Robot Classify

Release Alpha

Scott R Smith

Feb 18, 2020

CONTENTS

CHAPTER

ONE

INSTALATION AND OVERVIEW

1.1 Introduction

RobotClassify allows for non-data scientest such as citizen developers and other operational people involved with analizing and reporting on business data. The goal is to automate the entire ML process (feature-engineering, training, prediction).

This version of the app is optimized for loading datafiles to train with, and test files for predctiona, optimzed for submission in Kaggle competitions. Currently, we only support Machine Learning classification problems. The Machine Learning component is based upon mlLib, a library that I created to put into code, techniques I have learning during my ML course work. T

My motivation for project centers around my interest in machine learning for citizen developers. Taking the complecated tasked of feature engineering, model selection, and training and makeing it a simple point and click excersie without any prior machine learning training.

Using RobotClassify requires four simple steps that can all be accomplished via the RobotClassify.herokuapp.com.

CHAPTER

TWO

. LOAD A CSV DATA FILE. THIS IS DONE BE CREATING A PROJECT AND SPECIFING THE TRAINING AND TEST FILES (EXAMPLES ARE FOUNX IN THE EXAMPLES FOLDER)

CHAPTER

THREE

. CREATE A RUN. THE RUN RECORD DEFINES THE FILE ATTRIBUTES AND THE NATURE OF THE TRAINING. FOR THIS, WE NEED TO SPECIFY:

- The target variable that is to be predicted
- Record Key column
- Predict set out. These are the columns that are used to create the predict file in a format that can be used to submit the test results in a Kaggle comopetition
- Classification model to train
- Scoring method
- Algoritym type (There are two approaches used to automate feature engineering)

CHAPTER

FOUR

. RUN THE TRAINING

CHAPTER

FIVE

. REVIEW THE RESULTS

5.1 Implementation Ovweview

The application is build with Flask and Flask What-the-forms for the frontend.

Roles There are two roles: Editor: Able to create, update, train and delete projects and training runs Viewer: Only able to view results

5.1.1 API End Points

The following APIs endpoints are available. Detailed html documentation can be found at https://robotclassify.herokuapp.com/docs/index.html

– Home Page –

- GET / (home)

– Documentation Page –

- GET /docs/index.html

— Projects —

- GET /projects (List all projects) - get:project
- GET /projects/int:project_id (Project page) - get:project
- POST/GET /projects/create (create a new project) - post:project
- PATCH /projects/int:project_id/edit (edit a project) - patch:project
- DELETE /projects//delete (Delete a project) - delete:project

— Runs —

- GET /runs/int:run_id (Display a run results) - get:run
- GET/POST /runs/create/int:project_id (Create a run) - get:post
- DELETE /runs/int:run_id/delete (Delete a run) - delete:post
- PATCH /run/int:run_id/edit (edit a run) - patch:run

— Train —

- GET /train/int:run_id (run ML training for a run) post:train
- GET /train/int:run_id/download (download testing results file, .. code-block:

  kaggle file) get:train

5.2 Project dependencies, local development and hosting instructions

- Detailed instructions for scripts to install any project dependencies, and to run the development server.
- Documentation of API behavior and RBAC controls

5.3 Runing and Testing Instrunctions

URL: https://robotclassify.herokuapp.com/

There are three approaches to running and evaluating RobotClassify:

CHAPTER

SIX

. WEB

CHAPTER

SEVEN

. UNITTEST

CHAPTER

EIGHT

. CURL

There are scripts to help with each one.

For example, the Titanic Kaggle competeition (https://www.kaggle.com/c/titanic.com), provides two data sets, the Training set and test set. Loading these into Robot Classify, we would set the run paramatgers as follows:

- Target Variable: Survived
- Record Key: PassengerID
- Predict set out: Survuved, PassengerID
- Classification model: xgbc
- Scoring method: f1
- Use Algorithm I for feature engineering: True

This will give a training result that would put you in the top 8% of competitors.

Unittests are run using the script `test.sh`. This script requires Postgress on the local machine where the test is being run.

