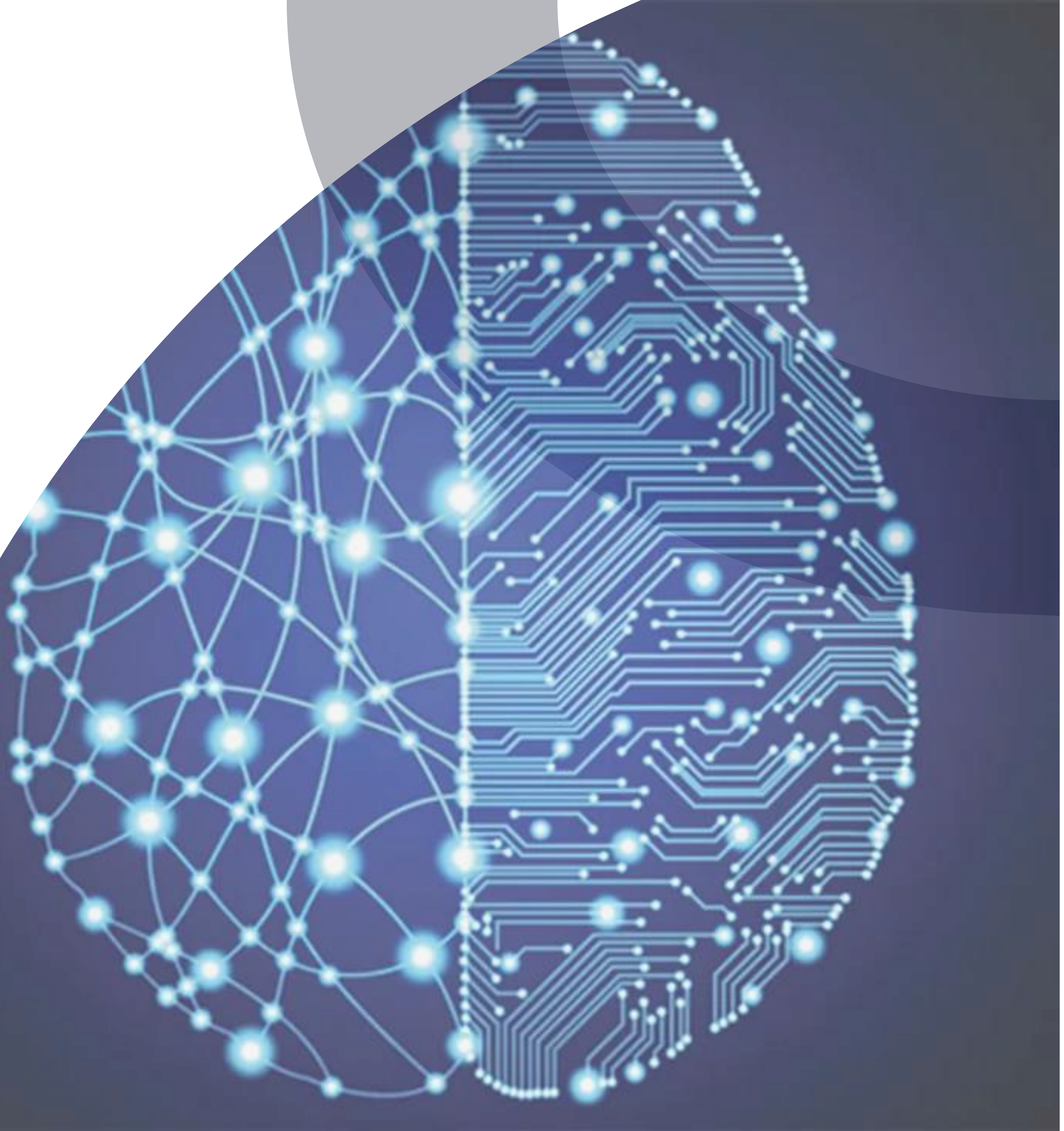
Machine Learning Approach for MACROS Classification

Shahar Zaidel, Maor Saadon, Eden Mor

Why using Machine Learning in Cyber Security is so important?

The ability to detect cyber attacks dynamically using machine learning algorithms can significantly bolster cybersecurity measures, enabling prompt identification and mitigation of threats, thereby enhancing overall network security posture.



Our solution to the problem

We used a combination of tools for preprocessing data, feature selection, validation and the prediction process itself

Preprocessing the data – Data Exploration

- **The Data-** The data is imported from a CSV file, with 2 columns containing VBA code and the corresponding labels.

