Student Name:_____Roll No:_____Section:_____

CS334 - Machine Learning

Lab 05 - Feature Selection Methods in ML (Part - 2)

Instructor: Ms. Maham Ashraf
E-mail: mashraf@uit.edu

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce feature selection methods for machine learning model. This lab is divided into two parts, in Part-1 we have used Filter methods for extracting important features from data sets. In Part-2, we will use Wrapper, Embedded and Hybrid methods for feature selections.

Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

How to Submit

- Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW01.py)
- Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
- Home work file name should be saved with your roll number (e.g. 19a-001-SE_HW01.py)

1 Wrapper Methods: for Features Selection

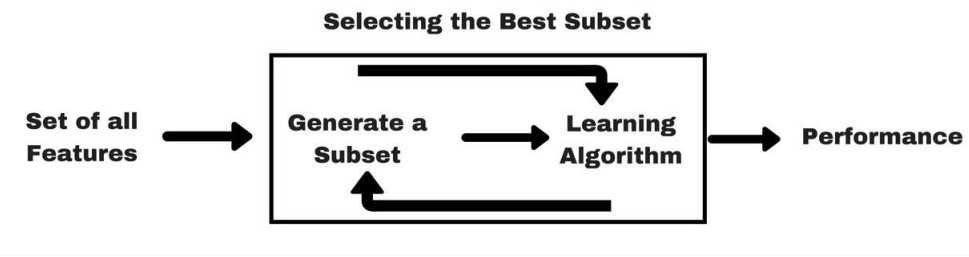

Figure 1: Wrapper Method

Wrappers require some method to search the space of all possible subsets of features, assessing their quality by learning and evaluating a classifier with that feature subset. The feature selection

process is based on a specific machine learning algorithm that we are trying to fit on a given dataset. It follows a greedy search approach by evaluating all the possible combinations of features against the evaluation criterion. The wrapper methods usually result in better predictive accuracy than filter methods.

The scikit-learn library also provides many different filtering methods once statistics have been calculated for each input variable with the target.
Two of the more popular methods include:

- Select the top k variables: *SelectKBest*
- Select the top percentile variables: *SelectPercentile*

1.1 Forward Feature Selection:

Forward elimination starts with no features, and the insertion of features into the regression model one-by-one. First, the regressor with the highest correlation is selected for inclusion, which coincidentally the regressor that produces the largest F-statistic value when testing the significance of the model. This is an iterative method wherein we start with the best performing variable against the target. Next, we select another variable that gives the best performance in combination with the first selected variable. This process continues until the preset criterion is achieved.

```
# 1. load Iris data
# 2. split data into descriptive and target features in X and y variables
    respectively.
# 3. load important libraries
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
# 4. Apply Model
lr=LogisticRegression(class_weight='balanced', solver='lbfgs', random_state
    =42, n_jobs=-1, max_iter=500)
lr.fit(X,Y)
# 5. Select best features
bfs = SFS(lr,
            k_features='best',
            forward = True ,
            floating = False ,
            verbose =2 ,
            scoring='accuracy',
            cv=0)
bfs.fit(X,Y)
# 6. print feature list
features = list(bfs.k_feature_names_)
print(features)
```

Listing 1: Forward Feature Selection

1.2 Backward Feature Elimination:

This method works exactly opposite to the Forward Feature Selection method. Here, we start with all the features available and build a model. Next, we the variable from the model which gives the best evaluation measure value. This process is continued until the preset criterion is achieved.

This method along with the one discussed above is also known as the Sequential Feature Selection method. We use `mlxtend.feature_selection` library and select `SequentialFeatureSelector` class for backward elimination method. what we need to change in Listing-1 of forward feature selection code is to make **"forward"** filed **"False"** there.

1.3 Exhaustive Feature Selection:

This exhaustive feature selection algorithm is a wrapper approach for brute-force evaluation of feature subsets; the best subset is selected by optimizing a specified performance metric given an arbitrary regressor or classifier.

```
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
knn = KNeighborsClassifier(n_neighbors=3)
efs1 = EFS(knn,
           min_features=1,
```

```
            max_features =4 ,
            scoring='accuracy',
            print_progress = True ,
            cv=5)

efs1 = efs1.fit(X_df, y_series)
print('Best accuracy score: %.2f' % efs1.best_score_)
print('Best subset (indices):', efs1.best_idx_)
print('Best subset (corresponding names):', efs1.best_feature_names_)
```

