Department of Computer Science and Engineering, SVNIT, Surat

MTech I (2^{nd} semester)

CSE614: Core Elective II: Machine Learning for Security

Lab Assignment No 2: ML Applications in Security #1

SPRING Semester 2022-23

Dated Uploaded: 21^{st} January 2023

Instructions:

1. *Right from the time the assignment is uploaded the students must start implementing the assignments.*
2. *You must use the specified toolkit and/or library and/or packages for your implementation.*
3. *All the assignments must be submitted in the form of a zip file containing the Program Source, the screenshot of the output that you obtained, the DataSet/Input Test data used and a ReadMe .txt file explaining what platform to use, what are the input parameters required for execution and how to execute it. Also write your conclusion in ReadMe file.*
4. *Perform usual error checking. Don't go overboard on this, but don't let your program die because of divide by zero.*
5. *Remember, your programs could be checked by a code-cheating program, so please follow the code of academic integrity.*
6. *There will be a viva for each assignment. This viva would be conducted for this assignment on a future date as specified.*
7. ***Maximum Points: 10 per question***

---

1. Machine Learning: Dataset splitting: Note that one of the major challenges for a machine learning practitioner is the danger of overfitting – creating a model that performs well on the training data but is not able to generalize to new, previously-unseen data. In order to combat the problem of over-fitting, machine learning practitioners set aside a portion of the data, called test data, and use it only to assess the performance of the trained model, as opposed to including it as part of the training dataset. This careful setting aside of testing sets is key to training classifiers in cybersecurity, where over-fitting is an omnipresent danger. One small oversight, such as using only benign data from one locale, can lead to a poor classifier. There are various other ways to validate model performance, such as cross-validation.
   However, for this problem you would use `scikit-learn`, `pandas` packages in `pip` and `train_test_split` module. The basic aim of this exercise is to create a program that is able to perform preparation of the train and test data.
   Thus, in this first assignment the focus will be mainly on *train-test splitting*. To start with,
   (a) First, you would install the `scikit-learn` and `pandas` packages in pip.
   (b) You have to use the North Korea missile test database (.csv file) (ref: `https://www.nti.org/analysis/articles/cns-north-korea-missile-test-database/`). It is already uploaded on the classroom.
   (c) Start by importing the `train_test_split` module and the `pandas` library, and read features into X and labels into y.
   (d) Standardize X using a `StandardScaler` instance:
   (e) Next, randomly split the dataset and its labels into a training set consisting 80% of the size of the original dataset and a testing set 20% of the size.
   (f) Apply the train_test_split method once more, to obtain a validation set, X_val and y_val.
   (g) Train-test the dataset.
2. Machine Learning: Classification, Network Attacks classification: This assignment problem requires you to study building a Predictive Model to classify network attacks i.e. demonstrate how to build *a network attack classifier* from scratch using machine learning. The dataset that you have to use the *NSL-KDD dataset*, that is an improvement to a classic network intrusion detection dataset used widely by security data science professionals. The original 1999 KDD Cup dataset was created for the DARPA Intrusion Detection Evaluation Program,

prepared and managed by MIT Lincoln Laboratory. This was studied and analyzed in the lab assignment #1, already. Your task now is to devise a general classifier that categorizes each individual sample as one of five classes: *benign, dos, r2l, u2r, or probe*. The training dataset contains samples that are labeled with the specific attack: `ftp_write` and `guess_passwd` attacks correspond to the `r2l` category, `smurf` and `udpstorm` correspond to the `dos` category, and so on. The mapping from attack labels to attack categories is specified in the file `training_attack_types.txt` on the classroom. Now, write appropriate code/commands to do the following:

(a) Write appropriate commands to readin and process the dataset. That is, write the commands that list out the 41 attack types specified and the category they belong to. To be specific your output should print the attack type and then list out the attacks in that type, as shown in the following partial example.

```
{
        'benign': ['normal'],
        'probe':  ['nmap', 'ipsweep', 'portsweep', 'satan',
        'mscan', 'saint', 'worm'],
        .... : ....
        ....: ....
}
```

(b) Now, since this data output is not very convenient to perform the mappings from attack labels to attack categories, invert this dictionary in preparation for data crunching, to get the output as follows. LEt there be two files viz. `train_file` and `test_file`, prepared as a result.

```
{
'apache2': 'dos',
'back': 'dos',
'guess_passwd': 'r2l',
'httptunnel': 'u2r',
'imap': 'r2l',
...
}
```

