Project

Machine Learning on CLAMP

Learning Goals of this Project:

Students will learn introductory level concepts about Data Science and Machine Learning as it can be applied to the Cybersecurity Domain. This lab develops understanding of the general data science process and commonly used python libraries like pandas and sci-kit learn.

The final deliverables:

Complete the Canvas Quiz.
Submit the 5 python files to Gradescope. The files should be named ***task1.py, task2.py, task3.py, task4.py and task5.py*** and should implement the functions described below (and in the starter code).

Important Reference Material :

- API Reference — scikit-learn 1.1.2 documentation
- https://www.kdnuggets.com/2016/03/data-science-process.html
- Getting started — pandas 1.4.3 documentation (pydata.org)
- Python Cheat Sheet for Data Science (elitedatascience.com)

Submission:

Gradescope (autograded)
Canvas Quiz

Introduction

You may be asking yourself “what is the importance of learning about Data Science and Machine Learning in a cybersecurity class?”, The short answer is that data science is a useful set of tools to handle the massive amount of data that flow through IT systems and it is used by many security teams either explicitly or within tools/programs they use so it is important to get a basic understanding of how it works. This Project will go through a simplified scenario where data science can be used, if this sparks your interest there are plenty of other ML focused classes at GaTech that you may be interested in taking, as well as a wealth of training materials on Youtube, Coursera, Udacity, Udemy, DataCamp etc that you could use to go deeper into the field.

Scenario:

You are an analyst on a security team for a midsized software company that runs a messaging app (a slack, gchat, microsoft teams competitor). It is Monday morning and you see an email from your manager setting up a meeting to discuss a new security feature that needs to be implemented in the product ASAP. You join the meeting and learn that recently there has been a big uptick in malicious executable files being sent over the chat app and it is starting to generate bad press for the company. A few analysts on the team already worked on analyzing a set of files sent over the app and classifying them as malicious or benign. They also used a python library (pefile) to get some attributes of each executable file and have created a CSV with those extracted attributes and a column with the name **class** with a 1 denoting a malicious file and a 0 denoting a benign files. They documented their preprocessing work in a readme in the git repo (urwithajit9/ClaMP: A Malware classifier dataset built with header fields' values of Portable Executable files (github.com)) and shared the repo with software engineers so they can get to work writing code that will generate those features for every executable file sent over the messaging app. Your boss turns to you and says I would like you to help us to understand a bit more about how big of a problem this is on our app and write a model that takes in these features and produces a propensity score from 0 to 1 where scores closer to 0 mean a low likelihood of the file being malicious and closer to a 1 means a higher likelihood of a file being malicious. Also since the team may want to reuse this type of work in the future for different types of files or with different extracted attributes you should create functions that can be used in the future with minimal rework. Once you produce a model, you will share your code and the trained model file with the software engineers who will integrate the model into the messaging app and will score all files uploaded to the app.

General Advice

- Develop locally then test in the autograder when you are confident your code runs without errors. You can run the python files locally, develop in a local vscode/jupyter notebook or on a hosted web notebook like google colab.
- Do **not** use print statements in your gradescope submissions. While print statements are useful to debug issues locally in an autograder context they can leak sensitive answer information. We have detections in gradescope that will block you from viewing scores/outputs from your code if you use print statements in any of your submitted code. If you try to bruteforce/hack/game the autograder or extract information we will give you a 0 for the whole assignment.
- Read the python library documentation. You will be using pandas, scikit-learn, yellowbrick
- Do not hard code solutions (see screenshot below for what not to do):

```python
def find_dataset_statistics(dataset:pd.DataFrame,target_col:str) -> tuple[int,int,int,int,float]:
    # TODO: Write the necessary code to generate the following dataset statistics given a dataframe
    # and a target column name.

    #Total number of records
    n_records = 4
    #Total number of columns
    n_columns = 2
    #Number of records where target is negative
    n_negative = 2
    #Number of records where where target is positive
    n_positive = 2
    # Percentage of instances of positive target value
    perc_positive =  50

    return n_records,n_columns,n_negative,n_positive,perc_positive
```

Task 0 (20 points)

We have a Canvas quiz that is meant to test that you have read the library documentation for the packages we use for this class. It is not meant to be tricky and can be completed before you start the project or after you finish it.