The `test.sh` script will:

CHAPTER

NINE

. CREATE THE ROBOTCLASSIY_TEST DATABASE

CHAPTER

TEN

. GET A TOKEN AND USER ID FOR THE API USER (PLACED INTO ENVIRONMENT VARIABLES)

CHAPTER

ELEVEN

. POPULATE THE TEST DATABASE WITH DATA FOR THE API USER

CHAPTER

TWELVE

. RUN THE TESTS

The current implementation is enabled for Curl. Curl will allow for operations against the product database (Hosted on Amazon/Heroku)

curl_pass.sh curl_fail.sh curl.sh (setup the environment variables to run Curl manually)

If you need to get an updated token, you need to login to the Web app and issue the following URL:

https://robotclassify.herokuapp.com/jwt

This will retrive the current jwt token for the logged in user.

12.1 Installation

12.1.1 Python 3.7

This project uses python 3.7

To Install Python

12.1.2 PIP Dependencies

Once you have your virtual environment setup and running, install dependencies by navigating to the root directory and running:

```
pip install -r requirements.txt
```

This will install all of the required packages we selected within the `requirements.txt` file.

12.1.3 Key Dependencies

- Flask is a lightweight backend microservices framework.
- SQLAlchemy is the Python SQL toolkit and ORM.
- Flask-CORS is the extension used to handle cross-origin requests from the frontend server.
- Auth0 Provides authentication and authorization as a service
- Postgres DOES XXX
- Heroku DOES XXX
- Flask-WTF DOES XXX

- mlLib DOES XXX
- InitTest DOES XXX
- FlaskMigrate DOES XXX

12.2 Database Setup

The app is running Postgres SQL.

12.3 Running the server

From within the `root` directory to run the server, execute:

```
export FLASK_APP=app.py
export FLASK_ENV=development
flask run
```

12.4 Documentation

12.4.1 HTML Documentation

Live documentation, including this readme, can be found at https://robotclassify.herokuapp.com/docs/index.html

12.4.2 PDF Documentation

The PDF version of the documentation is located in the root project directory. Named robotclassify.pdf

12.4.3 Generating documentation

Documentation is generated with Sphinx.

Installing Sphinx and support tools

```
To install Sphinx, reference the documents at https://www.sphinx-doc.org/en/master/
↪usage/installation.html

For example:

.. code-block:: bash

   pip install -U sphinx

Install dependencies by navigating to the ``root`` project directory and running:

.. code-block:: bash

   cd docs
   pip install m2r
```

(continues on next page)

(continued from previous page)

```
   pip install recommonmark
   pip install rinohtype
   pip install -r requirements.txt

Generating documentation
```

Documentation is generated with Sphinx. Use `docs.sh` in the docs folder to generate the documentation

12.5 Error Handling

Errors are returned as JSON objects in the following format:

```
{
    "success": False,
    "error": 400,
    "message": "Bad Request"
}
```

The API returns multiple error types when requests fail:

- 400: Bad Request
- 404: Resource Not Found
- 405: Method Not Allowed
- 422: Not Processable
- 500: Internal Server Error

12.6 Testing

Testing is done with UnitTest and curl. UnitTest is setup to create and use a local Postgres database while Curl is setup to run commands against the

12.6.1 Testing with UnitTest

12.6.2 Testing with Curl

12.7 Development Notes

- Flask Sessions are maintained between REST Calls for Web-based use of the API. The implementation is based upon
- CSRF protection is disabled for certain REST calls to faciliate testing cia CuRL.
- Patch and Delete functions are only avialable via API calls
- UnitTest uses a local postgres database
- UnitTest uses API App Auth0 credentials (verses using Auth0 Web App quickstart code) Auth0 Management API (Test Application)
- Tokens in the headers are used for API authentication

CHAPTER

THIRTEEN

ROBOT CLASSIFY'S API CONTROLLERS

CHAPTER

FOURTEEN

ROBOT CLASSIFY'S API MODEL

CHAPTER

FIFTEEN

ML LIB

15.1 Introduction

The projects module is is a high-level library to process ML jobs at the project level. All interactions can happen with this API

