Student Name:________________________Roll No:________________Section:___________

CS334 - Machine Learning

Lab 07 - Ensemble Models (Part - 1)

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

Semester: Fall, 2023

Objective

The purpose of this lab session is to introduce Ensemble models that use in Machine Learning for better performance.

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)

Growing and Pruning Ensembles in Python

Ensemble member selection refers to algorithms that optimize the composition of an ensemble.

This may involve growing an ensemble from available models or pruning members from a fully defined ensemble.

The goal is often to reduce the model or computational complexity of an ensemble with little or no effect on the performance of an ensemble, and in some cases find a combination of ensemble members that results in better performance than blindly using all contributing models directly.
In this lab, ww will perform following method of growing and pruning methods'

1. Ensemble Member Selection
2. Baseline Models and Voting
3. Ensemble Pruning Example
4. Ensemble Growing Example

1 Ensemble Member Selection

Voting and stacking ensembles typically combine the predictions from a heterogeneous group of model types.

Although the ensemble may have a large number of ensemble members, it is hard to know that the best combination of members is being used by the ensemble. For example, instead of simply using all members, it is possible that better results could be achieved by adding one more different model type or removing one or more models.

This can be addressed using a weighted average ensemble and using an optimization algorithm to find an appropriate weighting for each member, allowing some members to have a zero weight, which effectively removes them from the ensemble. The problem with a weighted average ensemble is that all models remain part of the ensemble, perhaps requiring an ensemble of greater complexity than is required to be developed and maintained.

An alternative approach is to optimize the composition of the ensemble itself. The general approach of automatically choosing or optimizing the members of ensembles is referred to as ensemble selection.

Two common approaches include ensemble growing and ensemble pruning.

- **Ensemble Growing:** Add members to the ensemble until no further improvement is observed.
- **Ensemble Pruning:** Remove members from the ensemble until no further improvement is observed.

Ensemble growing is a technique where the model starts with no members and involves adding new members until no further improvement is observed. This could be performed in a greedy manner where members are added one at a time only if they result in an improvement in model performance.

Ensemble pruning is a technique where the model starts with all possible members that are being considered and removes members from the ensemble until no further improvement is observed. This could be performed in a greedy manner where members are removed one at a time and only if their removal results in a lift in the performance of the overall ensemble.

2 Baseline Models and Voting

We will use five standard machine learning models, including logistic regression, naive Bayes, decision tree, support vector machine, and a k-nearest neighbor algorithm.

We can then define a function that takes a single model and the dataset and evaluates the performance of the model on the dataset. We will evaluate a model using repeated stratified k-fold cross-validation with 10 folds and three repeats, a good practice in machine learning.

```
# evaluate standard models on the synthetic dataset
from numpy import mean
from numpy import std
from sklearn . datasets  import  make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors  import KNeighborsClassifier
from sklearn . tree import Decision Tree Classifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot

# get the dataset
def get_dataset():
    X, y = make_classification(n_samples=5000, n_features=20, n_informative
   =10, n_redundant=10, random_state=1)
    return X, y
```

```
# get a list of models to evaluate
def get_models():
  models = list()
  models.append(('lr', LogisticRegression()))
  models.append(('knn', KNeighborsClassifier()))
  models.append(('tree', DecisionTreeClassifier()))
  models.append(('nb', GaussianNB()))
  models.append(('svm', SVC(probability=True)))
  return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
  # define the model evaluation procedure
  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
  # evaluate the model
  scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs
 =-1)
  return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models:
  # evaluate model
  scores = evaluate_model(model, X, y)
  # store results
  results.append(scores)
  names.append(name)
  # summarize result
  print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

Running the example evaluates each standalone machine learning algorithm on the synthetic binary classification dataset.
Now observe the result.
We can see that the KNN and SVM algorithms perform much better than the other algorithms, although all algorithms are skillful in different ways. This may make them good candidates to consider in an ensemble.

Next, we need to establish a baseline ensemble that uses all models. This will provide a point of comparison with growing and pruning methods that seek better performance with a smaller subset of models.
In this case, we will use a voting ensemble with soft voting. This means that each model will predict probabilities and the probabilities will be summed by the ensemble model to choose a final output prediction for each input sample.
This can be achieved using the VotingClassifier class where the members are set via the “estimators” argument, which expects a list of models where each model is a tuple with a name and configured model object, just as we defined in the previous section.
We can then set the type of voting to perform via the “voting” argument, which in this case is set to “soft.”

Now try to add following code into previous code after `models = get_models()` line,

