# Scikit-plot Documentation

**Reiichiro S. Nakano**

**Aug 19, 2018**

# Contents

CHAPTER 1
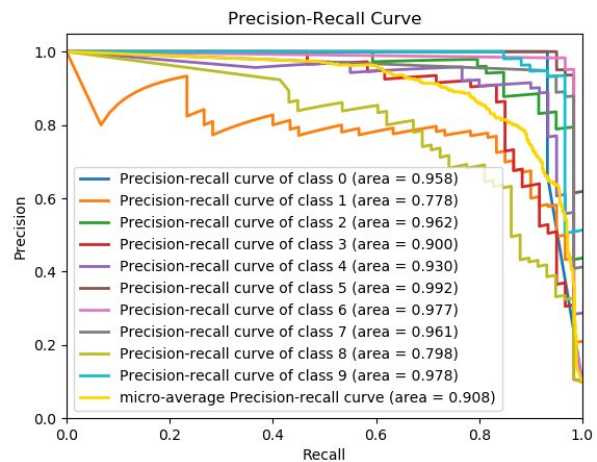
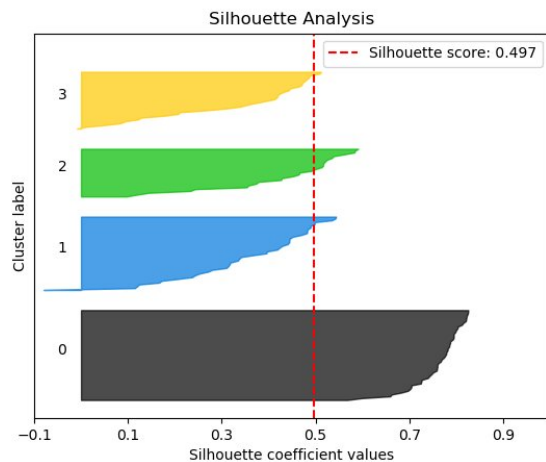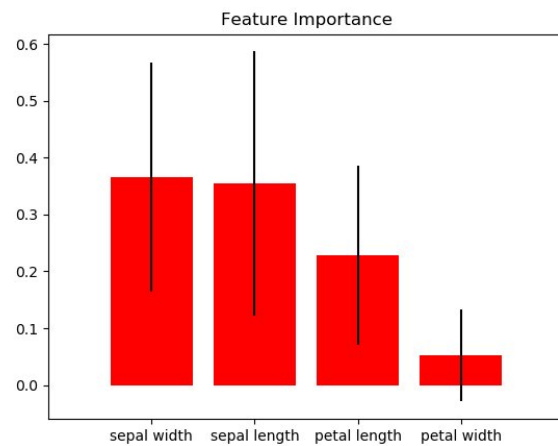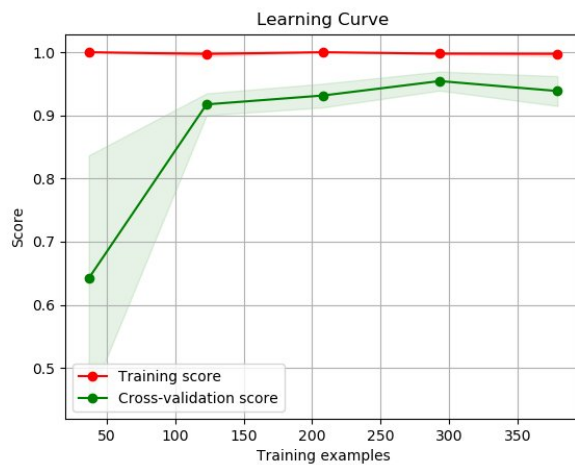The quickest and easiest way to go from analysis...

CHAPTER 2

. . . to this.

Scikit-plot is the result of an unartistic data scientist's dreadful realization that *visualization is one of the most crucial components in the data science process, not just a mere afterthought*.

Gaining insights is simply a lot easier when you're looking at a colored heatmap of a confusion matrix complete with class labels rather than a single-line dump of numbers enclosed in brackets. Besides, if you ever need to present your results to someone (virtually any time anybody hires you to do data science), you show them visualizations, not a bunch of numbers in Excel.

That said, there are a number of visualizations that frequently pop up in machine learning. Scikit-plot is a humble attempt to provide aesthetically-challenged programmers (such as myself) the opportunity to generate quick and beautiful graphs and plots with as little boilerplate as possible.

## 2.1 First steps with Scikit-plot

Eager to use Scikit-plot? Let's get started! This section of the documentation will teach you the basic philosophy behind Scikit-plot by running you through a quick example.

### 2.1.1 Installation

Before anything else, make sure you've installed the latest version of Scikit-plot. Scikit-plot is on PyPi, so simply run:

```
$ pip install scikit-plot
```

to install the latest version.

Alternatively, you can clone the source repository and run:

```
$ python setup.py install
```

at the root folder.

Scikit-plot depends on Scikit-learn and Matplotlib to do its magic, so make sure you have them installed as well.

### 2.1.2 Your First Plot

For our quick example, let's show how well a Random Forest can classify the digits dataset bundled with Scikit-learn. A popular way to evaluate a classifier's performance is by viewing its confusion matrix.

Before we begin plotting, we'll need to import the following for Scikit-plot:

```
>>> import matplotlib.pyplot as plt
```

`matplotlib.pyplot` is used by Matplotlib to make plotting work like it does in MATLAB and deals with things like axes, figures, and subplots. But don't worry. Unless you're an advanced user, you won't need to understand any of that while using Scikit-plot. All you need to remember is that we use the `matplotlib.pyplot.show()` function to show any plots generated by Scikit-plot.

Let's begin by generating our sample digits dataset:

```
>>> from sklearn.datasets import load_digits
>>> X, y = load_digits(return_X_y=True)
```

Here, `X` and `y` contain the features and labels of our classification dataset, respectively.

We'll proceed by creating an instance of a RandomForestClassifier object from Scikit-learn with some initial parameters:

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> random_forest_clf = RandomForestClassifier(n_estimators=5, max_depth=5, random_
→state=1)
```

Let's use `sklearn.model_selection.cross_val_predict()` to generate predicted labels on our dataset:

```
>>> from sklearn.model_selection import cross_val_predict
>>> predictions = cross_val_predict(random_forest_clf, X, y)
```

For those not familiar with what `cross_val_predict()` does, it generates cross-validated estimates for each sample point in our dataset. Comparing the cross-validated estimates with the true labels, we'll be able to get evaluation metrics such as accuracy, precision, recall, and in our case, the confusion matrix.

To plot and show our confusion matrix, we'll use the function *plot_confusion_matrix()*, passing it both the true labels and predicted labels. We'll also set the optional argument `normalize=True` so the values displayed in our confusion matrix plot will be from the range [0, 1]. Finally, to show our plot, we'll call `plt.show()`.