(c) It is always important to consider the class distribution within the training data and test data. In some scenarios, it can be difficult to accurately predict the class distribution of real-life data, but it is useful to have a general idea of what is expected. For instance, when designing a network attack classifier for deployment on a network that does not contain any database servers, one might expect to see very little `sqlattack` traffic.
One can make this conjecture through educated guessing or based on past experience. However, here in this example, since you have access to the test and training data, you have to use the training data to get its distribution, using appropriate commands. Look at the `attack_type` and `attack_category` distributions and use `matplotlib.pyplot` to plot the distributions. Your plot must show Training data class distribution (for a 5-category breakdown) and another plot for Training data class distribution (for a 22-category breakdown).

With this encounter with the data set now, give appropriate commands for the ML life cycle tasks to classify the data starting with the data preparation i.e.

(d) Begin with splitting the test and training DataFrames into data and labels.

(e) Then, before you can apply any algorithms to the data, you need to prepare the data for consumption. Hence, first encode the categorical/dummy variables (referred to in the dataset as symbolic variables). For convenience, generate the list of categorical variable names and a list of continuous variable names. Your commands should give a dictionary containing two keys, *continuous* and *symbolic*, each mapping to a list of feature names, as shown below:

```
{
continuous: [ duration, src_bytes, dst_bytes, wrong_fragment, ... ]
symbolic:   [ protocol_type, service, flag, land, logged_in, ... ]
}
```

(f) Further split the symbolic variables into nominal (categorical) and binary types. This is so because, you will preprocess them differently.

(g) Then,use the `pandas.get_dummies())` function to convert the nominal variables into dummy variables. For the purpose, first combine (`pd_combine(), train_x_raw and test_x_raw)`, and then run the dataset through `pandas.get_dummies()` function, and then separate it into training and test sets again. This is necessary because there might be some symbolic variable values that appear in one dataset and not the other, and separately generating dummy variables for them would result in inconsistencies in the columns of both datasets.

You should also observe that the function `pandas.get_dummies())` applies one-hot encoding (*find out what is it*) to categorical (nominal) variables such as *flag*, creating multiple binary variables for each possible value of flag that appears in the dataset. For instance, if a sample has value `flag=S2`, its dummy variable representation (for flag) will be as follows:

```
# flag_S0 , flag_S1 , flag_S2 , flag_S3 , flag_SF , flag_SH
[     0,         0,         1,         0,         0,         0     ]
```

For each sample, only one of these variables can have the value 1; hence the name *one-hot.*

(h) Next, observe and analyze the distribution of the training set features, and notice if there is anything that is of concern there or not. Give your comments, clearly. Specifically, give clear observations in whether is it required to perform *feature value standardization/normalization* or not. For example, is it that without performing feature value standardization/normalization, there is some feature that is dominating, causing the model to miss out on potentially important information in some other features ?

If so, first understand that Standardization is a process that rescales a data series to have a mean of 0 and a standard deviation of 1 (unit variance). It is a common, but frequently overlooked, requirement for many machine learning algorithms, and useful whenever features in the training data vary widely in their distribution characteristics. The `scikit-learn` library includes the `sklearn.preprocessing.StandardScaler` class that provides this functionality. Use it wherever required.

Note also that an alternative to *standardization* is *normalization*, which rescales the data to a given range - frequently [0,1] or [-1,1]. The `sklearn.preprocessing.MinMaxScaler` class scales a feature from its original range to `[min, max]`. One may choose to use `MinMaxScaler` over `StandardScaler` if one wants the scaling operation to preserve small standard deviations of the original series, or if one wants to preserve zero entries in sparse data.

You should also note and learn that *Outliers* in data can severely and negatively skews standard scaling and normalization results. If the data contains outliers, `sklearn.preprocessing, RobustScaler` could be more suitable for the job. `RobustScaler` uses robust estimates such as the *median* and *quantile* ranges, so it will not be affected as much by outliers.

In addition, whenever performing standardization or normalization of the data, one must apply consistent transformations to both the training and test sets (i.e., using the same mean, std, etc. to scale the data). Fitting a single Scaler to both test and training sets or having separate Scalers for test data and training data is incorrect, and will optimistically bias classification results.

