4. MODEL TRAINING

January 23, 2026

Importing Libraries

[1]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, StratifiedKFold

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

from hyperopt import hp, tpe, STATUS_OK, Trials, fmin

from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, precision_recall_curve, ConfusionMatrixDisplay

import matplotlib.pyplot as plt
import shap

import pickle

import warnings
warnings.filterwarnings('ignore')
```

Reading the data

[2]:
```
df = pd.read_csv('data/processed/processed_data.csv')

df.head()
```

[2]:
```
   has_https  url_len  domain_len  path_len  query_len  url_depth  \
0          1       27          19         0          0          0
1          0       34           0        13          0          2
2          1       51          20        10         11          1
3          1      175          15       103         43          5
4          1       79          21        47          0          3

   subdomain_count  tld_len  url_has_ipv4  url_has_port  …  spl_char_count  \
0                1        3             0             0  …               5
```

```
1                 0         0             1             0  …            9
2                 2         3             0             0  …           10
3                 1         3             0             0  …           21
4                 2         3             0             0  …           12

   url_entropy  domain_entropy  sld_entropy  path_entropy  domain_token_count  \
0     3.856196        3.431624     2.845351      0.000000                   3
1     3.962032        0.000000     0.000000      3.240224                   0
2     3.965393        3.008695     1.584963      2.913977                   4
3     5.569700        2.973557     1.918296      5.540696                   3
4     4.274946        3.748995     3.000000      3.845213                   4

   path_token_count  total_tokens  avg_token_length  class
0                 0             3          5.666667      0
1                 1             1          3.666667      1
2                 2             6          4.500000      0
3                 4             7          8.882353      1
4                 4             8          6.200000      0

[5 rows x 30 columns]
```

Splitting the data

[3]:
```
X = df.drop(columns=['class']).values
y = df['class'].values
```

[4]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.
↪75,random_state=6)
```

Training the Models

[5]:
```
# Function to evaluate the models
def evaluate_model(y_true,y_pred):
    accuracy = accuracy_score(y_true,y_pred)
    precision = precision_score(y_true,y_pred)
    recall = recall_score(y_true,y_pred)
    f1 = f1_score(y_true,y_pred)

    print(f'Accuracy: {accuracy}')
    print(f'Precision: {precision}')
    print(f'Recall: {recall}')
    print(f'F1-Score: {f1}')
```

Random Forest Classifier

[6]:
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train,y_train)

rf_pred_train = rf_model.predict(X_train)
```

```python
rf_pred_test = rf_model.predict(X_test)

print('Metrics of Random Forest on Training data:')
evaluate_model(y_train,rf_pred_train)

print()

print('Metrics of Random Forest on Testing data:')
evaluate_model(y_test,rf_pred_test)
```

```
Metrics of Random Forest on Training data:
Accuracy: 0.9980554379141887
Precision: 0.999406814221629
Recall: 0.9965742030092792
F1-Score: 0.9979884986548856

