SIT326 - 4.1p, 4.2c, 4.3d, 4.4hd

Contents:

4.1P - Feature Selection and Preprocessing

In the Week 3's tasks, I took five categories out of the captured certificates; issuer, notBefore, notAfter, subject, and subjectAltName. With these sections for both the good and bad certificates, I will split them into eight features for analysis:

Feature Name	Type	Description
IssuerOrgName	string	Organization name of the certificate issuer
IssuerComName	string	Common name of the certificate issuer
IssuerCountry	string	Country code of the certificate issuer
SubjectCountry	string	Country code of the certificate subject
SubjectDetails	integer	Number of certificate subject's non-name features present (ie. locality, busniess type, etc)
SubjectAltNames	integer	Number of alternate names for the subject within the certificate (Excludes wildcard and www. Variations of the same domain)
SubjectValidity	integer	Length of certificate validity in number of days
Malicious	boolean	A value for the algorithm to predict whether the certificate is likely from a malicious source or not

I have preprocessed the raw certificate data using two programs I have made that parses through each column using the Python Pandas module. It largely uses regex to carve out the data needed and, if required, processes it further to achieve an acceptable value.

```
def main():
    sorted_data = pd.DataFrame()
    data = pd.read_csv("GoodCertificates.csv")
    sorted_data['IssuerOrgName'] = IssuerOrgName(data.issuer)
    sorted_data['IssuerComName'] = IssuerComName(data.issuer)
    sorted_data['IssuerCountry'] = IssuerCountry(data.issuer)
    sorted_data['SubjectCountry'] = SubjectCountry(data.subject)
    sorted_data['SubjectDetails'] = SubjectDetails(data.subject)
    sorted_data['SubjectAltNames'] = SubjectAltNames(data.subjectAltName, data.subject)
    sorted_data['SubjectValidity'] = SubjectValidity(data.notBefore, data.notAfter)
    sorted_data['Malicious'] = 0
    print(sorted_data)
    sorted_data.to_csv("SortedGoodCerts.csv")

if __name__ == "__main__":
    main()
```

All functions simply carve out the required string with regex and append it to a list unless stated below:

- SubjectDetails(): Uses re.findall() to create a list of all entries within the subject that are not commonName or countryName and then len() is the quantifier used for the output

```
def SubjectDetails(subject_dataframe):
    dfoutput = []
    for row in subject_dataframe:
        try:
            output = re.findall("(?:\(\')((?!countryName|commonName)\w+)(?:',)", row)
            dfoutput.append(len(output))
        except:
            dfoutput.append("0")
    return dfoutput
```

- SubjectAltNames(): Uses re.search() to find the subject's domain name and then re.findall() to find all domains within the subjectAltName section of the certificate. For every entry of the domain name or entries that are simple variations of the same domain such as *.domain or www.domain, this number is reduced by one and the final integer is the output

```
def SubjectAltNames(subjectAltNames_dataframe, subject_dataframe):
    dfoutput = []
    i = 0
    count = 0
    outputnum = 0
    for row in subjectAltNames_dataframe:
        try:
            domain = re.search("(?:\(\('commonName', ')(?:\*\.|www\.|)([a-zA-Z.0-9]+)(?:'\),\))", subject_dataframe.iloc[i]).group(1)
            output = re.findall("(?:\('DNS', ')(?:\*\.|www\.|)([a-zA-Z.\-0-9]+)(?:'\))", row)
            count = len(output)
            # print(len(output))
            for url in output:
                if url == domain:
                    count -= 1
        except:
            count = 0
        dfoutput.append(count)
        i += 1
    return dfoutput
```

- SubjectValidity(): This function will use regex to find the year, month, and day from the notBefore and notAfter sections of a certificate and store them in a variable. If the month is in a worded format, it will find the number value from a dictionary and store that instead. After this, the Date module from DateTime is used to calculate the difference in number of days between these two dates and that integer is the output.

```
def SubjectValidity(notBefore_dataframe, notAfter_dataframe):
    months = {
        "Jan":1,
        "Feb":2,
        "Mar":3,
        "Apr":4,
        "May":5,
        "Jun":6,
        "Jul":7,
        "Aug":8,
        "Sep":9,
        "Oct":10,
        "Nov":11,
        "Dec":12,
    }
    dfoutput = []
    i = 0
    for row in notBefore_dataframe:
        before_year = int(re.search("([0-9]{4})", row).group(1))
        before_month = re.search("([A-Z][a-z]{2})", row).group(1)
        before_month = months[before_month]
        before_day = int(re.search("(?: )([0-9]{1}|[0-9]{2})(?: )", row).group(1))
        d0 = date(before_year, before_month, before_day)

        after_year = int(re.search("([0-9]{4})", notAfter_dataframe[i]).group(1))
        after_month = re.search("([A-Z][a-z]{2})", notAfter_dataframe[i]).group(1)
        after_month = months[after_month]
        after_day = int(re.search("(?: )([0-9]{1}|[0-9]{2})(?: )", notAfter_dataframe[i]).group(1))
        d1 = date(after_year, after_month, after_day)

        delta = d1 - d0
        dfoutput.append(delta.days)
        i+=1
    return dfoutput
```

