CS3802--Machine Learning Algorithms Lab

Adithya V | BTech CSE (IoT) - A | 21011102009

Exercise 6

Use the teleco-customer-churn dataset for the following:

1. Use the attached file and run SVM, Decision tree, Random Forest and any one boosting algorithm.

2. Find out the different tunable parameters for each algorithms mentioned above.

3. Apply gridsearchCV and randomizedsearchCV for all the above classification algorithms and get the best parameters.

Importing necessary libraries and reading the dataset

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

```
c:\Python311\Lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\Python311\Lib\site-packages\numpy\.libs\libopenblas64__v0.3.21-gcc_10_3_0.dll
c:\Python311\Lib\site-packages\numpy\.libs\libopenblas64__v0.3.23-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [ ]:

```
df = pd.read_csv("Telco-Customer-Churn.csv")
df = df.drop('customerID', axis = 1)
df.head()
```

Out[ ]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   object 
 19  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
```

In [ ]:

```
df = df[df["TotalCharges"] != " "]
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'])
```

Data Pre-Processing

In [ ]:

```
import pandas as pd

# Load your DataFrame `df` here

# Target Transformation
df['Churn'] = df['Churn'].map({"No": 0, "Yes": 1})
```

```
# Min-Max Scaling for 'TotalCharges'
df = df[df["TotalCharges"] != " "]
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'])
df['TotalCharges'] = (df['TotalCharges'] - df['TotalCharges'].min()) / (df['TotalCharges'].max() - df['TotalCharges'].min())

# Min-Max Scaling for 'MonthlyCharges'
df = df[df["MonthlyCharges"] != " "]
df['MonthlyCharges'] = pd.to_numeric(df['MonthlyCharges'])
df['MonthlyCharges'] = (df['MonthlyCharges'] - df['MonthlyCharges'].min()) / (df['MonthlyCharges'].max() - df['MonthlyCharges'

# Min-Max Scaling for 'tenure'
df = df[df["tenure"] != " "]
df['tenure'] = pd.to_numeric(df['tenure'])
df['tenure'] = (df['tenure'] - df['tenure'].min()) / (df['tenure'].max() - df['tenure'].min())

# One-Hot Encoding for categorical columns
df = pd.get_dummies(df)

data = df
data.head()
```

Out[ ]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	...	Stream
0	0	0.000000	0.115423	0.001275	0	1	0	0	1	1	...	
1	0	0.464789	0.385075	0.215867	0	0	1	1	0	1	...	
2	0	0.014085	0.354229	0.010310	1	0	1	1	0	1	...	
3	0	0.619718	0.239303	0.210241	0	0	1	1	0	1	...	
4	0	0.014085	0.521891	0.015330	1	1	0	1	0	1	...	

5 rows × 46 columns

Model Training

In [ ]:

```
x_columns = data.columns.drop('Churn').tolist()
x = data[x_columns]
y = data['Churn']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42, shuffle=True)
```

SVM

In [ ]:

```
svcModel = SVC()
svcModel.fit(x_train,y_train)
accuracy = svcModel.score(x_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7900473933649289
```

Decision Tree

In [ ]:

```
dtModel = DecisionTreeClassifier()
dtModel.fit(x_train, y_train)
accuracy_dt = dtModel.score(x_test, y_test)
print("Decision Tree Accuracy:", accuracy_dt)
```

```
Decision Tree Accuracy: 0.7327014218009479
```

Random Forest

In [ ]:

```
rfModel = RandomForestClassifier()
rfModel.fit(x_train, y_train)
accuracy_rf = rfModel.score(x_test, y_test)
print("Random Forest Accuracy:", accuracy_rf)
```

```
Random Forest Accuracy: 0.7777251184834123
```

Adaboost

In [ ]:

```
adaModel = AdaBoostClassifier()
adaModel.fit(x_train, y_train)
accuracy_ada = adaModel.score(x_test, y_test)
print("AdaBoost Accuracy:", accuracy_ada)
```

```
AdaBoost Accuracy: 0.7909952606635071
```