- **Feature Selection:** We have decided to compute six specific features for each entry in our dataset after researching and finding articles that highlighted their importance in identifying malicious VBA codes.

- **Labels-** We changed the label's values from { "mal", "White" } to {0, 1} for more comfort using the different functions.

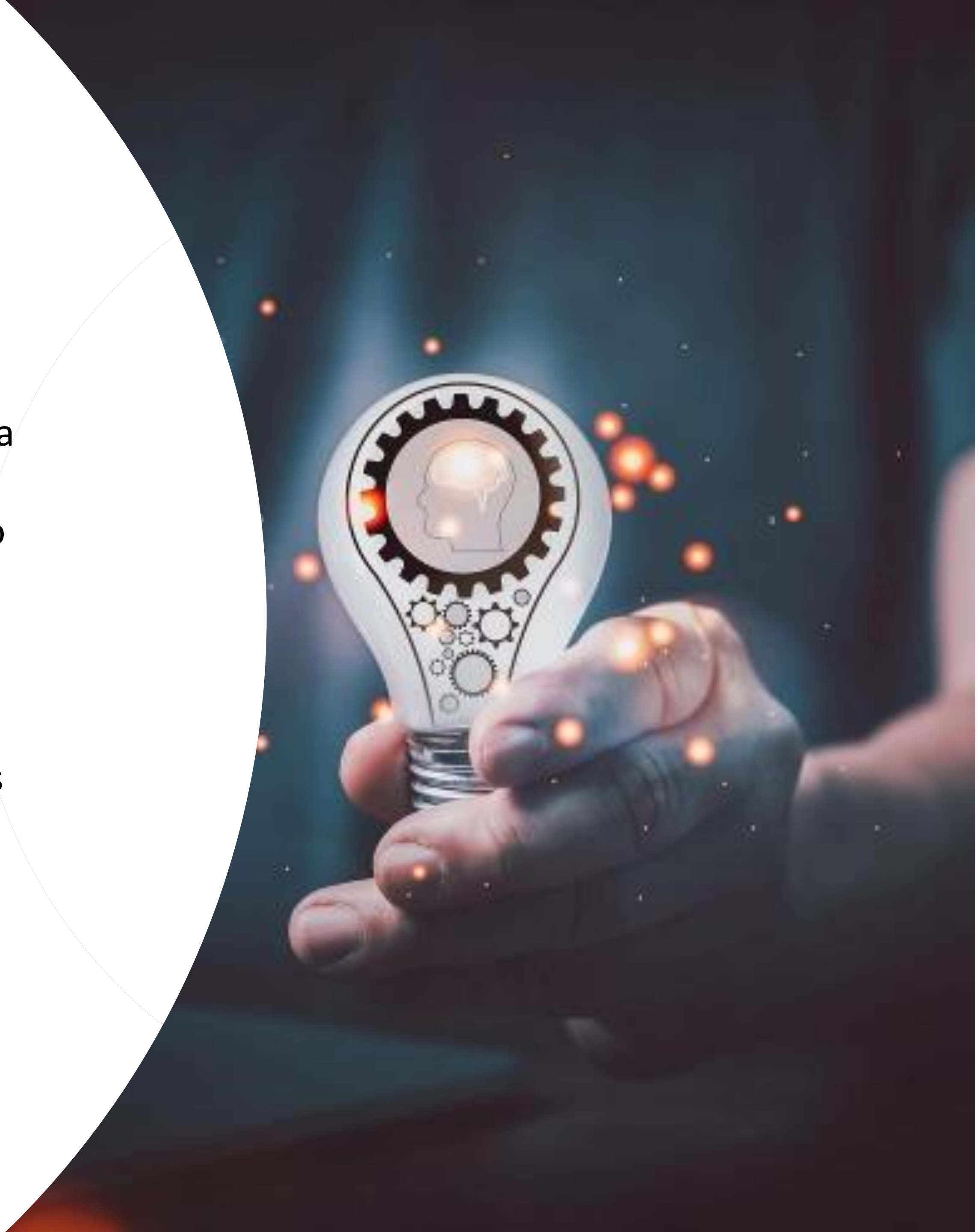
- **Creation and utilization of a pipeline-** Model training involves several steps. We utilized a pipeline, which enables the automatic execution of a sequence of stages.