Listing 2: Exahustive Feature Selection

1.4 Recursive Feature Elimination:

The goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import RFE
import numpy as np
from sklearn.ensemble import GradientBoostingClassifier
# initialize recursive feature elimination class
rfe = RFE(estimator=GradientBoostingClassifier(), n_features_to_select=6)
model = GradientBoostingClassifier()
# make a piline for execution
pipe = Pipeline([('Feature Selection', rfe), ('Model', model)])
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=5, random_state
  =36851234)
#compute and print score of each feature
n_scores = cross_val_score(pipe, X_train, y_train, scoring='accuracy', cv=
  cv, n_jobs=-1)
print(np.mean(n_scores))

# Execute RFE model
pipe.fit(X_train, y_train)
rfe.support_
```

Listing 3: Recursive Feature Elimination

2 C. Embedded Methods:

These methods encompass the benefits of both the wrapper and filter methods, by including interactions of features but also maintaining reasonable computational cost. Embedded methods are iterative in the sense that takes care of each iteration of the model training process and carefully extracts those features which contribute the most to the training for a particular iteration.

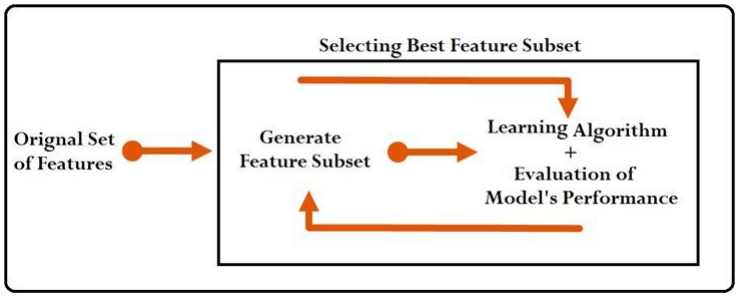

Figure 2: Embedded Method

2.1 LASSO Regularization (L1):

Regularization consists of adding a penalty to the different parameters of the machine learning model to reduce the freedom of the model, i.e. to avoid over-fitting. In linear model regularization, the penalty is applied over the coefficients that multiply each of the predictors. From the different types of regularization, Lasso or L1 has the property that is able to shrink some of the coefficients to zero. Therefore, that feature can be removed from the model.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
    random_state=42)
pipeline = Pipeline([
                      ('scaler',StandardScaler()),
                      ('model',Lasso())
])
search = GridSearchCV(pipeline,
                       {'model__alpha':np.arange(0.1,10,0.1)},
                       cv = 5, scoring="neg_mean_squared_error",verbose=3
                       )
search.fit(X_train,y_train)
coefficients = search.best_estimator_.named_steps['model'].coef_
importance = np.abs(coefficients)
print       (       importance       )
print(np.array(features)[importance > 0])
```

Listing 4: LASSO Regularization (L1)

Unlike Lasso, Ridge does not have zeroing coefficients as a goal, and you shouldn't expect applying ridge penalty to have this effect. Therefore, it is not good to use Ridge regression for feature selection.

2.2 Random forest features selection

Random Forests is a kind of a Bagging Algorithm that aggregates a specified number of decision trees. The tree-based strategies used by random forests naturally rank by how well they improve the purity of the node, or in other words a decrease in the impurity (Gini impurity) over all trees. Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features.

```
# 1. load housing data set
# 2. separate descriptive features and target features
X = df.iloc[:,1:12]
y = df . iloc[: , 13 ]
#perform train-test split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.33,
    random_state=42)
#  apply random forest regression algorithm
rf =  RandomForestRegressor(random_state=0)
rf.fit(X_train,y_train)
# observe the selected features
rf =  RandomForestRegressor(random_state=0)
rf.fit(X_train,y_train)
# print selected features
rf.feature_names_in_
```

Listing 5: Random Forest feature selection

3 Hybrid methods

The PCAs are popular with dimensionality reduction but the underlying assumptions of PCA depend on linearities. For nonlinear problems which is how real-world scenarios usually are, Genetic algorithms offer significant solutions.
A genetic algorithm (GA) is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. Some examples of GA applications include optimizing decision trees for better performance, solving sudoku puzzle, hyper-parameter optimization, etc.

