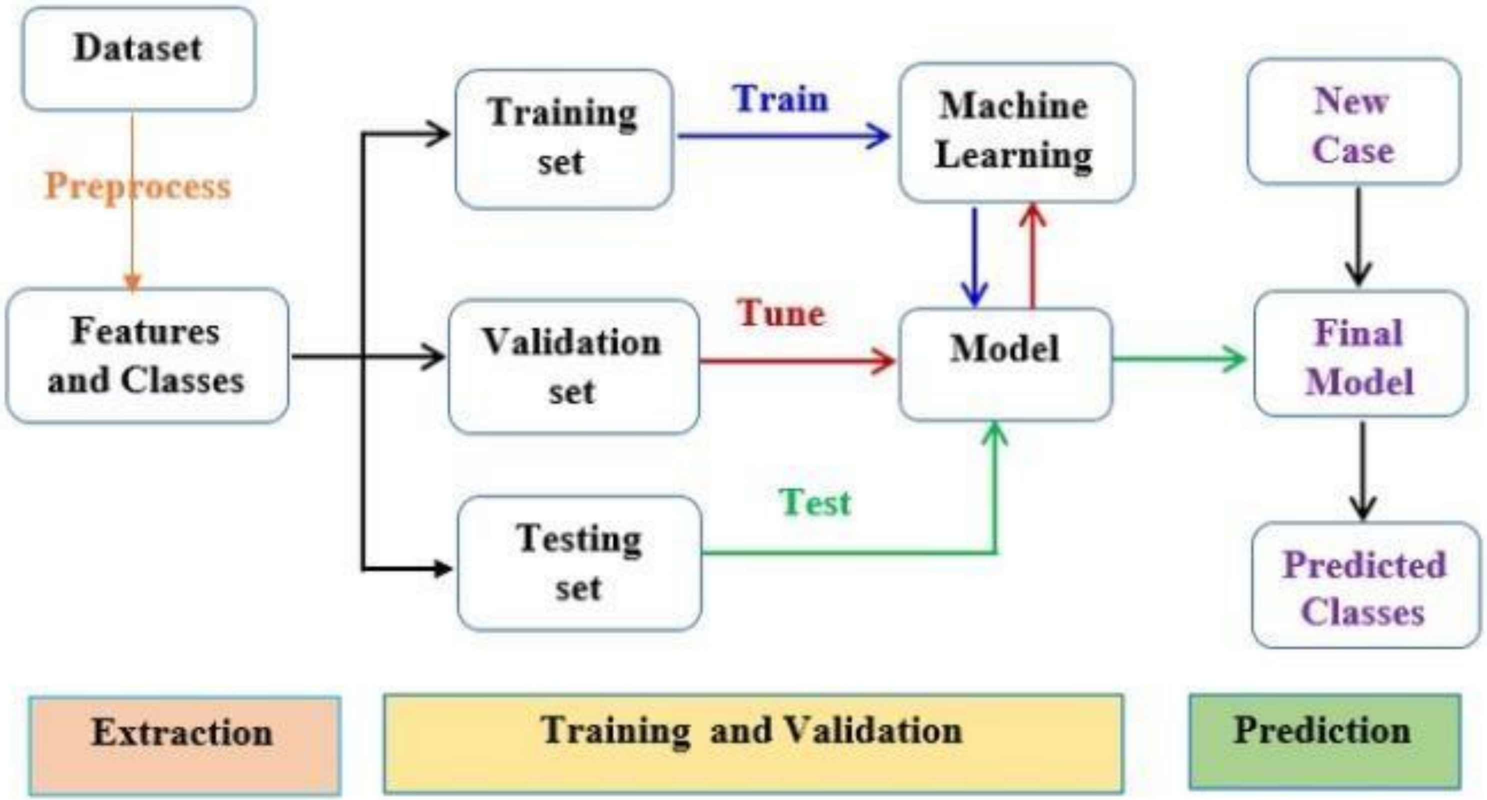
PREDICTING PASSWORD STRENGTH USING SUPERVISED MACHINE LEARNING TECHNIQUES

TEAM 14

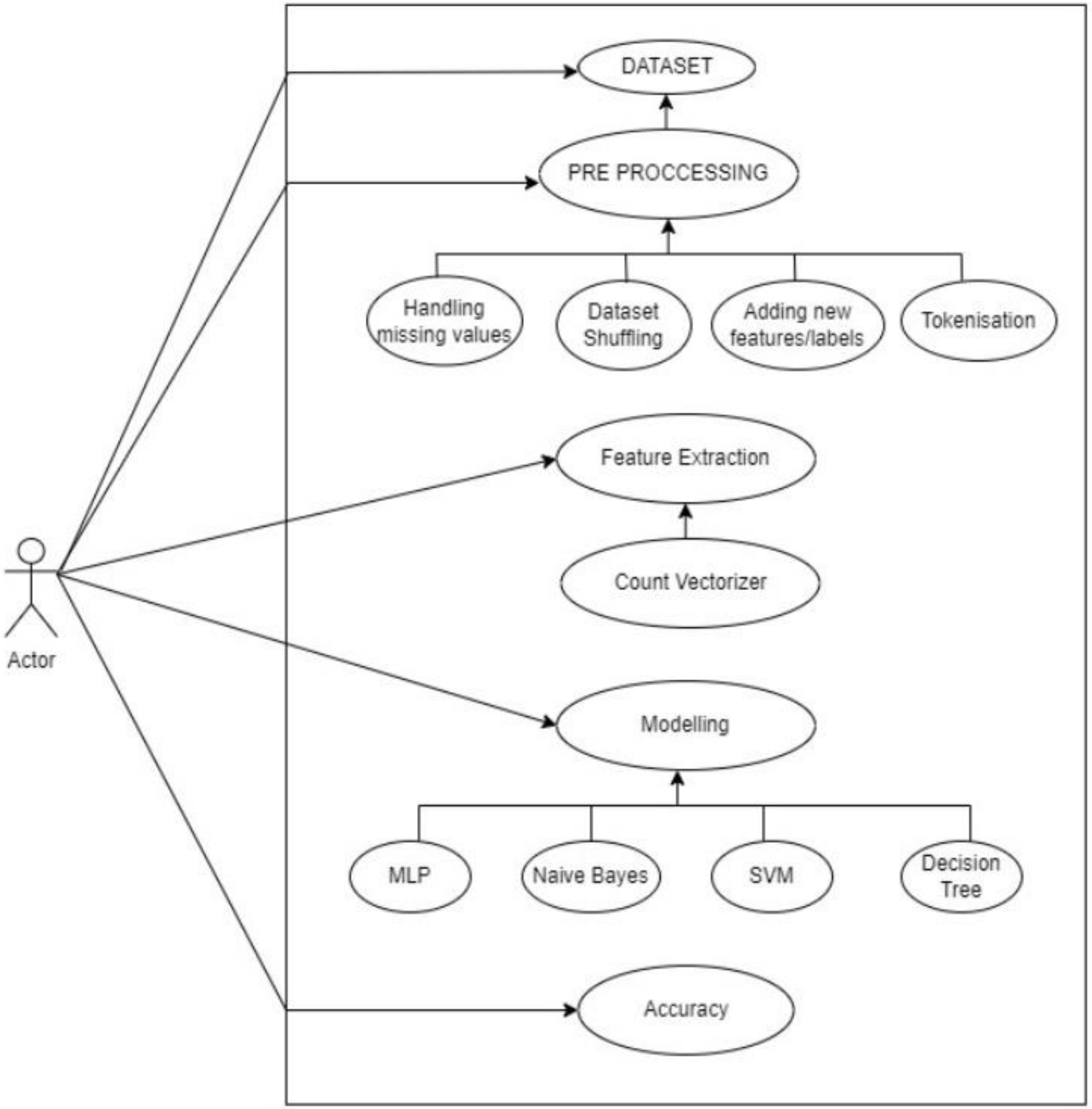
GROUP MEMBERS

1. JASWANTH REDDY .L	21MAI0016
2. ANUSHA .P	21MAI0050
3. PUNITHA .G	21MAI0056
4. BATTA VENKATA SAIKUMAR	21MAI0063