Metrics of Random Forest on Testing data:
Accuracy: 0.9782402519549773
Precision: 0.9872836719337847
Recall: 0.9676253687315635
F1-Score: 0.9773556797020484
```

XGBoost Classifier

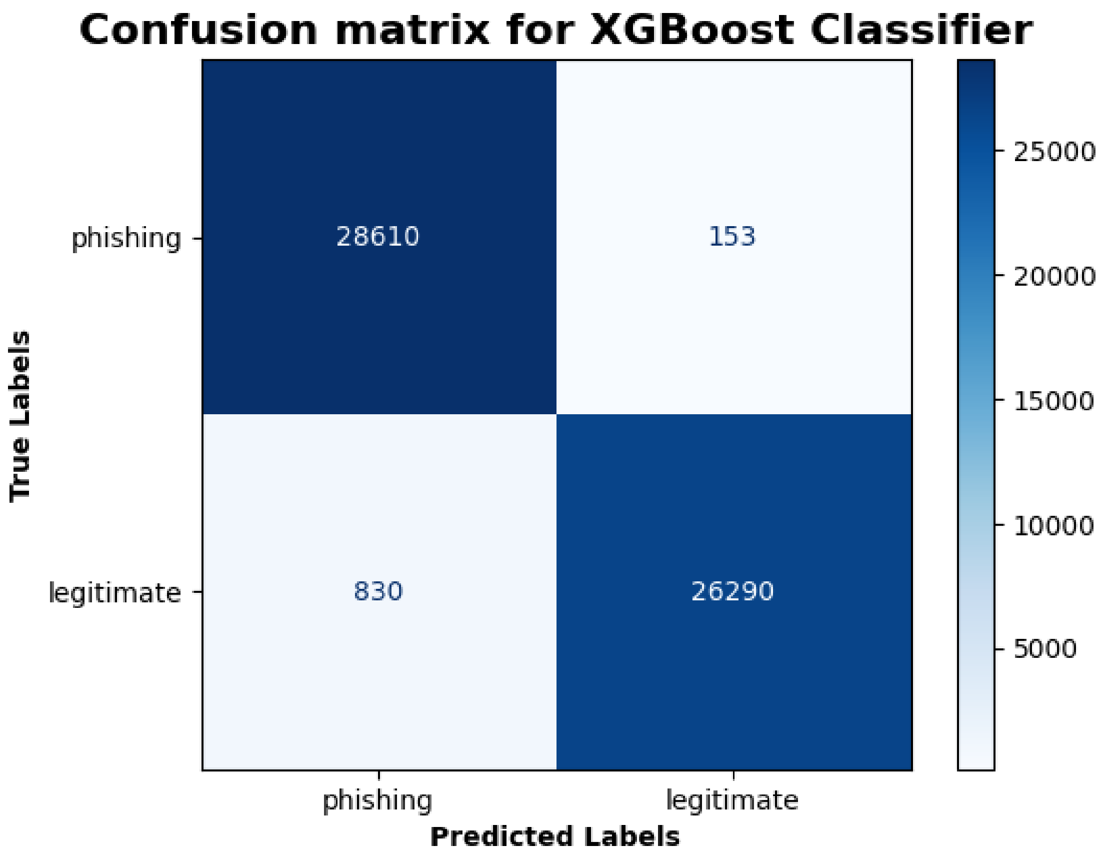
[7]:

```python
xg_model = XGBClassifier()
xg_model.fit(X_train,y_train)

xg_pred_train = xg_model.predict(X_train)
xg_pred_test = xg_model.predict(X_test)

print('Metrics of XGBoost Classifier on Training data:')
evaluate_model(y_train,xg_pred_train)

print()

print('Metrics of XGBoost Classifier on Testing data:')
evaluate_model(y_test,xg_pred_test)
```

```
Metrics of XGBoost Classifier on Training data:
Accuracy: 0.9860480652800229
Precision: 0.9966599445424754
Recall: 0.9744420756879321
F1-Score: 0.9854257924219105