```
>>> import scikitplot as skplt
>>> skplt.metrics.plot_confusion_matrix(y, predictions, normalize=True)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```
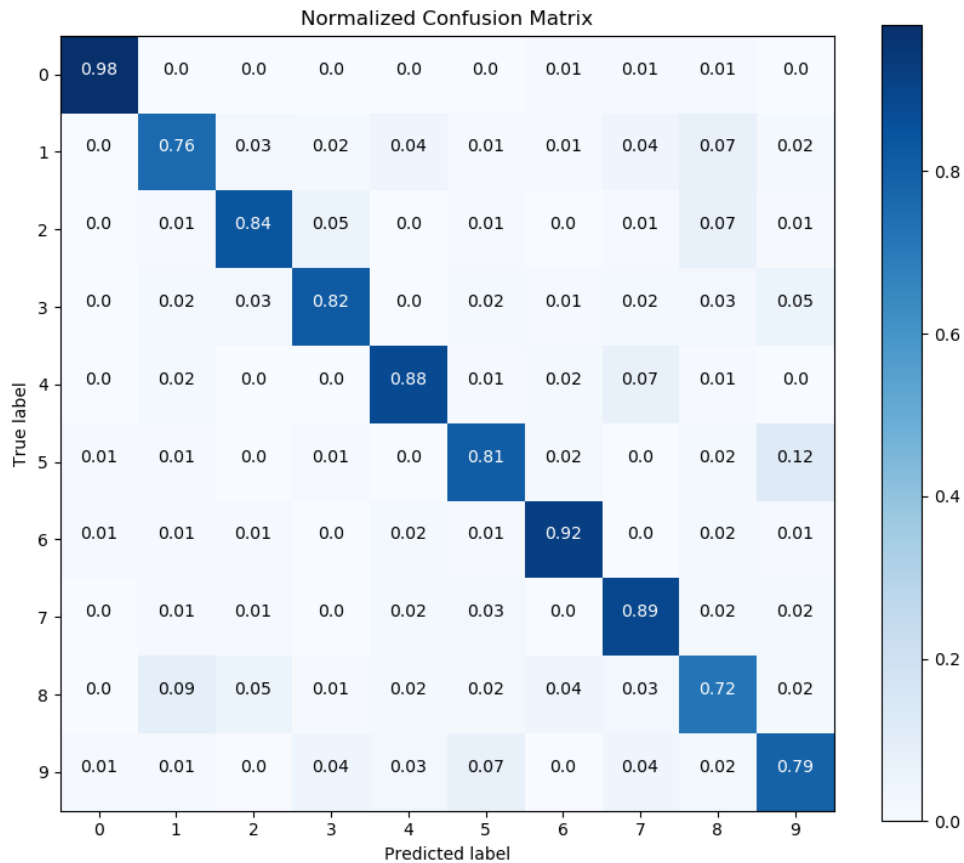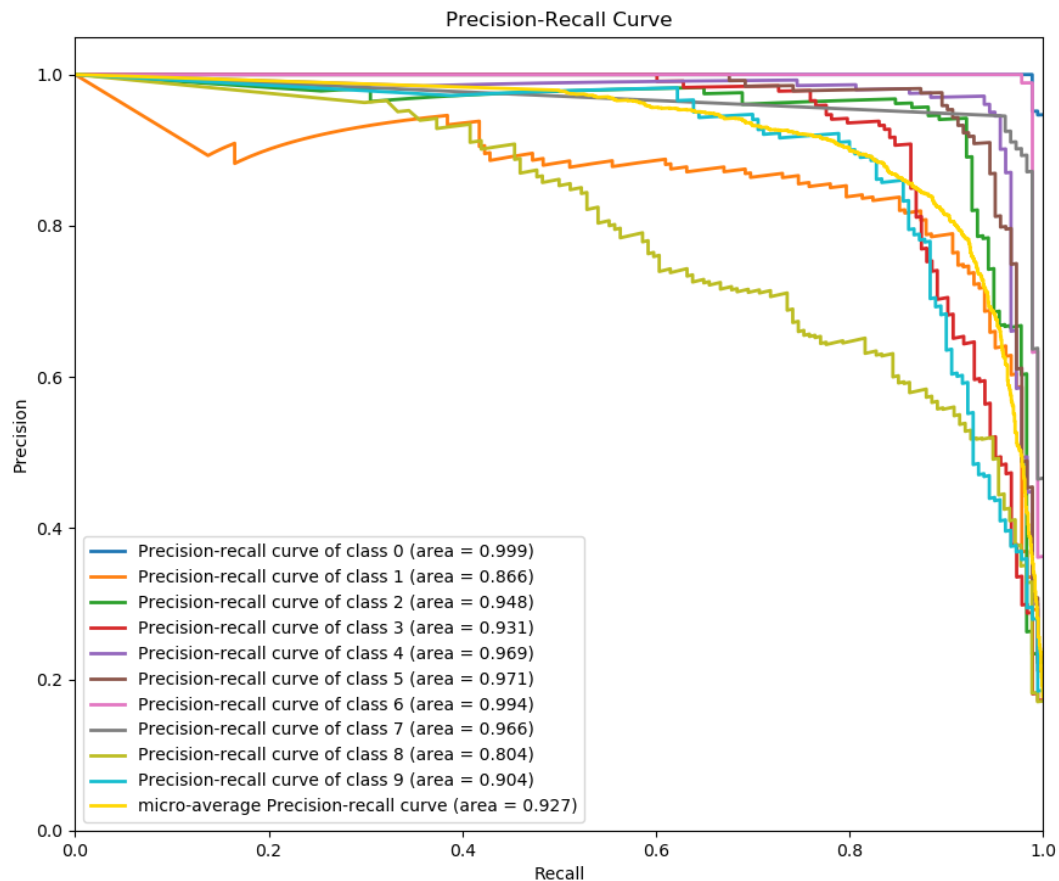
And that's it! A quick glance of our confusion matrix shows that our classifier isn't doing so well with identifying the digits 1, 8, and 9. Hmm. Perhaps a bit more tweaking of our Random Forest's hyperparameters is in order.

### 2.1.3 One more example

Finally, let's show an example wherein we *don't* use Scikit-learn.

Here's a quick example to generate the precision-recall curves of a Keras classifier on a sample dataset.

```
>>> # Import what's needed for the Functions API
>>> import matplotlib.pyplot as plt
>>> import scikitplot as skplt
>>> # This is a Keras classifier. We'll generate probabilities on the test set.
>>> keras_clf.fit(X_train, y_train, batch_size=64, nb_epoch=10, verbose=2)
>>> probas = keras_clf.predict_proba(X_test, batch_size=64)
>>> # Now plot.
>>> skplt.metrics.plot_precision_recall_curve(y_test, probas)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

Precision-Recall Curve

| Precision-recall curve of class 0 (area = 0.999) |
| Precision-recall curve of class 1 (area = 0.866) |
| Precision-recall curve of class 2 (area = 0.948) |
| Precision-recall curve of class 3 (area = 0.931) |
| Precision-recall curve of class 4 (area = 0.969) |
| Precision-recall curve of class 5 (area = 0.971) |
| Precision-recall curve of class 6 (area = 0.994) |
| Precision-recall curve of class 7 (area = 0.966) |
| Precision-recall curve of class 8 (area = 0.804) |
| Precision-recall curve of class 9 (area = 0.904) |
| micro-average Precision-recall curve (area = 0.927) |

And again, that's it! As in the example above, all we needed to do was pass the ground truth labels and predicted probabilities to `plot_precision_recall_curve()` to generate the precision-recall curves. This means you can use literally any classifier you want to generate the precision-recall curves, from Keras classifiers to NLTK Naive Bayes to XGBoost, as long as you pass in the predicted probabilities in the correct format.

### 2.1.4 Now what?

The recommended way to start using Scikit-plot is to just go through the documentation for the various modules and choose which plots you think would be useful for your work.

Happy plotting!

## 2.2 Metrics Module (API Reference)

The *scikitplot.metrics* module includes plots for machine learning evaluation metrics e.g. confusion matrix, silhouette scores, etc.