- Manage Projects
- Load Data
- Explore Data
- Auto-execute ML jobs
- Create cleaning rules
- Train the data, finding the best model
- Deploy model, to process ML transactions

mlLib.project.**autoFlaskEvaluateClassifier**(*projectName=None, trainingFile=None, testingFile=None, trainingFileDF=None, testingFileDF=None, targetVariable=None, key=None, predictSetOut=[None], trainingFileOut=None, logFileOut=None, transcriptFile=None, predictFileOut=None, resultsFile=None, modelList=None, confusionMatrixLabels=[], scoring='f1', useProba=False, bottomImportancePrecentToCut=None, setProjectGoals={'f1': (0.9, '>')}, runVerbose=0, recommendOnly=None, basicAutoMethod=None, doExplore=True, doTrain=True, doPredict=True, skewFactor=None, toTerminal=True*)

autoFlaskEvaluateClassifier

Auto-process an ML job for Flask Servers

XXXXX

- Sample Call:

```
Example Call
```

- Expected Success Response:

```
Example Success Response
```

- Expected Fail Response:

```
Example fail response
```

mlLib.project.**autoEvaluateClassifier**(*projectName=None, trainingFile=None, testingFile=None, targetVariable=None, key=None, predictSetOut=[None], trainingFileOut=None, logFileOut=None, transcriptFile=None, predictFileOut=None, resultsFile=None, modelList=None, confusionMatrixLabels=[], scoring='f1', useProba=False, bottomImportancePrecentToCut=None, setProjectGoals={'f1': (0.9, '>')}, runVerbose=1, recommendOnly=None, basicAutoMethod=None, doExplore=True, doTrain=True, doPredict=True, skewFactor=None, toTerminal=True*)

autoEvaluateClassifier

Auto-process an ML job for

XXXXX

- Sample Call:

```
Example Call
```

- Expected Success Response:

```
Example Success Response
```

- Expected Fail Response:

```
Example fail response
```

class mlLib.project.**mlProject**(*name*, *description=None*)

mlProject is the top level object for training and running a ML project. Various object mothods are used to load, review, and train the data, as well as manage running predictions

Example: project = mlProject('Customer Segements', 'clustering model should factor in both aggregate sales patterns and specific items purchased')

setTrainingPreferences(*crossValidationSplits=None*, *parallelJobs=None*, *modelType=None*, *modelList=None*, *testSize=None*, *randomState=None*, *uniqueThreshold=None*, *dropDuplicates=None*, *clusterDimensionThreshold=None*, *varianceThreshold=None*, *kmeansClusters=None*, *useStandardScaler=None*, *fbeta=None*, *runHyperparameters=None*, *runEstimatorHyperparameters=None*, *runMetaClassifier=None*, *runAutoFeaturesMode=None*, *smallSample=None*, *highDimensionality=None*, *gridSearchVerbose=None*, *gridSearchScoring=None*, *featuresToReport=None*, *logTrainingResultsFilename=None*, *useProbaForPredict=None*, *recommendOnly=None*, *basicAutoMethod=None*, *competitionMode=None*, *skewFactor=None*, *bottomImportancePrecentToCut=None*)

setTrainingPreferences

setTrainingPreferences for an ML job

XXXXX

- Sample Call:

```
Example sample call
```

- Expected Success Response:

```
Example Success Response
```

- Expected Fail Response:

```
Example fail response
```

setHyperparametersOverride(*modelName*, *override*, *forBaseEstimator=False*, *forMetaClassifier=False*)