Useful Links:

- scikit-learn Documentation

Deliverables:

- Complete Canvas Quiz

Task 1 (5 points)

Lets first get familiar with some pandas basics. Pandas is a library that handles data frames which you can think of as a python class that handles tabular data. In this section you will make a very simple function that takes in a pandas dataframe of the file attributes and classifications and returns some simple attributes. See the function skeleton and implement a count of rows, count of columns, count of rows where the classification is 1 (positive), count of rows where the classification is 0 (negative) and a percentage of classification of 1 (percent positive) in the dataset's target column. Generally in the real world you would also use plotting tools like PowerBi, Tableau, Data Studio, Matplotlib etc to create graphics and other visuals to better understand the dataset you are working with, this step is generally known as Exploratory Data Analysis. Since we are using an autograder for this class we will skip the plotting for this project. For this task we have released a test suite on ed discussion if you are struggling to run things locally please set that up and use it to debug your function.

Useful Links:

- pandas documentation — pandas 1.5.3 documentation (pydata.org)
- What is Exploratory Data Analysis? | IBM
- Top Data Visualization Tools | KDnuggets
- CS 6035 O01, OCY – Ed Discussion (edstem.org)

Deliverables:

- Complete find_dataset_statistics function in task1.py
- Submit task1.py to gradescope

Task 2 (10 points)

Now that you have a basic understanding of pandas and the dataset it is time to dive into some more complex data processing tasks. The first subtask in this task is splitting your dataset into both features and targets (columns) and splitting your dataset into training and test sets (rows). These are basic concepts in model building but at a high level it is important to hold out a subset of your data when you train a model so you can see what the expected performance is on unseen samples and so you can determine if the resulting model is overfit (performs much better on training data vs test data). Preprocessing data is important since most models only take in numerical values so categorical features need to be "encoded" to numerical values so models can use them. Numerical scaling can be more or less useful depending on the type of model used but is especially important in linear models. These preprocessing techniques will provide you options to augment your dataset and improve model performance.

For one hot encoding and scaling functions you should return a dataframe with the encoded/scaled columns concatenated to the 'other' columns you did not transform. For the PCA functions you should return just the PCA dataframe. For the feature engineering dataframe you should return the feature engineered column attached to the input dataframe.

Example Output for one hot encoding (where color and version are encoded):

Starting DF				Encoded DF					
color	version	price		color_red	color_blue	color_green	version_1	version_2	price
red	1	5.39		1	0	0	1	0	5.39
blue	1	5.39		0	1	0	1	0	5.39
green	2	10.55		0	0	1	0	1	10.55
red	2	10.55		1	0	0	0	1	10.55

Note: For these functions (and in data science/ML in general) you should be training/fitting to the train set and predicting/transforming on the train and test sets.

Note: for onehot encoding please use the format columnName_columnValue for the new encoded column names. Ie in the sklearn example of a column named `gender` and values `male` and `female`, you would put `gender_male` and `gender_female` as your new column names after one hot encoding. If the test set has a third gender then you would denote that value you didn't see at training time with 0's for each encoded column.

Note: When using sklearn.preprocessing.OneHotEncoder check the output from the transformation it may cause issues if it is not a format that pandas dataframes expect. To see this and to debug it, try running your code locally (or on a google colab notebook) using the training csv with a few categorical columns.

Note: for PCA you should drop columns with na values before running fitting or transforming methods (in the real world you could also try to fill those missing values but for this project that is out of our scope)

Note: if you see the error `Test Failed: Found unknown categories` for one hot encoding make sure you check the initialization of the OneHotEncoder and make sure you are handling values in the test set that were not in the training set (we want to set those unseen values encoded as all 0s for that encoded column as we describe in the 2nd Note above)

Note: For this task we have released a test suite on ed discussion if you are struggling to run things locally please set that up and use it to debug your function.

Useful Links:

- *Training and Test Sets | Machine Learning | Google Developers*
- Bias–variance tradeoff - Wikipedia
- https://en.wikipedia.org/wiki/Overfitting
- Categorical and Numerical Types of Data | 365 Data Science
- scikit-learn: machine learning in Python — scikit-learn 1.2.1 documentation
- CS 6035 O01, OCY – Ed Discussion (edstem.org)

Deliverables:

- Make use of the scikit-learn (sklearn) python package in your function implementations
- Complete *train_test_split* function
    - Using the train_test_split function from sklearn implement a function that given a dataset, target column, test size, random state and *True/False Value for stratify* will return train_features (DataFrame), test_features (DataFrame), train_targets (Series) and test_targets (Series)
    - Hint: write your code in a way that handles a case where we want to stratify vs where we don't want to stratify (dont use stratify directly as an input to the sklearn function)
- Complete *PreprocessDataset* class in task2.py
    - one_hot_encode_columns_train
        - Given training features (DataFrame), one_hot_encode_cols (list of column names) and using sklearn's OneHotEncoder. Split the data into columns that should be encoded and those that should be passed through then fit the encoder, transform the training data. You will keep the column names the same as their input column names.(**NOTE**: *make sure this new df uses the row indexes corresponding to the input dataframe*). Finally join the encoded columns with the columns from above that should be passed through. Your final results should be a Dataframe with columns in the one_hot_encode_cols list encoded and all other columns untouched.