Lastly, when performing any data preprocessing, one should pay careful attention to leaking information about the test set at any point in time. Using test data to scale training data will leak information about the test set to the training operation and cause test results to be unreliable. `Scikit-learn` provides a convenient way to do proper normalization for cross-validation processes — after creating the Scaler object and fitting it to the training data, one can simply reuse the same object to transform the test data.

(i) Now that the data is prepared and ready to go, analyze what options you have for actually classifying attacks. This is a five-class classification problem in which each sample belongs to one of the following classes: `benign, u2r, r2l, dos, probe`. It is to be noted that there are many different classification algorithms suitable for a problem like this. Many classification algorithms inherently support multi-class data (e.g., decision trees, nearest neighbors, Naive Bayes, multinomial logistic regression), whereas others do not (e.g., support vector machines). Keep in mind that developing machine learning solutions is an iterative process. Spending time and effort to iterate on a rough initial solution will almost always bring about surprising learnings and results. With this backdrop analyze and study how would you choose the ML classifier algorithm, for this application?

You may take help in that `Scikit-learn` provides a machine learning algorithm *cheat-sheet* that gives a good overview of how to choose a machine learning algorithm. The

cheat sheet, though not complete, provides some intuition on algorithm selection. In general, some of the questions you should ask when faced with machine learning algorithm selection, are as follows:

- What is the size of your training set?
- Are you predicting a sample's category or a quantitative value?
- Do you have labeled data? How much labeled data do you have?
- Do you know the number of result categories?
- How much time and resources do you have to train the model?
- How much time and resources do you have to make predictions?

(j) Answer all the above questions for this application/dataset as well as give your comments on which classifier to use. Now, subject the data to a simple decision tree classifier, `sklearn.tree.DecisionTreeClassifier`. What is the classification accuracy that you are obtaining? What is the confusion matrix you have used? How do you interpret the results and the confusion matrix?

(k) Next, use other classifiers
viz. `sklearn.neighbors.KNeighborsClassifier, sklearn.svm.LinearSVC`, and show your confusion matrix and interpret, compare and analyze the results in all the three cases.

3. Machine Learning: Classification: Missile Dataset classification:This assignment is also on working around with the classifiers. You would use the dataset as in the problem at Serial no 1, and classify the data. That is,

   (a) after completing the step 1(g) in assignment at Sr no 1, instantiate and train using the following classifiers (a) Logistic Regression (b) Decision Tree (c) isolation forest classifier (d) Support Vector Machine (e) K Nearest Neighbor (f) Naive Bias classifiers.

   (b) Score the classifiers on normal and anomalous observations.

   (c) Plot the scores for the normal set as well as for the anomalous observations for a visual examination.

   (d) Select a cut-off so as to separate out the anomalies from the normal observations. Examine this cut-off on the test set.

4. Machine Learning: Classification, Network Attacks classification: This assignment is also on working around with the classifiers. You would now use the dataset *Darknet.csv* dataset to be downloaded from https://www.unb.ca/cic/datasets/darknet2020.html. Your implementation should support two different classifications viz. i.e. the classifiers are trained to perform two tasks. The first one includes two classes: *Benign* and *Darknet*, and the second four classes: *Tor*, *Non Tor*, *VPN* and *Non VPN*. You may use a classifier that best suits such applications and justify your selection with technically sound justifications.

5. Machine Learning: Classification, Network Attacks classification: This assignment is also on working around with the classifiers. You would now use the dataset CIRA-CIC-DoHBrw-2020 dataset to be downloaded from https://www.unb.ca/cic/datasets/dohbrw-2020.html. The dataset concerns with the DNS over HTTPS (DoH) traffic. This dataset is used to analyze, test, and evaluate DoH traffic in covert channels and tunnels. Note that Domain Name System (DNS) is one of the early and vulnerable network protocols which has several security loopholes that have been exploited repeatedly over the years. DNS abuse has always been an area of great concern for cybersecurity researchers. However, providing security and privacy to DNS requests and responses is still a challenging task as attackers use sophisticated attack methodologies to steal data on the fly. To overcome some of the DNS vulnerabilities related to privacy and data manipulation, IETF introduced DNS over HTTPS (DoH) in RFC8484, a protocol that enhances privacy and combats eavesdropping and man-in-the-middle attacks by encrypting DNS queries and sending them in a covert channel/tunnel so that data is not hampered on the way. There is however always a need to evaluate the techniques that capture DoH traffic in a network topology using an appropriate dataset, for training. This exercise is aimed at the same. The network diagram to capture the traffic for the dataset is presented in Figure 1. Firstly, normal web browsing activity that includes non-DoH HTTPS and benign DoH is simulated using web browsers. Secondly, malicious DoH is generated by a combination of tools used to create DoH tunnels. Traffic generated by all these tools is captured for preprocessing and training the classifiers. In CIRA-CIC-DoHBrw-2020 dataset, a two-layered approach is used to capture benign and malicious DoH traffic along with non-DoH traffic. To generate the representative dataset, HTTPS (benign DoH and non-DoH) and DoH traffic is generated by accessing top 10k Alexa websites, and using browsers and DNS tunneling tools