```
scikitplot.metrics.plot_confusion_matrix(y_true, y_pred, labels=None, true_labels=None,
                                         pred_labels=None,    title=None,    normal-
                                         ize=False, hide_zeros=False, hide_counts=False,
                                         x_tick_rotation=0,    ax=None,    figsize=None,
                                         cmap='Blues',                title_fontsize='large',
                                         text_fontsize='medium')
```

Generates confusion matrix plot from predictions and true labels

> **Parameters**
>
> - **y_true** (`array-like, shape (n_samples)`) – Ground truth (correct) target values.
>
> - **y_pred** (`array-like, shape (n_samples)`) – Estimated targets as returned by a classifier.
>
> - **labels** (`array-like, shape (n_classes), optional`) – List of labels to index the matrix. This may be used to reorder or select a subset of labels. If none is given, those that appear at least once in `y_true` or `y_pred` are used in sorted order. (new in v0.2.5)
>
> - **true_labels** (`array-like, optional`) – The true labels to display. If none is given, then all of the labels are used.
>
> - **pred_labels** (`array-like, optional`) – The predicted labels to display. If none is given, then all of the labels are used.
>
> - **title** (`string, optional`) – Title of the generated plot. Defaults to "Confusion Matrix" if *normalize* is True. Else, defaults to "Normalized Confusion Matrix.
>
> - **normalize** (`bool, optional`) – If True, normalizes the confusion matrix before plotting. Defaults to False.
>
> - **hide_zeros** (`bool, optional`) – If True, does not plot cells containing a value of zero. Defaults to False.
>
> - **hide_counts** (`bool, optional`) – If True, doe not overlay counts. Defaults to False.
>
> - **x_tick_rotation** (`int, optional`) – Rotates x-axis tick labels by the specified angle. This is useful in cases where there are numerous categories and the labels overlap each other.
>
> - **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.
>
> - **figsize** (`2-tuple, optional`) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.
>
> - **cmap** (string or `matplotlib.colors.Colormap` instance, optional) – Colormap used for plotting the projection. View Matplotlib Colormap documentation for available options. https://matplotlib.org/users/colormaps.html
>
> - **title_fontsize** (`string or int, optional`) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".
>
> - **text_fontsize** (`string or int, optional`) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
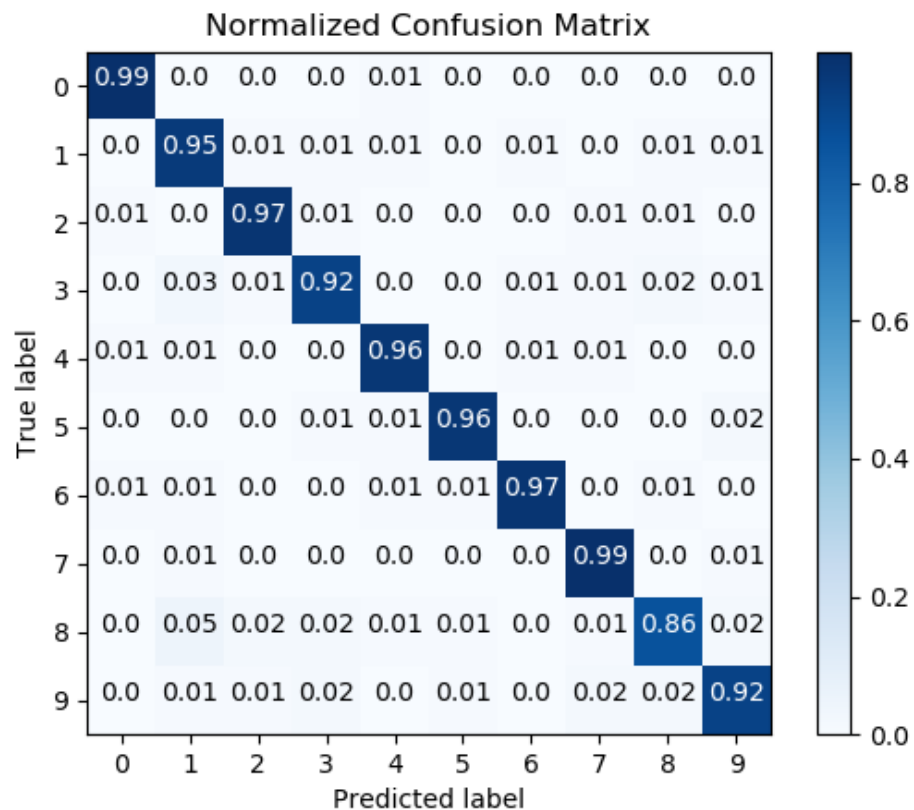>
> **Returns**
>
> > **The axes on which the plot was** drawn.
>
> **Return type** ax (`matplotlib.axes.Axes`)

**Example**

```
>>> import scikitplot as skplt
>>> rf = RandomForestClassifier()
>>> rf = rf.fit(X_train, y_train)
>>> y_pred = rf.predict(X_test)
>>> skplt.metrics.plot_confusion_matrix(y_test, y_pred, normalize=True)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```



scikitplot.metrics.**plot_roc**(*y_true, y_probas, title='ROC Curves', plot_micro=True, plot_macro=True, classes_to_plot=None, ax=None, figsize=None, cmap='nipy_spectral', title_fontsize='large', text_fontsize='medium'*)

Generates the ROC curves from labels and predicted scores/probabilities

> **Parameters**
>
> - **y_true** (*array-like, shape (n_samples)*) – Ground truth (correct) target values.
>
> - **y_probas** (*array-like, shape (n_samples, n_classes)*) – Prediction probabilities for each class returned by a classifier.
>
> - **title** (*string, optional*) – Title of the generated plot. Defaults to "ROC Curves".
>
> - **plot_micro** (*boolean, optional*) – Plot the micro average ROC curve. Defaults to True.

- **plot_macro** (*boolean, optional*) – Plot the macro average ROC curve. Defaults to `True`.

- **classes_to_plot** (*list-like, optional*) – Classes for which the ROC curve should be plotted. e.g. [0, 'cold']. If given class does not exist, it will be ignored. If `None`, all classes will be plotted. Defaults to `None`

- **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.

- **cmap** (string or `matplotlib.colors.Colormap` instance, optional) – Colormap used for plotting the projection. View Matplotlib Colormap documentation for available options. https://matplotlib.org/users/colormaps.html

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
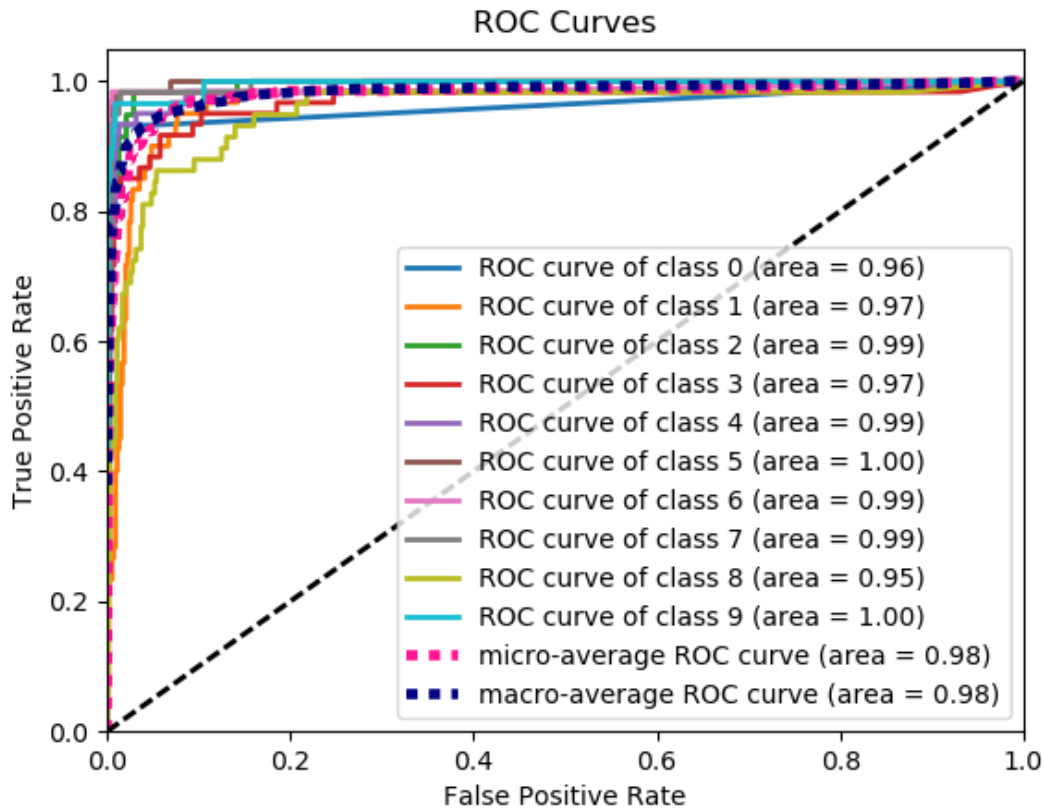
Returns

> **The axes on which the plot was** drawn.

