Malicious Web Page Prediction

Nadine Safwat - 900212508

Link to model scripts: 🗀 Machine Learning Project

INTRODUCTION

To recap, this project is attempting to create a malicious URL detection system using machine learning models. To begin I gathered research on existing systems that use the same technique and chose an appropriate dataset to train my models. Then I preprocessed the data so that it is suitable and allows the models to train and predict fairly.

THE MODELS

This is a binary classification problem where the label 0 means that the website is malicious and 1 means that it is benign. Thus the models that were considered are:

- KNN
- Logistic Regression
- Decision Trees (CART)
- Random Forest
- Naive Bayes
- Neural Network
- SVM

The dataset was already split into a test and train set so I simply preprocessed both CSVs in the same way and used the provided split. The train set consists of 1.2M data points and the test has 400K so there is a 75:25 split.

KNN

Setup:

To begin, we use GridSearchCV with 5-fold cross validation to fit the model on multiple Ks so that we can determine the best K to use. This has to be done because the KNN model is extremely sensitive to this parameter. Ideally we should test every K until the square root of the size of the dataset, however, due to its heavy computation and slow running speed I only ran Ks 1,3,5,7 and 9. GridSearchCV found the best k to be 9.

Parameter Choice:

For GridSearchCv the parameters i passed are the KNNClassifier as the estimator, the KFold object which defines 5 folds in CV and for default scoring was left as the default which is accuracy. I also added verbose = 3 so that the time for each fit and the score from each one is shown as they run.

For the actual classifier I used KNeighborsClassifier with all the default parameters except n_neightbours = best_k which was 9. The default parameters include a uniform weight function and the minkowski metric for distance computation.

Performance Evaluation:

As shown in the classification report, the stores for both the 0 and the 1 classes were extremely high which implies that the model did not overfit and could easily and accurately predict the label for new data.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.98      8062
           1       1.00      1.00      1.00    353872

    accuracy                           1.00    361934
   macro avg       1.00      0.98      0.99    361934
weighted avg       1.00      1.00      1.00    361934
```

Pros:

The model is simple to implement and can predict the label for new data with extremely high accuracy and recall.

Cons:

It is extremely computation and memory heavy, with each fit taking around 45-50 minutes to run showing that it is extremely inefficient for my dataset.

LOGISTIC REGRESSION

Setup:

Using the LogisticRegression and KFold classes from the sklearn library, the model was initialized with the default parameters and training with 5-fold cross validation in order to reduce overfitting. The dataset was converted into a NumPy array after being read from the CSV as it proved to run faster. The test data was then passed to the predict function on the fitted model and a classification report was printed.

Parameter Choice:

All the parameters were set to the default. These include the max_ter parameter that specifies the maximum number of iterations to get the weights is set to 100. The multi_class parameter which is default auto will choose ‘ovr’ as my data binary. I did not use verbose, instead I used the sklearn functions to print out the metrics for each

fold to get a general understanding of the performance of the model.

Performance Evaluation:

The classification report shows that the Logistic Regression model provided extremely good scores for each class, with the lowest being 0.93. The model was also quick to run with about a run time of 10 to 11 minutes for all the folds. However, when running the folds, I got a memory error multiple times before being able to run all the folds in one go. This may be an issue from VSCODE or it might imply that the model uses more memory than I would like.

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.93	0.96	8062
1	1.00	1.00	1.00	353872
accuracy			1.00	361934
macro avg	1.00	0.97	0.98	361934
weighted avg	1.00	1.00	1.00	361934

Pros:

The model can quickly and accurately predict new data which implies no overfitting. When i checked the fitted weights it shows that no one feature has an overly high weight compared to the rest so the model is compatible with the data.

Cons:

Threw multiple memory errors while running the code which might mean that it is memory heavy. It also assumes a linear relationship between the features which is not the case for my data.

DECISION TREES (CART)

Setup:

Using the DecisionTreeClassifier and KFold classes from the sklearn library, the model was initialized with the default parameters and training with 5-fold cross validation in order to reduce overfitting. The dataset was converted into a NumPy array after being read from the CSV as it proved to run faster. The test data was then passed to the predict function on the fitted model and a classification report was printed.

Parameter Choice:

All the parameters were set to the default choices. This means the function that was used to calculate gain was ‘gini’ and the strategy to choose each split is set to ‘best’. The max_depth is set to None

Performance Evaluation:

Again the performance of this model provided extremely good scores. The 1 class having perfect prediction and the 0 class only slightly worse. The model also ran extremely quickly (around 10 minutes) and with no memory errors.

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.97	0.97	8062
1	1.00	1.00	1.00	353872
accuracy			1.00	361934
macro avg	0.98	0.99	0.98	361934
weighted avg	1.00	1.00	1.00	361934

Pros:

The model has a very quick runtime and is very simple to implement. I also printed out the tree that was produced and, compared to the number of features in the data it does not seem to be overfitting.

Cons:

The model uses a greedy approach so the tree produced might not be the most optimal. Also my data is imbalanced and the difference in scores can be due to a bias towards the 1 class.

RANDOM FOREST

Setup:

Using the RandomForestClassifier and KFold classes from the sklearn library, the model was initialized with the default parameters and training with 5-fold cross validation in order to reduce overfitting. The dataset was converted into a NumPy array after being read from the CSV as it proved to run faster. The test data was then passed to the predict function on the fitted model and a classification report was printed.

Parameter Choice:

All the parameters were set to the default. The number of trees in the forest is set to 100, the default measure for gain is ‘gini’ and there is no max depth set.

Performance Evaluation:

The classification report shows that random forest was better at predicting the 0 class than decision trees were, but still both have increasingly good results. Run time for random first was a little slower however but only 3-5 minutes more. No memory errors came up in the model either.

Classification Report:	precision	recall	f1-score	support
0	0.99	0.96	0.98	8062
1	1.00	1.00	1.00	353872
accuracy			1.00	361934
macro avg	1.00	0.98	0.99	361934
weighted avg	1.00	1.00	1.00	361934

Pros:

The model has extremely accurate results and for both classes and is robust to noisy data and outliers. It also trains very quickly.

Cons:

Random forest is slightly more prone to overfitting than CART but the results imply that this is not a problem. Also it might consume more memory.

NAIVE BAYES

Setup:

Using the GaussianNB and KFold classes from the sklearn library, the model was initialized with the default parameters and training with 5-fold cross validation in order to reduce overfitting. The dataset was converted into a NumPy array after being read from the CSV as it proved to run faster. The test data was then passed to the predict function on the fitted model and a classification report was printed.

Parameter Choice:

Default parameters were used so the priors parameter is set to 'uniform' so every class is assumed to have equal likelihood. Var_smoothing is set to 1e-9 as a way to stabilize the computations

Performance Evaluation:

The results for this model were the worst results of all models. Neither class has good results overall and the weighted averages are all close to 0. The model ran very quickly (10-15 minutes) and no memory error was thrown.

Classification Report:	precision	recall	f1-score	support
0	0.02	0.99	0.05	8062
1	1.00	0.04	0.08	353872
accuracy			0.06	361934
macro avg	0.51	0.52	0.06	361934
weighted avg	0.98	0.06	0.08	361934

Pros:

The model is simple and easy and fast to implement and run. And is robust against irrelevant features.

Cons:

The scores from this model were not good which shows that it is not compatible with my data. There is also an assumption of feature independence which is not the case for my data.

NEURAL NETWORK

Setup:

For this model Tensorflow's Keras library was used to build an MLP (Multi-Layer Perceptron) Classifier with two hidden layers. I used tensorflow instead of sklearn as it was much faster than the sklearn class. 5-Fold cross validation was used in the fitting process as it can reduce performance bias and reduce overfitting. The dataset was converted into a NumPy array after being read from the CSV as it proved to run faster. The test data was then passed to the predict function on the fitted model and a classification report was printed.

Parameter Choice:

The sequential model is used, then we add two dense layers which will be the hidden layers, the first one has 128 neurons and the second 64 neurons and the activation function is 'relu'. The dropout layers are added to prevent overfitting with 0.2 meaning 20% of input units are dropped. For the output layer it has 2 neurons for the two output classes and a softmax activation function. For model compilation we use the adam optimizer, and for the loss function we assign the 'sparse_categorical_crossentropy' for multi-class classification. Metrics = ['accuracy'] is the metric that is printed for evaluation.

Performance Evaluation:

Like the rest of the models the scores are extremely good for both classes despite the imbalance between the labels. No memory errors are thrown when running this model, however, compared to the other models (other than KNN) it takes the most time to run (around 25 minutes)

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.98      8062
           1       1.00      1.00      1.00    353872

    accuracy                           1.00    361934
   macro avg       1.00      0.98      0.99    361934
weighted avg       1.00      1.00      1.00    361934
```