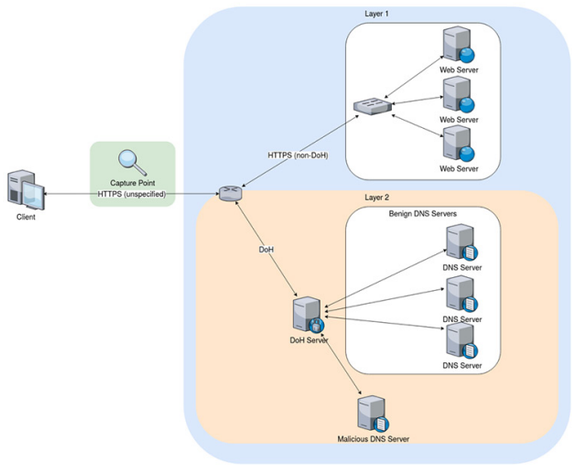

Figure 1: Network topology used to capture
Ref: https://www.unb.ca/cic/datasets/dohbrw-2020.html

that support DoH protocol respectively. At the first layer, the captured traffic is classified as DoH and non-DoH by using statistical features classifier. At the second layer, DoH traffic is characterized as benign DoH and malicious DoH by using time-series classifier.

Your implementation has to use time series classification for this problem. Time series classification uses supervised machine learning to analyze multiple labeled classes of time series data and then predict or classify the class that a new data set belongs to. The main objective of your implementation should be to capture benign as well as malicious DoH traffic in three classes viz. *Benign-DoH, Malicious-DoH* and *non-DoH* traffic. The semantics of these traffic, briefly, is as follows:

(a) Non-DoH: Traffic generated by accessing a website that uses HTTPS protocol is captured and labeled as non-DoH traffic. In order to capture ample traffic to balance the dataset, thousands of websites from Alexa domain are browsed.

(b) Benign-DoH: Benign DoH is non-malicious DoH traffic generated using the same technique as mentioned in non-DoH by using Mozilla Firefox and Google Chrome web browsers.

(c) Malicious-DoH: DNS tunneling tools such as dns2tcp, DNSCat2, and Iodine are used to generate malicious DoH traffic. These tools can send TCP traffic encapsulated in DNS queries. In other words, these tools create tunnels of encrypted data. Therefore, DNS queries are sent using TLS-encrypted HTTPS requests to special DoH servers.

You may use a classifier that best suits such applications and justify your selection with technically sound justifications.

6. For the problem at Sr no 1, now you will use Principal Component Analysis (PCA) to take the features and return a smaller number of new features, formed from original ones, with maximal explanatory power. In addition, since the new features are linear combinations of the old features, this allows one to anonymize our data, useful in many applications. Hence,

   (a) Instantiate a PCA object instance using appropriate method available in scikit-learn and pandas packages and use it to reduce the dimensionality of data.

   (b) Assess the effectiveness of your implementation of dimensionality reduction, using appropriate analysis and shown your results/conclusion.