**Return type** ax (`matplotlib.axes.Axes`)

**Example**

```
>>> import scikitplot as skplt
>>> nb = GaussianNB()
>>> nb = nb.fit(X_train, y_train)
>>> y_probas = nb.predict_proba(X_test)
>>> skplt.metrics.plot_roc(y_test, y_probas)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

scikitplot.metrics.**plot_ks_statistic**(*y_true*, *y_probas*, *title='KS Statistic Plot'*, *ax=None*, *figsize=None*, *title_fontsize='large'*, *text_fontsize='medium'*)

Generates the KS Statistic plot from labels and scores/probabilities

**Parameters**

- **y_true** (*array-like, shape (n_samples)*) – Ground truth (correct) target values.

- **y_probas** (*array-like, shape (n_samples, n_classes)*) – Prediction probabilities for each class returned by a classifier.

- **title** (*string, optional*) – Title of the generated plot. Defaults to "KS Statistic Plot".

- **ax** (matplotlib.axes.Axes, optional) – The axes upon which to plot the learning curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to None.

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
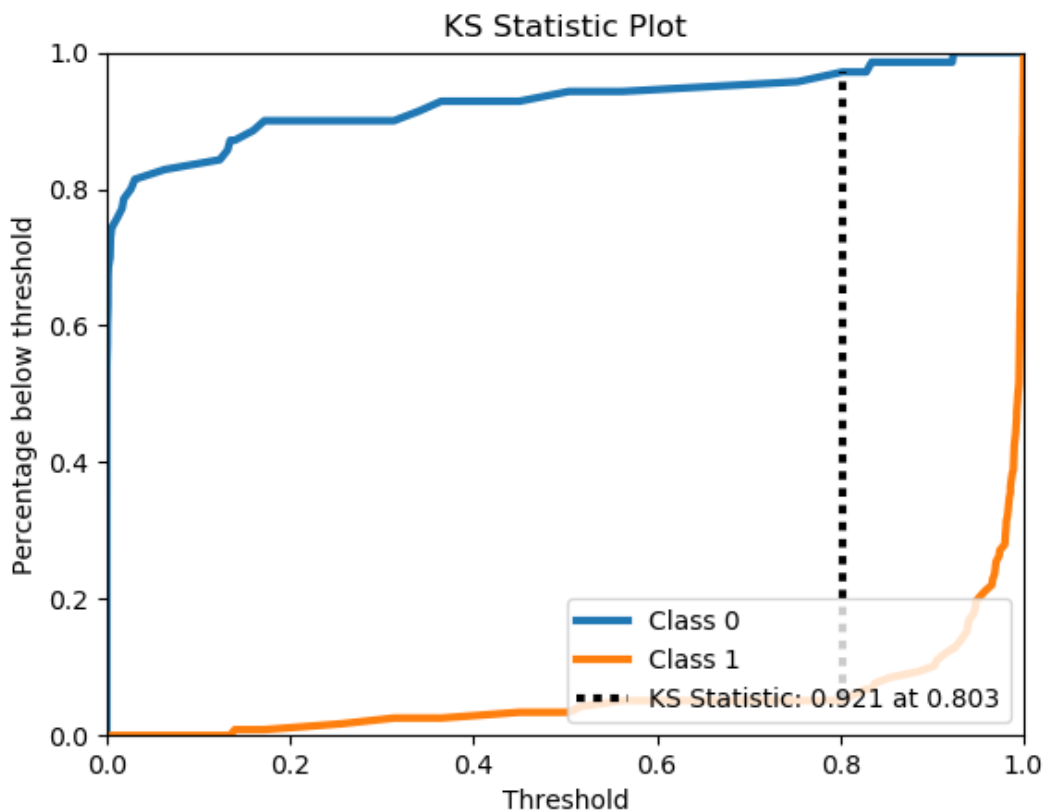
**Returns**

**The axes on which the plot was** drawn.

---

**Return type** ax (`matplotlib.axes.Axes`)

**Example**

```
>>> import scikitplot as skplt
>>> lr = LogisticRegression()
>>> lr = lr.fit(X_train, y_train)
>>> y_probas = lr.predict_proba(X_test)
>>> skplt.metrics.plot_ks_statistic(y_test, y_probas)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```



scikitplot.metrics.**plot_precision_recall**(*y_true*, *y_probas*, *title='Precision-Recall Curve'*, *plot_micro=True*, *classes_to_plot=None*, *ax=None*, *figsize=None*, *cmap='nipy_spectral'*, *title_fontsize='large'*, *text_fontsize='medium'*)

Generates the Precision Recall Curve from labels and probabilities

> **Parameters**
>
> - **y_true** (*array-like, shape (n_samples)*) – Ground truth (correct) target values.
>
> - **y_probas** (*array-like, shape (n_samples, n_classes)*) – Prediction probabilities for each class returned by a classifier.

- **title** (*string, optional*) – Title of the generated plot. Defaults to "Precision-Recall curve".

- **plot_micro** (*boolean, optional*) – Plot the micro average ROC curve. Defaults to `True`.

- **classes_to_plot** (*list-like, optional*) – Classes for which the precision-recall curve should be plotted. e.g. [0, 'cold']. If given class does not exist, it will be ignored. If `None`, all classes will be plotted. Defaults to `None`.

- **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.

- **cmap** (string or `matplotlib.colors.Colormap` instance, optional) – Colormap used for plotting the projection. View Matplotlib Colormap documentation for available options. https://matplotlib.org/users/colormaps.html

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
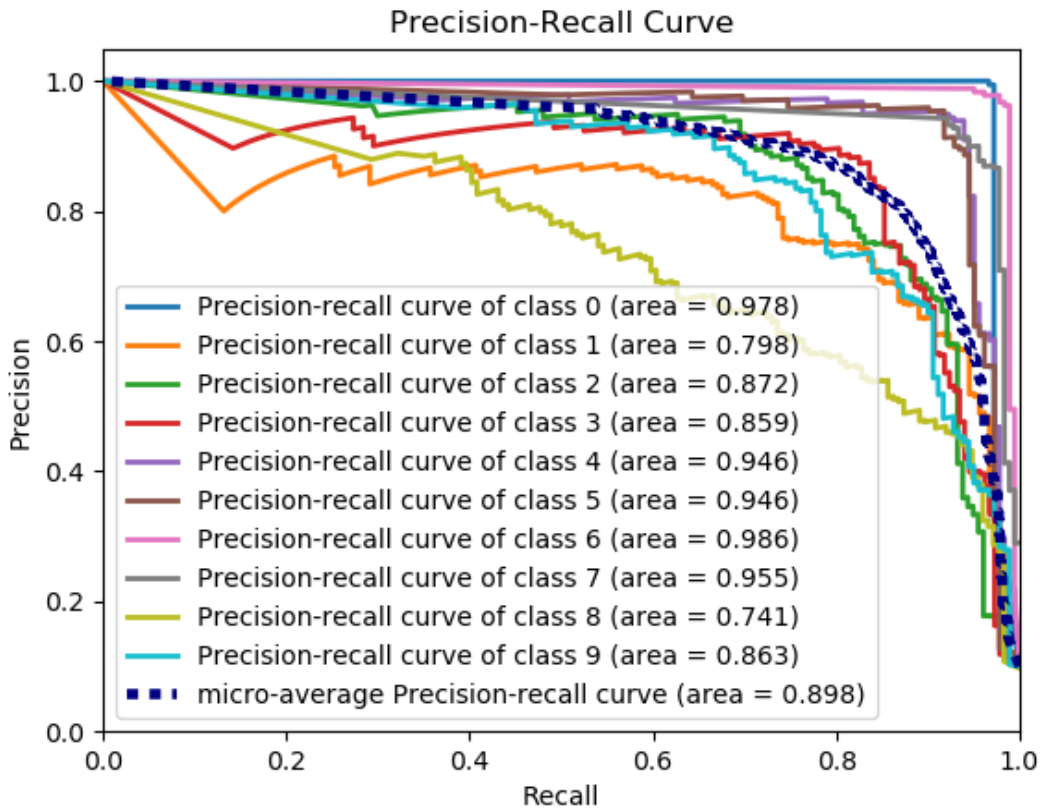
**Returns**

> The axes on which the plot was drawn.

**Return type** ax (`matplotlib.axes.Axes`)

**Example**

```
>>> import scikitplot as skplt
>>> nb = GaussianNB()
>>> nb.fit(X_train, y_train)
>>> y_probas = nb.predict_proba(X_test)
>>> skplt.metrics.plot_precision_recall(y_test, y_probas)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

scikitplot.metrics.**plot_silhouette**(*X*, *cluster_labels*, *title='Silhouette Analysis'*, *met-*
*ric='euclidean'*, *copy=True*, *ax=None*, *figsize=None*,
*cmap='nipy_spectral'*, *title_fontsize='large'*,
*text_fontsize='medium'*)

Plots silhouette analysis of clusters provided.

> **Parameters**
>
> - **X** (`array-like, shape (n_samples, n_features)`) – Data to cluster, where n_samples is the number of samples and n_features is the number of features.
>
> - **cluster_labels** (`array-like, shape (n_samples,)`) – Cluster label for each sample.
>
> - **title** (`string, optional`) – Title of the generated plot. Defaults to "Silhouette Analysis"
>
> - **metric** (`string or callable, optional`) – The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by sklearn.metrics.pairwise.pairwise_distances. If X is the distance array itself, use "precomputed" as the metric.
>
> - **copy** (`boolean, optional`) – Determines whether `fit` is used on **clf** or on a copy of **clf**.
>
> - **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.
>
> - **figsize** (`2-tuple, optional`) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.

- **cmap** (string or `matplotlib.colors.Colormap` instance, optional) – Colormap used for plotting the projection. View Matplotlib Colormap documentation for available options. https://matplotlib.org/users/colormaps.html

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
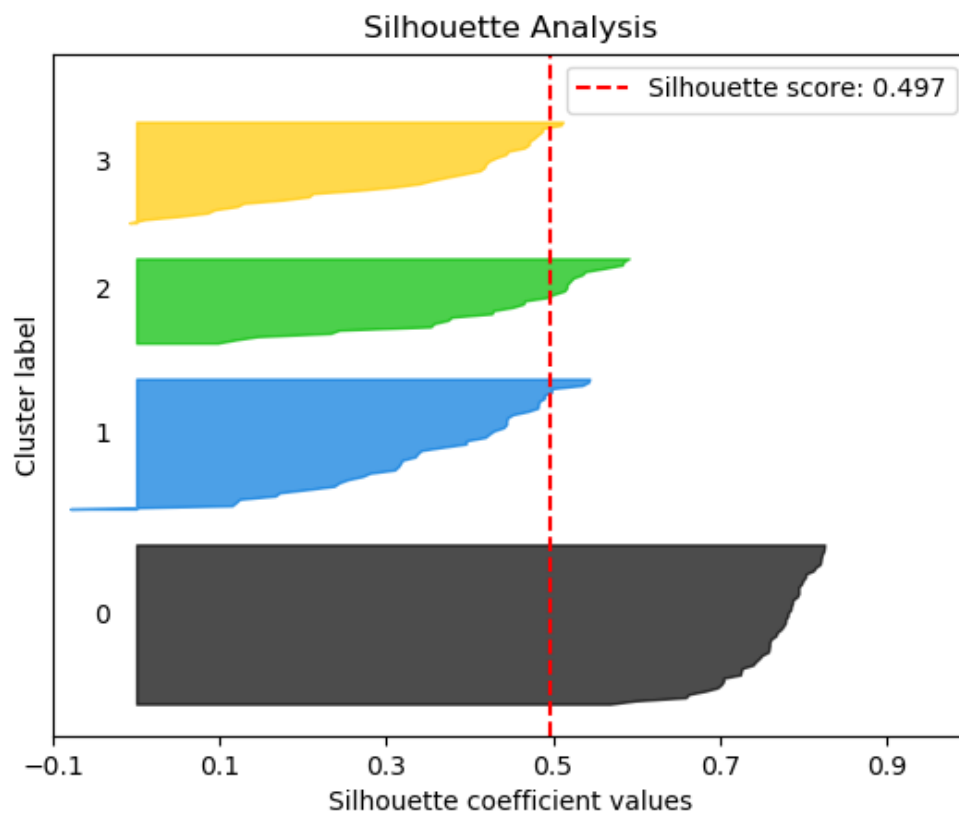
**Returns**

The axes on which the plot was drawn.

**Return type** ax (`matplotlib.axes.Axes`)

**Example**

