<u>Report</u>

Task 3 & Task 4

Task 3

Parameter analysis (only for digits dataset):

1. Perceptron

max_iter	eta0	Accuracy for training data	Accuracy for testing data
50	0.1	0.979	0.933
50	0.001	0.967	0.926
50	0.00001	0.967	0.926
10	0.1	0.975	0.931
50	0.1	0.979	0.933
100	0.1	0.979	0.933

Based on the different set of max_iter and eta0, the accuracy doesn't vary too much. Therefore, max_iter and eta0 are set as 50 and 0.1.

2. svm_linear

C1	Accuracy for training data	Accuracy for testing data
1	1	0.978
5	1	0.978
10	1	0.978

Three different C1 values lead to the same accuracy and thus C1 is set as 1.

3. svm_nonlinear

C2	gamma	Accuracy for training data	Accuracy for testing data
1	0.1	1	0.948
10	0.1	1	0.948
20	0.1	1	0.948
10	0.1	1	0.948
10	0.2	1	0.813
10	0.5	1	0.369

The magnitude of C2 doesn't influence the accuracy too much. However, the accuracy of testing decreases as the increase of gamma. Therefore, C2 and gamma are set as 1 and 0.1.

4. tree_model

max_depth	Accuracy for training data	Accuracy for testing data
10	0.982	0.841
50	1	0.857
100	1	0.857

Three values of max_depth are selected and the highest accuracy is achieved with the max_dpeth of 50.

5. knn

n_neighbor	Accuracy for training data	Accuracy for testing data
2	0.983	0.985
5	0.983	0.985
10	0.983	0.985

Three numbers of n_neighbor are selected and the same accuracy can be obtained. Thus 2 is set in the classifier.

Run result of data2:

```
#############  Result of Perceptron classifier #############
The running time of Perceptron classifier is 1.88176 s

Training data:
Misclassified samples: 196
Accuracy for training data: 0.945

Testing data:
Misclassified samples: 3589
Accuracy for testing data: 0.568
##########################  End ############################
```

```
#############  Result of svm_linear classifier #############
The running time of svm_linear classifier is 6.22267 s

Training data:
Misclassified samples: 2
Accuracy for training data: 0.999

Testing data:
Misclassified samples: 3613
Accuracy for testing data: 0.565
##########################  End ############################
```

```
#############  Result of svm_nonlinear classifier #############
The running time of svm_nonlinear classifier is 47.43926 s

Training data:
Misclassified samples: 0
Accuracy for training data: 1.000

Testing data:
Misclassified samples: 2508
Accuracy for testing data: 0.698
##########################  End ############################
```

```
#############  Result of tree_model classifier ##############
The running time of tree_model classifier is 2.07972 s

Training data:
Misclassified samples: 152
Accuracy for training data: 0.958

Testing data:
Misclassified samples: 5440
Accuracy for testing data: 0.345
##########################  End #############################
```

```
#############  Result of knn classifier ##############
The running time of knn classifier is 16.51690 s

Training data:
Misclassified samples: 150
Accuracy for training data: 0.958

Testing data:
Misclassified samples: 2050
Accuracy for testing data: 0.753
##########################  End #############################
```

Task 4

Strategies:

1. **Pre-prune:**

Provide parameters such as min-samples_leaf, min_impurity_split, max_leaf_nodes, max_depth, etc. to prevent a tree from overfitting.

2. **Post-prune**:

Compute the pruning path during Minimal Cost-Complexity Pruning. Complexity parameter (ccp_alpha) is used for Minimal Cost-Complexity Pruning.

3. The lines of code for **pre-prune**: lines: [205-319]

The code of min-sample_leaf and min_sample_split are attached below.

```
205         if isinstance(self.min_samples_leaf, numbers.Integral):
206             if not 1 <= self.min_samples_leaf:
207                 raise ValueError("min_samples_leaf must be at least 1 "
208                                  "or in (0, 0.5], got %s"
209                                  % self.min_samples_leaf)
210             min_samples_leaf = self.min_samples_leaf
211         else:  # float
212             if not 0. < self.min_samples_leaf <= 0.5:
213                 raise ValueError("min_samples_leaf must be at least 1 "
214                                  "or in (0, 0.5], got %s"
215                                  % self.min_samples_leaf)
216             min_samples_leaf = int(ceil(self.min_samples_leaf * n_samples))
```

```
218         if isinstance(self.min_samples_split, numbers.Integral):
219             if not 2 <= self.min_samples_split:
220                 raise ValueError("min_samples_split must be an integer "
221                                  "greater than 1 or a float in (0.0, 1.0]; "
222                                  "got the integer %s"
223                                  % self.min_samples_split)
224             min_samples_split = self.min_samples_split
225         else:  # float
226             if not 0. < self.min_samples_split <= 1.:
227                 raise ValueError("min_samples_split must be an integer "
228                                  "greater than 1 or a float in (0.0, 1.0]; "
229                                  "got the float %s"
230                                  % self.min_samples_split)
231             min_samples_split = int(ceil(self.min_samples_split * n_samples))
232             min_samples_split = max(2, min_samples_split)
233 
234         min_samples_split = max(min_samples_split, 2 * min_samples_leaf)
```

4. The lines of code for **post-prune**: lines [523-560]

```
523     def cost_complexity_pruning_path(self, X, y, sample_weight=None):
524         """Compute the pruning path during Minimal Cost-Complexity Pruning.
525 
526         See :ref:`minimal_cost_complexity_pruning` for details on the pruning
527         process.
528 
529         Parameters
530         ----------
531         X : {array-like, sparse matrix} of shape (n_samples, n_features)
532             The training input samples. Internally, it will be converted to
533             ``dtype=np.float32`` and if a sparse matrix is provided
534             to a sparse ``csc_matrix``.
535 
536         y : array-like of shape (n_samples,) or (n_samples, n_outputs)
537             The target values (class labels) as integers or strings.
538 
539         sample_weight : array-like of shape (n_samples,), default=None
540             Sample weights. If None, then samples are equally weighted. Splits
541             that would create child nodes with net zero or negative weight are
542             ignored while searching for a split in each node. Splits are also
543             ignored if they would result in any single class carrying a
544             negative weight in either child node.
545 
546         Returns
547         -------
548         ccp_path : Bunch
549             Dictionary-like object, with attributes:
550 
551             ccp_alphas : ndarray
552                 Effective alphas of subtree during pruning.
553 
554             impurities : ndarray
555                 Sum of the impurities of the subtree leaves for the
556                 corresponding alpha value in ``ccp_alphas``.
557         """
558         est = clone(self).set_params(ccp_alpha=0.0)
559         est.fit(X, y, sample_weight=sample_weight)
560         return Bunch(**ccp_pruning_path(est.tree_))
```