RobotClassify

Release 0.20

Scott R Smith

Feb 23, 2020

CONTENTS

CHAPTER

ONE

INTRODUCTION

RobotClassify allows for non-data scientists such as citizen developers and other operational people involved with analyzing 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 data files to train with, and test files for predictions. Prediction files are optimized 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, put into code techniques I have learning during my ML studies.

My motivation for RobotClassify centers around my interest in making machine learning accessible for citizen developers. Taking the complicated task of feature engineering, model selection, and training and making it a simple point and click exercise without any prior machine learning training.

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

- Load a CSV data file. This is done by creating a project and specifying the training and test files (examples are found in the examples folder)
- 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 competition
  - Classification model to train
  - Scoring method
  - Algorithm type (There are two approaches used to automate feature engineering)
- Run the training
- Review the results

CHAPTER

TWO

RUNNING AND TESTING INSTRUCTIONS

RobotClassify can be accessed from the URL: https://robotclassify.herokuapp.com/.

2.1 Running on the Web

The web interface provides a 4 step approach to completing training and getting a result:

- Load the training and test files by creating a project
- Create a run record. The run record describes the test attributes
- Run the training
- Download the results file from the predictions

For example, the Titanic Kaggle competition (https://www.kaggle.com/c/titanic) provides two data sets, the training set and a test set. Loading these into RobotClassify, we would set the run parameters as follows:

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

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

CHAPTER

THREE

IMPLEMENTATION OVERVIEW

The application was written with Flask as the backend and Flask What-the-forms for the frontend.

3.1 Roles

DISABLED FOR NOW - ALL PERMISSIONS AVAILABLE FOR ALL USERS.

There are two roles:

- Viewer Role: Viewers can only view projects, runs, and their results.
- Editor Role: Editors can create projects, runs, and perform training

permissions	Editor	Viewer	Description
get:project	Yes	Yes	get a single, or list of projects
post:project	Yes		Create a new project or search
patch:project	Yes		Update a project attributes
delete:project	Yes		Delete a project and its runs
get:run	Yes	Yes	Get a run or download run results
post:run	Yes		Create a new run
patch:run	Yes		Update a run's attributes
delete:run	Yes		Delete a run
get:train	Yes		Run ML Training

3.2 API End Points

The following APIs endpoints are available. Detailed HTML documentation on these end points, including this file, can be found at https://robotclassify.herokuapp.com/docs/index.html

These are the end-points, with the short description and role.

– Home Page –

- GET / (home)

– Documentation Page –

- GET /docs/index.html

— Projects —

- GET /projects (List all projects) - get:project

- GET /projects/int:project_id (List a single project) - 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) get:train
- GET /train/int:run_id/download (download testing results file, .. code-block:

  kaggle file) get:run

CHAPTER

FOUR

INSTALLATION AND DEPENDENCIES

RobotClassify source is loacted at: https://github.com/scottrsmith/RobotClassify

4.1 Python

This project uses python 3.7

To Install Python

4.2 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.

4.3 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 Postgres SQL database
- Heroku App Hosting
- Flask-WTF Flask What-the-forms
- mlLib Machine Learning Training lib. Included in robot classify
- InitTest Test automation for Python
- FlaskMigrate Manages SQLAlchemy database migrations for Flask applications using Alembic
- scikit-learn Simple and efficient tools for predictive data analysis

4.4 Database Setup

The UnitTest is running Postgres SQL as the local souce database.

How to start/stop: https://stackoverflow.com/questions/7975556/how-to-start-postgresql-server-on-mac-os-x

4.5 Running the flask server

On a local machine, from within the `root` directory to run the server, execute `dev.sh`

CHAPTER

FIVE

DOCUMENTATION

5.1 HTML Documentation

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

5.2 PDF Documentation

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

5.3 Generating documentation

Documentation is generated with Sphinx.