```
>>> import scikitplot as skplt
>>> kmeans = KMeans(n_clusters=4, random_state=1)
>>> cluster_labels = kmeans.fit_predict(X)
>>> skplt.metrics.plot_silhouette(X, cluster_labels)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

scikitplot.metrics.**plot_calibration_curve**(*y_true*, *probas_list*, *clf_names=None*, *n_bins=10*, *title='Calibration plots (Reliability Curves)'*, *ax=None*, *figsize=None*, *cmap='nipy_spectral'*, *title_fontsize='large'*, *text_fontsize='medium'*)

Plots calibration curves for a set of classifier probability estimates.

Plotting the calibration curves of a classifier is useful for determining whether or not you can interpret their predicted probabilities directly as as confidence level. For instance, a well-calibrated binary classifier should classify the samples such that for samples to which it gave a score of 0.8, around 80% should actually be from the positive class.

This function currently only works for binary classification.

> **Parameters**
>
> - **y_true** (`array-like, shape (n_samples)`) – Ground truth (correct) target values.
>
> - **probas_list** (`list of array-like, shape (n_samples, 2) or (n_samples,)`) – A list containing the outputs of binary classifiers' `predict_proba()` method or `decision_function()` method.
>
> - **clf_names** (`list of str, optional`) – A list of strings, where each string refers to the name of the classifier that produced the corresponding probability estimates in *probas_list*. If `None`, the names "Classifier 1", "Classifier 2", etc. will be used.
>
> - **n_bins** (`int, optional`) – Number of bins. A bigger number requires more data.
>
> - **title** (`string, optional`) – Title of the generated plot. Defaults to "Calibration plots (Reliabilirt Curves)"
>
> - **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.
>
> - **figsize** (`2-tuple, optional`) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.
>
> - **cmap** (string or `matplotlib.colors.Colormap` instance, optional) – Colormap used for plotting the projection. View Matplotlib Colormap documentation for available options. https://matplotlib.org/users/colormaps.html
>
> - **title_fontsize** (`string or int, optional`) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".
>
> - **text_fontsize** (`string or int, optional`) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
>
> **Returns** The axes on which the plot was drawn.
>
> **Return type** `matplotlib.axes.Axes`

### Example

```
>>> import scikitplot as skplt
>>> rf = RandomForestClassifier()
>>> lr = LogisticRegression()
>>> nb = GaussianNB()
>>> svm = LinearSVC()
>>> rf_probas = rf.fit(X_train, y_train).predict_proba(X_test)
```

(continues on next page)

```
>>> lr_probas = lr.fit(X_train, y_train).predict_proba(X_test)
>>> nb_probas = nb.fit(X_train, y_train).predict_proba(X_test)
>>> svm_scores = svm.fit(X_train, y_train).decision_function(X_test)
>>> probas_list = [rf_probas, lr_probas, nb_probas, svm_scores]
>>> clf_names = ['Random Forest', 'Logistic Regression',
...              'Gaussian Naive Bayes', 'Support Vector Machine']
>>> skplt.metrics.plot_calibration_curve(y_test,
...                                      probas_list,
...                                      clf_names)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```
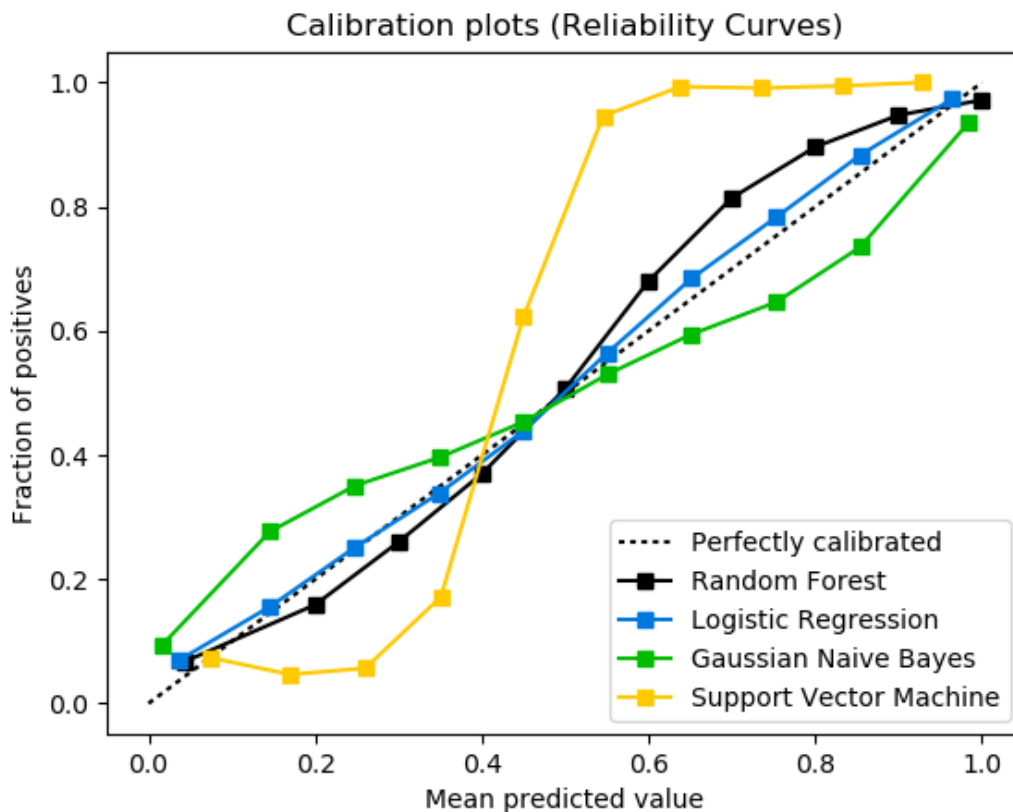


scikitplot.metrics.**plot_cumulative_gain**(*y_true*, *y_probas*, *title='Cumulative Gains Curve'*, *ax=None*, *figsize=None*, *title_fontsize='large'*, *text_fontsize='medium'*)

Generates the Cumulative Gains Plot from labels and scores/probabilities

The cumulative gains chart is used to determine the effectiveness of a binary classifier. A detailed explanation can be found at http://mlwiki.org/index.php/Cumulative_Gain_Chart. The implementation here works only for binary classification.

> **Parameters**
>
> - **y_true** (*array-like, shape (n_samples)*) – Ground truth (correct) target values.
>
> - **y_probas** (*array-like, shape (n_samples, n_classes)*) – Prediction

probabilities for each class returned by a classifier.

- **title** (*string, optional*) – Title of the generated plot. Defaults to "Cumulative Gains Curve".

- **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the learning curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
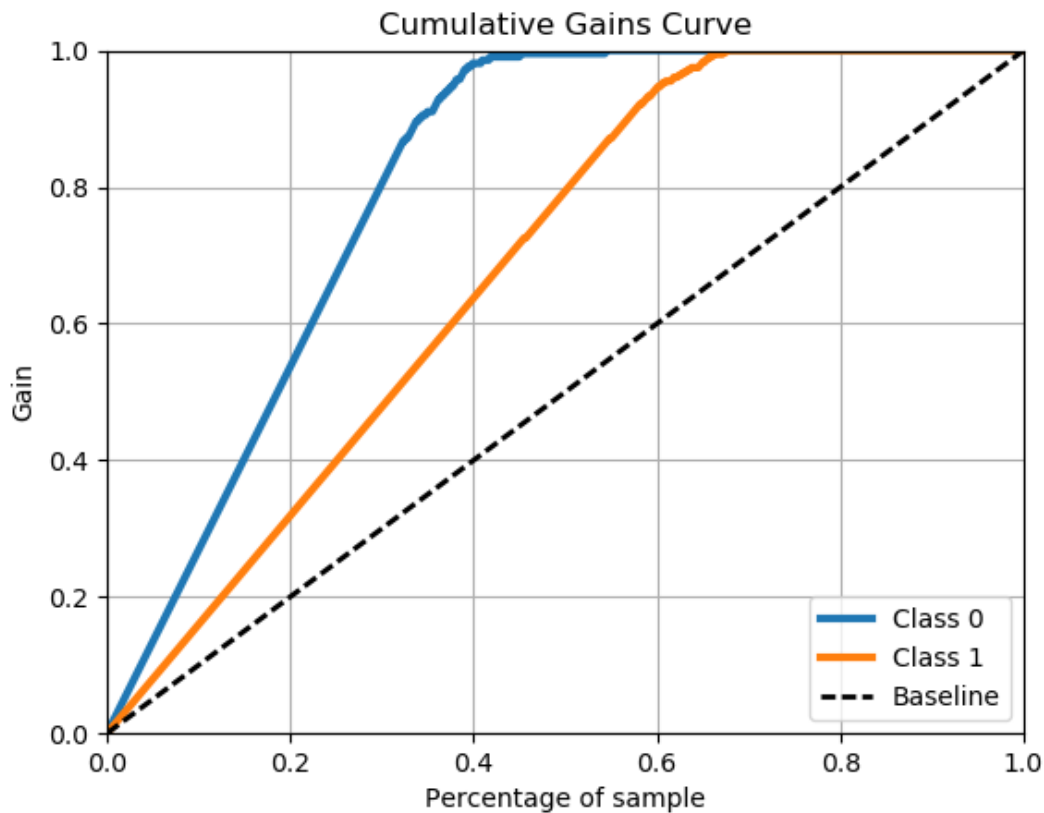
**Returns**

The axes on which the plot was drawn.

**Return type** ax (`matplotlib.axes.Axes`)

**Example**