- one_hot_encode_columns_test
  - Given training features (DataFrame), one_hot_encode_cols (list of column names) and using sklearn's OneHotEncoder, split the data into columns that should be encoded and those that should be passed through then using the encoder fit on the training data, transform the test data. You will keep the column names the same as their input column names. (**NOTE**: *make sure this new df uses the row indexes corresponding to the input dataframe*). Finally join the encoded columns with the columns from above that should be passed through. Your final results should be a Dataframe with columns in the one_hot_encode_cols list encoded and all other columns untouched.
- min_max_scaled_columns_train
  - Given training features (DataFrame), min_max_scale_cols (list of column names) and using sklearn's MinMaxScaler, Split the data into columns that should be scaled and those that should be passed through then fit the scaler, transform the training data and create a dataframe with column names the same as the pre-scaled feature.(**NOTE**: *make sure this new df uses the row indexes corresponding to the input dataframe*) Finally join the scaled columns with the columns from above that should be passed through. Your final results should be a Dataframe with columns in the min_max_scale_cols list scaled and all other columns untouched.
- min_max_scaled_columns_test
  - Given training features (DataFrame), min_max_scale_cols (list of column names) and using sklearn's MinMaxScaler, split the data into columns that should be scaled and those that should be passed through then using the scaler fit on the training data, transform the test data and create a dataframe with column names the same as the pre-scaled feature. (**NOTE**: *make sure this new df uses the row indexes corresponding to the input dataframe*) Finally join the scaled columns with the columns from above that should be passed through. Your final results should be a Dataframe with columns in the min_max_scale_cols list scaled and all other columns untouched.
- pca_train
  - Given training features (DataFrame), n_components (int) and using sklearn's PCA, initialize PCA with a random seed of 0 and n_components, train PCA on the training features dropping any columns that have NA values then transform the training set using PCA and create a DataFrame with column names component_1, component_2 .. component_n for each component you created (**NOTE**: *this new df (for the autograder this semester) should have an index from 0 to n which will not match the the row indexes corresponding to the input dataframe*)
- pca_test
  - Given test features (DataFrame), n_components (int) and using sklearn's PCA, using the above trained PCA and dropping any columns that have

NA values then transform the test set using PCA and create a DataFrame with column names component_1, component_2 .. component_n for each component you created (**NOTE**: *this new df (for the autograder this semester) should have an index from 0 to n which will not match the the row indexes corresponding to the input dataframe*)

- feature_engineering_train
  - Given training features (DataFrame) and a with the feature engineering functions passed in a dict with the format {'feature_name':function,} for each 'feature_name' in the dict, create columns of name in the training DataFrame by passing the training feature dataframe to the associated function. The Returned Dataframe will consist of the input dataframe with the additional feature engineered columns from the dict (**NOTE**: *make sure this new df uses the row indexes corresponding to the input dataframe*)
- feature_engineering_test
  - Given test features (DataFrame) and a with the feature engineering functions passed in a dict with the format {'feature_name':function,} for each 'feature_name' in the dict, create columns of name in the test DataFrame by passing the test feature dataframe to the associated function. The Returned Dataframe will consist of the input dataframe with the additional feature engineered columns from the dict (**NOTE**: *make sure this new df uses the row indexes corresponding to the input dataframe*)
- preprocess
  - Given a Training Features (DataFrame), Test Features (DataFrame) and the functions you created above, return Training and Test Dataframes with the one_hot_encode_cols encoded, min_max_scale_cols scaled, features described in the feature_engineering_functions engineered and any columns not affected by the above functions passed through to the output the same as they were in the input. (**NOTE**: *make sure this new df uses the row indexes corresponding to the input dataframe*)

- Submit task2.py to Gradescope

Task 3 (15 points)

So far we have functions to split the data and preprocessed it. Now we will run a basic model on the data to cluster files (rows) with similar attributes together. We will use an unsupervised (model with no target column) model, Kmeans, since it is simple to use and understand. Please use scikit-learn to create the model and Yellowbrick to determine the optimal value of k for our dataset.

