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 for a variety of state-of-the-art 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 from 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 for multiple 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 they're a lot of nuisances (that a novice may be unfamilar with) involved when developing machine learning pipelines (?) and if they are not taken into consideration one may not receive satisfactory results. In addition to this, an experience user may not want to deal with [REASONS HERE]. In this paper we introduce Standard Machine Learning Language (SML) which helps to combat these two use cases.

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 researchers and industry professionals without a background in this area to use 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.

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

```
READ "/path/to/data.csv" 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.

2. Related Works

TPOT (?) is a tool implemented in python that creates and optimizies machine learning pipelines. 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 TPOT from SML is that in addition feature, model, and parameter selection/optimization a framework is explicitly 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 a typical example of how one would perform classifcation on a dataset. The query in Figure 1 reads in a dataset, peforms a split 80/20 split of training and testing data respectively, and performs classifcation of the 5th column of the hypothetical dataset using columns 1-4 as predictors.

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:

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

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

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

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. Lastly, an Arugment can have a multitude of options (3) where an Option consist of an OptionName with either an option value or option value list. 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.

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

Figure 2: Example using the *READ* keyword in SML.

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

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

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

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

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

Figure **??**

3.2 Keywords

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

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. Figure 2 shows examples of the *READ* keyword being used. *READ* also accepts an OptionList, for a full list of the optional arguments can visit **??**.

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. Figure 3 shows can example of the *REPLACE* keyword being used.

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. Figure 4 shows an example utilizing the *SPLIT* keyword.

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

Figure 4: 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 5: Example using the *CLASSIFY* keyword in SML.

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 5 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 6 demonstrates how to use the *CLUSTER* keyword in a query.

3.2.6 Using Regression Algorithms

Regression algorithms can be invoked by using the *REGRESS* keyword. SML currently has the following regression algorithms implemented:Simple Linear Regression, Ridge Regression, Lasso Regression, and Elastic Net Regression. Figure 7 demonstrates how to use the *REGRESS* keyword in a query.

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 8 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 9 shows can example of how to use the *PLOT* keyword in a query.

```
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 6: Example using the *CLUSTER* keyword in SML.

```
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 7: Example using the *REGRESS* keyword in SML.

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

Figure 8: Example using the *LOAD* and *SAVE* keywords in SML.

4. 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.

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 prexisting 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 beneifical 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. Figure 10 displays a block diagram of the metric phase of SML. SML performs a dictionary lookup with the average complexity of O(1) to find specific terms that are in the query. For the example in Figure 10 we specified 'PLOT' which instructs SML to create visualizations, with the 'READ' keyword SML will create a lattice plot containing kernel density estimates for each feature. Given an algorithm type such as Classification SML generates plots such as ROC Curves and Validation and Learning Curves. For a comprehensive list for the type of plots that SML can generate visit ().

4.0.1 Connector

Lastly, another significant component of SML's architecture is the connector. The connector connects drivers from different languages to achieve an action we want. For instance, consider applying linear regression on a dataset. During the model phase, SML calls the connector to retrieve the linear regression library. SML uses scikit learn's **??** implementa-

```
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 9: Example using the *PLOT* keyword in SML.

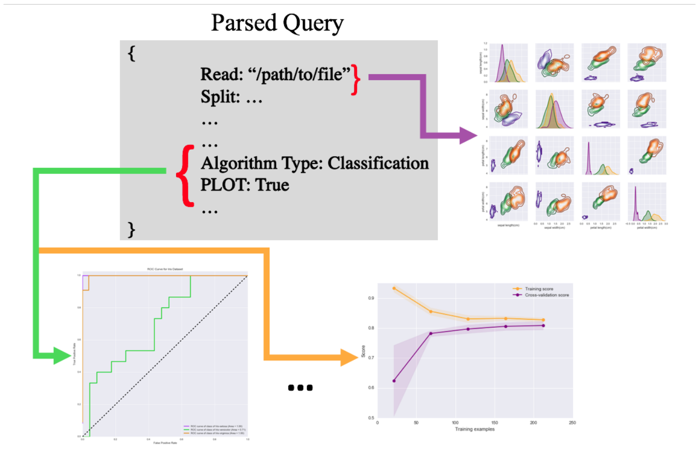


Figure 10: Block Diagram of Metric Phase Architecture

tion for Linear Regression however if we wanted to use a HMM scikit learn removed this algorithm from it's library ergo 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 the user to interact with SML. There'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. We then selected two problems that researchers have attempted to solve in different domains that utlized machine learning to apply SML.