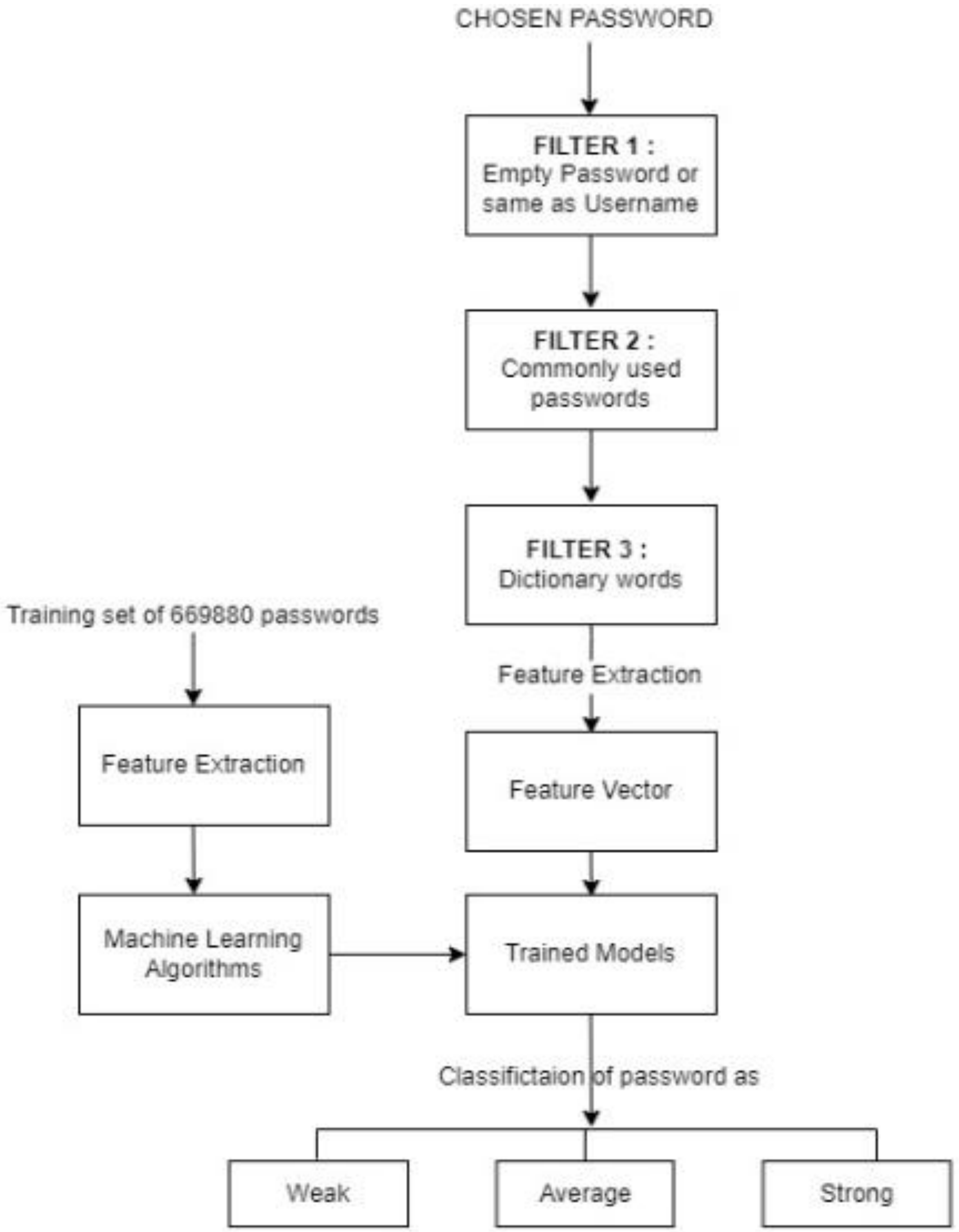
WORKFLOW DIAGRAM



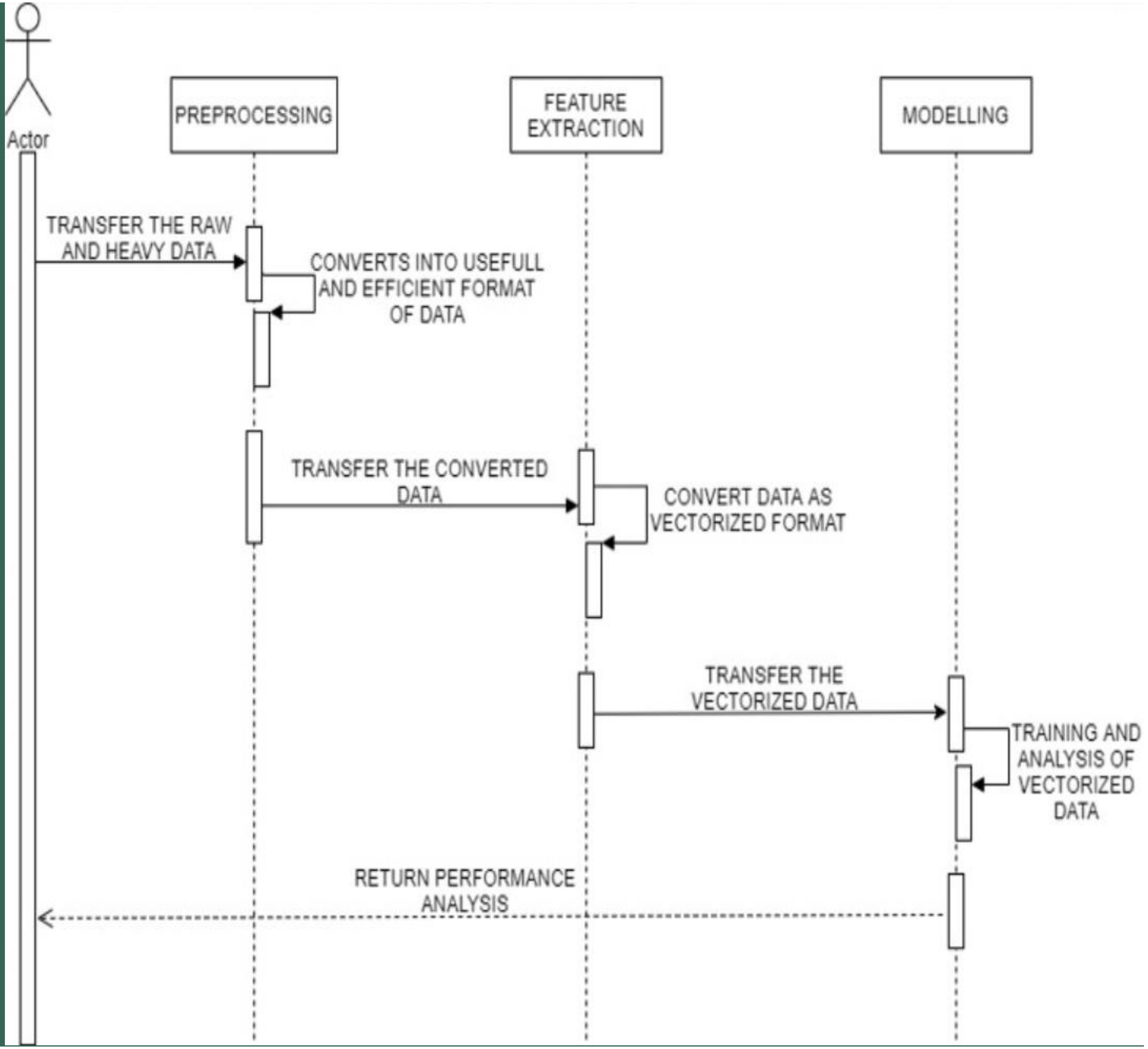
USE CASE DIAGRAM



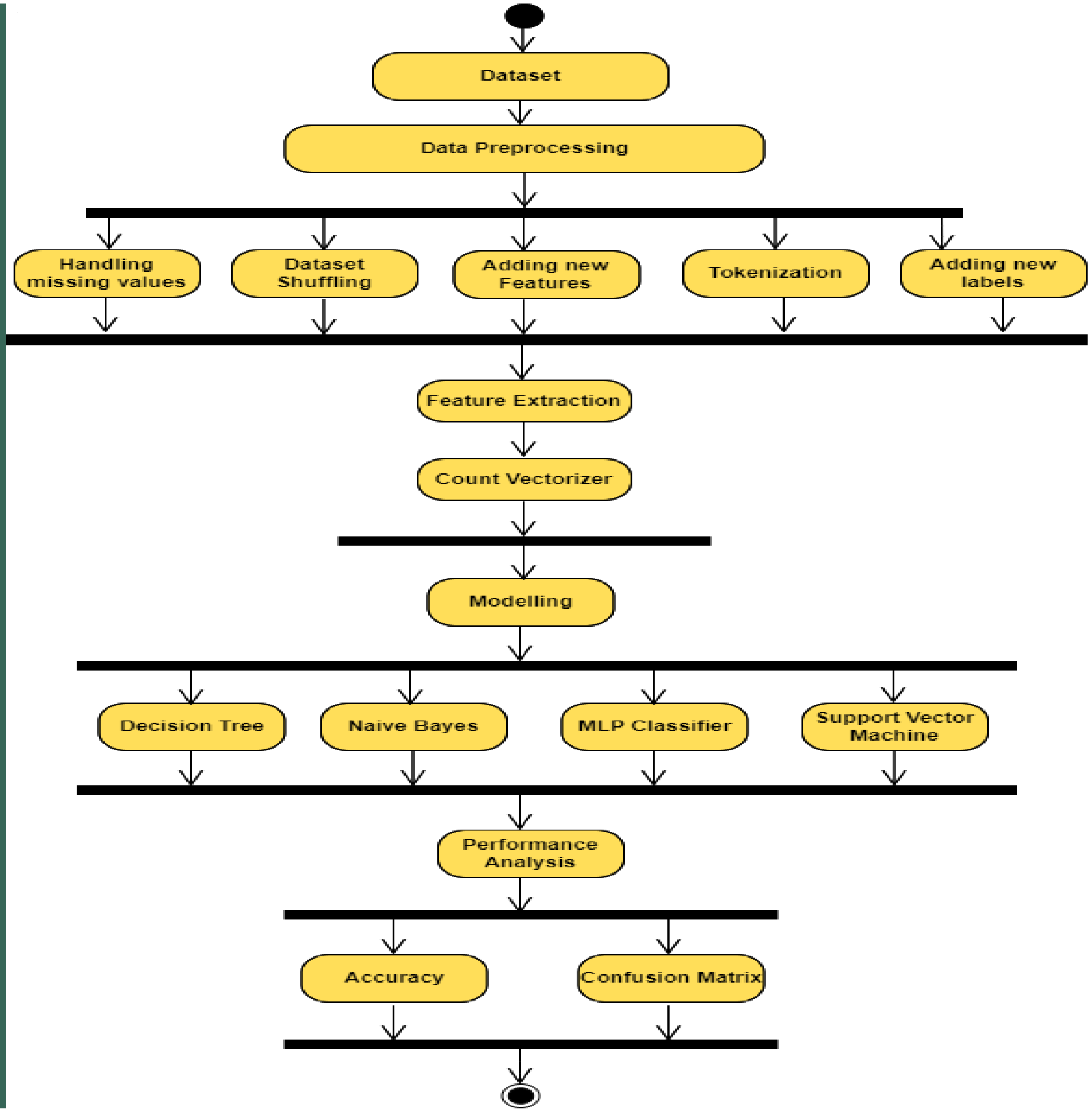
PROCESS FLOW DIAGRAM



SEQUENCE DIAGRAM



ACTIVITY DIAGRAM



MODULES

- DATA PRE-PROCESSING
- FEATURE EXTRACTION
- ML MODELLING
- DEPLOYMENT USING DJANGO FRAMEWORK IN HEROKU CLOUD

PRE-PROCESSING

- Data Pre-processing we transform the raw data into a useful and efficient format by removing noise and inconsistent data. So, it creates the reliable consistent data that improves the efficiency of the training data for analysis and also enables accurate decision – making.

- Handling missing values
- Dataset Shuffling
- Adding new features/labels
- Tokenization

HANDLING MISSING VALUES

Checking missing values

```
In [8]: pswd_data.isna().sum()

Out[8]: password    0
        strength    0
        dtype: int64

In [9]: pswd_data.shape

Out[9]: (20000, 2)
```

Removing missing values

```
In [10]: pswd_data.dropna(inplace=True)
```

Checking whether missing values removed or not

```
In [11]: pswd_data.isna().sum()

Out[11]: password    0
         strength    0
         dtype: int64
```