5.3.1 Installing Sphinx and support tools

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

5.3.2 Generating documentation

Documentation is generated with Sphinx. Use `docs.sh` in the docs folder to generate the documentation. Generated docs are located at https://robotclassify.herokuapp.com/docs/index.html

CHAPTER SIX

ERROR HANDLING

Errors are returned as JSON objects in the following format:

```
{
    "success": False,
    "error": 401,
    "message": "Premission Error"
    "description": "401: Authorization header is expected."
}
```

The API returns multiple error types when requests fail:

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

CHAPTER

SEVEN

TESTING

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

CHAPTER EIGHT

DEVELOPMENT NOTES

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

CHAPTER

NINE

ROBOT CLASSIFY'S API CONTROLLERS

9.1 Introduction

RobotClassify has a number of APIs that are under RBAC control. These APIs generally return HTML data (web pages) for human interaction. These APIs can also be called machine-to-machine.

app.**index**()

Home Page

Display the home page. No access restrictions.

- **Sample Call::** curl https://robotclassify.herokuapp.com/
- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 405
{
 "description":"405 Method Not Allowed: .... requested URL.",
 "error":405,
 "message":"Method Not Allowed",
 "success":false
}
```

app.**send_documents**(*path*)

Documentation Page

Display the documetnation pages. No access restrictions.

- Sample Call:

```
curl -X GET https://robotclassify.herokuapp.com/docs/index.html
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found: The requested URL..."
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

`app.projects(payload)`

List Projects

Display a list of projects for the current user. Returns as a web page. Requires get:project auth (Editor or viewer roles.)

- Sample Call:

```
export TOKEN=...
curl -X GET https://robotclassify.herokuapp.com/projects
     -H "Authorization: Bearer $TOKEN"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 302
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
```

- Expected Fail Response:

```
HTTP Status Code: 405
{
 "description": "405 Method Not Allowed...",
 "error": 405,
 "message": "Method Not Allowed",
 "success": false
}
```

`app.show_project(payload, project_id)`

Project

Display a single project. Must be owned by the user.

Requires get:project auth (Editor or viewer roles.)

- **Sample Call::**

  curl -X GET https://robotclassify.herokuapp.com/projects/4 -H “Authorization: Bearer $TOKEN”

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
 HTTP Status Code: 404
{
 "description": "404 Not Found: The requested URL..."
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

app.**create_projects_submission**(*payload*)

Create Project

Create Project for the current user. Uploads the files and stores the uploaded files in the database.

Requires post:project auth (Editors Only). Data is sent as a form.

- Sample Call:

```
export TOKEN="edfgdfgd..."
curl -X POST https://robotclassify.herokuapp.com/projects/create
     -H "Authorization: Bearer $TOKEN"
     -F "form-project-name=New Test Project"
     -F "form-project-description=Testing Project Create"
     -F "form-project-trainingFile=@examples/titanic_train.csv"
     -F "form-project-testingFile=@examples/titanic_test.csv"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

app.**edit_project_submission**(*payload*, *project_id*)

Edit Project

Edit Project. Data is passed as a form. Requires pacht:project auth (Editors Only). Data is sent as a form.

- Sample Call to edit:

```
curl -X PATCH https://robotclassify.herokuapp.com/projects/4/edit
     -H "Authorization: Bearer $TOKEN"
     -F "form-project-name=Titanic Disaster Patch"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

- Expected Fail Response:

```
HTTP Status Code: 405
{
 "description":
 "405 Method Not Allowed: The method is not allowed...",
 "error":405,
 "message":"Method Not Allowed",
 "success":false
}
```

app.**search_projects**(*payload*)

Search Projects

Search Projects, returning a list of projects with matching criteria. Requires post:project auth (Editors Only). Search is sent as a form post.