6.1 10 Use Cases

We applied SML to the following datasets: Iris Dataset (), Auto-MPG Dataset (), Seeds Dataset (), Computer Hardware Dataset (), Boston Housing Dataset (), Wine Dataset (), US Census Dataset (), Chronic Kidney Disease (), Spam Detection (), and the Titanic Dataset (). 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

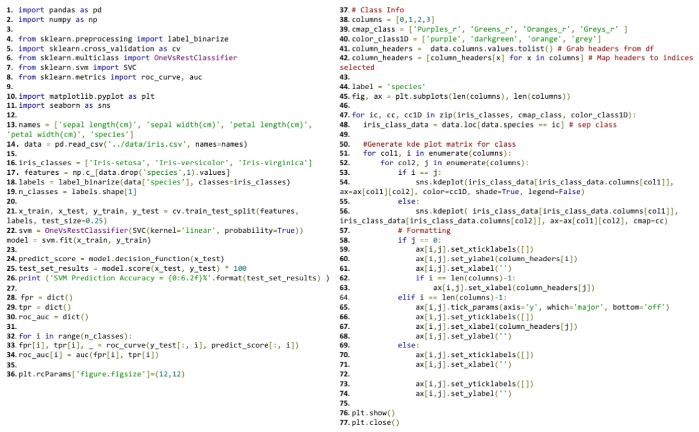
Figure 11 shows all of the code required to perform classification on the Iris dataset. In Figure 11 We read data from a specified path, perform a 80/20 split, use the first 4 columns to predict the 5th column, use support vector machines as the algorithm to perform classi-

```
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 11: SML query that performs classification on the iris dataset using support vector machines.



```
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 12: This shows the code required to replicate the same actions of the SML query in Figure 11

fication and finally plot distributions of our dataset and metrics of our algorithm. Figure 12 illustrates what is required to perform the same operations using scikit learn. The result for both snippets of code are the same and can be seen in Figure 13.

6.1.2 Auto-Mpg Dataset

Figure 14 shows all of the code required to perform regression on the Auto-MPG dataset. In Figure 14 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 regression using columns 2-8 to predict the 1st label. Lastly, we visualize distributions of our dataset and metrics of our algorithm. Figure 15 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 16

7. Future Work

future work stuff

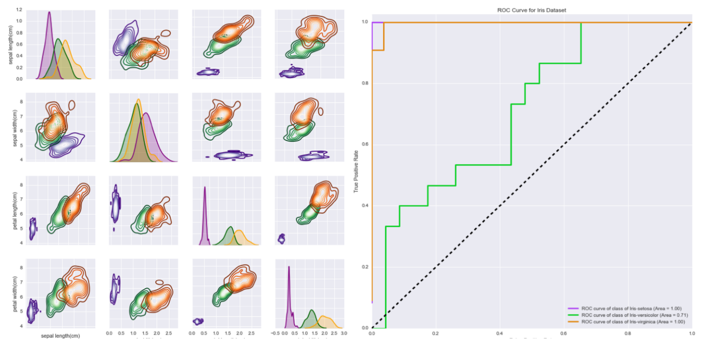


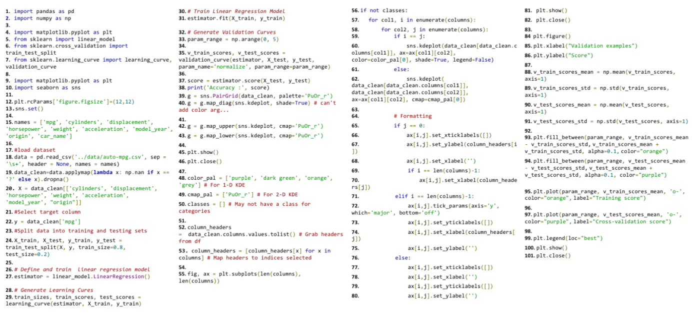
Figure 13: The SML query in Figure 11 and the code in Figure 12 produce these results. The subgraph on the left is a lattice plot showing the density estimates of each feature used. The graph on the right shows the ROC curves for each class of the iris dataset.