DATASET SHUFFLING

Convert dataset into array

```
In [42]: pswd = np.array(pswd_data)

In [43]: pswd

Out[43]: array([['furkozy------182928', 2],
                ['V02IgjjM5Mgleau7', 2],
                ['ocbvdwu764', 1],
                ...,
                ['crossmoci963', 1],
                ['felipe5691', 1],
                ['martina90', 1]], dtype=object)
```

Dataset shuffling

```
In [44]: import random

In [45]: random.shuffle(pswd)
```

ADDING NEW FEATURES/LABELS

Adding features and labels

```
In [46]: ylabels  = [s[1] for s in pswd]
         allpasswords = [s[0] for s in pswd]

In [47]: len(ylabels)  #checking length of labels

Out[47]: 20000

In [48]: len(allpasswords)   #Checking length of allpasswords

Out[48]: 20000
```

TOKENIZATION

Implementing tokenization

In [49]:
```
def createTokens(f):

    tokens = []
    for i in f:
        tokens.append(i)
    return tokens
```

FEATURE EXTRACTION

- Feature extraction refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set.

Initializing CountVectorizer

```
In [51]: from sklearn.feature_extraction.text import CountVectorizer
         vectorizer_state = CountVectorizer()

In [52]: #allpasswords = np.array(allpasswords).reshape(-1,1)

In [53]: X = vectorizer_state.fit_transform(allpasswords)
```

SPLITTING VECTORIZED DATA

Splitting vectorized dataset

```
In [55]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, ylabels, test_size=0.2, random_state=42)

In [56]: print(X_train.shape, X_test.shape) #y_train.shape, y_test.shape)

         (16000, 10105) (4000, 10105)
```

ML MODELS

ML models are used to predict the vulnerability of passwords.

- Support Vector Machine
- Naive Bayes
- Multi-Layer Perceptron Classifier
- Decision Tree

SUPPORT VECTOR MACHINE

Support Vector Machines(SVM)

In [36]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# define model and parameters
model = SVC()
kernel = ['poly', 'rbf', 'sigmoid', 'linear']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']

# define grid search
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train, y_train)

# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Tuned SVC Parameters: C = 50, gamma = 'scale', kernel = 'rbf'
```

SUPPORT VECTOR MACHINE

In [34]:
```
from sklearn.svm import SVC
svm_model = SVC(C = 50, gamma = 'scale', kernel = 'rbf')
svm_model.fit(X_train, y_train)
y_test_pred_SVM = svm_model.predict(X_test)
y_train_pred_SVM = svm_model.predict(X_train)
print(accuracy_score(y_train, y_train_pred_SVM))
print(accuracy_score(y_test, y_test_pred_SVM))
```

```
0.999875
0.9155
```

SUPPORT VECTOR MACHINE

In [48]:
```
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
print(classification_report(y_train,y_train_pred_SVM))
print(classification_report(y_test,y_test_pred_SVM))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2220
           1       1.00      1.00      1.00     11830
           2       1.00      1.00      1.00      1950

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

              precision    recall  f1-score   support

           0       1.00      0.69      0.82       557
           1       0.90      1.00      0.95      2922
           2       0.99      0.69      0.81       521

    accuracy                           0.92      4000
   macro avg       0.96      0.79      0.86      4000
weighted avg       0.92      0.92      0.91      4000
```

NAÏVE BAYES

Naive bayes

```
In [31]: from sklearn.model_selection import GridSearchCV
         from sklearn.naive_bayes import GaussianNB

         nb_classifier = GaussianNB()

         params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
         gs_NB = GridSearchCV(estimator=nb_classifier,
                          param_grid=params_NB,
                          verbose=1,
                          scoring='accuracy')
         gs_NB.fit(X_train.todense(), y_train)

         #print the tuned parameters
         print("Tuned MLPClassifier Parameters: {}".format(gs_NB.best_params_))
         print("Best score is {}".format(gs_NB.best_score_))
```

```
In [32]: from sklearn.naive_bayes import GaussianNB
         naive_model = GaussianNB()
         naive_model.fit(X_train.toarray(), y_train)
         y_test_pred_naive = naive_model.predict(X_test.toarray())
         y_train_pred_naive = naive_model.predict(X_train.toarray())
         print(accuracy_score(y_train, y_train_pred_naive))
         print(accuracy_score(y_test, y_test_pred_naive))
```

