. Figure 11 shows a block diagram of this.

The model phase is generally for constructing a model. The keywords that generally invoke the model phase are: *READ*, *REPLACE*, *CLASSIFY*, *REGRESS*, and *CLUSTER*. The apply phase is generally for applying a preexisting model to new data. The keywords that generally invoke the apply phase are: *LOAD*, and *APPLY*. It's often useful to visualize the data that one works with and beneficial to see the performance metrics of a machine learning algorithm. By default if you specify the *PLOT* keyword in a query, SML will execute the metrics phase.

```
READ "/path/to/data" (separator = ",", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND REGRESS
(predictors = [1,2,3,4,5,6,7,8,9], label = 10,
algorithm = ridge)
```

Figure 8: Example using the *REGRESS* keyword in SML.

```
SAVE "path/to/save/model"
LOAD "path/to/load/model"
```

Figure 9: Example using the *LOAD* and *SAVE* keywords in SML.

```
READ "/path/to/data" (separator = ",", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND CLUSTER
(predictors = [1,2,3,4,5,6,7], algorithm = kmeans)
AND PLOT
```

Figure 10: Example using the *PLOT* keyword in SML.

4.0.1 Connector

The last significant component of SML's architecture is the connector. The connector connects drivers from different libraries and languages to achieve an action a user want during a particular phase (see Figure 12). If one considers applying linear regression on a dataset, during the model phase SML calls the connector to retrieve the linear regression library in this case SML uses sci-kit learn's implementation however, if we wanted to use an an algorithm not available in sci-kit learn such as a Hidden Markov Model (HMM) SML will use the connector to call another library that supports HMM.

5. Interface

They're multiple interfaces available for working with SML. We've developed a web tool that's publicly available which allows in a query andThere's also a REPL environment available that allows the user to interactively use SML. Lastly, users have the option to import SML into an existing pipeline to simplify the development process.

6. Use Cases

We tested the SML framework against ten popular machine learning problems with publicly available data sets.

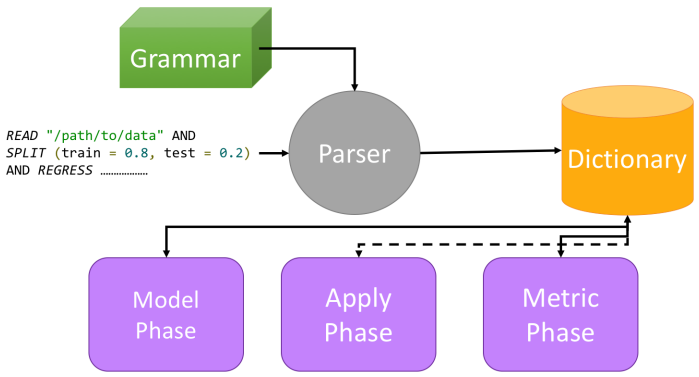

Figure 11: Block Diagram of SML's Architecture

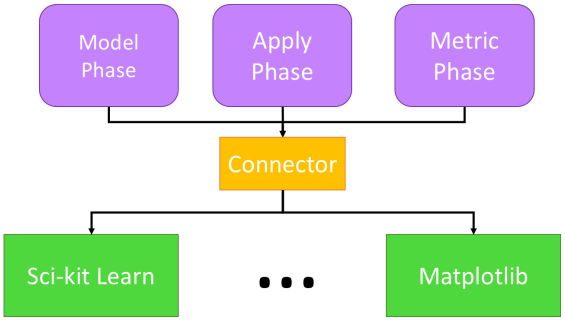

Figure 12: Block Diagram of SML's Connector

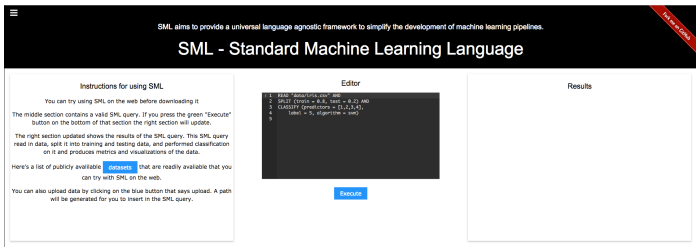

Figure 13: Interface of SML website. The user to interact with SML through a web interface (see Figure 13). Currently users can read instructions on the right to see examples of how to use SML. In the middle pane users can type SML Queries and then hit the execute button. The results of the query are explained on the right pane.

```
from sml import execute

query = 'READ "../data/iris.csv" AND \
SPLIT (train = 0.8, test = 0.2) AND \
CLASSIFY (predictors = [1,2,3,4], label = 5, algorithm = svm) AND \
PLOT'

execute(query, verbose=True)
```

Figure 14: SML query that performs classification on the iris dataset using support vector machines.