7. Isolation Forest for anomaly detection:Anomaly detection is the identification of events in a dataset that do not conform to the expected pattern. In applications, these events may be of critical importance. For instance, they may be occurrences of a network intrusion or of fraud. In this exercise, you have to use *Isolation Forest* to detect such anomalies. Isolation Forest relies on the observation that it is easy to isolate an outlier, while more difficult to describe a normal data point. For this problem, you will use `matplotlib`, `pandas`, `scipy` packages in `pip`. Then, follow the steps as instructed further:

   (a) First, Import the required libraries and set a random seed:

```
import numpy as np
import pandas as pd
```

```
random_seed = np.random.RandomState(12)
```

(b) Next, generate a set of normal observations, to be used as training data:

```
X_train = 0.5 * random_seed.randn(500, 2)
X_train = np.r_[X_train + 3, X_train]
X_train = pd.DataFrame(X_train, columns=["x", "y"])
```

(c) Then, generate a set of outlier observations. These are generated from a different distribution than the normal observations with the following set of commands:

```
X_outliers = random_seed.uniform(low=-5, high=5, size=(50, 2))
X_outliers = pd.DataFrame(X_outliers, columns=["x", "y"])
```

(d) Then inspect the data that is generated, using `matplotlib` i.e.

```
%matplotlib inline
import matplotlib.pyplot as plt
p1 = plt.scatter(X_train.x, X_train.y, c="white",
s=50, edgecolor="black")
p2 = plt.scatter(X_test.x, X_test.y, c="green",
s=50, edgecolor="black")
p3 = plt.scatter(X_outliers.x, X_outliers.y, c="blue",
s=50, edgecolor="black")
plt.xlim((-6, 6))
plt.ylim((-6, 6))
plt.legend(
[p1, p2, p3], ["training set", "normal testing set",
"anomalous testing set"], loc="lower right",
)
plt.show()
```

Take the snapshot of the output and analyze/give inferences on your output.

(e) Next, train an Isolation Forest model on the training data using the commands viz.

```
from sklearn.ensemble import IsolationForest
clf = IsolationForest()
clf.fit(X_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_outliers = clf.predict(X_outliers)
```

(f) Next, check how the algorithm performs. Append the labels to X_outliers:

```
X_outliers = X_outliers.assign(pred=y_pred_outliers)
X_outliers.head()
```

(g) Obtain the output, in terms of values of x, y and *prediction*

(h) Lastly, write appropriate commands to plot the Isolation Forest predictions on the outliers. Check how many it caught. Plot the graph to show how many it caught.

(i) Then, check how it performs on the normal testing data. Append the predicted label to `X_test`

```
X_test = X_test.assign(pred=y_pred_test)
X_test.head()
```

(j) Obtain the output in terms of values of x, y and *prediction* and analyze the output.

(k) Next, plot the results to see whether the classifier labeled the normal testing data correctly, using the following set of commands:

```
p1 = plt.scatter(X_train.x, X_train.y, c="white", s=50,
edgecolor="black")
p2 = plt.scatter(
X_test.loc[X_test.pred == 1, ["x"]],
X_test.loc[X_test.pred == 1, ["y"]],
c="blue",
s=50,
edgecolor="black",
)
```

```
p3 = plt.scatter(
X_test.loc[X_test.pred == -1, ["x"]],
X_test.loc[X_test.pred == -1, ["y"]],
c="red",
s=50,
edgecolor="black",
)
plt.xlim((-6, 6))
plt.ylim((-6, 6))
plt.legend(
[p1, p2, p3],
[
"training observations",
"correctly labeled test observations",
"incorrectly labeled test observations",
],
loc="lower right",
)
plt.show()
```

(l) Obtain the snapshot of your output graph. Analyze the graph to comment whether your Isolation Forest model performed well at capturing the anomalous points or not, whether there were a few false negatives (instances where normal points were classified as outliers) or not. Check whether you can improve the performance by tuning your model's parameters, depending on how many false negatives this application can tolerate - use scientific justification to prove your inferences. Give clear inferences based on your experimentations.

8. Markov chains for spam detection: Markov chains are simple stochastic models in which a system can exist in a number of states. To know the probability distribution of where the system will be next, it suffices to know where it currently is. This is in contrast with a system in which the probability distribution of the subsequent state may depend on the past history of the system. This simplifying assumption allows Markov chains to be easily applied in many domains, surprisingly fruitfully.
   In this exercise, you have to use Markov chains to generate fake reviews, which is useful for pen-testing a review system's spam detector. For this problem, you will use `markovify` and `pandas` packages in `pip`. And you will use the `airport_reviews.csv` - a CSV dataset, uploaded on the classroom. Then,

   (a) First, start by importing the `markovify` library and a text file whose style is to be imitated - e.g. you could use - a collection of airport reviews with some text like *"The airport is certainly tiny! ..."*

   (b) Next, join the individual reviews into one large text string and build a Markov chain model using the airport review text. Note that, when you do so, behind the scenes, the library would have computed the transition word probabilities from the text.