HyperParameter Tuning

Random Forest Classifier

GridSearchCV

In [ ]:

```
param_grid_rf = {'n_estimators': [100, 200, 300], 'criterion': ['gini', 'entropy'],
                  'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10],
                  'min_samples_leaf': [1, 2, 4]}
grid_search_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5)
grid_search_rf.fit(x_train, y_train)
best_params_rf = grid_search_rf.best_params_
print("Best Parameters for Random Forest:", best_params_rf)
```

```
Best Parameters for Random Forest: {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 10,
'n_estimators': 300}
```

RandomizedSearch CV

In [ ]:

```
param_dist_ada = {'n_estimators': [50, 100, 150], 'learning_rate': [0.01, 0.1, 1.0]}
random_search_ada = RandomizedSearchCV(AdaBoostClassifier(), param_dist_ada, cv=5, n_iter=10)
random_search_ada.fit(x_train, y_train)
best_params_rand_ada = random_search_ada.best_params_
print("Best Parameters for AdaBoost (RandomizedSearchCV):", best_params_rand_ada)
```

```
c:\Python311\Lib\site-packages\sklearn\model_selection\_search.py:307: UserWarning: The total space of parameters 9 is smalle
r than n_iter=10. Running 9 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
Best Parameters for AdaBoost (RandomizedSearchCV): {'n_estimators': 150, 'learning_rate': 0.1}
```

Decision Tree classifier

GridSearch CV

In [ ]:

```
param_grid_dt = {'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'],
                 'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10],
                 'min_samples_leaf': [1, 2, 4]}
grid_search_dt = GridSearchCV(DecisionTreeClassifier(), param_grid_dt, cv=5)
grid_search_dt.fit(x_train, y_train)
best_params_dt = grid_search_dt.best_params_
print("Best Parameters for Decision Tree:", best_params_dt)
```

```
Best Parameters for Decision Tree: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'spl
itter': 'random'}
```

Randomized Search CV

In [ ]:

```
param_dist_dt = {'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'],
                 'max_depth': [None, 10, 20], 'min_samples_split': [2, 5, 10],
                   'min_samples_leaf': [1, 2, 4]}
random_search_dt = RandomizedSearchCV(DecisionTreeClassifier(), param_dist_dt, cv=5, n_iter=10)
random_search_dt.fit(x_train, y_train)
best_params_rand_dt = random_search_dt.best_params_
print("Best Parameters for Decision Tree (RandomizedSearchCV):", best_params_rand_dt)
```

```
Best Parameters for Decision Tree (RandomizedSearchCV): {'splitter': 'random', 'min_samples_split': 5, 'min_samples_leaf': 4,
'max_depth': 10, 'criterion': 'gini'}
```

SVM Classifier

GridSearch CV

In [ ]:

```
param_grid_svc = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']}
grid_search_svc = GridSearchCV(SVC(), param_grid_svc, cv=5)
grid_search_svc.fit(x_train, y_train)
```

```
best_params_svc = grid_search_svc.best_params_
print("Best Parameters for SVC:", best_params_svc)
```

```
Best Parameters for SVC: {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
```

Randomized Search CV

In [ ]:

```
from scipy.stats import uniform
param_dist_svc = {'C': uniform(loc=0, scale=10), 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']}
random_search_svc = RandomizedSearchCV(SVC(), param_dist_svc, cv=5, n_iter=10)
random_search_svc.fit(x_train, y_train)
best_params_rand_svc = random_search_svc.best_params_
print("Best Parameters for SVC (RandomizedSearchCV):", best_params_rand_svc)
```

```
Best Parameters for SVC (RandomizedSearchCV): {'C': 4.075240301647072, 'gamma': 'auto', 'kernel': 'rbf'}
```