Certificates before Preprocessing:

issuer
((('countryName', 'US'),), (('organizationName', 'DigiCert Inc'),), (('organizationalUnitName', 'www.digicert.com'),), (('commonName', 'DigiCer
((('countryName', 'US'),), (('organizationName', 'Google Trust Services LLC'),), (('commonName', 'GTS CA 1C3'),))
((('countryName', 'US'),), (('organizationName', 'Google Trust Services LLC'),), (('commonName', 'GTS CA 1C3'),))
((('countryName', 'US'),), (('organizationName', 'Google Trust Services LLC'),), (('commonName', 'GTS CA 1C3'),))
((('countryName', 'US'),), (('organizationName', 'DigiCert Inc'),), (('commonName', 'DigiCert TLS RSA SHA256 2020 CA1'),))
((('countryName', 'US'),), (('organizationName', 'DigiCert Inc'),), (('organizationalUnitName', 'www.digicert.com'),), (('commonName', 'DigiCer
((('countryName', 'US'),), (('organizationName', 'Google Trust Services LLC'),), (('commonName', 'GTS CA 1C3'),))
((('countryName', 'GB'),), (('stateOrProvinceName', 'Greater Manchester'),), (('localityName', 'Salford'),), (('organizationName', 'Sectigo Limi
((('countryName', 'US'),), (('organizationName', 'DigiCert Inc'),), (('commonName', 'DigiCert SHA2 Secure Server CA'),))
((('countryName', 'US'),), (('organizationName', 'Google Trust Services LLC'),), (('commonName', 'GTS CA 1C3'),))

notAfter	notBefore	subject
Apr 4 23:59:59 2023 GMT	Jan 10 00:00:00 2023 GMT	((('countryName', 'US'),), (('stateOrProvinceName', 'California'),), (('localityNam
May 29 08:18:45 2023 GMT	Mar 6 08:18:46 2023 GMT	((('commonName', 'upload.video.google.com'),),)
May 29 08:16:20 2023 GMT	Mar 6 08:16:21 2023 GMT	((('commonName', '*.google.com'),),)
May 29 08:16:20 2023 GMT	Mar 6 08:16:21 2023 GMT	((('commonName', '*.google.com'),),)
Jan 12 23:59:59 2024 GMT	Jan 12 00:00:00 2023 GMT	((('countryName', 'US'),), (('stateOrProvinceName', 'California'),), (('localityNam
Apr 4 23:59:59 2023 GMT	Jan 11 00:00:00 2023 GMT	((('countryName', 'US'),), (('stateOrProvinceName', 'California'),), (('localityNam
May 29 08:16:19 2023 GMT	Mar 6 08:16:20 2023 GMT	((('commonName', '*.google-analytics.com'),),)
Jan 6 23:59:59 2024 GMT	Dec 6 00:00:00 2022 GMT	((('commonName', '*.w.org'),),)
Sep 7 23:59:59 2023 GMT	Mar 7 00:00:00 2023 GMT	((('countryName', 'US'),), (('stateOrProvinceName', 'California'),), (('localityNam
May 29 08:18:44 2023 GMT	Mar 6 08:18:45 2023 GMT	((('commonName', '*.gstatic.com'),),)

Certificates after Preprocessing:

	IssuerOrgName	IssuerComName	IssuerCountry	SubjectCountry	SubjectDetails	SubjectAltNames	SubjectValidity	Malicious
0	DigiCert Inc	DigiCert SHA2 High Assurance Server CA	US	US	3	9	84	0
1	Google Trust Services LLC	GTS CA 1C3	US	None	0	16	84	0
2	Google Trust Services LLC	GTS CA 1C3	US	None	0	132	84	0
3	Google Trust Services LLC	GTS CA 1C3	US	None	0	132	84	0
4	DigiCert Inc	DigiCert TLS RSA SHA256 2020 CA1	US	US	3	0	365	0
5	DigiCert Inc	DigiCert SHA2 High Assurance Server CA	US	US	3	6	83	0
6	Google Trust Services LLC	GTS CA 1C3	US	None	0	0	84	0
7	Sectigo Limited	Sectigo ECC Domain Validation Secure Server CA	GB	None	0	0	396	0
8	DigiCert Inc	DigiCert SHA2 Secure Server CA	US	US	3	25	184	0
9	Google Trust Services LLC	GTS CA 1C3	US	None	0	3	84	0
10	Let's Encrypt	R3	US	None	0	0	90	0
11	Google Trust Services LLC	GTS CA 1C3	US	None	0	16	84	0
12	Google Trust Services LLC	GTS CA 1C3	US	None	0	132	84	0
13	Google Trust Services LLC	GTS CA 1C3	US	None	0	132	84	0
14	Google Trust Services LLC	GTS CA 1C3	US	None	0	132	84	0
15	Google Trust Services LLC	GTS CA 1C3	US	None	0	132	84	0