Pros:

Cons:

SVM

Setup:

Using the LinearSVC and KFold classes from the sklearn library, the model was initialized with the default parameters and training with 5-fold cross validation in order to reduce overfitting. The dataset was converted into a NumPy array after being read from the CSV as it proved to run faster. The test data was then passed to the predict function on the fitted model and a classification report was printed.

Parameter Choice:

The default parameters used are 'squared_hinge' for the loss function. For dual we set it to false because the number of samples is greater than the number of features. The tolerance for stopping is 1e-4 and multi-class is set to 'ovr'.

Performance Evaluation:

As we can see from the classification report the scores for linearSVM are very good, and the run time was also quite fast taking around 15 minutes to finish running with no memory errors being thrown.

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97      8062
           1       1.00      1.00      1.00    353872

    accuracy                           1.00    361934
   macro avg       1.00      0.97      0.98    361934
weighted avg       1.00      1.00      1.00    361934
```

Pros:

Provides good results for both classes and is good for both linearly or non linearly separable data. Its good when dealing with high dimensional feature spaces which is applicable to my data

Cons:

This model is not suitable for large datasets as the computational complexity increases.

COMPARATIVE ANALYSIS OF TOP 3 MODELS AND TOP CHOICE

Seeing as most of the models provided top results the models I chose for the top three is based on run time speed and memory usage and these are:

1. Random Forest
2. Decision Trees
3. Logistic Regression

Random Forest is my chosen model because, despite being slightly more memory heavy it still did not throw any memory errors and ran smoothly, as well as quickly. The scores that were produced were of the best and the parameters can still be adjusted to improve the performance even more. Due to the imbalance in my data I felt that Random Forest was the most appropriate model to choose because it deals with this imbalance better than Decision Trees.

Decision Trees are my second choice because it's the fastest running model with some of the best results. I printed out the tree that was created to ensure that no overfitting or single feature dependencies are happening. My only concern with this model is that it may be affected more by the 2:98 split in my label. Otherwise it is more memory efficient and simpler to implement

Logistic Regression is my third best option because it is the second fastest running model and has extremely good scores as well. The implementation is simple and the output weights show that no one feature has higher weights than the rest. This means there's no overfitting and a good split between feature dependencies. The biggest issue with this model is the memory errors that occurred when running the model which could be an error when using it for a bigger system.