   (c) Next, generate five sentences using the Markov chain model. Since, you are using airport reviews, you will have the following sample as the output after executing your previous code:
   `On the positive side it's a clean airport transfer from A to C gates and outgoing gates is truly enormous - but why when we arrived at about 7.30 am for our connecting flight to Venice on TAROM. The only really bother: you may have to wait in a polite manner. Why not have bus after a short wait to check-in there were a lots of shops and less seating. Very inefficient and hostile airport. This is one of the time easy to access at low price from city center by train. The distance between the incoming gates and ending with dirty and always blocked by never ending roadworks.`

   (d) Next, generate 3 sentences with a length of no more than 140 characters. You should be able to see the output as follows:
   `However airport staff member told us that we were put on a connecting code share flight. Confusing in the check-in agent was friendly. I am definitely not keen on coming to the lack of staff. Lack of staff. Lack of staff at boarding pass at check-in.`

9. Clustering for malware classification: Clustering is a collection of unsupervised machine learning algorithms in which parts of the data are grouped based on similarity. For example,

clusters might consist of data that is close together in `n-dimensional Euclidean space`. Clustering is *useful in cybersecurity* for distinguishing between *normal* and *anomalous network activity*, and for helping *to classify malware into families*.
In this exercise you have to experiment and observe/analyze how `scikit-learn`'s `K-means clustering algorithm` performs on a toy PE type (Portable Executable format is the standard file format for executables, object code and Dynamic Link Libraries (DLLs) used in 32- and 64-bit versions of Windows operating systems) malware classification. Your eventual results must show how the clustering algorithm has performed, and analyze whether the results are perfect/not perfect, and whether the clustering algorithm captured much of the structure in the dataset or not. For this problem, you will use `scikit-learn` and implementation of `K-means` clustering algorithm therein and the dataset uploaded on the classroom viz. `file_pe_headers.csv`. Then,

(a) First, start with the following statements importing and plotting the dataset:

```
import pandas as pd
import plotly.express as px
df = pd.read_csv("file_pe_headers.csv", sep=",")
fig = px.scatter_3d(
df,
x="SuspiciousImportFunctions",
y="SectionsLength",
z="SuspiciousNameSection",
color="Malware",
)
fig.show()
```

This dataset consists of two classes of PE files: *malware* and *benign*. Use *plotly* to create a nice-looking interactive 3D graph (step 1).

(b) Study the output. Extract the features and target labels. Next, import scikit-learn's clustering module and fit a K-means model with two clusters to the data.

(c) Predict the cluster using our trained algorithm.

(d) To see how the algorithm did, plot the algorithm's clusters. Analyze and show how the clustering algorithm captured much of the structure in the dataset.

(e) Finally, enlist what all implementations for clustering are provided in the `scikit-learn` toolkit ? Which one is preferred ? Prepare a table with the following columns for all the clustering algorithm available in scikit learn viz. *Method name, Parameters, Scalability, Usecase, Geometry (metric used).*

(f) Repeat the clustering now with the *Agglomerative clustering and DBSCAN* clustering algorithms. Analyze the difference with respect to the earlier clustering obtained and comment the reasons thereof.

10. Gradient boosting for malware detection: Gradient boosting is widely considered the most reliable and accurate algorithm for generic machine learning problems. `XGBoost` is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. `XGBoost` can be used to create malware detectors too.
In this exercise, you have to use implement an `XG-Boost` classifier, by instantiating it initially with default parameters and fit it to your training set. Finally, your `XGBoost` classifier implementation is to be used to predict on the testing set. For this problem, you will use `scikit-learn`, `pandas`, and `xgboost` packages in `pip`. And you will use the `file_pe_header.csv` - a CSV dataset, uploaded on the classroom. Then,

(a) First, start by importing the `train_test_split` module and the `pandas` library

(b) Next, read in the data from the dataset as mentioned above, into the appropriate object instances. That is, read features into X and labels into y.

(c) Next, randomly split the dataset and its labels into a training set consisting 70% of the size of the original dataset and a testing set 30% of the size.

(d) Now, create one instance of an `XGBoost` model with default parameters and fit it to your training set

(e) Finally, assess its performance on the testing set.

over to next & last assignment
*****