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 = 4.67` and `SSE_std = 0.18` for 20 replicates
- **2-predictor model:**'
  - `SSE_mean = 3.86` and `SSE_std = 0.057` for 20 replicates
- *The cross validation scores for for 2-predictor model are lower than that of 8-predictor model. The standard deviation across the twenty replicates gives us and idea about the amount variation we can expect in the final model.*
- This inconsistency is a result of the the other variables ($x_3, \cdots, x_4$) not having enough explanability about the dependent variable y. Also, these variables have a low correlation with x_1 and x_2. For this assignment, we limit the analysis to checking only the linear relationships. It is possible that these models have a non-linear relationship with the other predicted variables and the dependent variable. We can therefore conclude that these "additional" variables are just adding noise into linear the model and therefore do not improve the performance.

`Correlation matrix:`

	y	x1	x2	x3	x4	x5	x6	x7	x8
y	1.00	0.67	0.64	0.22	-0.10	0.31	-0.02	0.01	0.03
x1	0.67	1.00	0.60	0.26	-0.10	0.23	-0.08	-0.08	0.06
x2	0.64	0.60	1.00	0.28	-0.11	0.20	0.02	0.03	0.06
x3	0.22	0.26	0.28	1.00	0.12	-0.02	-0.01	0.16	0.16
x4	-0.10	-0.10	-0.11	0.12	1.00	-0.02	0.05	0.07	0.00
x5	0.31	0.23	0.20	-0.02	-0.02	1.00	0.00	-0.09	0.09
x6	-0.02	-0.08	0.02	-0.01	0.05	0.00	1.00	-0.09	-0.25
x7	0.01	-0.08	0.03	0.16	0.07	-0.09	-0.09	1.00	0.07
x8	0.03	0.06	0.06	0.16	0.00	0.09	-0.25	0.07	1.00

- Thus, we would chose the 2-predictor model over the 8-predictor model.

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
```

```python
        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
    """

    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)]  # noqa
        remaining_samples = self.num_rows - sum(self.num_samples_per_part)
        self.num_samples_per_part.append(remaining_samples)  # add remaining samples  # noqa
        logging.debug(f"Number of samples per part: {self.num_samples_per_part}")  # noqa

        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._compute_num_samples_per_part()
    self._get_split_indices()

    # 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), :]
        _ = model.fit(X, y)
        # Evaluate
        indices_for_testing = self.split_indices[holdout_idx]
        X_test = self.X[indices_for_testing]
        y_test = self.y[indices_for_testing]
        y_predicted = model.predict(X_test)
        performance = performance_eval_func(
            y_actual=y_test, y_predicted=y_predicted
        )
        performances.append(performance)
        logging.debug(f"Holdout part: {holdout_idx} | {performance_eval_func.__name__}: {performance}")  # n

    mean_performance = np.mean(performances)
    logging.info(f"Seed: {seed} | Mean CV {performance_eval_func.__name__}: {mean_performance}")  # noqa

    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 [utils.py:141] NumExpr defaulting to 8 threads.
INFO [main.py:29] Model: 8-Predictor Model
INFO [k_fold_cross_validation.py:111] Seed: 0 | Mean CV sum_of_squared_error: 4.428942653481051
INFO [k_fold_cross_validation.py:111] Seed: 1 | Mean CV sum_of_squared_error: 4.741664914315521
INFO [k_fold_cross_validation.py:111] Seed: 2 | Mean CV sum_of_squared_error: 4.477409252562606
INFO [k_fold_cross_validation.py:111] Seed: 3 | Mean CV sum_of_squared_error: 4.485621912531902
INFO [k_fold_cross_validation.py:111] Seed: 4 | Mean CV sum_of_squared_error: 4.55443355001513
INFO [k_fold_cross_validation.py:111] Seed: 5 | Mean CV sum_of_squared_error: 4.460291335366071
INFO [k_fold_cross_validation.py:111] Seed: 6 | Mean CV sum_of_squared_error: 4.722469620052055
INFO [k_fold_cross_validation.py:111] Seed: 7 | Mean CV sum_of_squared_error: 4.637311676817276
INFO [k_fold_cross_validation.py:111] Seed: 8 | Mean CV sum_of_squared_error: 4.94463752479322
INFO [k_fold_cross_validation.py:111] Seed: 9 | Mean CV sum_of_squared_error: 4.626793975723169
INFO [k_fold_cross_validation.py:111] Seed: 10 | Mean CV sum_of_squared_error: 4.8319705545090255
INFO [k_fold_cross_validation.py:111] Seed: 11 | Mean CV sum_of_squared_error: 4.7892806783624255
INFO [k_fold_cross_validation.py:111] Seed: 12 | Mean CV sum_of_squared_error: 5.022567004736908
```

```
INFO [k_fold_cross_validation.py:111] Seed: 13 | Mean CV sum_of_squared_error: 4.600990923801492
INFO [k_fold_cross_validation.py:111] Seed: 14 | Mean CV sum_of_squared_error: 5.002736585439514
INFO [k_fold_cross_validation.py:111] Seed: 15 | Mean CV sum_of_squared_error: 4.512915622096051
INFO [k_fold_cross_validation.py:111] Seed: 16 | Mean CV sum_of_squared_error: 4.749370262425966
INFO [k_fold_cross_validation.py:111] Seed: 17 | Mean CV sum_of_squared_error: 4.711962280614829
INFO [k_fold_cross_validation.py:111] Seed: 18 | Mean CV sum_of_squared_error: 4.561723914991142
INFO [k_fold_cross_validation.py:111] Seed: 19 | Mean CV sum_of_squared_error: 4.4370429534891365
INFO [main.py:38] Mean CV sum_of_squared_error across 20 replicates: 4.665006859806225
INFO [main.py:39] STD of CV sum_of_squared_error across 20 replicates: 0.18045085033996033
INFO [main.py:40]

INFO [main.py:29] Model: 2-Predictor Model
INFO [k_fold_cross_validation.py:111] Seed: 0 | Mean CV sum_of_squared_error: 3.7698751863201574
INFO [k_fold_cross_validation.py:111] Seed: 1 | Mean CV sum_of_squared_error: 3.9073299820179166
INFO [k_fold_cross_validation.py:111] Seed: 2 | Mean CV sum_of_squared_error: 3.7772143721206533
INFO [k_fold_cross_validation.py:111] Seed: 3 | Mean CV sum_of_squared_error: 3.847395697331033
INFO [k_fold_cross_validation.py:111] Seed: 4 | Mean CV sum_of_squared_error: 3.9148120220675544
INFO [k_fold_cross_validation.py:111] Seed: 5 | Mean CV sum_of_squared_error: 3.867411669385355
INFO [k_fold_cross_validation.py:111] Seed: 6 | Mean CV sum_of_squared_error: 3.844436996030572
INFO [k_fold_cross_validation.py:111] Seed: 7 | Mean CV sum_of_squared_error: 3.8903085251660974
INFO [k_fold_cross_validation.py:111] Seed: 8 | Mean CV sum_of_squared_error: 3.8174219040341297
INFO [k_fold_cross_validation.py:111] Seed: 9 | Mean CV sum_of_squared_error: 3.8623260409133273
INFO [k_fold_cross_validation.py:111] Seed: 10 | Mean CV sum_of_squared_error: 3.8462834447009984
INFO [k_fold_cross_validation.py:111] Seed: 11 | Mean CV sum_of_squared_error: 3.899690286887517
INFO [k_fold_cross_validation.py:111] Seed: 12 | Mean CV sum_of_squared_error: 4.012354016735182
INFO [k_fold_cross_validation.py:111] Seed: 13 | Mean CV sum_of_squared_error: 3.850081157648
INFO [k_fold_cross_validation.py:111] Seed: 14 | Mean CV sum_of_squared_error: 3.8189218531201163
INFO [k_fold_cross_validation.py:111] Seed: 15 | Mean CV sum_of_squared_error: 3.9148643304386974
INFO [k_fold_cross_validation.py:111] Seed: 16 | Mean CV sum_of_squared_error: 3.83914788348861
INFO [k_fold_cross_validation.py:111] Seed: 17 | Mean CV sum_of_squared_error: 3.905615359327728
INFO [k_fold_cross_validation.py:111] Seed: 18 | Mean CV sum_of_squared_error: 3.817382928556681
INFO [k_fold_cross_validation.py:111] Seed: 19 | Mean CV sum_of_squared_error: 3.7673693359695735
INFO [main.py:38] Mean CV sum_of_squared_error across 20 replicates: 3.858512149612995
INFO [main.py:39] STD of CV sum_of_squared_error across 20 replicates: 0.05746842654665062
INFO [main.py:40]
```