```
>>> import scikitplot as skplt
>>> lr = LogisticRegression()
>>> lr = lr.fit(X_train, y_train)
>>> y_probas = lr.predict_proba(X_test)
>>> skplt.metrics.plot_cumulative_gain(y_test, y_probas)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

scikitplot.metrics.**plot_lift_curve**(*y_true*,         *y_probas*,        *title='Lift*        *Curve'*,           *ax=None*,      *figsize=None*,      *title_fontsize='large'*,       *text_fontsize='medium'*)

Generates the Lift Curve from labels and scores/probabilities

The lift curve is used to determine the effectiveness of a binary classifier. A detailed explanation can be found at [http://www2.cs.uregina.ca/~dbd/cs831/notes/lift_chart/lift_chart.html](http://www2.cs.uregina.ca/~dbd/cs831/notes/lift_chart/lift_chart.html). The implementation here works only for binary classification.

> **Parameters**
>
> - **y_true** (*array-like, shape (n_samples)*) – Ground truth (correct) target values.
>
> - **y_probas** (*array-like, shape (n_samples, n_classes)*) – Prediction probabilities for each class returned by a classifier.
>
> - **title** (*string, optional*) – Title of the generated plot. Defaults to "Lift Curve".
>
> - **ax** (*matplotlib.axes.Axes, optional*) – The axes upon which to plot the learning curve. If None, the plot is drawn on a new set of axes.
>
> - **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to None.
>
> - **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".
>
> - **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
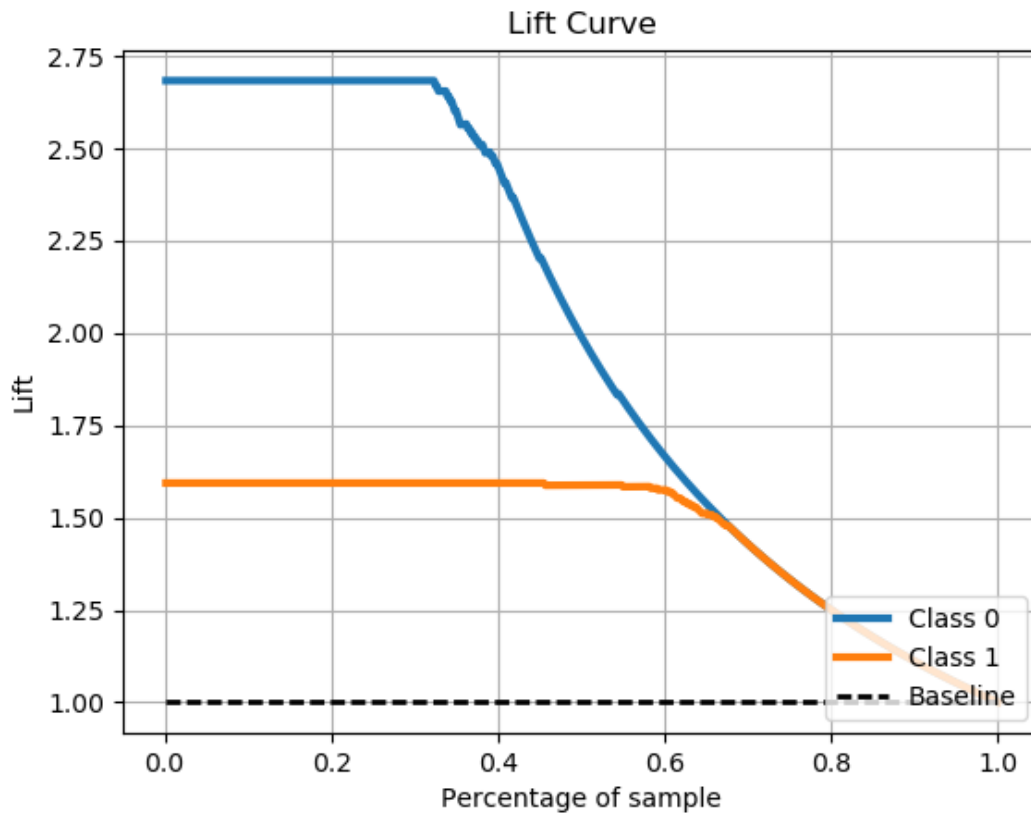
**Returns**

> **The axes on which the plot was** drawn.