Finally, I dummy encoded the categorical data, which leads to a set of data at 9624 rows by 241 columns plus one extra column for the Y variables.

```
└─$ python3 cert_analysis.py
      SubjectDetails  SubjectAltNames  SubjectValidity  ...  SubjectCountry_SE  SubjectCountry_TW  SubjectCountry_US
0                  3                9               84  ...                  0                  0                  1
1                  0               16               84  ...                  0                  0                  0
2                  0              132               84  ...                  0                  0                  0
3                  0              132               84  ...                  0                  0                  0
4                  3                0              365  ...                  0                  0                  1
...              ...              ...              ...  ...                ...                ...                ...
9619               0                0               90  ...                  0                  0                  0
9620               0                0               90  ...                  0                  0                  0
9621               0                0               90  ...                  0                  0                  0
9622               0                0               90  ...                  0                  0                  0
9623               0                0               90  ...                  0                  0                  0

[9624 rows x 241 columns]
```

```
  0        0
1        0
2        0
3        0
4        0
        ..
9619     1
9620     1
9621     1
9622     1
9623     1
Name: Malicious, Length: 9624, dtype: int64
```

4.2C - Decision Tree

My decision tree algorithm is fairly straightforward. It begins by opening the preprocessed dataset as a dataframe, then dummy encodes all the categorical data within it. The categorical columns are dropped from the original data and the new encoded columns are added.
Next, the “Malicious” Boolean column is taken from data and set as the y-axis while everything else is the x-axis.
The data is split into a mix of 80% Training data and 20% Testing data. This is then split again into 60% Training data, 20% Testing data, and 20% Validation data.
A decision tree is then fit and predicts the malicious state of SSL certificates given all the input data. A figure of the decision tree process is also created to show the most important features.
Finally, a cross-validation score is calculated using the validation data against the Training data to calculate any feature over-fitting present.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn import preprocessing
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

data = pd.read_csv("FullCertListSorted.csv")
# Dummy Encode the string data
data_encoded = pd.get_dummies(data[["IssuerOrgName", "IssuerComName", "IssuerCountry", "SubjectCountry"]])
# Drop old categorical data
data = data.drop(columns=["IssuerOrgName", "IssuerComName", "IssuerCountry", "SubjectCountry"])
# Combine old data with new encoded data
data = pd.concat([data, data_encoded], axis=1)

data_x = data.drop("Malicious", axis=1)
data_y = data["Malicious"]

depth = 3

# Perform the Training and Test the data
X_train, X_test, y_train, y_test = train_test_split(data_x, data_y, test_size=0.2)
# Use Cross-Validation to test for overfitting or anomalies
X_train, X_val, y_train, y_val =  train_test_split(X_train, y_train, test_size=0.25)
# for depth in range(1,4):
dtree = DecisionTreeClassifier(max_depth=depth)
dtree.fit(X_train, y_train)
predictions = dtree.predict(X_test)
acc = accuracy_score(y_test, predictions)
print("Test accuracy score for ", depth," depth is: ", round(acc*100, 2), "%", sep="")
fig = plt.figure(figsize=(50,50))
plot_tree(dtree, feature_names = data_x.columns, class_names = ["Malicious", "Not malicious"], impurity=False, proportion=False, filled=True)
fig.savefig("DTreeFig.png")

dtree = DecisionTreeClassifier(max_depth=depth)
scores = (cross_val_score(dtree, X_val, y_val, cv=5))
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean()*100, scores.std()*100))
```

Results:

Running this algorithm and changing the depth hyperparameter from 1 to 50 many times each resulted in a trend of an optimal depth of 10 for this dataset. Any further than this and the accuracy did not significantly improve if it did at all.

At the optimal depth of 10, the algorithm achieved on average a 96.88% testing accuracy with similar validation accuracies and a standard deviation of the latter being between 0.3% and 0.6%. Overall, this is a fairly good result and a low chance of over-fitting the algorithm with too much data.

```
┌──(kali㉿kali)-[~/Documents/SIT326/4.1p]
└─$ python3 cert_analysis.py
Test accuracy score for 3 depth is: 96.88%
96.57 validation accuracy with a standard deviation of 0.38
```

4.3D – Random Forest

The data preparation and train-test-validation split is all the same as the previous Decision Tree model analysis. The one simple change for this model is to use a Random Forest classifier instead and everything else is the same.

