IE-7374 ST:Machine Learning in Engineering
Group - 11
Lab 2

Akash Bansal   Divya Bandapalle   Rohit Bokade

K-fold cross validation

- The dataset contains 8 predictor variables and 79 observations
- 10-fold cross validation is performed by dividing the dataset into 10 parts, 9 of which are of size 8 and 1 of size 7
- **Performance evaluation metric:** Sum of squared errors ($SSE_{CV} = \frac{1}{k}\sum_{i=1}^{k} SSE_i$)

Experiements:

1. Using all the eight predictor variables $(x_1, \cdots, x_8)$
2. Using the first two predictor variables (x_1, x_2)

Each experiement is replicated 20 times with different seeds, which are consistent across both the experiements to avoid bias. We decided to use multi linear regression (from Lab-1) of normal form for this lab assignment.

Results:

- **8-predictor model:**
  - `SSE_mean = 30.029` and `SSE_std = 0.073` for 20 replicates
- **2-predictor model:**'
  - `SSE_mean = 32.027` and `SSE_std = 0.026` for 20 replicates
- *The cross validation scores are fairly consistent across 20 replicates with the SSE for 8-predictor model being lower than that of 2-predictor model. The standard deviation across the twenty replicates gives us and idea about the amount variation we can expect in the final model.*
- Thus, we would chose the 8-predictor model over the 2-predictor model.
- However, without further analysis we cannot be sure that adding 6 more variables in the model is worth the reduction of $\approx$ 2 SSE, since there is no upper bound to SSE (losely speaking)
- A more reliable metric could be R-squared, which could be used to estimate the benefit of adding each extra variable in the model.

- Another simpler approach would be to simply check the correaltion of each predictor with the dependent variable and drop the ones with low correlation.

Code for K-fold cross validation

```
class KFoldCrossValidation:
    """
    - Divide the data into K parts of roughly equal size
    - Evaluation:
        - For "i"-th in "k" parts, set one part "j" aside
        - Train the model on remaining parts
        - Use this model to train on "j" part and evaluate the given
        performance metric (e.g., SSE_i)
        - The final cross-validation score is the average of
        the performance metrics (e.g. mean({SSE_{1}, ..., SSE_{k}))


    Parameters:
    -----------
        @parameter k: (int) Number of parts to divide the data into
        @parameter X: (np.ndarray) Values of independent variables
        @parameter y: (np.ndarray) Values of predictor variable
        @parameter seed: (int) Seed for reproducible results
    """

    _logger = logging.getLogger(__name__)

    def __init__(self, X, y, k):
        self.X = X
        self.y = y
        self.k = k
        assert self.k > 1, "k must be greater than 1."
        self.num_rows, self.num_cols = self.X.shape

    def _compute_num_samples_per_part(self):
        """
        Splits the data into k equal parts. If the data cannot be split into
        equal parts, then the last part would contain the remainder of the data
        """
        # Split equally
        q, r = divmod(self.num_rows, self.k)

        self.num_samples_per_part = [
            ceil(q + (r / self.k)) for _ in range(self.k - 1)
        ]
```

```
        self.num_samples_per_part.append(
                self.num_rows - sum(self.num_samples_per_part)
        )  # add remaining samples
        self._logger.info(
            f"Number of samples per part: {self.num_samples_per_part}"
        )

        return self.num_samples_per_part

    def _get_split_indices(self):
        """
        Computes indices for each part. Storing indices in memory is less
        expensive than storing the data.
        """
        indices = np.arange(0, self.num_rows)

        self.split_indices = []
        for i in range(self.k - 1):
            # Get a random sample of indices for the part
            sampled_indices = np.random.choice(
                indices, size=self.num_samples_per_part[i],
                replace=False
            )
            self.split_indices.append(sampled_indices)
            # Remove the selected indices
            indices = np.setdiff1d(indices, sampled_indices)

        # Add the remaining indices in the last part
        self.split_indices.append(indices)
        self.split_indices = np.array(self.split_indices, dtype="object")

    def get_cv_performance(self, model, performance_eval_func, seed=42):
        """
        Performs k-fold cross validation of the dataset.

        Parameters:
        -----------
            @param model: (class) model with fit() and predict() methods
            @param performance_metric: (func) function to evaluate the
                performance of the model
            @param seed: (int) Seed
        """
        np.random.seed(seed)  # set random seed for reproducibility
        self._logger.info(f"Seed: {seed}")
        self._compute_num_samples_per_part()
        self._get_split_indices()
```

```python
        # Holding out each part (j), training on remaining parts
        # and evaluating the performance on part (j)
        performances = []
        for holdout_idx in range(self.k):
            # Train
            indices_for_training = np.delete(
                self.split_indices, holdout_idx, axis=0
            )
            indices_for_training = np.concatenate(
                    indices_for_training, axis=0
            )
            X = self.X[tuple(indices_for_training), :]
            if self.y.ndim == 1:
                y = self.y[indices_for_training]
            else:
                y = self.y[tuple(indices_for_training), :]
            coefficients = model.fit(X, y)
            # Evaluate
            indices_for_testing = self.split_indices[holdout_idx]
            y_pred = model.predict(X)
            performance = performance_eval_func(y_pred=y_pred, y=y)
            performances.append(performance)
            self._logger.debug(
                f"Holdout part: {holdout_idx} | Perforamance: {performance}"
            )
        mean_performance = np.mean(performances)
        self._logger.info(f"Mean CV performance: {mean_performance}")

        return mean_performance
```

