Weekly Report 2 - Random Forest

Ganji Varshitha
AI20BTECH11009

Introduction

Random Forest is an ensemble classifier which combines multiple classifiers to achieve better accuracy. It trains several models using bootstrapped dataset and selects the majority vote for classification problems and average for regression problems.

Algorithm

Algorithm 1 Random Forest Algorithm

```
Given a training set S
for i = 1 to k do
    Build subset S_i by sampling with replacement from S
    Learn tree T_i from S_i
    for each node do
        Choose best split from random subset of F features
        Each tree grows to the largest extent, and no pruning
    end for
end for
Make predictions according to majority vote of the set of k trees.
```

The value of F needs to be constant during the algorithm and it should be very less compared to total number of features M.
Possible values of F are $\frac{1}{2}\sqrt{M}, \sqrt{M}, 2\sqrt{M}$.

Why does bagging work?

Decision trees are prone to overfit which results in high variance of the model. Bagging reduces the variance of the model.
Let S be the training dataset.
Let S_k be a sequence of training sets containing a sub-set of S.
Let P be the underlying distribution of S.
Bagging replaces the prediction of the model with the majority of the predictions given by the classifiers S.

$$\phi(x, P) = \mathbb{E}_s(\phi(x, S_k))) \tag{1}$$

```python

class Random_Forest():
  def __init__(self,n_trees, bootstrap_samples , n_features):
    '''Constructor for maximum depth'''
    self.n_trees = n_trees
    self.bootstrap_samples = bootstrap_samples
    self.n_features = n_features

  # Function to get a subset of data with replacement
  def Subset(self,data):
    indices = np.random.choice(data.shape[0], size=self.bootstrap_samples ,replace = True)
    train_data = data[indices]
    OOB_data = np.delete(data,indices,0)
    return train_data,OOB_data

  def get_label(self,data):
    label_column = data[:, -1]
    unique_classes, counts_unique_classes = np.unique(label_column, return_counts=True)

    index = counts_unique_classes.argmax()
    classification = unique_classes[index]

    return classification

  def test_error(self,y_true, y_pred):
    misclassified = 0

    for i in range(len(y_true)):
      if y_pred[i] != y_true[i]:
        misclassified += 1
    return misclassified / len(y_true)

  def learn_trees(self,data):
    # Creating an empty list for storing trees
    trees_list = []
    Decision_tree = Decision(self.n_features)
    for i in range(self.n_trees):
      train,OOB_data = self.Subset(data)
      tree_learnt = Decision_tree.learn(train,{},0)
      trees_list.append(tree_learnt)
    return trees_list

  def OOB_score(self,data):
    OOB_error= []
    Decision_tree = Decision(self.n_features)
    for i in range(self.n_trees):
      train,OOB_data = self.Subset(data)
      tree_learnt = Decision_tree.learn(train,{},0)

      Y_oob = Decision_tree.predict_test(OOB_data,tree_learnt)
      OOB_error.append(self.test_error(OOB_data[:,-1],Y_oob))
```

```python
    OOB_score = np.mean(OOB_error)
    return OOB_score


   # Bagging - most important part
  def predict(self,test,trees_list):
    Decision_tree = Decision(self.n_features)
    len_samples = len(test)
    Preds_all = np.empty((len(trees_list),len_samples))
    Preds = []
    for i in range(len(trees_list)):
      predict = Decision_tree.predict_test(test,trees_list[i])
      Preds_all[i]=predict



    for p in range(len_samples):
      list1 = list(Preds_all[:,p])

      Preds.append(max(set(list1), key=list1.count))


    return Preds
```

Listing 1: Logistic Regression Code