Metrics of XGBoost Classifier on Testing data:
Accuracy: 0.9824096773616305
Precision: 0.9942139696706123
Recall: 0.9693952802359882
F1-Score: 0.9816477792506021
```

By comparing the metrics of Random Forest Classifier & XGBoost Classifier on both training set and testing set, XGBoost Classifier is best. Both the model's metrics are good on training set but on testing set, Random Forest model's metrics are slightly lower than XGBoost model's metrics telling that Random Forest model has less Generalization. Therefore, we prefer XGBoost Classifier.

[8]:
```
disp = ConfusionMatrixDisplay.
  ↪from_estimator(xg_model,X_test,y_test,display_labels=['phishing','legitimate'],cmap=plt.
  ↪cm.Blues)
plt.xlabel('Predicted Labels',weight='bold')
plt.ylabel('True Labels',weight='bold')
plt.title('Confusion matrix for XGBoost Classifier',weight='bold',fontsize=16)
plt.show()
```

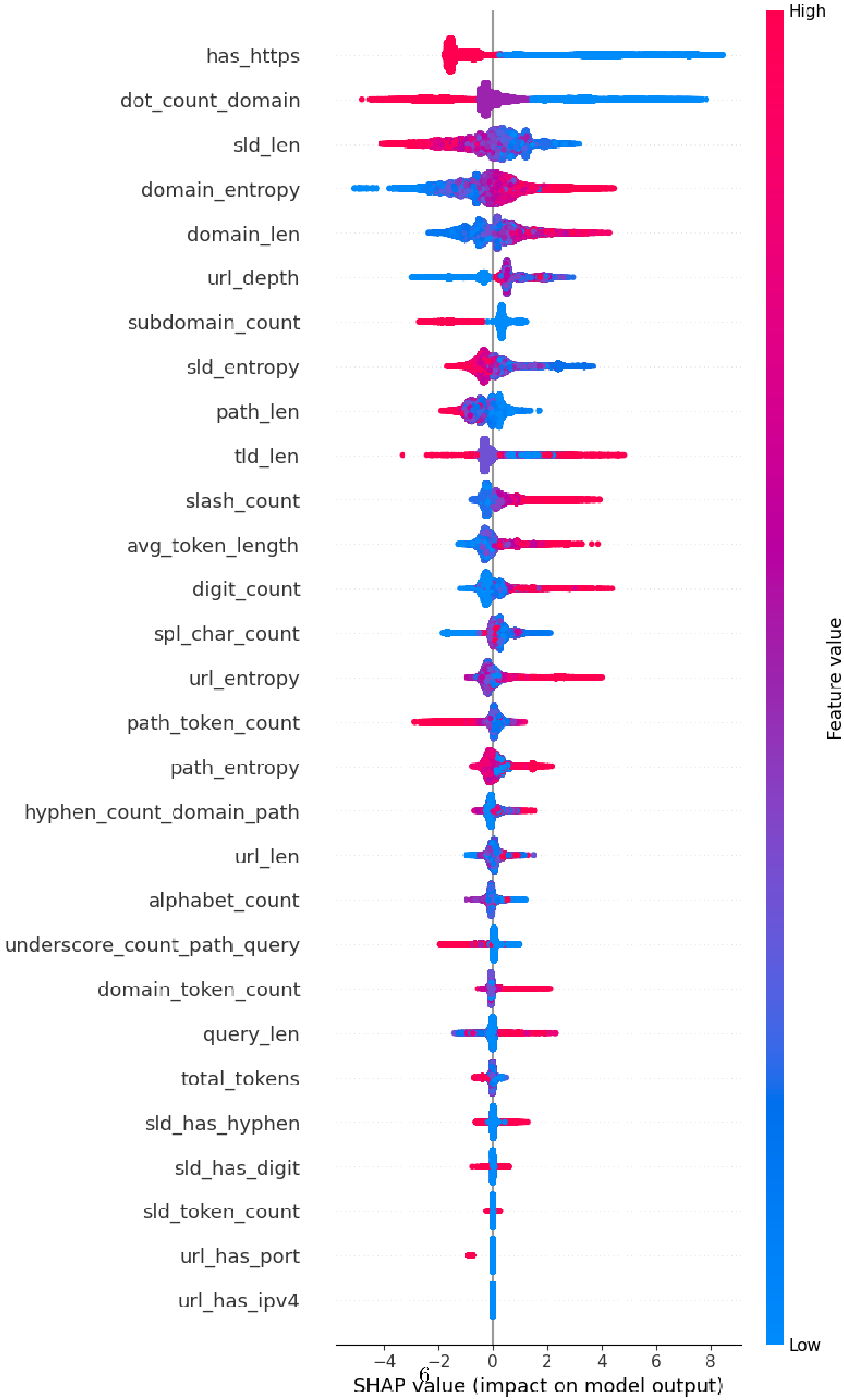


Feature Importance

[9]:
```
features = df.drop(columns='class').columns
explainer = shap.TreeExplainer(xg_model)
shap_values = explainer.shap_values(X_test)

shap.initjs()
```

<IPython.core.display.HTML object>

[10]:
```
X_test_df = pd.DataFrame(X_test,columns=features)
shap.summary_plot(shap_values,X_test_df,max_display=30)
```


has_https
dot_count_domain
sld_len
domain_entropy
domain_len
url_depth
subdomain_count
sld_entropy
path_len
tld_len
slash_count
avg_token_length
digit_count
spl_char_count
url_entropy
path_token_count
path_entropy
hyphen_count_domain_path
url_len
alphabet_count
underscore_count_path_query
domain_token_count
query_len
total_tokens
sld_has_hyphen
sld_has_digit
sld_token_count
url_has_port
url_has_ipv4
−4
−2
0
2
4
6
8
SHAP value (impact on model output)
High
Low
Feature value

From the above plot, most of the features have some impact in predicting phishing URLs. But there are some features which does not show any impact on predictions: - total_tokens - sld_token_count - url_has_ipv4 - url_has_port

So, we will ignore these features.

Therefore the final features considered to train the model are: - has_https - url_len - domain_len - path_len - query_len - url_depth - subdomain_count - tld_len - sld_len - sld_has_digit - sld_has_hyphen - dot_count_domain - hyphen_count_domain_path - underscore_count_path_query - slash_count - digit_count - alphabet_count - spl_char_count - url_entropy - domain_entropy - sld_entropy - path_entropy - domain_token_count - path_token_count - avg_token_length

[11]:
```
# Updated data
updated_df = df.
 ↪drop(columns=['total_tokens','sld_token_count','url_has_ipv4','url_has_port'])