**Return type** `ax (matplotlib.axes.Axes)`

**Example**

```
>>> import scikitplot as skplt
>>> lr = LogisticRegression()
>>> lr = lr.fit(X_train, y_train)
>>> y_probas = lr.predict_proba(X_test)
>>> skplt.metrics.plot_lift_curve(y_test, y_probas)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```



## 2.3 Estimators Module (API Reference)

The `scikitplot.estimators` module includes plots built specifically for scikit-learn estimator (classifier/regressor) instances e.g. Random Forest. You can use your own estimators, but these plots assume specific properties shared by scikit-learn estimators. The specific requirements are documented per function.

scikitplot.estimators.**plot_learning_curve**(*clf*, *X*, *y*, *title='Learning Curve'*, *cv=None*, *shuffle=False*, *random_state=None*, *train_sizes=None*, *n_jobs=1*, *scoring=None*, *ax=None*, *figsize=None*, *title_fontsize='large'*, *text_fontsize='medium'*)

Generates a plot of the train and test learning curves for a classifier.

**Parameters**

- **clf** – Classifier instance that implements `fit` and `predict` methods.

- **X** (*array-like, shape (n_samples, n_features)*) – Training vector, where n_samples is the number of samples and n_features is the number of features.

- **y** (*array-like, shape (n_samples) or (n_samples, n_features)*) – Target relative to X for classification or regression; None for unsupervised learning.

- **title** (*string, optional*) – Title of the generated plot. Defaults to "Learning Curve"

- **cv** (*int, cross-validation generator, iterable, optional*) – Determines the cross-validation strategy to be used for splitting.

  **Possible inputs for cv are:**

  – None, to use the default 3-fold cross-validation,

  – integer, to specify the number of folds.

  – An object to be used as a cross-validation generator.

  – An iterable yielding train/test splits.

  For integer/None inputs, if `y` is binary or multiclass, `StratifiedKFold` used. If the estimator is not a classifier or if `y` is neither binary nor multiclass, `KFold` is used.

- **shuffle** (*bool, optional*) – Used when do_cv is set to True. Determines whether to shuffle the training data before splitting using cross-validation. Default set to True.

- **random_state** (int `RandomState`) – Pseudo-random number generator state used for random sampling.

- **train_sizes** (*iterable, optional*) – Determines the training sizes used to plot the learning curve. If None, `np.linspace(.1, 1.0, 5)` is used.

- **n_jobs** (*int, optional*) – Number of jobs to run in parallel. Defaults to 1.

- **scoring** (*string, callable or None, optional*) – default: None A string (see scikit-learn model evaluation documentation) or a scorerbcallable object / function with signature scorer(estimator, X, y).

- **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
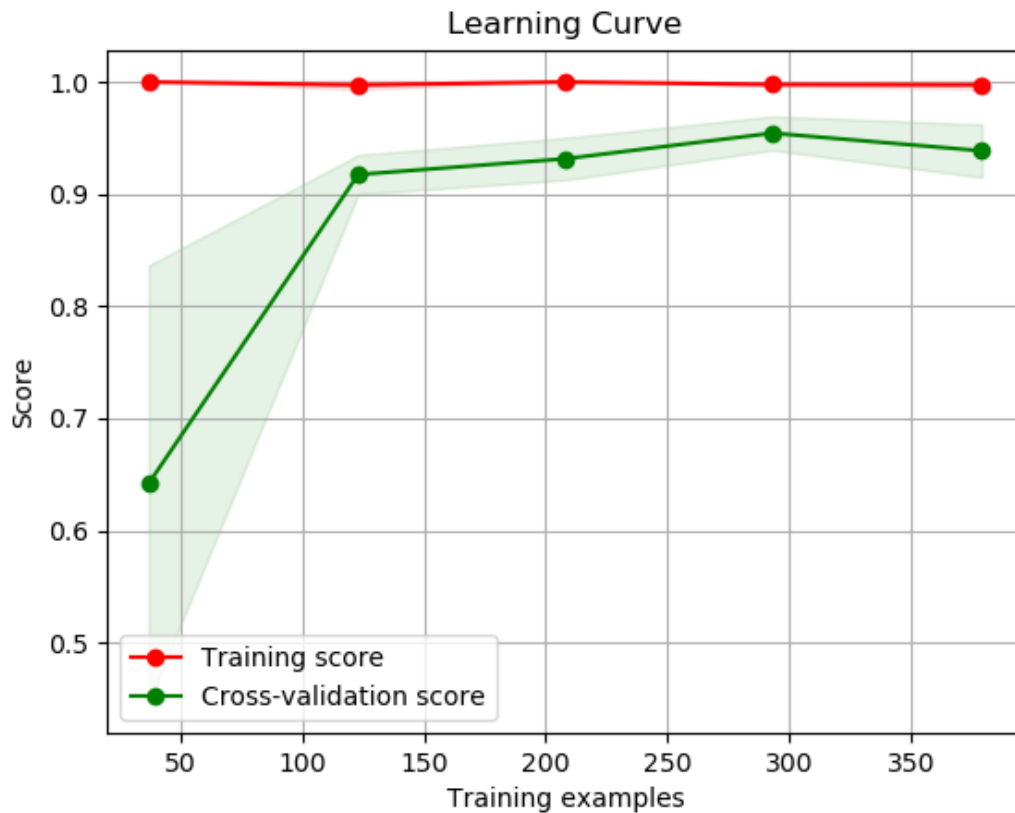
**Returns**

**The axes on which the plot was** drawn.

**Return type** ax (`matplotlib.axes.Axes`)

**Example**

```
>>> import scikitplot as skplt
>>> rf = RandomForestClassifier()
>>> skplt.estimators.plot_learning_curve(rf, X, y)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```



scikitplot.estimators.**plot_feature_importances**(*clf, title='Feature Importance', feature_names=None, max_num_features=20, order='descending', x_tick_rotation=0, ax=None, figsize=None, title_fontsize='large', text_fontsize='medium'*)

Generates a plot of a classifier's feature importances.

**Parameters**

- **clf** – Classifier instance that has a `feature_importances_` attribute, e.g. `sklearn.ensemble.RandomForestClassifier` or `xgboost.XGBClassifier`.

- **title** (*string, optional*) – Title of the generated plot. Defaults to "Feature importances".

---

- **feature_names** (None, `list` of string, optional) – Determines the feature names used to plot the feature importances. If None, feature names will be numbered.

- **max_num_features** (*int*) – Determines the maximum number of features to plot. Defaults to 20.

- **order** (*'ascending', 'descending', or None, optional*) – Determines the order in which the feature importances are plotted. Defaults to 'descending'.

- **x_tick_rotation** (*int, optional*) – Rotates x-axis tick labels by the specified angle. This is useful in cases where there are numerous categories and the labels overlap each other.

- **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
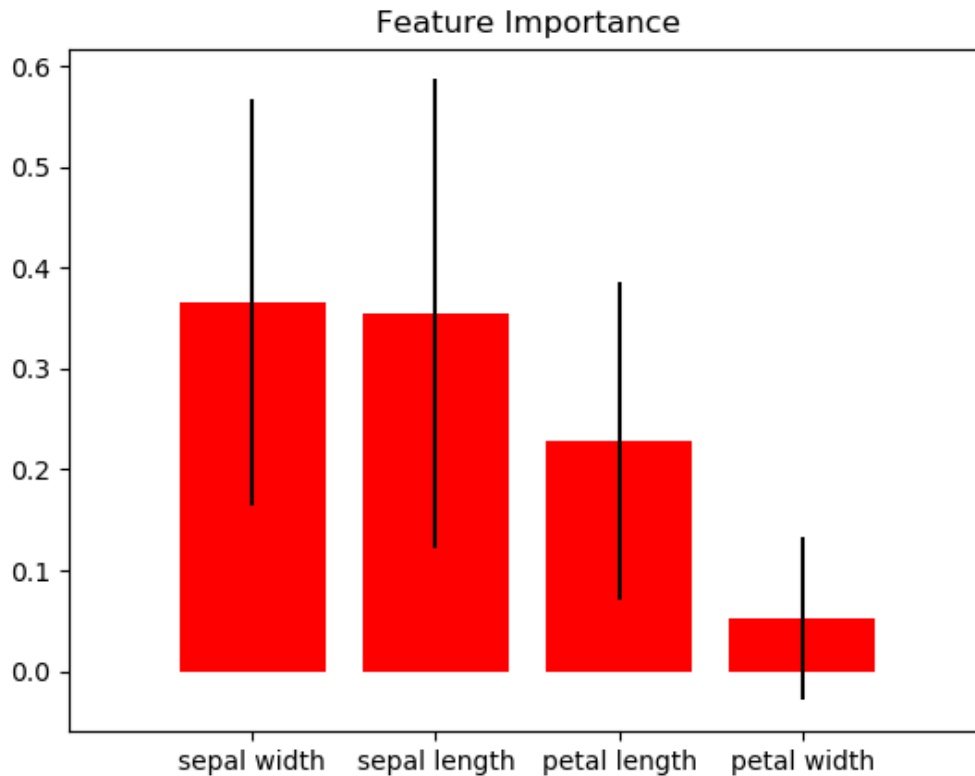
**Returns**

> **The axes on which the plot was** drawn.