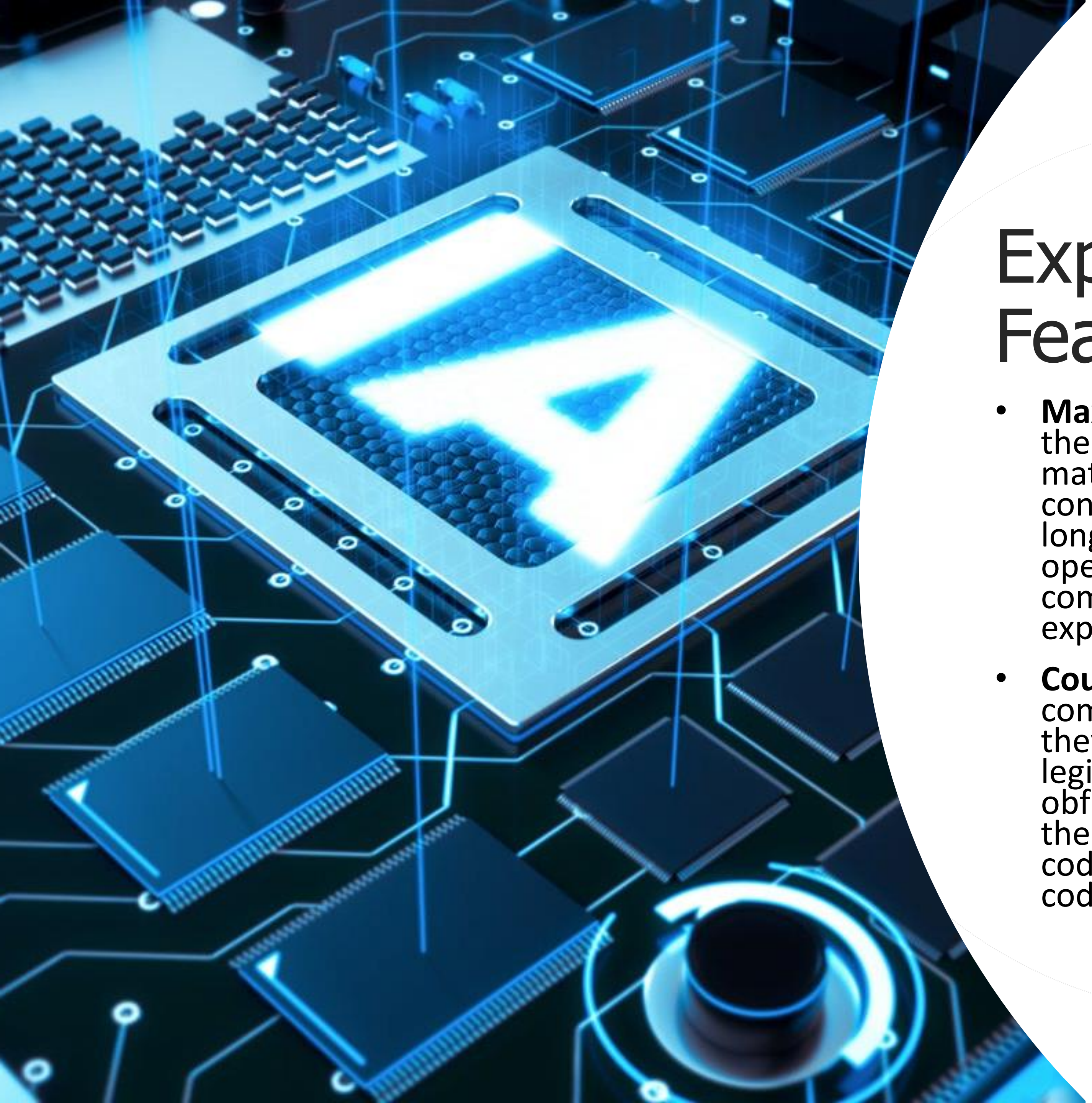
The Features:

- Average Variable Assignment Length
- Integer/String Variables Ratio
- Max Consecutive Math Operations
- Macro Keywords
- Casing Ratio in Variable Declarations
- Code Length

Explanation of The Features:

- **Casing Ratio in Variable Declarations-** Typically a programmer adopts a naming convention when it comes to variables. We would expect the ratio of upper-case to lower-case characters in variable names to be either close to zero or significantly greater than one.
- **Average Variable Assignment Length-** Research indicates that many malicious macro commands declare unusually long string variables. This is determined by comparing the average length of variables across both legitimate and malicious code. To capture this characteristic, the average length of string variables in the original VBA code is calculated.