```
0.9996875
0.728
```

NAÏVE BAYES

```
In [47]: from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
         print(classification_report(y_train,y_train_pred_naive))
         print(classification_report(y_test,y_test_pred_naive))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2220
           1       1.00      1.00      1.00     11830
           2       1.00      1.00      1.00      1950

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

              precision    recall  f1-score   support

           0       0.34      1.00      0.51       557
           1       1.00      0.69      0.81      2922
           2       1.00      0.67      0.80       521

    accuracy                           0.73      4000
   macro avg       0.78      0.79      0.71      4000
weighted avg       0.91      0.73      0.77      4000
```

MULTILAYER PERCEPTRON CLASSIFIER

MLPClassifier

```
In [35]: from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(max_iter=100)
parameter_space = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
clf.fit(X_train, y_train)

#print the tuned parameters
print("Tuned MLPClassifier Parameters: {}".format(clf.best_params_))
print("Best score is {}".format(clf.best_score_))
```

```
Tuned MLPClassifier Parameters: hidden_layer_sizes=(6,5), random_state=5, verbose=True, learning_rate='adaptive', act
ivation='relu', solver='adam', alpha=0.05
```

MULTILAYER PERCEPTRON CLASSIFIER

```
In [30]: # Create model object
         from sklearn.neural_network import MLPClassifier
         clf = MLPClassifier(hidden_layer_sizes=(6,5),
                             random_state=5,
                             verbose=True,
                             learning_rate='adaptive',
                             activation='relu',
                             solver='adam',
                             alpha=0.05)

         # Fit data onto the model
         clf.fit(X_train,y_train)
         y_test_pred_MLP = clf.predict(X_test)
         y_train_pred_MLP = clf.predict(X_train)
         print(accuracy_score(y_train, y_train_pred_MLP))
         print(accuracy_score(y_test, y_test_pred_MLP))
```

```
Iteration 113, loss = 0.04361594
Iteration 114, loss = 0.04358863
Iteration 115, loss = 0.04354080
Iteration 116, loss = 0.04348845
Iteration 117, loss = 0.04338801
Iteration 118, loss = 0.04335436
Iteration 119, loss = 0.04329096
Iteration 120, loss = 0.04338493
Iteration 121, loss = 0.04339499
Iteration 122, loss = 0.04326376
Iteration 123, loss = 0.04318888
Iteration 124, loss = 0.04313876
Iteration 125, loss = 0.04314115
Iteration 126, loss = 0.04314795
Iteration 127, loss = 0.04314126
Iteration 128, loss = 0.04321860
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
0.99975
0.91325
```

MULTILAYER PERCEPTRON CLASSIFIER

```
In [46]: from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
         print(classification_report(y_train,y_train_pred_MLP))
         print(classification_report(y_test,y_test_pred_MLP))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2220
           1       1.00      1.00      1.00     11830
           2       1.00      1.00      1.00      1950

    accuracy                           1.00     16000
   macro avg       1.00      1.00      1.00     16000
weighted avg       1.00      1.00      1.00     16000

              precision    recall  f1-score   support

           0       0.99      0.69      0.81       557
           1       0.89      1.00      0.94      2922
           2       1.00      0.66      0.80       521

    accuracy                           0.91      4000
   macro avg       0.96      0.79      0.85      4000
weighted avg       0.92      0.91      0.91      4000
```

DECISION TREE

1.Decision Tree

In [27]:

```
from scipy.stats import randint
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

# Setup the parameters and distributions to sample from: param_dist
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}

# Instantiate a Decision Tree classifier: tree
tree = DecisionTreeClassifier()

# Instantiate the RandomizedSearchCV object: tree_cv
tree_cv = RandomizedSearchCV(tree, param_dist, cv=5)

# Fit it to the data
tree_cv.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
```

```
Tuned Decision Tree Parameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': 7, 'min_samples_leaf': 4}
Best score is 0.7394375
```

DECISION TREE

```
In [28]: from sklearn.metrics import accuracy_score
         dect_model = DecisionTreeClassifier(criterion = 'gini', max_depth = 3, max_features = 1, min_samples_leaf = 2)
         dect_model.fit(X_train, y_train)
         y_test_pred_dect = dect_model.predict(X_test)
         y_train_pred_dect = dect_model.predict(X_train)
         print(accuracy_score(y_train, y_train_pred_dect))
         print(accuracy_score(y_test, y_test_pred_dect))