- Sample Call to search:

```
curl -X POST https://robotclassify.herokuapp.com/projects/search
     -H "Authorization: Bearer $TOKEN"
     -F "search_term=Titanic"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

app.**delete_project**(*payload*, *project_id*)

Delete Project

Delete Project

- Sample Call:

```
curl -X DELETE https://robotclassify.herokuapp.com/                        ␣
↪projects/15/delete                 -H "Authorization: Bearer $TOKEN"
```

- Expected Success Response:

```
HTTP Status Code: 200
{"success"}
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found:  If you entered....",
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

app.**show_run**(*payload*, *run_id*)

Runs

Display a single run result. Requires get:run auth (Editors and Viewers).

- Sample Call:

```
curl -X GET https://robotclassify.herokuapp.com/runs/8
     -H "Authorization: Bearer $TOKEN"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found:  If you entered....",
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401 Unauthorized: Token not found.",
    "error": 401,
    "message": "Premission Error",
    "success": false
}
```

app.**create_run_submission**(*payload*, *project_id*)

Create Run

Create Run for a project. Requires post:run auth (Editors Only). Data is sent as a form.

- Sample Call:

```
curl -X POST https://robotclassify.herokuapp.com/runs/create/4
     -H "Authorization: Bearer $TOKEN"
     -F "form-run-name=New Curl Run"
```

(continues on next page)

(continued from previous page)

```
     -F "form-run-description=Via curl"
     -F "form-run-targetVariable=Survived"
     -F "form-run-key=PassengerId"
     -F "form-run-predictSetOut=PassengerId"
     -F "form-run-predictSetOut=Survived"
     -F "form-run-scoring=f1"
     -F "form-run-modelList=xgbc"
     -F "form-run-basicAutoMethod=True"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found:  If you entered....",
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401 Unauthorized: Token not found.",
    "error": 401,
    "message": "Premission Error",
    "success": false
}
```

app.**delete_run**(*payload*, *run_id*)

Delete Run

Delete a Run. Requires delete:run auth (Editors Only).

- Sample Call:

```
curl -X DELETE https://robotclassify.herokuapp.com/runs/17/delete
     -H "Authorization: Bearer $TOKEN"
```

- Expected Success Response:

```
 HTTP Status Code: 200
{'success'}
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found: The requested URL was....",
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

app.**edit_run_submission**(*payload*, *run_id*)

Edit Run

Edit Run record. Requires pacht:run auth (Editors Only). Data is sent as a form.

- Sample Call to edit:

```
curl -X PATCH https://robotclassify.herokuapp.com/runs/15/edit
     -H "Authorization: Bearer $TOKEN"
     -F "form-run-name=Updated Curl Run Patch"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 401
{
    "description": "401: Authorization header is expected.",
    "error": 401,
    "message": "Unauthorized",
    "success": false
}
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found: The requested URL was....",
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

app.**run_submission**(*payload*, *run_id*)

Exec Run

Run ML Training based upon run record attributes. Requires get:train auth (Editors and viewers).

- Sample Call to display:

```
curl -X GET https://robotclassify.herokuapp.com/train/13
     -H "Authorization: Bearer $TOKEN"
```

- Expected Success Response:

```
HTTP Status Code: 200
<!doctype html>...</html>
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found: The requested URL... try again.",
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

app.**download**(*payload*, *run_id*)

download results file Run ML Training based upon run record attributes

- Sample Call to display:

```
curl -X GET https://robotclassify.herokuapp.com//train/13/download
     -H "Authorization: Bearer $TOKEN"
```

- Expected Success Response:

```
HTTP Status Code: 200
File Download. Example:
    PassengerId,Survived
    892,0
    893,0
    894,1
```

- Expected Fail Response:

```
HTTP Status Code: 404
{
 "description": "404 Not Found: The requested URL... try again.",
 "error": 404,
 "message": "Not Found",
 "success": false
}
```

CHAPTER

TEN

ROBOT CLASSIFY'S API MODEL

Introduction

There are three models: - Venue - Artists - Shows

class models.**Project**(***kwargs*)

An ML Project. Projects are the top-orginizing layer for running ML problems

id

Id Primary Key - Auto assigned

account_id

account_id Account owner of the record. Restricts viewing from other users.

name

name Name of the project

description

description Description of the project

trainingFile

Training file. CSV file to be loaded for training as uploaded in the upload folder. It stored in the project record in 'savedTrainingFile' as a python pickle.

testingFile

Testing file. CSV file to be loaded for testing the trained model as uploaded in the upload folder. The testing file is designed for Kaggle comperition submissions. It is stored in the project record in 'savedTestingFile' as a python pickle

savedTrainingFile

savedTrainingFile file. The saved training file stored as a python pickle. Generated automatically.

savedTestingFile

savedTestingFile file. The saved testing file stored as a python pickle. Generated automatically.

columns

The columns of the training table. Used to populate picklist in the UI. Generated automatically.

runs

Runs associated with the project. Runs have a many to one relationship to projects.

insert()

Inserts a new model into a database. The model must have a unique id and title.

EXAMPLE:

```
p = Project()
p.insert()
```

delete()

deletes a new model into a database the model must exist in the database

EXAMPLE:

```
p = Project()
p.delete()
```

update()

updates a new model into a database the model must exist in the database

EXAMPLE:

```
p = Project.query.filter(p.id == id).one_or_none()
p.name = 'Regression Test'
p.update()
```

class models.**Run**(***kwargs*)

id

Id Primary Key - Auto assigned

name

name Name of the run.

description

description Description of the projerunct

results

results The results data for a run. This contains the output of a run. This includes the feature evaluation, feature engineering, and model training stats.

account_id

account_id Account owner of the record. Restricts viewing from other users.

project_id

project_id Parent record of the run.

targetVariable

targetVariable The variable that is to be predicted.

key

key The unique key that defines the training record.

predictSetOut

predictSetOut Defines the sheet columns to be used in the predict file. Usually two, the target variable and key. This is designed to be used for Kaggle competition submissions.

predictFile

predictFile Stores the results of the prediction for the test file. Used for downloads.

modelList

modelList List of models used in training. Best is evaluated on the scoreing choice. Valid models can include: l2, rfc, gbc, decisiontree, kneighbors, sgd, bagging, adaboost, gaussiannb, etc, svc, xgbc, stack, vote

scoring

scoring Scoring methods for model evaluation. Options include f1, r2, Perecision, Recall, and Accuracy

basicAutoMethod

basicAutoMethod There are two algorithems for feature engineering and attribute evaliuation. This selects between the two models. True for Model I False for model II

Project

Project Back reference to the parent project

insert()

Inserts a new model into a database. The model must have a unique id and title.

EXAMPLE:

```
p = Project()
p.insert()
```

delete()

deletes a new model into a database the model must exist in the database

EXAMPLE:

```
p = Project()
p.delete()
```

update()

updates a new model into a database the model must exist in the database

EXAMPLE:

```
p = Project.query.filter(p.id == id).one_or_none()
p.name = 'Regression Test'
p.update()
```

CHAPTER ELEVEN

MLLIB

11.1 *Introduction*

The Project module 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

Python Example:

```
from project import mlProject
import pandas as pd
import numpy as np
from getData import getData
import exploreData as ed
from exploreData import exploreData
from cleanData import cleanData, cleaningRules
from prepData import prepData
from trainModels import trainModels

# Create the project and set the training preferences
project = mlProject('Predict Home Values', 'EDS Real Estate Project 2')
project.setTrainingPreferences(crossValidationSplits=10,
                               parallelJobs=-1,
                               modelType='regression',
                               modelList=['lasso', 'ridge',
                                          'enet', 'rf', 'gb'])

# Import the file and set the training target to predict
project.importFile('Property Data',
                   type='GoogleSheet',
                   description='Real Estate Project 2',
                   location='1QC2v9jPJFTLYxjm-x6ic-utw0CBskU8',
                   hasHeaders=False, range='A1:Z1884')
```

(continues on next page)

(continued from previous page)