```
# create the ensemble
ensemble = VotingClassifier(estimators=models, voting='soft')
# define the evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the ensemble
scores = cross_val_score(ensemble, X, y, scoring='accuracy', cv=cv, n_jobs
    =-1)
# summarize the result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Running the example evaluates the soft voting ensemble of all models using repeated stratified k-fold

cross-validation and reports the mean accuracy across all folds and repeats.
In this case, we can see that the voting ensemble achieved a mean accuracy of about 92.8 percent. This is lower than SVM and KNN models used alone that achieved an accuracy of about 95.3 percent.

This result highlights that a simple voting ensemble of all models results in a model with higher complexity and worse performance in this case. Perhaps we can find a subset of members that performs better than any single model and has lower complexity than simply using all models.
Next, we will explore pruning members from the ensemble.

3 Ensemble Pruning Example

We will use a greedy algorithm in this case, which is straightforward to implement. This involves removing one member from the ensemble and evaluating the performance and repeating this for each member in the ensemble. The member that, if removed, results in the best improvement in performance is then permanently removed from the ensemble and the process repeats. This continues until no further improvements can be achieved.

First, we can define a function to evaluate a candidate list of models. This function will take the list of models and the dataset and construct a voting ensemble from the list of models and evaluate its performance using repeated stratified k-fold cross-validation, returning the mean classification accuracy.

Next, we can define a function that performs a single round of pruning.

First, a baseline in performance is established with all models that are currently in the ensemble. Then the list of models is enumerated and each is removed in turn, and the effect of removing the model is evaluated on the ensemble. If the removal results in an improvement in performance, the new score and specific model that was removed is recorded.

Importantly, the trial removal is performed on a copy of the list of models, not on the main list of models itself. This is to ensure we only remove an ensemble member from the list once we know it will result in the best possible improvement from all the members that could potentially be removed at the current step.

Next, we need to drive the pruning process.

This involves running a round of pruning until no further improvement in accuracy is achieved by calling the prune_round() function repeatedly.

If the function returns None for the model to be removed, we know that no single greedy improvement is possible and we can return the final list of models. Otherwise, the chosen model is removed from the ensemble and the process continues.

```
# evaluate a list of models
def evaluate_ensemble(models, X, y):
  # check for no models
  if len(models) == 0:
    return 0.0
  # create the ensemble
  ensemble = VotingClassifier(estimators=models, voting='soft')
  # define the evaluation procedure
  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
  # evaluate the ensemble
  scores = cross_val_score(ensemble, X, y, scoring='accuracy', cv=cv, n_jobs
   =-1)
  # return mean score
  return mean(scores)

# perform a single round of pruning the ensemble
def prune_round(models_in, X, y):
  # establish a baseline
  baseline = evaluate_ensemble(models_in, X, y)
  best_score, removed = baseline, None
  # enumerate removing each candidate and see if we can improve performance
  for m in models_in:
    # copy the list of  chosen models
    dup  =  models_in.copy()
    #  remove  this  model
    dup.remove(m)
    #  evaluate  new  ensemble
    result =  evaluate_ensemble ( dup , X , y)
    #  check  for  new  best
    if  result  >  best_score:
      #  store  the  new  best
      best_score , removed = result , m
  return  best_score,  removed

# prune an ensemble from  scratch
def prune_ensemble(models,  X,  y):
  best_score  =  0.0
  #  prune  ensemble  until  no  further  improvement
  while  True:
    #  remove  one  model  to  the  ensemble
    score,  removed  =  prune_round(models,  X,  y)
    #  check  for  no  improvement
    if  removed  is  None:
      print('>no  further  improvement')
      break
    #  keep  track  of  best  score
    best_score = score
    #  remove  model  from  the  list
    models.remove(removed)
    #  report  results  along  the  way
    print('>%.3f  (removed:  %s)'  %  (score,  removed[0]))
  return  best_score,  models

#  define  dataset
X,  y  =  get_dataset()
# get the models to evaluate
models  =  get_models()
#  prune  the  ensemble
score,  model_list  =  prune_ensemble(models,  X,  y)
names = ','. join ([ n for n , _ in model_list ])
print('Models:  %s'  %  names)
print('Final  Mean  Accuracy:  %.3f'  %  score)
```

Running the example performs the ensemble pruning process, reporting which model was removed each round and the accuracy of the model once the model was removed.