6.1 10 Use Cases

We applied SML to the following datasets: Iris Dataset [3], Auto-MPG Dataset [4], Seeds Dataset [5], Computer Hardware Dataset [6], Boston Housing Dataset [7], Wine Recognition Dataset [8], US Census Dataset [9], Chronic Kidney Disease [10], Spam Detection [11], and the Titanic Dataset [12]. In this paper we dicuss applying SML to the Iris Dataset and the Auto-MPG dataset. (?) provides detailed explanations and examples that solve problems all 10 data sets.

6.1.1 Iris Dataset

Figure 14 shows all of the code required to perform classification on the Iris dataset. In Figure 14 We read data from a specified path, perform a 80/20 split, use the first 4 columns to predict the 5th column, use support vector machines as the algorithm to perform classification and finally plot distributions of our dataset and metrics of our algorithm. Figure 15 illustrates what is required to perform the same operations using scikit learn. It's worth noting that the code is publicly available and well documented [13] and it is out of the scope of this paper. We just want to online the complexity required to produce such results. The result for both snippets of code are the same and can be seen in Figure 16.

6.1.2 Auto-Mpg Dataset

Figure 17 shows the SML query required to perform regression on the Auto-MPG dataset. In Figure 17 we read data from a specified path, the dataset is separated by fixed width spaces and we choose not to provide a header for the dataset. Next we perform a 80/20 split, replace all occurrences of "?" with the mode of the column. We then perform linear

3. https://archive.ics.uci.edu/ml/datasets/Iris
4. https://archive.ics.uci.edu/ml/datasets/Auto+MPG
5. https://archive.ics.uci.edu/ml/datasets/seeds
6. https://archive.ics.uci.edu/ml/datasets/Computer+Hardware
7. https://archive.ics.uci.edu/ml/datasets/Housing
8. https://archive.ics.uci.edu/ml/datasets/Wine
9. https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990)
10. https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease
11. https://archive.ics.uci.edu/ml/datasets/Spambase
12. https://www.kaggle.com/c/titanic
13. https://github.com/lcdm-uiuc/sml/blob/master/dataflows/plot/iris_svm-READ-SPLIT-CLASSIFY-PLOT.ipynb

```
import pandas as pd
import numpy as np

from sklearn.preprocessing import label_binarize
import sklearn.cross_validation as cv
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as plt
import seaborn as sns

names = ['sepal length(cm)', 'sepal width(cm)', 'petal length(cm)',
'petal width(cm)', 'species']
data = pd.read_csv('../data/iris.csv', names=names)

iris_classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
Features = np.c_[data.drop('species',1).values]
labels = label_binarize(data['species'], classes=iris_classes)
n_classes = labels.shape[1]

x_train, x_test, y_train, y_test = cv.train_test_split(features,
labels, test_size=0.25)
svm = OneVsRestClassifier(SVC(kernel='linear', probability=True))
model = svm.fit(x_train, y_train)

predict_score = model.decision_function(x_test)
test_set_results = model.score(x_test, y_test) * 100
print ('SVM Prediction Accuracy = {0:6.2f}%'.format(test_set_results) )

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
fpr[i], tpr[i], _ = roc_curve(y_test[:, i], predict_score[:, i])
roc_auc[i] = auc(fpr[i], tpr[i])

plt.rcParams['figure.figsize']=(12,12)
```

```
# Class Info
columns = [0,1,2,3]
cmap_class = ['Purples_r', 'Greens_r', 'Oranges_r', 'Greys_r' ]
color_class1D = ['purple', 'darkgreen', 'orange', 'grey']
column_headers =  data.columns.values.tolist() # Grab headers from df
column_headers = [column_headers[x] for x in columns] # Map headers to indices
selected

label = 'species'
fig, ax = plt.subplots(len(columns), len(columns))

for ic, cc, cc1D in zip(iris_classes, cmap_class, color_class1D):
  iris_class_data = data.loc[data.species == ic] # sep class

  #Generate kde plot matrix for class
  for col1, i in enumerate(columns):
      for col2, j in enumerate(columns):
          if i == j:
              sns.kdeplot(iris_class_data[iris_class_data.columns[col1]],
ax=ax[col1][col2], color=cc1D, shade=True, legend=False)
          else:
              sns.kdeplot( iris_class_data[iris_class_data.columns[col1]],
iris_class_data[iris_class_data.columns[col2]], ax=ax[col1][col2], cmap=cc)
          # Formatting
          if j == 0:
              ax[i,j].set_xticklabels([])
              ax[i,j].set_ylabel(column_headers[i])
              ax[i,j].set_xlabel('')
              if i == len(columns)-1:
                  ax[i,j].set_xlabel(column_headers[j])
          elif i == len(columns)-1:
              ax[i,j].tick_params(axis='y', which='major', bottom='off')
              ax[i,j].set_yticklabels([])
              ax[i,j].set_xlabel(column_headers[j])
              ax[i,j].set_ylabel('')
          else:
              ax[i,j].set_xticklabels([])
              ax[i,j].set_xlabel('')

              ax[i,j].set_yticklabels([])
              ax[i,j].set_ylabel('')

plt.show()
plt.close()
```