```
project.setTarget('tx_price')

# Explore the data and print out the heatmap that
# will illustrate the recommended feature engineering
# plot other statistical data
project.exploreData()
project.explore['Property Data'].plotExploreHeatMap()
project.explore[TRAININGFILENAME].plotFeatureImportance()
project.explore[TRAININGFILENAME].plotColumnImportance()
project.explore[TRAININGFILENAME].plotHistogramsAll(10)
project.explore[TRAININGFILENAME].plotCorrelations()

# Init the setup of the cleaning rules
project.runCleaningRules()


# Add manual cleaning Rules. These rules perform the feature engineering
project.addManualRuleForDefault(ed.EXPLORE_REBUCKET, 'roof', [
                                ['asphalt,shake-shingle','shake-shingle'],
                                 'Shake Shingle'])
project.addManualRuleForDefault(ed.EXPLORE_REBUCKET, 'exterior_walls', [
                                ['Rock, Stone'], 'Masonry'])
project.addManualRuleForDefault(ed.EXPLORE_REBUCKET, 'exterior_walls', [
                                ['Concrete', 'Block'], 'Concrete Block'])
project.addManualRuleForDefault(
    ed.EXPLORE_REMOVE_ITEMS_ABOVE, 'lot_size', 1220551)

# values are sparse classes and prompt to combine
project.addManualRuleForDefault(ed.EXPLORE_REBUCKET, 'exterior_walls', [
                                ['Wood Siding', 'Wood Shingle'], 'Wood'])
project.addManualRuleForDefault(ed.EXPLORE_REBUCKET, 'exterior_walls', [
                                ['Concrete Block', 'Stucco', 'Masonry',
                                 'Asbestos shingle'], 'Other'])
project.addManualRuleForDefault(ed.EXPLORE_REBUCKET, 'roof', [
                                ['Composition', 'Wood Shake/ Shingles'],
                                 'Composition Shingle'])
project.addManualRuleForDefault(ed.EXPLORE_REBUCKET, 'roof', [
                                ['Gravel/Rock', 'Roll Composition','Slate',
                                'Built-up', 'Asbestos', 'Metal'], 'Other'])

# indicator variables
# Create indicator variable for properties with 2 beds and 2 baths
project.addManualRuleForDefault(ed.EXPLORE_NEW_INDICATOR_VARIABLE,
                                'two_and_two',
                                '( beds == 2 ) & ( baths == 2 )')

# Create indicator feature for transactions between 2010 and 13 resession
project.addManualRuleForDefault(ed.EXPLORE_NEW_INDICATOR_VARIABLE,
                            'during_recession',
                            '( tx_year >= 2010 ) & ( tx_year <= 2013 )')

# Create a school score feature that num_schools * median_school
project.addManualRuleForDefault(ed.EXPLORE_NEW_VARIABLE,
                               'school_score',
                               'num_schools * median_school')

# Create a property age feature
```

(continues on next page)

(continued from previous page)

```
project.addManualRuleForDefault(ed.EXPLORE_NEW_VARIABLE,
                                'property_age', 'tx_year - year_built')
project.addManualRuleForDefault(ed.EXPLORE_REMOVE_ITEMS_BELOW,
                                'property_age', -1)

# Drop 'tx_year' and 'year_built' from the dataset
project.addManualRuleForDefault(ed.EXPLORE_DROP_COLUMN, 'tx_year', None)
project.addManualRuleForDefault(ed.EXPLORE_DROP_COLUMN, 'year_built', None)

# Now run these rules to reformat the data
project.cleanAndExploreProject()

# Prep the data for training
project.prepProjectByName('Property Data')

# Train the model
project.trainProjectByName('Property Data')

# Save the best model and one named model
project.exportBestModel('realEstateBestModel.plk')
project.exportNamedModel('gb', 'gbRealEstateModel.plk')