Explanation Of The Features:

- **Max Consecutive Math Operations-** refers to the maximum consecutive sequence of mathematical operations presented in a given context or dataset. This feature quantifies the longest uninterrupted chain of such operations, providing insight into the complexity and structure of mathematical expressions within the dataset.
- **Count of Integer/String Variables-** Another common attribute of malicious macros is that they define more integer/String variables than legitimate macros, complicating and obfuscating the code. To capture this feature, the number of integer variables in the source code of the macro is calculated as part of the code's overall length.

Explanation Of The Features:

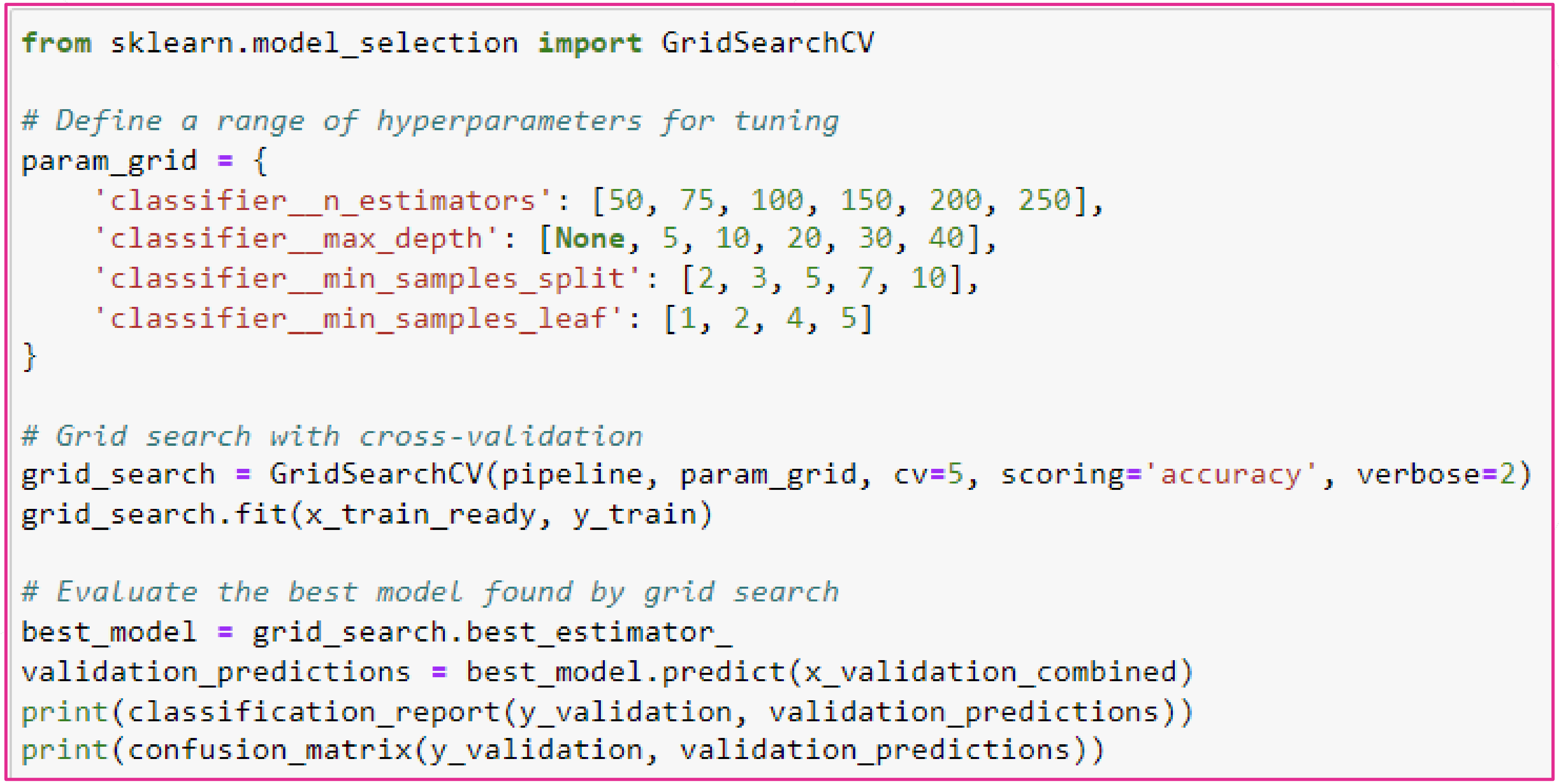
- **Code Length-** This feature simply refers to the length of the code itself. In the context of analyzing macros, the length of the code can provide valuable insights. Malicious macros often contain extensive and convoluted code compared to legitimate ones
- **Macro Keywords-** A binary attribute encoding the presence of certain keywords found to be significantly more frequent among the malicious subset, both in literature and in our dataset. Examples of such keywords include AutoOpen, AutoClose, DocumentOpen, and DocumentClose.