Execution

- The script can be executed by running `python main.py` in command line
- Dependencies can be found in `requirements.txt`

Output

```
INFO:Lab-2:Model: 8-predictor
INFO:k_fold_cv:Seed: 0
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.116726528851125
INFO:k_fold_cv:Seed: 1
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 29.999741045267463
INFO:k_fold_cv:Seed: 2
```

```
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.104237143747305
INFO:k_fold_cv:Seed: 3
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.095519326689907
INFO:k_fold_cv:Seed: 4
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.07904835940726
INFO:k_fold_cv:Seed: 5
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.10766988690827
INFO:k_fold_cv:Seed: 6
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.00271499662232
INFO:k_fold_cv:Seed: 7
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.05329572565531
INFO:k_fold_cv:Seed: 8
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 29.921421357981256
INFO:k_fold_cv:Seed: 9
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.05229636492313
INFO:k_fold_cv:Seed: 10
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 29.978396709077515
INFO:k_fold_cv:Seed: 11
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 29.98836339696294
INFO:k_fold_cv:Seed: 12
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 29.8819902038899
INFO:k_fold_cv:Seed: 13
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.037968667213057
INFO:k_fold_cv:Seed: 14
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 29.88634051269827
INFO:k_fold_cv:Seed: 15
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.090145238172532
INFO:k_fold_cv:Seed: 16
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 29.98761569736042
INFO:k_fold_cv:Seed: 17
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
```

```
INFO:k_fold_cv:Mean CV performance: 30.005221505383872
INFO:k_fold_cv:Seed: 18
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.069790549229527
INFO:k_fold_cv:Seed: 19
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 30.13195604534938
INFO:Lab-2:Mean CV score across 20 replicates: 30.029522963069542
INFO:Lab-2:STD of CV score across 20 replicates: 0.07254638987719794
INFO:Lab-2:


INFO:Lab-2:Model: 2-predictor
INFO:k_fold_cv:Seed: 0
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.06700943490037
INFO:k_fold_cv:Seed: 1
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.00462146631315
INFO:k_fold_cv:Seed: 2
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.063964267238255
INFO:k_fold_cv:Seed: 3
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.033155605332105
INFO:k_fold_cv:Seed: 4
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.00143247534919
INFO:k_fold_cv:Seed: 5
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.02242518396899
INFO:k_fold_cv:Seed: 6
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.031790674888555
INFO:k_fold_cv:Seed: 7
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.012416321247045
INFO:k_fold_cv:Seed: 8
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.04609539279691
INFO:k_fold_cv:Seed: 9
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.02682796456653
INFO:k_fold_cv:Seed: 10
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.032017066682215
```

```
INFO:k_fold_cv:Seed: 11
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.006925149014286
INFO:k_fold_cv:Seed: 12
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 31.956864537342017
INFO:k_fold_cv:Seed: 13
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.03264089960253
INFO:k_fold_cv:Seed: 14
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.042843795319655
INFO:k_fold_cv:Seed: 15
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.00103126542281
INFO:k_fold_cv:Seed: 16
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.03449234014179
INFO:k_fold_cv:Seed: 17
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.00474669335625
INFO:k_fold_cv:Seed: 18
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.045935747894546
INFO:k_fold_cv:Seed: 19
INFO:k_fold_cv:Number of samples per part: [8, 8, 8, 8, 8, 8, 8, 8, 8, 7]
INFO:k_fold_cv:Mean CV performance: 32.067042033726636
INFO:Lab-2:Mean CV score across 20 replicates: 32.02671391575518
INFO:Lab-2:STD of CV score across 20 replicates: 0.026073680941034988
```