3.1 Genetic Algorithm

The Genetic Algorithm (GA) selection has great benefit for applying machine learning in applications with many nuisance features. GA selection is more likely to find the best model among all possible subsets. Constraints from model restrictions, data transformations, data encoding are naturally incorporated into the algorithm. Although the time needed to find the best solution is higher than shrinkage methods, in most cases it is acceptable when compared to the improved selection and confidence in the selected features.
In machine learning, GA's have two main uses. The first is for optimization, such as finding the best weights for a neural network. The second is for supervised feature selection. In this use case, "genes" represent individual features and the "organism" represents a candidate set of features. Each organism in the "population" is graded on a fitness score such as model performance on a hold-out set. The fittest organisms survive and reproduce, repeating until the population converges on a solution some generations later.
Let's train the model and predicting the accuracy using the Genetic Algorithm in the Logistics regression technique.

```
#defining various steps required for the genetic algorithm
def initilization_of_population(size,n_feat):
    population = []
    for i in range(size):
        chromosome  = np.ones(n_feat,dtype=np.bool)
        chromosome [: int ( 0 . 3 * n_feat ) ]= False
        np . random . shuffle ( chromosome )
        population.append(chromosome)
    return population

def fitness_score(population):
    scores = []
    for        chromosome        in        population       :
        logmodel . fit ( X_train . iloc [: , chromosome ], y_train )
        predictions = logmodel.predict(X_test.iloc[:,chromosome])
        scores.append(accuracy_score(y_test,predictions))
    scores, population = np.array(scores), np.array(population)
    inds = np.argsort(scores)
    return list(scores[inds][::-1]), list(population[inds,:][::-1])

def selection(pop_after_fit,n_parents):
    population_nextgen = []
    for i in range ( n_parents ):
        population_nextgen . append ( pop_after_fit [i])
    return population_nextgen

def crossover ( pop_after_sel ):
    population_nextgen = pop_after_sel
    for i in range(len(pop_after_sel)):
        child        =       pop_after_sel        [        i]
        child[3:7]=pop_after_sel[(i+1)%len(pop_after_sel)][3:7]
        population_nextgen.append(child)
    return population_nextgen

def mutation(pop_after_cross,mutation_rate):
    population_nextgen = []
    for i in range(0,len(pop_after_cross)):
        chromosome = pop_after_cross[i]
        for j in range(len(chromosome)):
            if random.random() < mutation_rate:
                chromosome[j]= not chromosome[j]
        population_nextgen . append ( chromosome )
    #print(population_nextgen)
    return population_nextgen

def generations(size,n_feat,n_parents,mutation_rate,n_gen,X_train,
                                    X_test, y_train, y_test):
    best_chromo= []
    best_score= []
    population_nextgen=initilization_of_population ( size , n_feat )
    for i in range(n_gen):
        scores, pop_after_fit = fitness_score(population_nextgen)
        print(scores[:2])
```

```
        pop_after_sel = selection ( pop_after_fit , n_parents )
        pop_after_cross = crossover ( pop_after_sel )
        population_nextgen = mutation(pop_after_cross,mutation_rate)
        best_chromo . append ( pop_after_fit [ 0 ])
        best_score.append(scores[0])
    return best_chromo,best_score

#                          Implementing                          GA
chromo , score = generations ( size =200 , n_feat =30 , n_parents =100 , mutation_rate
    =0.10 , n_gen =5 , X_train = X_train , X_test = X_test , y_train = y_train , y_test =
    y_test )
logmodel . fit ( X_train . iloc [: , chromo [ -1]] , y_train )
predictions = logmodel.predict(X_test.iloc[:,chromo[-1]])
print (" Accuracy score after genetic algorithm is = " + str ( accuracy_score (
    y_test,predictions)))

print('list of important features', X_train.iloc[:,chromo[-1]].columns)
```

4 Homework

1. Apply *Forward Feature Selection* methods on **housing** data set and discover the selected features.
2. Apply *Backward Feature Elimination* methods on **diabetes** data set and discover the selected features.
3. Apply *Exhaustive Feature Selection* methods on **Motor Insurance Fraud** data set and discover the selected features.
4. Apply *Recursive Feature Elimination* methods on **Breast Cancer** data set and discover the selected features.
5. Apply *Lasso regression (l_1)* method for feature selection on **housing** data set and discover the selected features.
6. Apply *Random forest features selection* method with 50 random states on **diabetes** data set and discover the selected features.
7. Apply *Genetic Algorithm* on **diabetes** data set and for feature selection using following parameters.
    a. number of parents = 100
    b. mutation rate = 0.10,
    c. number of generations = 5