Useful Links:

- Unsupervised learning - Wikipedia

- What is Clustering? | Machine Learning | Google Developers
- ML | K-means++ Algorithm - GeeksforGeeks
- scikit-learn: machine learning in Python — scikit-learn 1.2.1 documentation
- Yellowbrick: Machine Learning Visualization — Yellowbrick v1.5 documentation (scikit-yb.org)

Deliverables:

- Make use of the scikit-learn (sklearn) and yellowbrick python packages in your function implementations
- Complete the *KmeansClustering* class in task3.py
    - kmeans_train
        - Initialize a sklearn Kmeans model using random_state, n_init =10. Initialize a yellowbrick KElbowVisualizer to search for the optimal value of k (between 1 and 10). Train the KElbowVisualizer on the training data and determine the optimal k value. Then Train a Kmeans model with the proper initialization for that optimal value of k and return the cluster ids for each row of the training set as a list.
    - kmeans_test
        - Using the model you trained in the previous function return the cluster ids for each row of the test set as a list.
    - train_add_kmeans_cluster_id_feature
        - Using kmeans_train add an additional column to the training features and return the training dataframe with all input features untouched and the additional cluster id column with the column name “kmeans_cluster_id”
    - test_add_kmeans_cluster_id_feature
        - Using kmeans_test add an additional column to the test features and return the test dataframe with all input features untouched and the additional cluster id column with the column name “kmeans_cluster_id”
- Submit task3.py to Gradescope

Task 4 (25 points)

Finally we are ready to try a few different supervised classification models. We have chosen a few commonly used models for you to use here but there are many options and in the real world specific algorithms may fit a specific dataset better. You also won’t be doing any hyperparameter tuning yet to better focus on writing the code. You will train a model using the training set, predict on the training/test sets and calculate performance metrics and return a *ModelMetrics* object and trained scikit-learn model from each model function. (Note: You should use RFE for determining feature importance of the **logistic regression** model but do **NOT** use RFE for **random forest** or **gradient boosting** models to determine feature importance please use their built in values for this)

Useful Links:

- Supervised Learning | Machine Learning | Google Developers

- scikit-learn: machine learning in Python — scikit-learn 1.2.1 documentation
- Classification | Machine Learning | Google Developers
- Classification: True vs. False and Positive vs. Negative | Machine Learning | Google Developers
- Classification: Accuracy | Machine Learning | Google Developers
- Classification: Precision and Recall | Machine Learning | Google Developers
- Classification: ROC Curve and AUC | Machine Learning | Google Developers

Deliverables:

- Make use of the scikit-learn (sklearn) python package in your function implementations
- Complete the Following Functions in task4.py:
    - *calculate_naive_metrics*
        - Given a train dataframe, test dataframe, target_col and naive assumption split out the target column from the training and test dataframes to create a feature dataframes and a target series then calculate (rounded to 4 decimal places) accuracy, recall, precision and f1 score using the sklearn functions, the train and test target values and the naive assumption.
    - *calculate_logistic_regression_metrics*
        - Given a train dataframe, test dataframe, target_col and logreg_kwargs split out the target column from the training and test dataframes to create a feature dataframes and a target series. Then train a logistic regression model (initialized using the kwargs) on the training data and predict (both binary predictions and probability estimates) on the training and test data. Then using those predictions and estimates along with the target values calculate (rounded to 4 decimal places) accuracy, recall, precision, f1 score, false positive rate, false negative rate and area under the reciever operator curve (using probabilities for roc auc) for both training and test datasets.
        - For Feature Importance use the top 10 features selected by RFE and sort by absolute value of the coefficient from biggest to smallest (make sure you use the same feature and importance column names as ModelMetrics shows in feat_name_col and imp_col and the index is 0-9 you can do that with `df.reset_index(drop=True)`)
    - *calculate_random_forest_metrics*
        - Given a train dataframe, test dataframe, target_col and rf_kwargs split out the target column from the training and test dataframes to create a feature dataframes and a target series. Then train a random forest model (initialized using the kwargs) on the training data and predict (both binary predictions and probability estimates) on the training and test data. Then using those predictions and estimates along with the target values calculate (rounded to 4 decimal places) accuracy, recall, precision, f1 score, false positive rate, false negative rate and area under the reciever operator curve (using probabilities for roc auc) for both training and test datasets