Purpose: Set the hyperparameters to override the defaults for a model

Example:

```
hyperparameters = {
        'lasso__alpha' : [0.001, 0.01, 0.1, 1, 5, 10]
        }
project.setHyperparametersOverride(self, 'lasso', hyperparameters)


hyperparameters = {
    'lasso__alpha': [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10]
    }
hyperparameters = {
    'ridge__alpha': [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10]
}
hyperparameters = {
    'elasticnet__alpha':
        [0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10],
    'elasticnet__l1_ratio' : [0.1, 0.3, 0.5, 0.7, 0.9]
}
hyperparameters = {
    'randomforestregressor__n_estimators': [50, 100, 200, 500],
    'randomforestregressor__max_features': ['auto', 'sqrt', 0.33]}
hyperparameters = {
    'gradientboostingregressor__n_estimators': [50, 100, 200, 500],
    'gradientboostingregressor__learning_rate':
            [0.001, 0.05, 0.1, 0.5],
    'gradientboostingregressor__max_depth': [1, 5, 10, 50]}
hyperparameters = {'decisiontreeregressor__max_depth':
                    [1, 8, 16, 32, 64, 200]}
hyperparameters = {
        'logisticregression__C' : np.linspace(1e-4, 1e3, num=50),
        'logisticregression__max_iter': [25, 50, 100, 300, 500]
        }
hyperparameters = {
        'logisticregression__C' : np.linspace(1e-4, 1e3, num=50),
        'logisticregression__max_iter': [25, 100, 300, 500]
        }
hyperparameters = {'randomforestclassifier__n_estimators':
                    [100, 200],
                    'randomforestclassifier__max_features':
                    ['auto', 'sqrt', 0.33]}
hyperparameters = {'gradientboostingclassifier__n_estimators':
```

(continues on next page)

(continued from previous page)

```
                    [50, 100, 200, 500],
    'gradientboostingclassifier__max_depth': [1, 10, 50, 100],
    'gradientboostingclassifier__learning_rate':
    [.1, .01, .001,.0001]}
```

setConfusionMatrixLabels(*list*)

Example: project.setConfusionMatrixLabels([(0,'Paid'), (1, 'Default')])

setTarget(*value*, *boolean=False*, *trueValue=None*, *convertTable=None*, *tableName=None*)
Purpose: Set the target variable for supervised learning.

Call: setTarget(self, value, boolean=False, trueValue=None, convertTable=None, tableName=None):

Example:

project.setTarget('loan_status')

trueValue = what is the true values boolean = is this a boolean value convertTable = a table of how to convert values

importFile(*name*, *type=None*, *description=None*, *location=None*, *fileName=None*, *sheetName=None*, *df=None*, *hasHeaders=False*, *range=None*, *isDefault=False*)

def importFile(self, name, type=None, description=None, location=None, fileName=None, sheetName=None, hasHeaders = False, range=None, isDefault=False):

project.importFile('Loan Data', type='csv', description='Lending Club Data from 2017-2018', fileName='LendingClub2017_2018ready.csv', hasHeaders = True, isDefault=True)

exportFile(*name*, *filename*)
Purpose: Export the named file. (Projects can have multiuple files associated with them)

Call: def exportFile(self, name, filename):

Example: project.exportFile('Loan Data', 'fileout.csv'):

getColumn(*name*, *columnName*)
Purpose: Get a columns from the data file

Call: def getColumn(self, name, column):

Example: project.getColumn('Loan Data','Name')

exploreData(*fileName=None*)

Purpose: Run the explore data function. This will review the data and make recommendations

Call: exploreData(self):

Example: project.exploreData()

initCleaningRules(*fileName=None*)
Before adding any cleaning rules you must init

project.initCleaningRules()

project.addManualRuleForDefault(ed.CLEANDATA_REBUCKET_TO_BINARY, 'term', [['36 months', ' 36 months'], '36'])

project.addManualRuleForDefault('ed.CLEANDATA_REBUCKET_TO_BINARY, 'term', [['60 months', ' 60 months'], '60'])

cleanProject (*fileName=None*)
Purpose: Run the cleaning rules established for a project.

Call: cleanProject(self)

Example: project.cleanProject()

cleanAndExploreProject (*fileName=None*)
Purpose: Run clean and explore together

Call: def cleanAndExploreProject(self)

Example: project.cleanAndExploreProject()

prepProjectByName (*tableName=None*, *outFile=None*)

Purpose: Prepare the 'table' for training. This will one-hot encode, for example.