X_updated = updated_df.drop(columns='class').values
y_updated = updated_df['class']

X_train_updated,X_test_updated,y_train_updated,y_test_updated =␣
 ↪train_test_split(X_updated,y_updated,train_size=0.75,random_state=6)
```

[12]:
```
# Saving the updated data
updated_df.to_csv('data/feature_refined/feature_refined_data.csv',index=False)
print('Updated data saved')
```

```
Updated data saved
```

[13]:
```
model = XGBClassifier()
model.fit(X_train_updated,y_train_updated)
```

[13]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, …)
```

[14]:
```
print('Metrics of XGBoost Classifier on Training data:')
evaluate_model(y_train_updated,model.predict(X_train_updated))
```

```
print()

print('Metrics of XGBoost Classifier on Testing data:')
evaluate_model(y_test_updated,model.predict(X_test_updated))
```

```
Metrics of XGBoost Classifier on Training data:
Accuracy: 0.986012275793781
Precision: 0.9966471715236462
Recall: 0.9743804606341422
F1-Score: 0.9853880424961834

Metrics of XGBoost Classifier on Testing data:
Accuracy: 0.981944419590931
Precision: 0.9937223461785728
Recall: 0.9689159292035399
F1-Score: 0.9811623695461419
```

Hyperparameter Tuning

[15]:
```
search_space = {
    "max_depth" : hp.quniform("max_depth",3,10,1),
    "learning_rate" : hp.loguniform("learning_rate",np.log(0.01),np.log(0.2)),
    "n_estimators" : hp.quniform("n_estimators",100,600,50),
    "subsample" : hp.uniform("subsample",0.6,1.0),
    "colsample_bytree" : hp.uniform("colsample_bytree",0.6,1.0),
    "gamma" : hp.uniform("gamma",0,5),
    "min_child_weight" : hp.qloguniform("min_child_weight",1,10,1),
    "reg_alpha" : hp.loguniform("reg_alpha",np.log(1e-3),np.log(1)),
    "reg_lambda" : hp.loguniform("reg_lambda",np.log(1),np.log(10))
}
```

[16]:
```
def objective(params):

    params['max_depth'] = int(params['max_depth'])
    params['n_estimators'] = int(params['n_estimators'])
    params['min_child_weight'] = int(params['min_child_weight'])

    params.update({
        "objective" : "binary:logistic",
        "eval_metric" : "logloss",
        "random_state" : 42,
        "tree_method" : "hist",
        "n_jobs" : -1
    })

    skf = StratifiedKFold(n_splits=5,shuffle=True,random_state=42)

    recalls = []
```

```
    for train_idx,val_idx in skf.split(X_train,y_train):
        X_tr, X_val = X_train[train_idx], X_train[val_idx]
        y_tr, y_val = y_train[train_idx], y_train[val_idx]

        model = XGBClassifier(**params)
        model.fit(X_tr,y_tr)

        y_pred = model.predict(X_val)
        recalls.append(recall_score(y_val, y_pred))

    mean_recall = np.mean(recalls)

    return {"loss" : -mean_recall, "status" : STATUS_OK}
```

[17]:
```
trials = Trials()

best_params = fmin(
    fn = objective,
    space = search_space,
    algo = tpe.suggest,
    max_evals = 50,
    trials = trials
)
```

```
100%|      | 50/50 [12:20<00:00, 14.80s/trial, best loss:
-0.9720883684672247]
```