# Now run the predections
predict = project.createPredictFromBestModel('Property Data')
predict.importPredictFromDF(project.PullTrainingData(),
                            readyForPredict=True)
keyName, keyData = project.getKey()

# Prep the predict file
predict.prepPredict()
answer = predict.runPredict()

# Prepare the predict file for Kaggle upload
# This means taking the predicted answer and adding
# to the file for later export
predict.addToPredictFile(keyName, keyData)
if useProba:
    pass
else:
    answer = [int(x) for x in answer]
predict.addToPredictFile(targetVariable, answer)

# Prep the file for export, only keeping
# columns needed
predict.keepFromPredictFile(predictSetOut)
predict.exportPredictFile(resultsFile)
```

`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

Purpose: Auto-process an ML job for Flask Servers

Takes a data file and a few data points about the file and automatically does feature engineering and model evaluation.

Example:

```
results, pred = autoFlaskEvaluateClassifier(
     projectName=run.name,
     trainingFile=run.Project.trainingFile,
     testingFile=run.Project.testingFile,
     trainingFileDF=run.Project.savedTrainingFile,
     testingFileDF=run.Project.savedTestingFile,
     targetVariable=run.targetVariable,
     key=run.key,
     predictSetOut=run.predictSetOut,
     logFileOut=None,
     transcriptFile=None,
     trainingFileOut=None,
     predictFileOut=None,
     resultsFile='KaggleSubmitFile.csv',
     modelList=run.modelList,
     confusionMatrixLabels=None,
     scoring=run.scoring,
     setProjectGoals={'f1': (0.9, '>')},
     runVerbose=0,
     recommendOnly=True,
     basicAutoMethod=True,
     skewFactor=40.0,
     doExplore=True,
     doTrain=True,
     doPredict=True,
     toTerminal=True,
 )
```

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

mlProject is the top-level object for training and running a ML project. Various object methods are used to load, review, and train the data, as well as manage running predictions. autoFlaskEvaluateClassifier is built using this class and class methods.

Example:

```
project = mlProject('Customer Segements',
                    'clustering model should factor')
```

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 for an ML job.

Example:

```
project.setTrainingPreferences (
               crossValidationSplits=5,
               parallelJobs=-1,
               modelType=tm.TRAIN_CLASSIFICATION,
               modelList=['l2','xgbc','etc','stack'],
               useStandardScaler=True,
               gridSearchScoring='accuracy',
               testSize=.25,
               logTrainingResultsFilename=resultsFile,
               gridSearchVerbose=1,
               runHyperparameters=runHyper,
               runEstimatorHyperparameters=runEstimatorHyper,
               runMetaClassifier=runMeta)
```

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

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)
```

Example Hyper Paramaters:

```
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]
}
```

(continues on next page)

(continued from previous page)

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

setConfusionMatrixLabels(*list*)

Set the labels for the confusion matrix plot.

Example:

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

setTarget(*value*, *boolean=False*, *trueValue=None*, *convertTable=None*, *tableName=None*)

Set the target variable for supervised learning.

Call::

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

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

Example:

```
project.setTarget('loan_status')
```

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

Import a file to be used in training or predicting

Example:

```
project.importFile('Loan Data', type='csv',
        description='Lending Club Data from 2017-2018',
```

(continues on next page)

(continued from previous page)

```
        fileName='LendingClub2017_2018ready.csv',
        hasHeaders = True, isDefault=True)
```

exportFile (*name*, *filename*)

Export the named file. (Projects can have multiuple files associated with them)

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

getColumn (*name*, *columnName*)

Get a column from the data.

Example:

```
project.getColumn('Loan Data','Name')
```

dropColumn (*name*, *columnName*)

Drop column from the data/

Example:

```
project.dropColumn('Loan Data','Name')
```

saveKey (*filename*, *columnName*)

Save a column as a key.

Example:

```
project.saveKey('Loan Data','ID')
```

getKey ()

Get the key column

Example:

```
keyName, keyData = project.getKey()
```

MergeFilesAsTrainAndTest (*trainingFile*, *testingFile*)

Merge both the training and test files. Useful when doing Statistical evaluation of the complete data set.