The Pipeline:

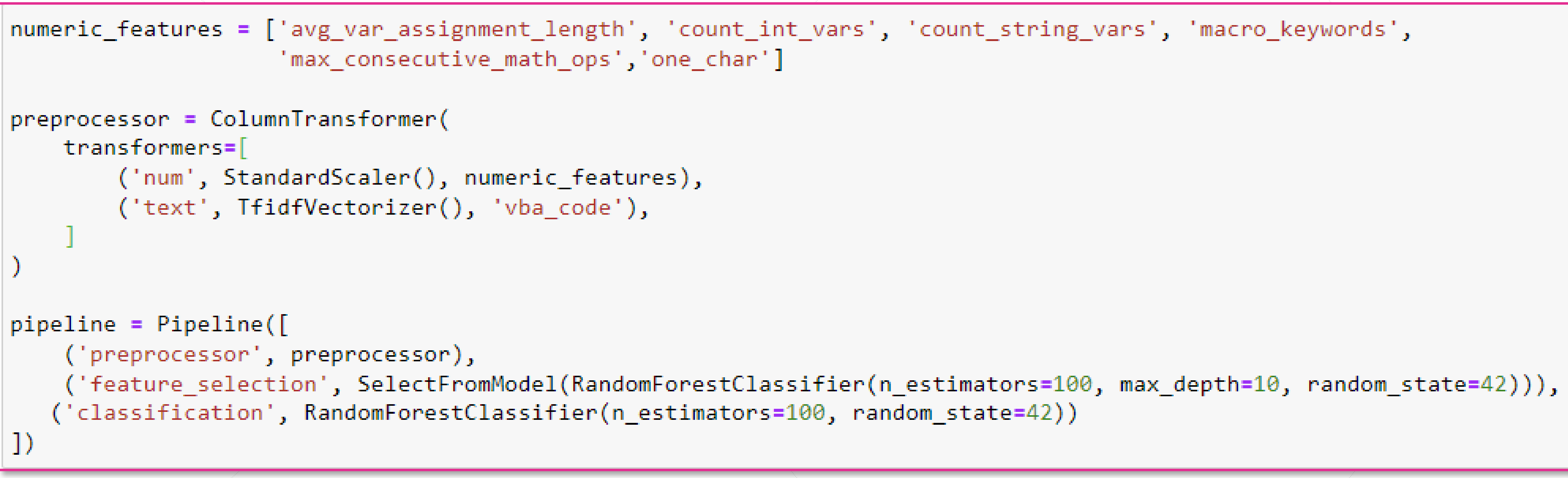
The pipeline comprises several key components:

• **StandardScalar-** Normalizes the data to facilitate easier classification between them.
• **TfidfVectorizer-** Converts each VBA code into a matrix of important words.
• **Feature selection-** Selects the most important features based on a random forest model. This allows for filtering features and simplifying the model's execution with minimal cost.
• **RandomForestClassifier-** The classifier we used.

We used a grid search in order to determine which parameters give the best result