[18]:
```
best_params
```

[18]:
```
{'colsample_bytree': 0.8941098594943607,
 'gamma': 0.4766670950284926,
 'learning_rate': 0.17447844329778114,
 'max_depth': 9.0,
 'min_child_weight': 9.0,
 'n_estimators': 600.0,
 'reg_alpha': 0.038632516672771545,
 'reg_lambda': 3.0628993814390015,
 'subsample': 0.8153616462720845}
```

[19]:
```
best_params['max_depth'] = int(best_params['max_depth'])
best_params['n_estimators'] = int(best_params['n_estimators'])
best_params['min_child_weight'] = int(best_params['min_child_weight'])

best_params.update({
    'objective' : 'binary:logistic',
    'eval_metric' : 'logloss',
    'random_state' : 42,
```

```
    'n_jobs' : -1
})

final_model = XGBClassifier(**best_params)
final_model.fit(X_train,y_train)

print('Metrics of XGBoost Classifier on Training data:')
evaluate_model(y_train,final_model.predict(X_train))

print()

print('Metrics of XGBoost Classifier on Testing data:')
evaluate_model(y_test,final_model.predict(X_test))
```

```
Metrics of XGBoost Classifier on Training data:
Accuracy: 0.9927884185222521
Precision: 0.997572513382298
Recall: 0.9875044670913997
F1-Score: 0.9925129583413324

Metrics of XGBoost Classifier on Testing data:
Accuracy: 0.9819265250612887
Precision: 0.9903838930208099
Recall: 0.9721976401179941
F1-Score: 0.9812065051542556
```

Handling False Negatives

[20]:

```
import numpy as np
from sklearn.metrics import precision_recall_curve, f1_score

def recall_and_threshold_constrained_selection(
    y_true,
    y_prob,
    min_recall=0.9750,
    min_threshold=0.45
):
    try:
        precision, recall, thresholds = precision_recall_curve(y_true, y_prob)

        # precision_recall_curve returns one extra precision/recall value
        precision = precision[:-1]
        recall = recall[:-1]

        thresholds = np.array(thresholds)

        # Apply BOTH constraints
        valid_idxs = np.where(
```

```
            (recall >= min_recall) &
            (thresholds >= min_threshold)
        )[0]

        if len(valid_idxs) == 0:
            raise ValueError(
                f"No threshold satisfies recall >= {min_recall} "
                f"and threshold >= {min_threshold}"
            )

        # Among valid thresholds, maximize precision
        best_idx = valid_idxs[np.argmax(precision[valid_idxs])]
        best_threshold = thresholds[best_idx]

        return {
            "threshold": float(best_threshold),
            "precision": float(precision[best_idx]),
            "recall": float(recall[best_idx]),
            "f1": float(
                f1_score(
                    y_true,
                    (y_prob >= best_threshold).astype(int)
                )
            ),
            "accuracy": accuracy_score(
                    y_true,
                    (y_prob >= best_threshold).astype(int)
                )
        }
    except Exception:
        fallback_threshold = np.percentile(y_prob,95)

        return {
            "threshold": fallback_threshold,
            "accuracy": float(accuracy_score(y_true, (y_prob >=␣
↪fallback_threshold).astype(int))),
            "precision": float(precision_score(y_true, (y_prob >=␣
↪fallback_threshold).astype(int))),
            "recall": float(recall_score(y_true, (y_prob >= fallback_threshold).
↪astype(int))),
            "f1_score": float(f1_score(y_true, (y_prob >= fallback_threshold).
↪astype(int))),
            "fallback_used": True
        }
```

[81]:
```
y_prob = final_model.predict_proba(X_test)[:,1]
```

[82]:
```
recall_and_threshold_constrained_selection(y_test,y_prob,0.9725,0.45)
```

[82]:
```
{'threshold': 0.7680490612983704,
 'precision': 0.9987181313203247,
 'recall': 0.9737085724865766,
 'f1': 0.9860547964469027,
 'accuracy': 0.9866908244978303}
```

After applying a recall-constrined threshold (0.45) slightly increased recall to 0.9725 while maintaining very high precision, resulting in a unchanged F1-score and accuracy compared to the initial model. Now the threshold is 0.48.

Saving the model

[80]:
```
with open('models/XGBoostModel.pkl','wb') as file:
    pickle.dump(final_model,file=file)
```