9,5 | 10

Sheet09

June 21, 2022

Exercise 18

a) Explain in general which features can be useful for a classification. In particular, address features that should not be used and discuss whether the features we provide are useful

Features are useful if they provide information that is relevant for predicting the correct label. The number of pixel of the detector dont provide helpful information, as they are the same for all the datapoints.

The event-id is not useful as well, as it is arbitrary and is not connected to the datapoint. If the events are generated in order, first tracks and then order, the algortihm would use mainly the id to identify the kind of particle. E.g id<10000 -> track. The model would then perform poorly on test data, where the events are randomly ordered.

The mean pixel value is the most useful generated feature, as it provides additional information, which the raw data cannot provide. Also it is mostly unique for every data point, compared to the number of pixels of the detector.

b)

[1]:
```python
def analyse(self, event):
        """Reconstruct an event measured in the assigned detector.

        Parameters
        ----------
        event : Event
            The event which we want to reconstruct.

        Returns
        -------
        Features: Dict
            A dictionary of arbitrary features.
            Values are to be scalar to be fed into a random forest
        """

        # ---------
        # Exercise:
        # ---------
        # Feature Generation (exercises/feature_generation.py)
        #
```

```
        # ------------------------------------------------------------------
        # Replace Code here
        # ------------------------------------------------------------------

        # Dummy solution to pass unit tests (this solution is not correct)
        # You can access the event and detector here.

        #print(vars(self.detector))

        x_arrays = np.max(np.array([event.pixels[:,i] for i in range(64)]),␣
↪axis = 0)
        y_arrays = np.max(np.array([event.pixels[i,:] for i in range(64)]),␣
↪axis = 0)

        x_max = np.where(x_arrays == np.max(x_arrays))
        y_max = np.where(y_arrays == np.max(y_arrays))

       # print(x_max, y_max)


        #x_pos_array = np.where(x_pos_array == np.max(x_pos_array, ))

        features = {
            "pixel_sum": event.pixels.sum(),
            "pixel_max": np.max(event.pixels),
            "x_mean": np.mean(x_max),
            "y_mean": np.mean(y_max),
            }


        return features
```

c) AUC score of 0.849 achieved -> area under the curve ✓

d) What would a "roc auc score" of 0.5 and a "roc auc score" of 1 mean respectively?

0.5: Curve would be straigh line, algorithm just guesses the class -> 50/50 chance thus same amount of true positives as false positives ✓

1: optimal classifier, for every threshold the algorithm predicts every sample accurately -> overfitting

Getting a roc-auc of 1 is the goal of any classification. As the AUC-score is evaluated on test-data an overfitted model should produce an AUC < 1.

e) How could a "roc auc score" lower than 0.5 occur and give a solution to this problem.

auc lower than 0.5 -> more false positives than true positives

Problem: Test data is wrongly labeled

Solution: Just switch the labels of the classes ✓

f) How do accuracy and sensitivity change with increasing "prediction threshold"? Take a look

at the “Precision and Recall” graph in the overview PDF.

With higher thresholds the precision increases -> the algortihm needs a higher confidence for the prediction to be accepted. With a higher threshold the number of false positives decreases. As fp is in the denumerator the accuracy increases. At a threshold of zero the precision is at 50%, as there is a 50/50 chance for the sample to be accepted. With a threshold of 1 only samples with 100% confidence are accepted. This results in a precision of almost 1.

The recall falls with higher thresholds as the number of tp falls as they need higher and higher confidence to be accepted, resulting in less accepted tp overall

We start out at a recall of slightly less than 1, as fn is close to 0, because almost every prediction is accepted -> only tp exist

The recall stops at slightly more than zero for a threshold of 1 as tp becomes zero, almost no data point is predicted with a confidence of 100 %

Exercise 19

a) Change the function to use the random forest regressor in define_model. For this purpose, use the python package sklearn.

```
[ ]: def define_model(seed):
        """A helper function to retrieve a model for the energy regression
        exercise. The parameter seed gets used to set the random state of the model
        and ensure reproducible results"""

        # ---------
        # Exercise:
        # ---------
        # Energy_regression (exercises/energy_regression.py)
        #
        # ------------------------------------------------------------------------
        # Add Code here
        # ------------------------------------------------------------------------

        model =  ensemble.RandomForestRegressor(n_estimators=100, min_samples_leaf=␣
     ↪5, random_state=seed)


        return model
```

b) Implement the 5-fold cross validation in the function cross_validate_model.

```
[ ]: def cross_validate_model(X, y, model, seed):
        """This function implements a cross validation on a given model.

        Required return values:
        -----------------------
        predictions: np.array of the same shape as y
            (These are the predictions on the test sets combined into one array)
```

```
    true_values: np.array of the same shape as y
        (These are the y's chosen in the different cross validation steps
        combined into one array. This equals y but with different order.)
    models: list of (n_cross_validation_steps) models
        These are used to calculate the feature importances later
    """
    # ---------
    # Exercise:
    # ---------
    # Energy_regression (exercises/energy_regression.py)
    #
    # --------------------------------------------------------------------------
    # Add Code here
    # --------------------------------------------------------------------------

    # This does not perform a cross validation!
    # It merely lets the script run through without an error
    true_values = y

    cf = KFold(5, shuffle=True, random_state=seed)

    predictions = []
    true_values = []
    models = []

    for train_index, test_index in cf.split(X):
        X_train, y_train = X.iloc[train_index], y[train_index]
        X_test, y_test = X.iloc[test_index], y[test_index]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        predictions.append(y_pred)
        true_values.append(y_test)
        models.append(model)

    predictions = np.concatenate(predictions)
    true_values = np.concatenate(true_values)

    return predictions, true_values, models
```

c) You will find the graph "Regressor Confusion" in the overview PDF. Briefly describe what a migration matrix is. Which properties of your regressor can you infer from the migration matrix?

The migration matrix is to the regressor, what the confusion matrix is to the classifier. It shows how well the predictor works. When there is a wide distribution around the diagonal, the regressor performs the worst. The more the graphs resembles a diagonal, the better the energies of that area

are predicted.

The matrix is a good measure to check, which energies are prediced acurrately or less accurately. if the energies are above the diagonal the energy is predicted to high. Below the diagonal they are predicted too low.

d) Which properties of your regressor can you infer from the value "Bias" and "Resolution"? (Graph "Bias and Resolution")

The higher the the absolute of the bias, the bigger the effect of underfitting. The model is biased towards a certain energy. Energies with a high absolute bias are predicted less accurately.

The Resolution describes how well the algorithm can differentiate energies, e.g the smalles ΔE between two energies.

e) Which three features in this example are most important for the energy estimation? (Graph "Feature Importance")

Most important is the pixel sum, followed by the maximum pixel count and the mean y-position

The pixel sum is the best feature here, as the sum depends on the light distribution generated by the detected particle. The light distribution is characteristic for its energy, thus the sum is the most important for predicting the energy.

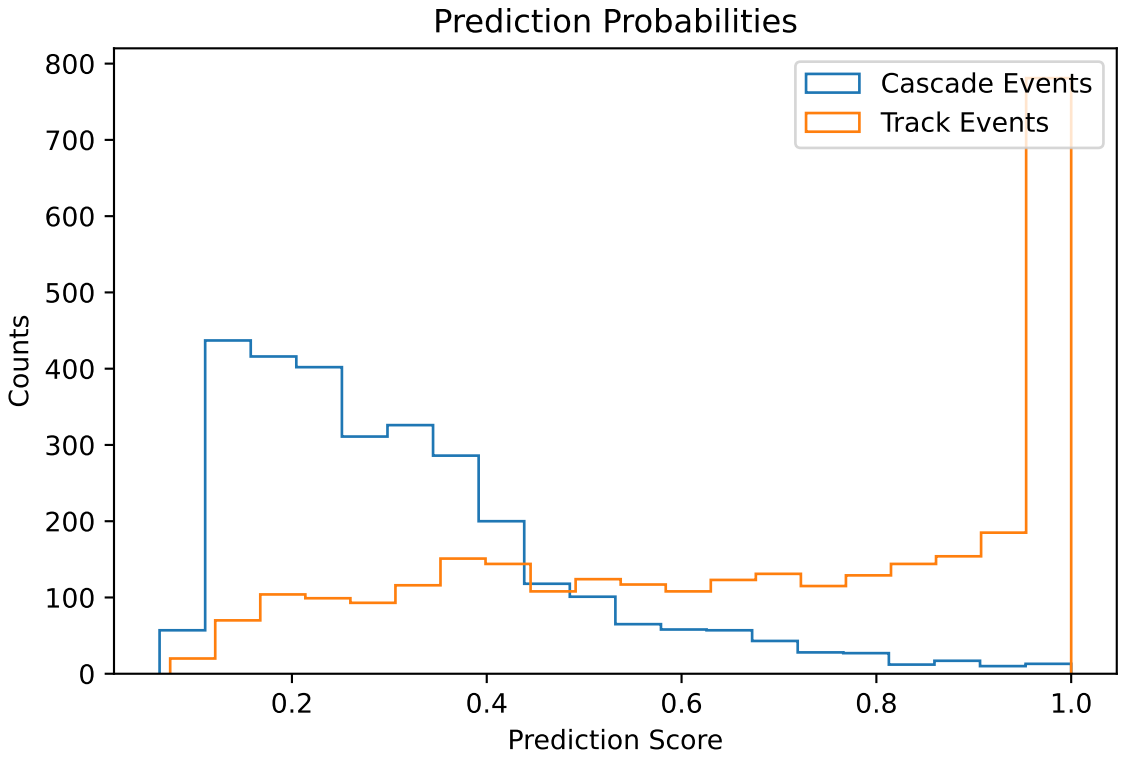
f) What is the effect of predicting the logarithmic energy instead of a linear prediction?

logarithmic prediction: R2 score: 0.77 Mean Squared Error: 0.03 Mean Absolute Error: 0.14

linear prediction: R2 score: 0.77 Mean Squared Error: 135637926.00 Mean Absolute Error: 1903.74

The mean squared errors and absolute errors are now in a reasonable size. In the linear prediction the errors of higher energies are overrepresented as they are magnitudes bigger than the errors of smaller energies. With logarthmic prediction the errors of the energies are treated more "evenly"

15/15



Information:

Exercise: Feature Generation

Group name: project_c3

Random Seed: 1337

Track Events: 10000

Cascade Events: 10000

Background Level: 10

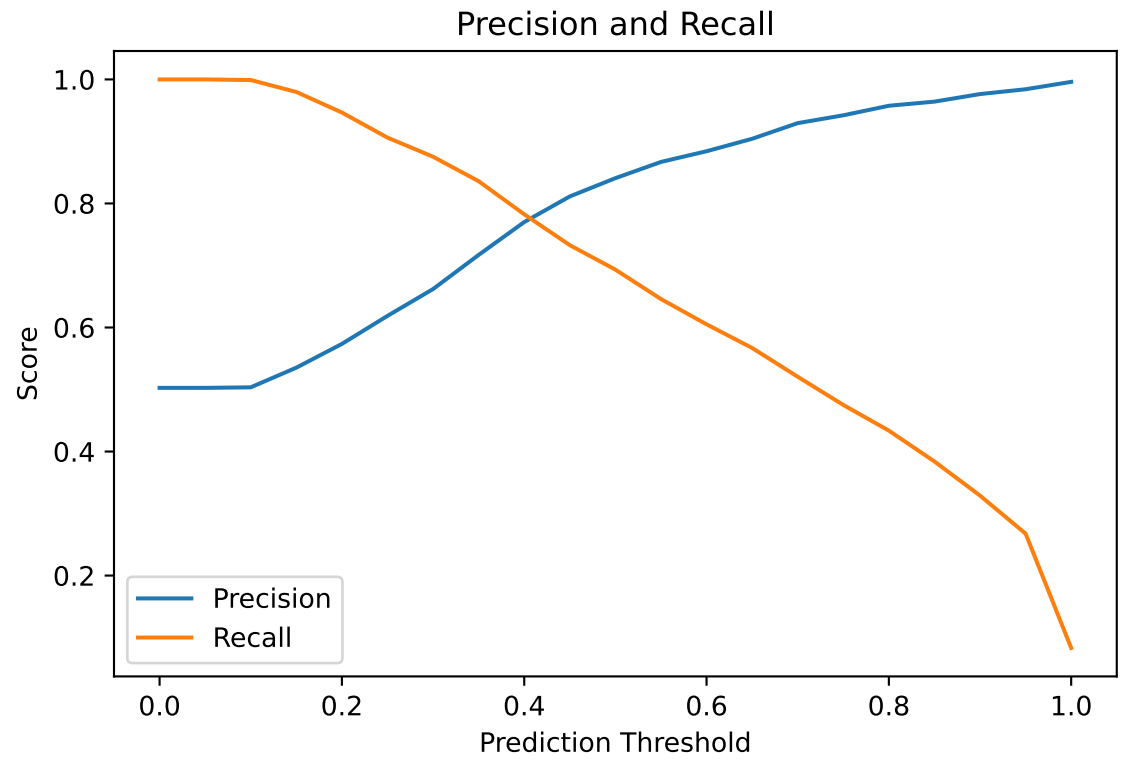
Runtime: 29.398s

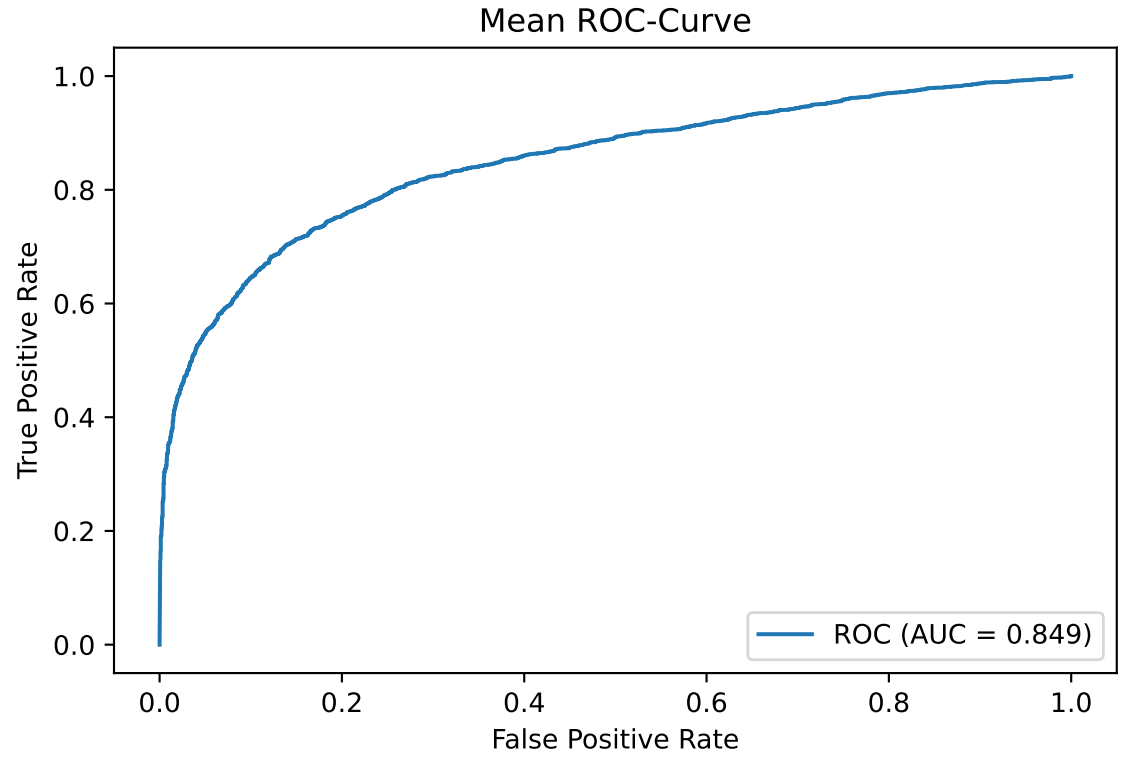
Test Accuracy: 0.78

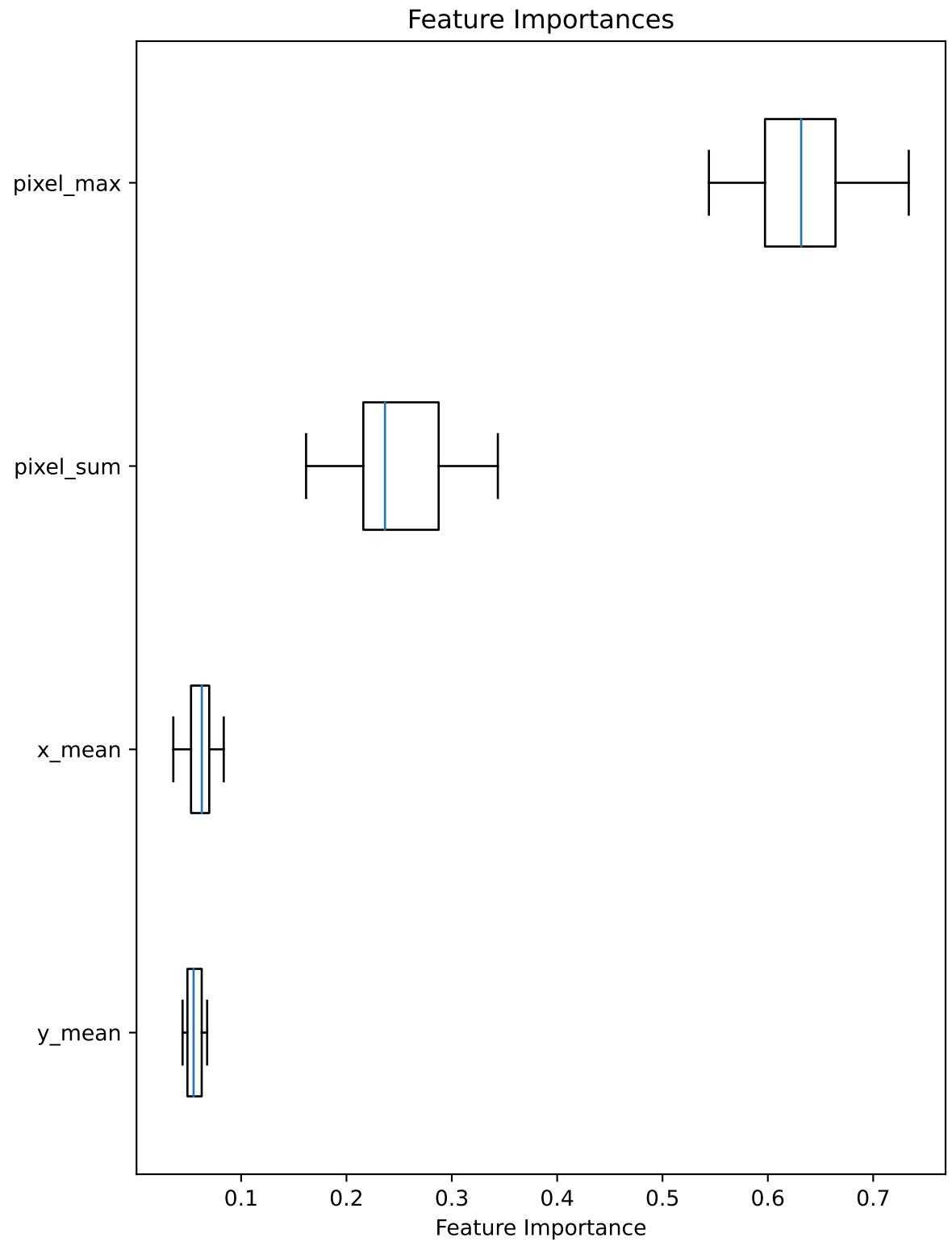
Tests

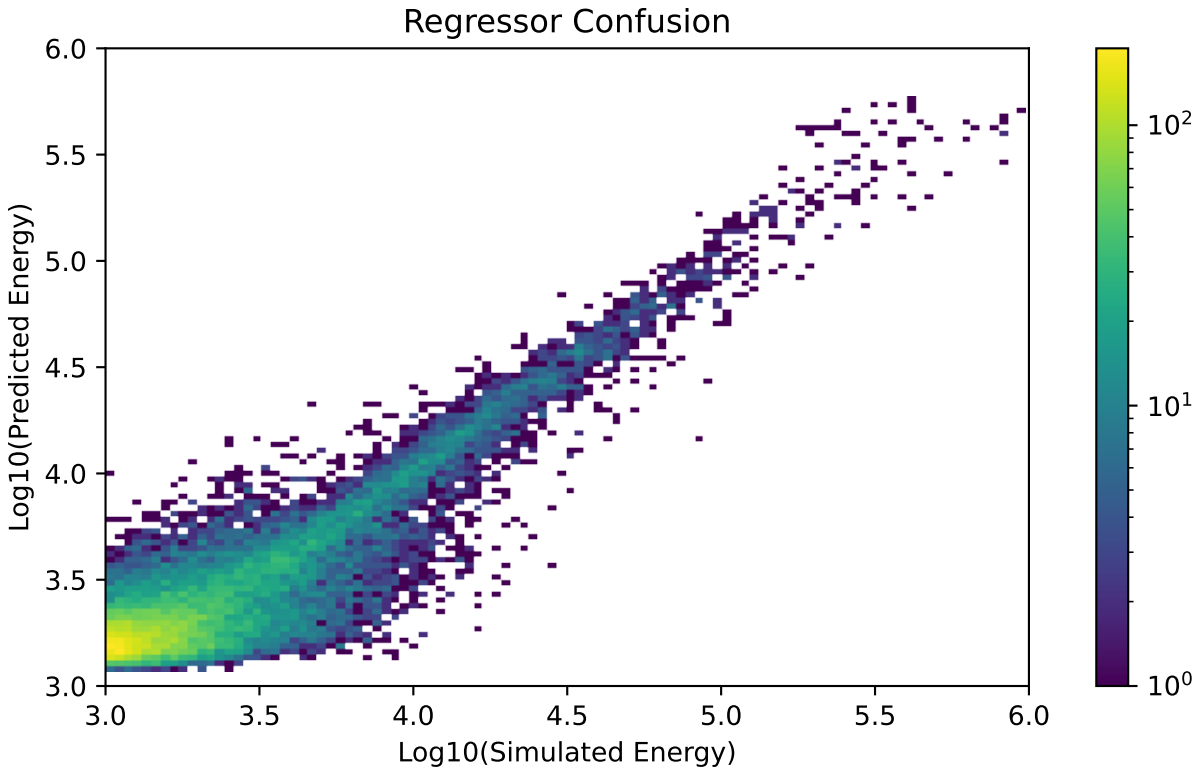
Passed nan test: True

Passed settings test: True







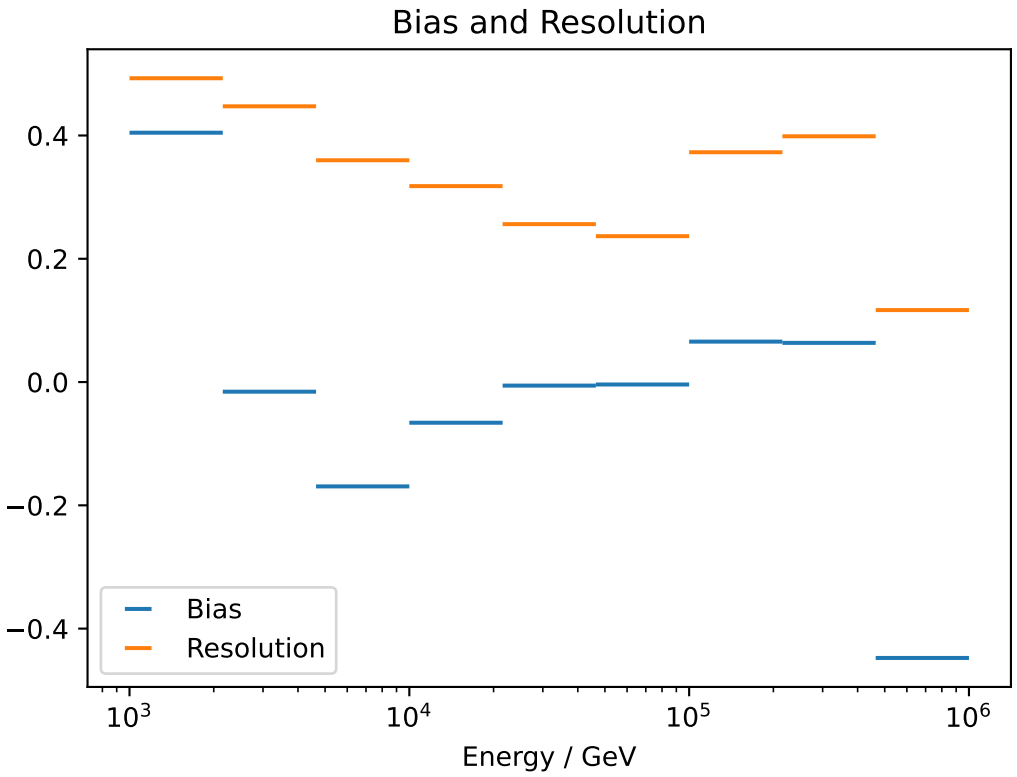


Information:

- Exercise: Energy Regression
- Group name: project_c3
- Random Seed: 1337
- # Cascade Events: 10000
- Background Level: 10
- Runtime: 51.790s
- Trained models: 5

Tests

- Passed nan test: True
- Passed settings test: True
- Passed scaling test: True



More Metrics:

- R^2 score: 0.77
- Mean Squared Error: 135637926.00
- Mean Absolute Error: 1903.74