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



```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.learning_curve import learning_curve, validation_curve

import matplotlib.pyplot as plt
import seaborn as sns

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

names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', 'origin', 'car_name']

#load dataset
data = pd.read_csv('../data/auto-mpg.csv', sep = '\s+', header = None, names = names)
data_clean=data.applymap(lambda x: np.nan if x == '?' else x).dropna()
X = data_clean[['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year', "origin"]]
#Select target column
y = data_clean['mpg']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2)

# Define and train  linear regression model
estimator = linear_model.LinearRegression()

# Generate Learning Cures
train_sizes, train_scores, test_scores = learning_curve(estimator, X_train, y_train)
# Train Linear Regression Model
estimator.fit(X_train, y_train)

# Generate Validation Curves
param_range = np.arange(0, 5)

v_train_scores, v_test_scores = validation_curve(estimator, X_test, y_test, param_name='normalize', param_range=param_range)

score = estimator.score(X_test, y_test)
print('Accuracy :', score)
g = sns.PairGrid(data_clean, palette='PuOr_r')
g = g.map_diag(sns.kdeplot, shade=True) # can't add color arg...

g = g.map_upper(sns.kdeplot, cmap='PuOr_r')
g = g.map_lower(sns.kdeplot, cmap='PuOr_r')

plt.show()
plt.close()

color_pal = ['purple', 'dark green', 'orange', 'grey'] # For 1-D KDE
cmap_pal = ['PuOr_r'] # For 2-D KDE
classes = [] # May not have a class for categories

column_headers =  data_clean.columns.values.tolist() # Grab headers from df
column_headers = [column_headers[x] for x in columns] # Map headers to indices selected

fig, ax = plt.subplots(len(columns), len(columns))
if not classes:
   for col1, i in enumerate(columns):
       for col2, j in enumerate(columns):
           if i == j:
               sns.kdeplot(data_clean[data_clean.columns[col1]], ax=ax[col1][col2], color=color_pal[0], shade=True, legend=False)
           else:
               sns.kdeplot(data_clean[data_clean.columns[col1]], data_clean[data_clean.columns[col2]], ax=ax[col1][col2], cmap=cmap_pal[0])

           # 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()

plt.figure()
plt.xlabel("Validation examples")
plt.ylabel("Score")

v_train_scores_mean = np.mean(v_train_scores, axis=1)
v_train_scores_std = np.std(v_train_scores, axis=1)
v_test_scores_mean = np.mean(v_test_scores, axis=1)
v_test_scores_std = np.std(v_test_scores, axis=1)

plt.fill_between(param_range, v_train_scores_mean - v_train_scores_std, v_train_scores_mean + v_train_scores_std, alpha=0.1, color="orange")
plt.fill_between(param_range, v_test_scores_mean - v_test_scores_std, v_test_scores_mean + v_test_scores_std, alpha=0.1, color="purple")

plt.plot(param_range, v_train_scores_mean, 'o-', color="orange", label="Training score")

plt.plot(param_range, v_test_scores_mean, 'o-', color="purple", label="Cross-validation score")

plt.legend(loc="best")
plt.show()
plt.close()
```

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

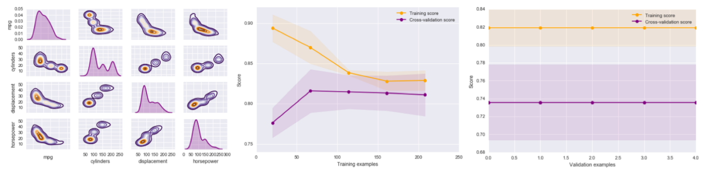


Figure 16: The SML query in Figure 14 and the code in Figure 15 produce these results. The subgraph on the left is a lattice plot showing the density estimates of each feature used. The the centered shows the learning curve of the model and the graph on right shows the validation curve.

8. Conclusion

conclusion stuff

Acknowledgments

acknowledgments go here

Appendix X...

Appendix goes here if needed.