Figure 15: This shows the code required to replicate the same actions of the SML query in Figure 14

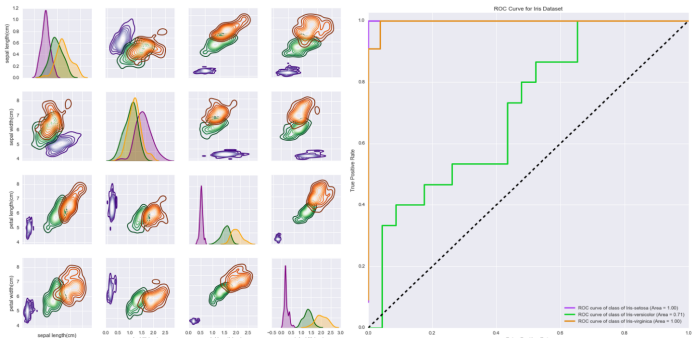

Figure 16: The SML query in Figure 14 and the code in Figure 15 produce these results. The subgraph on the left is a lattice plot showing the density estimates of each feature used. The graph on the right shows the ROC curves for each class of the iris dataset.

```
from sml import execute


query = 'READ "../data/auto-mpg.csv" (separator = "\s+", header = None) AND \
REPLACE (missing = "?", strategy = "mode") AND \
SPLIT (train = .8, test = .2, validation = .0) AND \
REGRESS (predictors = [2,3,4,5,6,7,8], label = 1, algorithm = simple) AND \
PLOT'

execute(query, verbose=True)
```

Figure 17: SML query that performs classification on the Auto-MPG dataset using support vector machines.

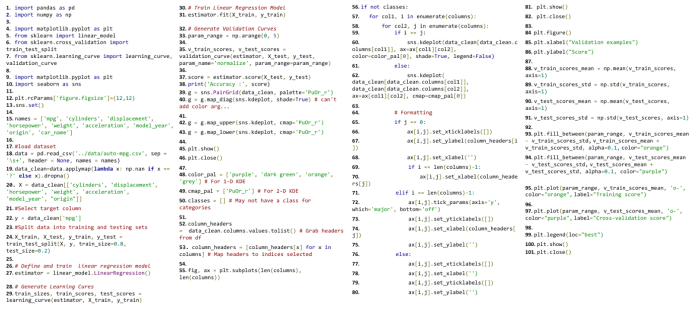

Figure 18: This shows the code required to replicate the same actions of the SML query in Figure 17

regression using columns 2-8 to predict the 1st label. Lastly, we visualize distributions of our dataset and metrics of our algorithm. Figure 18 demonstrates what's required to perform the same operations using scikit learn. The outcome for both process are the same and can be seen in Figure 19.

7. Conclusion

To summarize we introduced an agnostic framework that integrates a query-like language to simplify the development of machine learning pipelines. We provided a high level overview

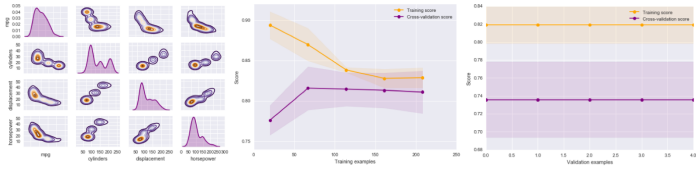

Figure 19: The SML query in Figure 17 and the code in Figure 18 produce these results. The subgraph on the left is a lattice plot showing the density estimates of each feature used. The the centered shows the learning curve of the model and the graph on right shows the validation curve.

of it's architecture and grammar. We then applied SML to machine learning problems and demonstrated how the complexity of the code one has to write significantly decreases when SML is used. The source code and detailed documentation for SML is open sourced and publicly available on github [14]. In the future we plan to extend the connector to support more machine learning libraries from additional languages. We plan to expand the web application to make SML easier to use.

If we want to researchers from other domain areas to utlize machine learning without understanding the complexities required for machine learning a tool like SML is needed. The concepts presented in this paper as a whole is sound. The details may change but the core principals will remain the same. Abstracting the complexities of machine learning from users are needed to increase the use of machine learning by researchers in other areas.

References

Domingos, P. (2012). A few useful things to know about machine learning.. Vol. 55, pp. 78–87, New York, NY, USA. ACM.

Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. *CoRR*, *abs/1603.06212*.

14. https://github.com/lcdm-uiuc/sml