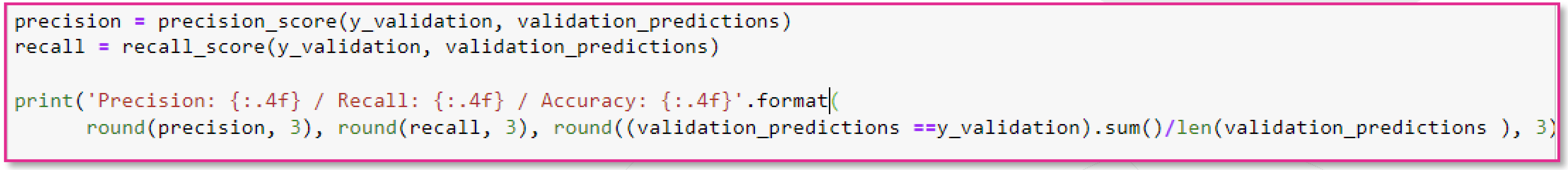


```python
from sklearn.model_selection import GridSearchCV

# Define a range of hyperparameters for tuning
param_grid = {
    'classifier__n_estimators': [50, 75, 100, 150, 200, 250],
    'classifier__max_depth': [None, 5, 10, 20, 30, 40],
    'classifier__min_samples_split': [2, 3, 5, 7, 10],
    'classifier__min_samples_leaf': [1, 2, 4, 5]
}

# Grid search with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', verbose=2)
grid_search.fit(x_train_ready, y_train)

# Evaluate the best model found by grid search
best_model = grid_search.best_estimator_
validation_predictions = best_model.predict(x_validation_combined)
print(classification_report(y_validation, validation_predictions))
print(confusion_matrix(y_validation, validation_predictions))
```



```
numeric_features = ['avg_var_assignment_length', 'count_int_vars', 'count_string_vars', 'macro_keywords',
                    'max_consecutive_math_ops','one_char']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('text', TfidfVectorizer(), 'vba_code'),
    ]
)

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('feature_selection', SelectFromModel(RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42))),
   ('classification', RandomForestClassifier(n_estimators=100, random_state=42))
])
```

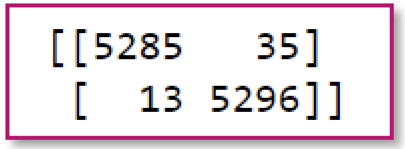
The Results:



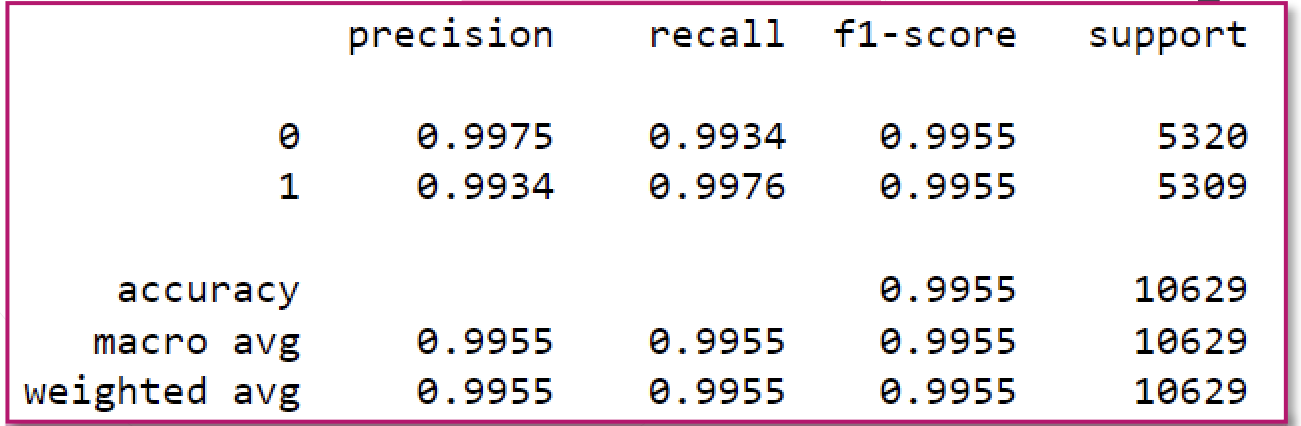
```
precision = precision_score(y_validation, validation_predictions)
recall = recall_score(y_validation, validation_predictions)

print('Precision: {:.4f} / Recall: {:.4f} / Accuracy: {:.4f}'.format(
      round(precision, 3), round(recall, 3), round((validation_predictions ==y_validation).sum()/len(validation_predictions ), 3)
```

Model evaluation- The model is assessed using various metrics, including precision, recall, and accuracy. Dealing with False

Positives and False Negatives-To minimize the number of errors made by the learning machine, we attempted to incorporate different features and evaluated comparisons between different models and methods to see what improves the model's performance the most.



```
[[5285   35]
 [  13 5296]]
```



```
              precision    recall  f1-score   support

           0     0.9975    0.9934    0.9955      5320
           1     0.9934    0.9976    0.9955      5309

    accuracy                         0.9955     10629
   macro avg     0.9955    0.9955    0.9955     10629
weighted avg     0.9955    0.9955    0.9955     10629
```

Related Work

https://ceur-ws.org/Vol-2259/aics_34.pdf- Detection of malicious VBA macros using Machine Learning methods Ed Aboud, Darragh O'Brien, Dublin City University.

Thank You For Listening