Example:

```
project.MergeFilesAsTrainAndTest('Training','Test')
```

PullTrainingData ()

Return training and test files merged with MergeFilesAsTrainAndTest.

Example:

```
project.PullTrainingData()
```

groupByValue (*filename*, *columnList*, *value='mean'*)

Refactor data that is grouped. Uses pandas groupby function. Can recalculate grouped data as a mean of the data. (default)

Example:

```
project.groupByValue('Training',['col1','col2'])
```

exploreData (*fileName=None*)

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

Example:

```
project.exploreData()
```

initCleaningRules (*fileName=None*)

Before adding any cleaning rules you must initialize.

Example:

```
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*)

Run the cleaning rules established for a project.

Call: cleanProject(self)

Example:

```
project.cleanProject()
```

cleanAndExploreProject (*fileName=None*)

Run clean and explore together

Example:

```
project.cleanAndExploreProject()
```

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

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

Example:

```
project.prepProjectByName('Loan Data')
```

writePreppedFileByName (*filename*, *tableName=None*)

Once a file has been cleaned and explored it can be exported with this command.

Example:

```
project.writePreppedFileByName('results.csv','Titanic')
```

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

Write out the training set file by filename.

Example:

```
project.writeTrainingSetFileByName('results.csv','Titanic')
```

trainProjectByName (*tableName=None*)

Train table by tablename.

Example:

```
project.trainProjectByName('Titanic')
```

prepProjectByBatch()

Prep all of the files for training.

Example:

```
project.prepProjectByBatch()
```

trainProjectByBatch()

Train all of the files at once

Example:

```
project.trainProjectByBatch()
```

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

Exports the model with the best training results to a file.

Example:

```
project.exportBestModel('bestmodel.plk')
```

createPredictFromBestModel(*tableName=None*)

Using the best trained model in the project, create the predict object to allow for raw data to be cleaned and predicted based upon the model.

Example:

```
predict = project.createPredictFromBestModel('Property Data')
```

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

Using a named trained model from the project, create the predict object to allow for raw data to be cleaned and predicted based upon the model.

Example:

```
predict = project.createPredictFromNamedModel('xgbc')
```

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

Export the named project to a file.

Example:

```
project.exportNamedModel('gbc', 'gbRealEstateModel.plk')
```

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

Manually add a cleaning rule for the named table. Cleaning rules are used to feature engineer and refactor data.

Example:

```
project.addManualRuleForTableName('Property Data',
                                  ed.EXPLORE_NEW_VARIABLE,
                                  'school_score',
                                  'num_schools * median_school')
```

(continues on next page)

(continued from previous page)

```
# Create a property age feature
project.addManualRuleForTableName('Property Data',
                                  ed.EXPLORE_NEW_VARIABLE,
                                  'property_age',
                                  'tx_year - year_built')
project.addManualRuleForTableName('Property Data',
                                  ed.EXPLORE_REMOVE_ITEMS_BELOW,
                                  'property_age', -1)

# Drop 'tx_year' and 'year_built' from the dataset
project.addManualRuleForTableName('Property Data',
                                  ed.EXPLORE_DROP_COLUMN,
                                  'tx_year', None)
project.addManualRuleForTableName('Property Data',
                                  ed.EXPLORE_DROP_COLUMN,
                                  'year_built', None)
```

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

Manually add a cleaning rule for the default table. Cleaning rules are used to feature engineer and refactor data.

Example:

```
project.addManualRuleForDefault(ed.EXPLORE_NEW_VARIABLE,
                               'school_score',
                               'num_schools * median_school')

# Create a property age feature
project.addManualRuleForDefault(ed.EXPLORE_NEW_VARIABLE,
                                'property_age',
                                'tx_year - year_built')
project.addManualRuleForDefault(ed.EXPLORE_REMOVE_ITEMS_BELOW,
                                'property_age', -1)