0.739375
0.7305
```

DECISION TREE

```
In [45]: from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
         print(classification_report(y_train,y_train_pred_dect))
         print(classification_report(y_test,y_test_pred_dect))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2220
           1       0.74      1.00      0.85     11830
           2       0.00      0.00      0.00      1950

    accuracy                           0.74     16000
   macro avg       0.25      0.33      0.28     16000
weighted avg       0.55      0.74      0.63     16000

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       557
           1       0.73      1.00      0.84      2922
           2       0.00      0.00      0.00       521

    accuracy                           0.73      4000
   macro avg       0.24      0.33      0.28      4000
weighted avg       0.53      0.73      0.62      4000
```

PERFORMANCE ANALYSIS

Displaying all test and train accuracies

In [44]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["S.no", "Model", "Train_accuracy", "Test_accuracy"]
x.add_row(['1', 'Naive Bayes', accuracy_score(y_train, y_train_pred_naive), accuracy_score(y_test, y_test_pred_naive)])
x.add_row(['2', 'SVM', accuracy_score(y_train, y_train_pred_SVM), accuracy_score(y_test, y_test_pred_SVM)])
x.add_row(['3', 'MLP', accuracy_score(y_train, y_train_pred_MLP), accuracy_score(y_test, y_test_pred_MLP)])
x.add_row(['4', 'Decision Tree', accuracy_score(y_train, y_train_pred_dect), accuracy_score(y_test, y_test_pred_dect)])
print(x)
```

S.no	Model	Train_accuracy	Test_accuracy
1	Naive Bayes	0.9996875	0.728
2	SVM	0.999875	0.9155
3	MLP	0.99975	0.91325
4	Decision Tree	0.739375	0.7305

AFTER DEPLOYMENT

M*j$f0ipl#4s87

Nice password!

- Your password is hack-resistant.
- Your password does not appear in any databases of leaked passwords

REFERENCES

[1] F Bergadano, B.Crispo, G.Ruffo, "Proactive password checking with decision trees",Proc. of the 4th ACM conference on computer and communications security, Zurich, Switzerland, 1997, pp 67-77

[2] Giancarlo Ruffo, Francesco Bergadano, "EnFilter :A Password Enforcement and Filter Tool Based on Pattern Recognition Techniques", Springer Berlin /Heidelberg, 1611-3349 (Online),Volume 3617/2005

[3] Dell'amico, Matteo, Michiardi, Pietro and Roudier, Measuring Password Strength: An Empirical Analysis., Yves. Jul 20, 2009.

[4] Forget,Alain, Improving text passwords through persuasion, et al. s.l. :ACM, 2008. pp. 1--12.

[5] Narayanan, Arvind and Shmatikov, Vitaly. Fast dictionary attacks on passwords using time-space tradeoff. s.l. :ACM, 2005. pp. 364--372.

[6] Ur, P.G. Kelley, S. Komanduri, M. M. J.Lee, M. Mazurek, T. Passaro, R. Shay, T. Vidas and L. Bauer, "How does your password measure up? the effect of strength meters on password creation," in Proc. USENIX SEC 2012, 2012

REFERENCES

[7] Wang, Z. Zhang, J.Y. P. Wang and X. Huang, "Targeted online password guessing: an underestimated threat," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security,Vienna Austria, 2016.

[8] Florencio, C. Herley and P.C. V. Oorschot, "Password portfolios and the finite-effort user: Sustainably managing large numbers of accounts," in Proc. USENIX SEC 2014, 2014.

[9] Ma, W. Yang, M. Luo and N. L, "A study of probabilistic password models," in 2014 IEEE Symposium on Security and privacy, San Jose, CA, USA, 2014.

[10] L. Gong-Shen, Q. Wei-Dong and L. Jian-Hua, "Password vulnerability assessment and recovery based on rules mined from large-scale real data," Chinese Journal of Computers, vol. 39, no. 3, p. 454–467, 2016.

[11] D.Wang and P.Wang, "The emperor’s new password creation policies," in in Proc. ESORICS 2015, 2015.

[12] M. Weir, S. Aggarwal, M. Collins and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," in Proc.ACM CCS 2010, 2010.

THANK YOU !!