In this case, we can see that three rounds of pruning were performed, removing the naive Bayes, decision tree, and logistic regression algorithms, leaving only the SVM and KNN algorithms that achieved a mean classification accuracy of about 95.7 percent. This is better than the 95.3 percent achieved by SVM and KNN used in a standalone manner, and clearly better than combining all models together.

Now that we are familiar with developing and evaluating an ensemble pruning method, let's look at the reverse case of growing the ensemble members.

4 Ensemble Growing Example

In this section, we will explore how to develop a greedy ensemble growing algorithm from scratch.

The structure of greedily growing an ensemble is much like the greedy pruning of members, although in reverse. We start with an ensemble with no models and add a single model that has the best performance. Models are then added one by one only if they result in a lift in performance over the ensemble before the model was added.

Much of the code is the same as the pruning case so we can focus on the differences.

First, we must define a function to perform one round of growing the ensemble. This involves enumerating all candidate models that could be added and evaluating the effect of adding each in turn to the ensemble. The single model that results in the biggest improvement is then returned by the function, along with its score.

The grow_round() function below implements this behavior.

```
# perform a single round of growing the ensemble
def grow_round(models_in, models_candidate, X, y):
  # establish a baseline
  baseline = evaluate_ensemble(models_in, X, y)
  best_score, addition = baseline, None
  # enumerate adding each candidate and see if we can improve performance
  for m in models_candidate:
    # copy the list of chosen models
    dup = models_in.copy()
    # add the candidate
    dup.append(m)
    # evaluate new ensemble
    result = evaluate_ensemble(dup, X, y)
    # check for new best
    if result > best_score:
      # store the new best
      best_score, addition = result, m
  return best_score, addition
```

Next, we need a function to drive the growing procedure.

This involves a loop that runs rounds of growing until no further additions can be made resulting in an improvement in model performance. For each addition that can be made, the main list of models in the ensemble is updated and the list of models currently in the ensemble is reported along with the performance.

The grow_ensemble() function implements this and returns the list of models greedily determined to result in the best performance along with the expected mean accuracy.

```
# grow an ensemble from scratch
def grow_ensemble(models, X, y):
  best_score, best_list = 0.0, list()
  # grow ensemble until no further improvement
  while True:
    # add one model to the ensemble
    score, addition = grow_round(best_list, models, X, y)
    # check for no improvement
    if addition is None:
      print('>no further improvement')
      break
    # keep track of best score
    best_score = score
    # remove new model from the list of candidates
    models.remove(addition)
    # add new model to the list of models in the ensemble
    best_list.append(addition)
    # report results along the way
    names = ','.join([n for n,_ in best_list])
    print('>%.3f (%s)' % (score, names))
  return best_score, best_list
```

Running the example incrementally adds one model at a time to the ensemble and reports the mean classification accuracy of the ensemble of the models.

In this case, we can see that ensemble growing found the same solution as greedy ensemble pruning where an ensemble of SVM and KNN achieved a mean classification accuracy of about 95.6 percent, an improvement over any single standalone model and over combining all models.

5 Lab Tasks

1. Execute tasks of Section, 2, 3, and 4 on *make_classification* dataset of sklearn library.
2. Train and evaluate models as discussed in Section 2, 3, and 4 on *Diabetes* data set select the bestmodel based on evaluation scores.

6 Homework - Project Assignment - 1

1. Download SEER data file from MS Team.
2. Download Data Dictionary of SEER data set from MS Team
3. Load Data into datafram
4. Implment ensemble method using following ML alogorithms,
   a. LogisticRegression
   b. KNeighborsClassifier
   c. DecisionTreeClassifier
   d. GaussianNB
   e. Support Vector Machine (SVM)

   your target variable should be **Behavior recode for analysis** (`https://seer.cancer.gov/behavrecode/`)
5. Compare the scores and select best model
6. Provide the evidence why your selected model is best from other model