# Drop 'tx_year' and 'year_built' from the dataset
project.addManualRuleForDefault(ed.EXPLORE_DROP_COLUMN,
                                'tx_year', None)
project.addManualRuleForDefault(ed.EXPLORE_DROP_COLUMN,
                                'year_built', None)
```

setGoals (*goals*)

Set project goals for model training. Does not change how a model is trained, just added to reporting

Example:

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

setOngoingReporting (*flag*, *fileName*)

Allow for the training data to be recorded.

Example:

```
project.setOngoingReporting(True,'Loan Data')
```

displayAllScores (*fileName*, *short=False*)

Display the results of the model training. This is an extensive report.

Example:

```
project.displayAllScores('Loan Data')
```

reportResultsOnTrainedModel (*fileName*, *modelName*)

Report the funak results on the traibn mdoel

Example:

```
project.reportResultsOnTrainedModel('Titanic','xgbc')
```

showFeatureImportances (*fileName*, *modelName*)

Report on the feature importance of columns in a datafile based upon the model.

Example:

```
project.showFeatureImportances('Titanic', 'xgbc')
```

logTrainingResultsRunDescription (*description='None'*)

Give a run a name to be added to the training log results

Example:

```
project.logTrainingResultsRunDescription('First Run')
```

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

Log training results to a file.

Example:

```
project.logTrainingResults('Training File','TrainingLog.csv')
```

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

Predict a previously trained project. Predict objects allow for raw data to be loaded and cleaned with cleaning rules from the project as well as run predictions from the previously trained model.

Example:

```
predict = project.createPredictFromBestModel('Property Data')
```

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

Import a file for prediction.

Example:

```
predict = importPredictFile('Titanic','/files/titanic_train.csv')
```

importPredictFileFromProject (*project*, *tableName*)

Get a file loaded in a project and load it as the predict file.

Example:

```
predict = importPredictFileFromProject(project, 'Titanic')
```

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

From an existing dataframne, import the data for prediction. (Rather than from a file)

Example:

```
predict = importPredictFromDF(df)
```

prepPredict()

Get the predict data ready for prediction.

Example:

```
predict = prepPredict()
```

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

Export a preped predict file. (Post cleaning rules.)

Example:

```
predict = exportPreppedFile('readydata.csv')
```

getColumn(*columnName*)

Get a columns from the data file

Example:

```
predict.getColumn('Name')
```

exportPredictClass(*filename*)

Export the predict class object to a file (for later use).

Example:

```
predict = exportPredictClass('save.pkl')
```

addToPredictFile(*columnName*, *columnData*)

Add a column to the predict file.

Example:: id = project.getKey() predict = addToPredictFile('id',id)

removeFromPredictFile(*columns*)

Remove listed columns from the predict file.

Example:

```
predict = removeFromPredictFile(['tmp','description'])
```

keepFromPredictFile(*columns*)

Remove all columns from the predict file except those that are listed.

Example:

```
predict = keepFromPredictFile(['ID','Survived'])
```

exportPredictFile(*filename*)

Export the predicted file as a csv.

Example:

```
predict = exportPredictFile('Kaggle.csv')
```

getPredictFileDF()

Get, from the dataframe, the predicted file.

Example:

```
predict = predictProject()
```

runPredict()

After setup, run the prediction on the defined file.

Example:

```
predict = project.createPredictFromBestModel('Property Data')
predict.importPredictFromDF(project.PullTrainingData(),
                            readyForPredict=True)
keyName, keyData = project.getKey()

# Prep the predict file
predict.prepPredict()
answer = predict.runPredict()
```

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

Load the predit object (used to run predictions) from a saved file. The file is stored as a Python pickle file.

Example:

```
predict = loadPrecictProject('')
```

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

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

Example:

```
plot_confusion_matrix(matrix, Normalize=True)
```

CHAPTER TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX