Documentation for lung cancer drug reponse prediction using quantum machine learning

Timothy Geiger

Aug 19, 2023

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

class lung_cancer.**BaseClassifier**

Abstract base class for classifiers.

This class serves as the base class for all classifier implementations. Subclasses should override the abstract methods and provide their own implementation.

class lung_cancer.**ClassicLivePlotter**(*classifier:* BaseClassifier, *data_wrapper:* DataWrapper | *None* = *None*)

This class helps to plot the training and validation loss and accuracy during the training process of a classical classifier.

Parameters

- **classifier** (BaseClassifier) – The classifier to use.
- **data_wrapper** (*Optional[DataWrapper]*) – The data wrapper for validation data. Default is None.

class lung_cancer.**CorrMatrixPlotter**(*dataset:* DatasetWrapper)

Initialize the CorrMatrixPlotter.

Parameters

dataset (DatasetWrapper) – The dataset wrapper object.

show(*cols: List[str]*, *show_labels: bool = True*, *size: tuple = (6, 6)*, *calc: bool = True*) → None

Display the correlation matrix plot.

Parameters

- **cols** (*list[str]*) – The columns to include in the correlation matrix.
- **show_labels** (*bool, optional*) – Whether to show the labels or not. Defaults to True.
- **size** (*tuple, optional*) – The size of the plot.
- **calc** (*bool*) – If the correlation should be calculated.

Returns

None

class lung_cancer.**DataWrapper**(*X: DataFrame*, *y: Series*, *batch_size: int*, *shuffle: bool*)

A class that wraps data for training. This wrapper is needed since qiskit and pytorch want the data in different formats. Pytorch needs the data to be in the pytorch dataloader, where as quskit wants just the data (features & targets) itself.

Parameters

- **X** (*pd.DataFrame*) – Feature values.
- **y** (*pd.Series*) – Target values.
- **batch_size** (*int*) – Number of samples per batch.
- **shuffle** (*bool*) – Whether to shuffle the data.

Examples

```
>>> import pandas as pd
>>> X = pd.DataFrame({'feature1': [1, 2, 3], 'feature2': [4, 5, 6]})
>>> y = pd.Series([0, 1, 0])
>>> wrapper = DataWrapper(X, y, batch_size=2, shuffle=True)
```

get_data() → Tuple[ndarray, ndarray]

Get the raw data.

Returns

A tuple containing the input
features and target values as numpy arrays.

Return type
tuple[np.ndarray, np.ndarray]

Examples

```
>>> wrapper = DataWrapper(X, y, batch_size=2, shuffle=True)
>>> X_values, y_values = wrapper.get_data()
```

get_loader() → DataLoader

Get the pytorch data loader object.

Returns
A data loader object for iterating over the data.

Return type
DataLoader

Examples

```
>>> wrapper = DataWrapper(X, y, batch_size=2, shuffle=True)
>>> loader = wrapper.get_loader()
>>> for batch_X, batch_y in loader:
...     # Use the batched data for training
```

class lung_cancer.**DatasetWrapper**(*features: DataFrame | None = None*, *labels: DataFrame | None = None*, *train_X: DataFrame | None = None*, *test_X: DataFrame | None = None*, *val_X: DataFrame | None = None*, *train_y: DataFrame | None = None*, *test_y: DataFrame | None = None*, *val_y: DataFrame | None = None*, *train_size: float = 0.65*, *test_size: float = 0.2*, *val_size: float = 0.15*, *batch_size: int = 64*)

A wrapper class for managing datasets. The user has to specify either features and labels or the train, test and validation datasets to create this object. If the user inputs features and labels he can also specfiy how to split the data. The numbers must add up to 1.

Parameters

- **features** (*pd.DataFrame, optional*) – Input features. Defaults to None.
- **labels** (*pd.DataFrame, optional*) – Input labels. Defaults to None.
- **train_X** (*pd.DataFrame, optional*) – Training features. Defaults to None.
- **test_X** (*pd.DataFrame, optional*) – Testing features. Defaults to None.
- **val_X** (*pd.DataFrame, optional*) – Validation features. Defaults to None.
- **train_y** (*pd.DataFrame, optional*) – Training labels. Defaults to None.
- **test_y** (*pd.DataFrame, optional*) – Testing labels. Defaults to None.
- **val_y** (*pd.DataFrame, optional*) – Validation labels. Defaults to None.
- **train_size** (*float, optional*) – Size of the training data. Defaults to 0.65.
- **test_size** (*float, optional*) – Size of the testing data. Defaults to 0.2.
- **val_size** (*float, optional*) – Size of the validation data. Defaults to 0.15.
- **batch_size** (*int, optional*) – Batch size. Defaults to 64.

Examples

```
>>> import pandas as pd
>>> X = pd.DataFrame({'feature1': [1, 2, 3], 'feature2': [4, 5, 6]})
>>> y = pd.DataFrame({'target': [0, 1, 0]})
>>> dataset = DatasetWrapper(X, y)
```

apply_transformer(*transformer: Any*, *fit_on: List[str] = ['X']*, *fit_data: str = 'train'*, *fit_method: str = 'fit'*, *transform_on: List[str] = ['X']*, *transform_data: List[str] = ['train', 'test', 'val']*, *transform_method: str = 'transform'*, *cols=None*) → None

Applies a specified transformer on the dataset.

Parameters

- **transformer** – The transformer object.

- **fit_on** (*list, optional*) – List of data to fit the transformer on. Valid is X or/and y. Defaults to ['X'].
- **fit_data** (*str, optional*) – Data to fit the transformer on. Valid is train, test and val. Defaults to 'train'.
- **fit_method** (*str, optional*) – Method name to fit the transformer. Defaults to 'fit'.
- **transform_on** (*list, optional*) – List of data to transform using the transformer. Valid is X or/and y. Defaults to ['X'].
- **transform_data** (*list, optional*) – List of data subsets to transform. Valid is train or/and test or/and val. Defaults to ['train', 'test', 'val'].
- **transform_method** (*str, optional*) – Method name to transform the data. Defaults to 'transform'.
- **cols** (*list, optional*) – List of columns to apply the transformer on. Defaults to None (= all columns).

Returns
None

Raises
ValueError – If the dataset is already generated.

Examples

A simple example using a MinMax Scaler. The MinMax Scaler should be fitted to the train data und should transform the train, test and valdiation data.

```
>>> import pandas as pd
>>> from sklearn.preprocessing import MinMaxScaler
>>>
>>> X = pd.DataFrame({
>>>     'feature1': [1, 2, 3],
>>>     'feature2': [4, 5, 6]})
>>> y = pd.DataFrame({'target': [0, 1, 0]})
>>>
>>> dataset = DatasetWrapper(
>>>     features, labels,
>>>     train_size=0.7,
>>>     val_size=0.15,
>>>     test_size=0.15)
>>> dataset.apply_transformer(MinMaxScaler())
```

An Example using SMOTE as an transformer. Only the train data should be transformed. However this time not only the features but the targets as well. In addition the fit and transform function name is different compared to the standart values.

```
>>> import pandas as pd
>>> from imblearn.combine import SMOTE
>>>
>>> X = pd.DataFrame({
>>>     'feature1': [1, 2, 3],
>>>     'feature2': [4, 5, 6]})
>>> y = pd.DataFrame({'target': [0, 1, 0]})
```

(continues on next page)

(continued from previous page)

```
>>>
>>> dataset = DatasetWrapper(
>>>     features, labels,
>>>     train_size=0.7,
>>>     val_size=0.15,
>>>     test_size=0.15)
>>>
>>> smote = SMOTE(random_state=random_state)
>>> dataset.apply_transformer(
>>>     smote,
>>>     fit_on=[],
>>>     transform_on=['X', 'y'],
>>>     transform_data=['train'],
>>>     transform_method='fit_resample')
```

generate() → None

Generates the dataset.

Returns
None

Raises
ValueError – If the dataset is already generated.

Examples

```
>>> dataset = DatasetWrapper(
>>>     features, labels,
>>>     train_size=0.7,
>>>     val_size=0.15,
>>>     test_size=0.15)
>>> dataset.generate()
```

get_column_infos(*colname: str | int*)

Prints information about a specified column.

Parameters
colname (*Union[str, int]*) – Name of the column.

Returns
None

Examples

```
>>> dataset = DatasetWrapper(X, y)
>>> dataset.get_column_infos('feature1')
```

get_test() → *DataWrapper*

Returns the testing data.

Returns
The testing data.

Return type
DataWrapper

Raises
ValueError – If the dataset is not yet generated.

Examples

```
>>> dataset = DatasetWrapper(
>>>     features, labels,
>>>     train_size=0.7,
>>>     val_size=0.15,
>>>     test_size=0.15)
>>>
>>> test = dataset.get_test()
```

get_train() → *DataWrapper*

Returns the training data.

Returns
The training data.

Return type
DataWrapper

Raises
ValueError – If the dataset is not yet generated.

Examples

```
>>> dataset = DatasetWrapper(
>>>     features, labels,
>>>     train_size=0.7,
>>>     val_size=0.15,
>>>     test_size=0.15)
>>>
>>> train = dataset.get_train()
```

get_val() → *DataWrapper*

Returns the validation data.

Returns
The validation data.

Return type
DataWrapper

Raises
ValueError – If the dataset is not yet generated.

Examples

```
>>> dataset = DatasetWrapper(
>>>     features, labels,
>>>     train_size=0.7,
>>>     val_size=0.15,
>>>     test_size=0.15)
>>>
>>> val = dataset.get_val()
```

static load(*filename: str*) → None

Loads a dataset from a file.

Parameters
filename (*str*) – Name of the file to load the dataset.

Returns
None

Examples

```
>>> dataset = DatasetWrapper.load(
>>>     '../data/preprocessed/test_data.pkl')
```

save(*filename: str*) → None

Saves the dataset to a file.

Parameters
filename (*str*) – Name of the file to save the dataset.

Returns
None

Examples

```
>>> dataset = DatasetWrapper(
>>>     features, labels,
>>>     train_size=0.7,
>>>     val_size=0.15,
>>>     test_size=0.15)
>>>
>>>  dataset.save('../data/preprocessed/test_data.pkl')
```

class lung_cancer.**DistrubutionPlotter**(*dataset:* DatasetWrapper)

Initialize the DistrubutionPlotter.

Parameters
dataset (DatasetWrapper) – The dataset wrapper object.

show(*cols: str | List[str]*) → None

Display the distribution plot.

Parameters
cols (*str or list[str]*) – The column(s) to include in the distribution plot.

Returns
None

class lung_cancer.**FacetGridPlotter**(*dataset:* DatasetWrapper)

Initialize the FacetGridPlotter.

Parameters
dataset (DatasetWrapper) – The dataset wrapper object.

show(*cat_col: str*, *num_col: str*, *label_pos: str = 'right'*, *cat_order: List[str] | None = None*) → None

Display the FacetGrid plot.

Parameters

- **cat_col** (*str*) – The categorical column for the rows and hue.
- **num_col** (*str*) – The numerical column for the plot.
- **label_pos** (*str, optional*) – The position of the labels. Defaults to 'right'.
- **cat_order** (*list[str], optional*) – The order of categories. Defaults to None.

Returns
None

class lung_cancer.**LivePlotter**(*classifier:* BaseClassifier, *data_wrapper:* DataWrapper | *None = None*)

An abstract base class for live plotting during training.

Parameters

- **classifier** (BaseClassifier) – The classifier to monitor.
- **data_wrapper** (*Optional[*DataWrapper*]*) – The data wrapper for validation data. Default is None.

class lung_cancer.**NNClassifier**(*classifier*, *optimizer*, *loss*)

Initialize the NNClassifier (Neural Network).

Parameters

- **classifier** (*nn.Module*) – The neural network classifier.
- **optimizer** (*Optimizer*) – The optimizer for training the classifier.
- **loss** (*Loss*) – The loss function used for training.

Returns
None

calc_metrics(*data_wrapper:* DataWrapper)

Shows different metrics of the classifier using the dataset specified as an argument. It shows Accuracy, Recall, Precision, ROC-AUC score and a confusion matrix.

Parameters
data_wrapper (DataWrapper) – The dataset that should be used for calculating the metrics.

cross_validate(*features: DataFrame*, *labels: Series*, *pipeline: Pipeline*, *iterations: int | None = 100*, *k: int | None = 5*, *weights: bool | None = False*, *smote: bool | None = False*, *kbest: bool | None = False*, *kbestNum: int | None = 0*, *dim_red: str | None = 'pca'*, *dimension: int | None = 6*) → None

Perform cross-validation on the given dataset using the specified parameters.

Parameters

- **features** (*np.ndarray*) – The input features as a NumPy array.
- **labels** (*np.ndarray*) – The target labels as a NumPy array.
- **pipeline** (*Pipeline*) – The data preprocessing pipeline.
- **iterations** (*int, optional*) – The number of training iterations. Defaults to 100.
- **k** (*int, optional*) – The number of folds for cross-validation. Defaults to 5.
- **weights** (*bool, optional*) – Whether to calculate class weights. Defaults to False.
- **smote** (*bool, optional*) – Whether to apply SMOTE oversampling. Defaults to False.
- **kbest** (*bool, optional*) – Whether to select kbest features. Defaults to False.
- **kbestNum** (*int, optional*) – Number of features to select if kbest=True. Defaults to 0.
- **dim_red** (*str, optional*) – The dimensionality reduction technique. Valid is 'pca' and 'agglo'. Defaults to 'pca'.
- **dimension** (*int, optional*) – The reduced dimension size. Defaults to 6.

Returns
None

predict(*X: tensor*) → LongTensor

Predict the targets of features using the classifier. It returns the predicted targets.

Parameters
X (*torch.tensor*) – Features that should be used for the classification.

Returns
Predicted targets.

Return type
torch.LongTensor

train(*nepochs: int*, *data_wrapper:* DataWrapper, *callback: Callable | None = None*) → None

Trains the classifier for a specified number of epochs using the provided data wrapper.

Parameters

- **nepochs** (*int*) – The number of epochs to train the classifier.
- **data_wrapper** (DataWrapper) – The data wrapper containing the training data.
- **callback** (*Optional[Callable], optional*) – A callback function for tracking the training progress. Defaults to None.

Returns
None

valid(*data_wrapper:* DataWrapper) → Tuple[float, float]

Perform validation on the given data using the trained classifier.

Parameters

- **data_wrapper** (DataWrapper) – The data wrapper object containing the input features and target labels.
- **weights** (*np.ndarray, optional*) – Weights for the classifier. Defaults to None.

Returns

A tuple containing the validation loss
and accuracy.

Return type
Tuple[float, float]

class lung_cancer.**QNNClassifier**(*feature_dimension: int*, *feature_map: QuantumCircuit | Instruction*, *ansatz: QuantumCircuit | Instruction*, *optimizer: Optimizer*, *loss: Loss*, *output_shape: int*, *parity: Callable*, *primitive: Sampler | None = None*)

Initializes a Quantum Neural Network (QNN) Classifier.

Parameters

- **feature_dimension** (*int*) – The dimensionality of the input features.
- **output_shape** (*int*) – The number of classes for classification.
- **feature_map** (*Union[QuantumCircuit, Instruction]*) – The feature map circuit or instruction for the QNN.
- **ansatz** (*Union[QuantumCircuit, Instruction]*) – The ansatz circuit or instruction for the QNN.
- **parity** (*Callable*) – The interpret function for the classifier.
- **optimizer** (*Optimizer*) – The optimizer for training the QNN.
- **loss** (*Loss*) – The loss function for the QNN.
- **qnn_type** (*str, optional*) – The type of QNN, either 'sampler' or 'estimator'. Defaults to 'sampler'.
- **primitive** (*Optional[Union[Sampler, Estimator]], optional*) – The primitive object for the QNN. Defaults to None.

Raises
ValueError – If qnn_type is not 'sampler' or 'estimator'.

calc_metrics(*data_wrapper:* DataWrapper)

Shows different metrics of the classifier using the dataset specified as an argument. It shows Accuracy, Recall, Precision, ROC-AUC score and a confusion matrix.

Parameters
data_wrapper (DataWrapper) – The dataset that should be used for calculating the metrics.

cross_validate(*features: DataFrame*, *labels: Series*, *pipeline: Pipeline*, *k: int | None = 5*, *weights: bool | None = False*, *smote: bool | None = False*, *kbest: bool | None = False*, *kbestNum: int | None = 0*, *dim_red: str | None = 'pca'*, *dimension: int | None = 6*) → None

Perform cross-validation on the given dataset using the specified parameters.

Parameters

- **features** (*pd.DataFrame*) – The input features as a NumPy array.
- **labels** (*pd.Series*) – The target labels as a NumPy array.
- **pipeline** (*Pipeline*) – The data preprocessing pipeline.
- **k** (*int, optional*) – The number of folds for cross-validation. Defaults to 5.
- **weights** (*bool, optional*) – Whether to calculate class weights. Defaults to False.
- **smote** (*bool, optional*) – Whether to apply SMOTE oversampling. Defaults to False.
- **kbest** (*bool, optional*) – Whether to select kbest features. Defaults to False.
- **kbestNum** (*int, optional*) – Number of features to select if kbest=True. Defaults to 0.

- **dim_red** (*str, optional*) – The dimensionality reduction technique. Valid is 'pca' and 'agglo'. Defaults to 'pca'.
- **dimension** (*int, optional*) – The reduced dimension size. Defaults to 6.

Returns
None

get_bit_strings_probs(*features: DataFrame*) → ndarray

This function returns the probability of each bitstring.

Parameters
features (*pd.DataFrame*) – Input features.

Returns
Probabilities of each bitstring.

Return type
np.ndarray

get_feature_importance(*features: DataFrame*, *labels: Series*, *pipeline: Pipeline*, *num_features_red: int*, *dim_red: str = 'agglo'*) → ndarray

This function returns the feature importance of the trained model.

Parameters

- **features** (*pd.DataFrame*) – Input features.
- **labels** (*pd.Series*) – Output labels.
- **pipeline** (*Pipeline*) – Preprocessing pipeline.
- **num_features_red** (*int*) – Number of features for dimension reduction.
- **dim_red** (*str, optional*) – Dimension reduction technique. Either 'agglo' or 'pca'. Default: 'agglo'.

Returns

Feature importance of each feature.
Order is the same as after applying the pipeline.

Return type
np.ndarray

predict(*X: ndarray*) → ndarray

Predict the targets of features using the classifier. It returns the predicted targets.

Parameters
X (*np.ndarray*) – Features that should be used for the classification.

Returns
Predicted targets.

Return type
np.ndarray

predict_proba(*X: ndarray*) → ndarray

Predict the targets of features using the classifier. It returns the probability of the sampe belonging to class 0.

Parameters
X (*np.ndarray*) – Features that should be used for the classification.

Returns
Probabilites.

Return type
np.ndarray

train(*data_wrapper:* DataWrapper, *callback: Callable | None = None*) → None

Train the classifier using the provided data.

Parameters

- **data_wrapper** (DataWrapper) – The data wrapper object containing the training data.
- **callback** (*Callable, optional*) – Optional callback function. Defaults to None.

Returns
None

valid(*data_wrapper:* DataWrapper, *weights: ndarray | None = None*) → Tuple[float, float]

Perform validation on the given data using the trained classifier.

Parameters

- **data_wrapper** (DataWrapper) – The data wrapper object containing the input features and target labels.
- **weights** (*np.ndarray, optional*) – Weights for the classifier. Defaults to None.

Returns

A tuple containing the validation loss
and accuracy.

Return type
Tuple[float, float]

class lung_cancer.**QuantumLivePlotter**(*classifier:* BaseClassifier, *data_wrapper:* DataWrapper | *None = None*)

This class helps to plot the training and validation loss and accuracy during the training process of a quantum classifier.

Parameters

- **classifier** (BaseClassifier) – The classifier to use.
- **data_wrapper** (*Optional[DataWrapper]*) – The data wrapper for validation data. Default is None.

class lung_cancer.**QubitPlotter**(*datawrapper:* DataWrapper, *feature_map: QuantumCircuit*, *num_qubits: int*, *cols: int = 5*)

Initializes a QubitPlotter object.

Parameters

- **datawrapper** (DataWrapper) – DataWrapper object containing the feature data.
- **feature_map** (*QuantumCircuit*) – Quantum circuit representing the feature map.
- **num_qubits** (*int*) – Number of qubits in the quantum circuit.
- **cols** (*int, optional*) – Number of columns in the subplot grid. Defaults to 5.

plot_qubits() → None

Plots how a feature map transforms the data onto the bloch sphere. This is done by converting the statevector to bloch sphere coordinates. The Bloch coordinates are then visualized using the Qutip library.

Returns
None

class lung_cancer.**StackedBarPlotter**(*dataset:* DatasetWrapper)

Initialize the StackedBarPlotter.

Parameters
dataset (DatasetWrapper) – The dataset wrapper object.

show(*col_x: str*, *col_y: str*) → None

Display the stacked bar chart.

Parameters

- **col_x** (*str*) – The column for the x-axis.
- **col_y** (*str*) – The column for the y-axis.

Returns
None

PYTHON MODULE INDEX

INDEX