Call: prepProjectByName(self, tableName=None)

Example: project.prepProjectByName('Loan Data')

writePreppedFileByName (*filename*, *tableName=None*)
Purpose: Once a file has been cleaned and explorred

Call:

Example:

writeTrainingSetFileByName (*filename*, *tableName=None*)
Purpose:

Call:

Example:

trainProjectByName (*tableName=None*)
Purpose:

Call:

Example:

prepProjectByBatch ()
Purpose:

Call:

Example:

trainProjectByBatch ()
Purpose:

Call:

Example:

exportBestModel (*filename*, *tableName=None*)
Purpose:

Call:

Example:

createPredictFromBestModel (*tableName=None*)
Purpose:

Call:

Example:

createPredictFromNamedModel (*namedModel*, *tableName=None*)
Purpose:

Call:

Example:

exportNamedModel (*namedModel*, *filename*, *tableName=None*)
Purpose:

Call:

Example:

addManualRuleForTableName (*tableName*, *functionName*, *columnName*, *value*, *forPredict=True*)
Purpose:

Call:

Example:

addManualRuleForDefault (*functionName*, *columnName=None*, *value=None*, *forPredict=True*)
Purpose:

Call:

Example:

setGoals (*goals*)
Purpose:

Call:

Example:

project.setGoals({'AUROC':(0.70,'>'),'Precision':(0.386,'>'), 'fbeta':(0.44,'>')})

setOngoingReporting (*True*, *'Loan Data'*)

displayAllScores (*'Loan Data'*)
def displayAllScores(self, fileName):

reportResultsOnTrainedModel (*fileName*, *modelName*)
Purpose:

Call:

Example:

showFeatureImportances (*fileName*, *modelName*)
Purpose:

Call:

Example:

logTrainingResultsRunDescription (*description='None'*)
Purpose:

Call:

Example:

logTrainingResults (*fileName*, *outputFileName*, *inputModelName=None*)
Purpose:

Call:

Example:

class mlLib.project.**predictProject**(*project*, *tableName=None*, *namedModel=None*)
Purpose: predictProject

Call:

Example:

importPredictFile(*name*, *type=None*, *description=None*, *location=None*, *fileName=None*, *sheetName=None*, *hasHeaders=False*, *range=None*)
Purpose:

Call:

Example:

importPredictFileFromProject(*project*, *tableName*)
Purpose:

Call:

Example:

importPredictFromDF(*df*, *readyForPredict=False*)
Purpose:

Call:

Example:

prepPredict()
Purpose:

Call:

Example:

exportPreppedFile(*filename*, *columnName=None*, *columnData=None*, *columnName2=None*, *columnData2=None*)
Purpose:

Call:

Example:

getColumn(*columnName*)
Purpose: Get a columns from the data file

Call: def getColumn(self, column):

Example: prdict.getColumn('Name')

exportPredictClass(*filename*)
Purpose:

Call:

Example:

addToPredictFile(*columnName*, *columnData*)
Purpose:

Call:

Example:

removeFromPredictFile(*columns*)
Purpose:

Call:

Example:

keepFromPredictFile(*columns*)
Purpose:

Call:

Example:

exportPredictFile(*filename*)
Purpose:

Call:

Example:

getPredictFileDF()
Purpose: Return a datraframe of the predict file.

Call:

Example:

runPredict()
Purpose:

Call:

Example:

mlLib.project.**loadPredictProject**(*filename*)
Purpose:

Call:

Example:

mlLib.project.**plot_confusion_matrix**(*cm*, *classes*, *normalize=False*, *title='Confusion matrix'*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*)

Purpose:

Call:

Example:

This function prints and plots the confusion matrix. Normalization can be applied by setting *normalize=True*.

mlLib.project.**makeStack**(*classifier*, *list*, *alias=None*)
Purpose:

Call:

Example:

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

R

S

T

W