        - For Feature Importance use the top 10 features using the built in feature importance attributes as sorted from biggest to smallest (make sure you use the same feature and importance column names as ModelMetrics shows in feat_name_col and imp_col and the index is 0-9 you can do that with `df.reset_index(drop=True)`)
    - *calculate_gradient_boosting_metrics*
        - Given a train dataframe, test dataframe, target_col and gb_kwargs split out the target column from the training and test dataframes to create a feature dataframes and a target series. Then train a gradient boosting model (initialized using the kwargs) on the training data and predict (both binary predictions and probability estimates) on the training and test data. Then using those predictions and estimates along with the target values calculate (rounded to 4 decimal places) accuracy, recall, precision, f1 score, false positive rate, false negative rate and area under the reciever operator curve (using probabilities for roc auc) for both training and test datasets
        - For Feature Importance use the top 10 features using the built in feature importance attributes as sorted from biggest to smallest (make sure you use the same feature and importance column names as ModelMetrics shows in feat_name_col and imp_col and the index is 0-9 you can do that with `df.reset_index(drop=True)`)
- Submit task4.py to Gradescope

Task 5 (25 points)

Now that you have written functions for different steps of the model building process you will put it all together. You will write code that trains a model with hyperparameters you determine (you should do any tuning locally or in a notebook ie don't tune your model in gradescope since the autograder will likely timeout). It will take in the CLAMP training data (reminder that the “class” column is the target for this dataset), train a model then predict on a test set and output values from 0 to 1 for each row and our autograder will compare your predictions with the correct answers and to get credit you will need a roc auc score of .9 or higher on the test set (should not require much hyperparameter tuning for this dataset). This is basically a simulation of how your model would perform in the “production” system using batch inference.

Deliverables:

- Make use of any of the techniques we covered in this project to train a model and return predicted probabilities for each row of the test set as a DataFrame with columns **index** (same as your index from the input test df) and **malware_score** (predicted probabilities).
- Complete the *train_model_return_scores* function in task5.py

- Submit task5.py to Gradescope

FAQs

- How many submissions do we have in Gradescope?
    - Answer: Unlimited

- How many Attempts do we get on the quiz?
    - You get one submission but the time is unlimited (until the due date) so don't submit until you are finished. Canvas will allow you to close the quiz window and should save your quiz progress but it may be a good idea to note your answers to the quiz somewhere on your computer if you are going to take multiple days to finish it just in case something happens to canvas.

- When are office hours for this project?
    - Answer: There will be a pinned Ed Discussion Post with office hour date/times as well as recordings after they take place.

- I need an extension for this project
    - Answer: Open a private Ed Discussion post with your situation and we will determine if it is an approved reason for an extension and decide how many extra days you will get. Note: The earlier you tell us the more likely that we can give you an extension (ie asking the day it is due will make it very unlikely that you will get an approved extension)

- I am overwhelmed and don't know where to start.
    - Answer: Start simple with reviewing the useful links we have provided and doing the coding tasks (tasks 1-5) in order. They will somewhat build on each other and will get progressively harder so early tasks are easier to complete.

- I am using RFE to find the feature importance of a random forest or gradient boosting model and it is running for a long time and timing out in the autograder
    - Answer: Only use RFE for logistic regression models and use the built in values of feature importance for random forest and gradient boosting models.
- Should I make my own post related to this project in Ed Discussion?
    - Answer: Please try to ask your question in one of the pinned project posts and remove answer data (ie don't post your code, even snippets) or other information that should not be publicly shared and ask in the public forum so others can benefit from your questions.

- I can't see any scores/output in the autograder is it broken?

    - Answer: We have a protection in the autograder to prevent printing sensitive information so if your code has print statements then you wont see your score or any outputs of the autograder. Please resubmit your code with print statements removed and you should see the normal outputs.

- Can you review my code in a private Ed Discussion Post?
    - Answer: Since we have a Gradescope autograder we will not be reviewing code of students and expect you to debug your code using information in public posts in Ed Discussion or via google searches/stack overflow.

- I think I found a bug in the Autograder
    - Answer: Open a private Ed Discussion Post and we can take a look. This is the first semester we are running this version of the project and while we tested it internally with the TAs there is a chance that we missed something. If this happens we will make an update to the autograder to fix it and will make a pinned post letting students know the autograder was changed.

- I have constructive feedback that can improve this project for next semester
    - Answer: Open a private Ed Discussion Post and we will review your ideas and may implement them in a future semester.