```
trees = 100
rf = RandomForestClassifier(n_estimators=trees)
rf.fit(X_train, y_train)
predictions = rf.predict(X_test)
acc = accuracy_score(y_test, predictions)
print("Test accuracy score for ", trees," depth is: ", round(acc*100, 2), "%", sep="")

rf = RandomForestClassifier(n_estimators=trees, criterion="log_loss")
rf.fit(X_train, y_train)
# Cross Validation block
scores = (cross_val_score(rf, X_val, y_val, cv=5))
print("%0.2f validation accuracy with a standard deviation of %0.2f" % (scores.mean()*100, scores.std()*100))
```

I had changed some more hyperparameters in this test to see the effects on the accuracy and all-in-all they had very marginal, insignificant effects.

Random Forest at default settings (for sklearn, n_estimators = 100, criterion = “gini”, etc.) obtained very similar results to the Decision Tree classifier but were, on average 0.3% more accurate (additive) while having about 0.03% less standard deviation in the cross validation.

```
┌──(kali㉿kali)-[~/Documents/SIT326/4.1p]
└─$ python3 cert_analysis.py
Test accuracy score for 100 depth is: 97.53%
96.21 validation accuracy with a standard deviation of 0.48
```

Several hyperparameters were edited to see the effect on the accuracy such as n_estimators, criterion, class_weight, and bootstrap. Below are the general trends from changing each hyperparameter:

- N_estimators: The accuracy seemed to average better when the trees were set to around 30 instead of the default 100, but it was so statistically insignificant that it can hardly be an observation. The only notable value was that anything below 10 consistently produced worse results than anything higher.
- Criterion: The default setting, “gini” outperformed both “entropy” and “log_loss” in consistency and overall score.
- Class_weight: Changing this to any option other than None dropped the accuracy by 10% and doubled the standard deviation.
- Bootstrap: Changing this value to False showed a greater variance in accuracy but also increased the max accuracies achieved during testing, while simultaneously provided further decreased low accuracy values.

Conclusion:

This classifier outperformed Decision Tree marginally and performed much more consistently too. For hyperparameters, the default that sklearn has set are more than sufficient for this data set and if anything could change, it is a lower n_estimators value for increased efficiency in program performance vs. accuracy.

4.4HD – Gradient Boosting

The data preparation and train-test-validation split is all the same as the previous Decision Tree model analysis. For this classifier, I used sklearn's GradientBoostingClassifier.

```
acc = 0
for i in range(10):
    gb = GradientBoostingClassifier()
    gb.fit(X_train, y_train)
    predictions = gb.predict(X_test)
    acc = acc + accuracy_score(y_test, predictions)
print("Test accuracy score: ", round(acc*10, 2), "%", sep="")

scores = (cross_val_score(gb, X_val, y_val, cv=5))
print("%0.2f validation accuracy with a standard deviation of %0.2f" % (scores.mean()*100, scores.std()*100))
```

The tests began with default hyperparameters, similar to the other classifiers. The gradient boosting classifier immediately did not perform as well as either of the other classifiers by default and it achieved an extremely consistent but comparatively poor testing score, although boasting a very low standard deviation in cross validation.

```
┌──(kali㉿kali)-[~/Documents/SIT326/4.1p]
└─$ python3 cert_analysis.py
Test accuracy score: 97.19%
96.62 validation accuracy with a standard deviation of 0.23
```

Below are various hyperparameters and the trends I noticed during analysis on the accuracy.

- Loss: "exponential" achieved a slightly higher accuracy score on average than the default log_loss.
- N_estimators: Similarly to Random Forest classifier, the value trended towards 30 being an effective value for this parameter for both performance and accuracy.
- Criterion: "squared_error" had much higher variance in the accuracies achieved and was more resource intensive too, however it did achieve a higher average accuracy than the default of "friedman_mse".
- Max_features: both "sqrt" and "log2" outperformed the default of None, with "log2" having a clearly higher accuracy on average.

Conclusion:

In my testing for Gradient Boosting, out of all hyperparameter combinations tested, changing max_features to "log2" and having everything else as sklearn default provided the highest accuracy values while also keeping acceptable validation accuracy and deviation. The accuracies gained with this has performed better than both Decision Tree classifier and Random Forest Classifier, ending up at 97.94% accuracy in detecting SSL certificates attached to malicious websites.

```
┌──(kali㉿kali)-[~/Documents/SIT326/4.1p]
└─$ python3 cert_analysis.py
Test accuracy score: 97.94%
96.42 validation accuracy with a standard deviation of 0.76
```