**Return type** ax (`matplotlib.axes.Axes`)

### Example

```
>>> import scikitplot as skplt
>>> rf = RandomForestClassifier()
>>> rf.fit(X, y)
>>> skplt.estimators.plot_feature_importances(
...     rf, feature_names=['petal length', 'petal width',
...                        'sepal length', 'sepal width'])
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

## 2.4 Clusterer Module (API Reference)

The *scikitplot.cluster* module includes plots built specifically for scikit-learn clusterer instances e.g. KMeans. You can use your own clusterers, but these plots assume specific properties shared by scikit-learn estimators. The specific requirements are documented per function.

scikitplot.cluster.**plot_elbow_curve**(*clf,   X,   title='Elbow   Plot',   cluster_ranges=None,*
                                                             *n_jobs=1,                          show_cluster_time=True,*
                                                             *ax=None,     figsize=None,     title_fontsize='large',*
                                                             *text_fontsize='medium'*)

    Plots elbow curve of different values of K for KMeans clustering.

    **Parameters**

- **clf** – Clusterer instance that implements fit,``fit_predict``, and score methods, and an n_clusters hyperparameter. e.g. sklearn.cluster.KMeans instance

- **X** (*array-like, shape (n_samples, n_features)*) – Data to cluster, where n_samples is the number of samples and n_features is the number of features.

- **title** (*string, optional*) – Title of the generated plot. Defaults to "Elbow Plot"

- **cluster_ranges** (None or list of int, optional) – List of n_clusters for which to plot the explained variances. Defaults to range(1, 12, 2).

- **n_jobs** (*int, optional*) – Number of jobs to run in parallel. Defaults to 1.

---

- **show_cluster_time**(*bool, optional*) – Include plot of time it took to cluster for a particular K.

- **ax** (matplotlib.axes.Axes, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to None.

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".
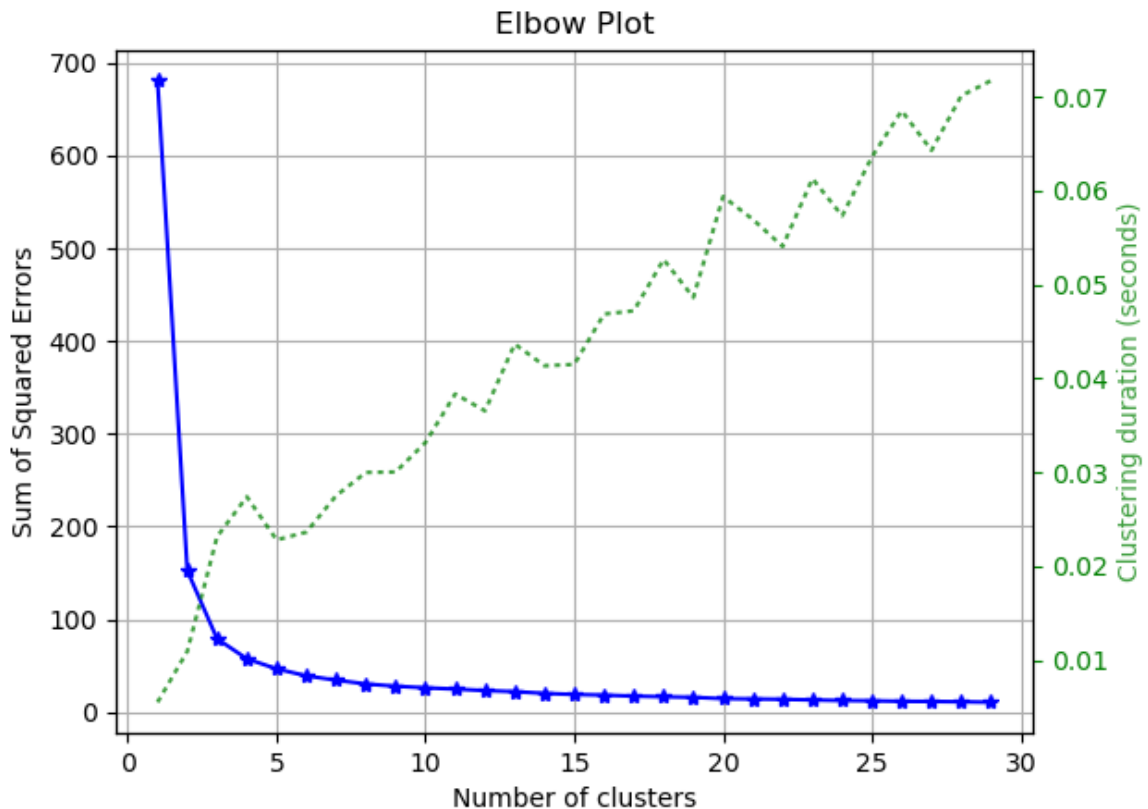
- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".

**Returns**

> The axes on which the plot was drawn.

**Return type** ax (matplotlib.axes.Axes)

## Example

```
>>> import scikitplot as skplt
>>> kmeans = KMeans(random_state=1)
>>> skplt.cluster.plot_elbow_curve(kmeans, cluster_ranges=range(1, 30))
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

# 2.5 Decomposition Module (API Reference)

The *scikitplot.decomposition* module includes plots built specifically for scikit-learn estimators that are used for dimensionality reduction e.g. PCA. You can use your own estimators, but these plots assume specific properties shared by scikit-learn estimators. The specific requirements are documented per function.

scikitplot.decomposition.**plot_pca_component_variance**(*clf*, *title='PCA Component Explained Variances'*, *target_explained_variance=0.75*, *ax=None*, *figsize=None*, *title_fontsize='large'*, *text_fontsize='medium'*)

Plots PCA components' explained variance ratios. (new in v0.2.2)

> **Parameters**
>
> > - **clf** – PCA instance that has the `explained_variance_ratio_` attribute.
> >
> > - **title** (`string, optional`) – Title of the generated plot. Defaults to "PCA Component Explained Variances"
> >
> > - **target_explained_variance** (`float, optional`) – Looks for the minimum number of principal components that satisfies this value and emphasizes it on the plot. Defaults to 0.75
> >
> > - **ax** (`matplotlib.axes.Axes`, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.
> >
> > - **figsize** (`2-tuple, optional`) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to `None`.
> >
> > - **title_fontsize** (`string or int, optional`) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".
> >
> > - **text_fontsize** (`string or int, optional`) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".
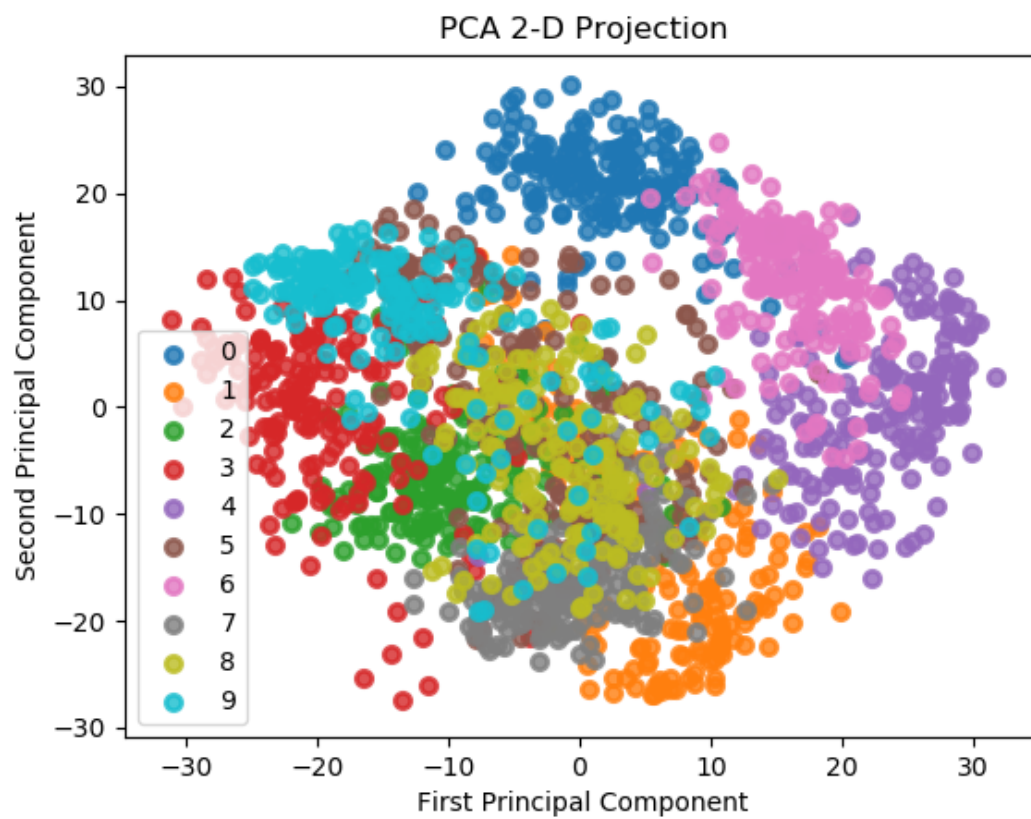>
> **Returns**
>
> > **The axes on which the plot was** drawn.
>
> **Return type** ax (`matplotlib.axes.Axes`)

## Example

```
>>> import scikitplot as skplt
>>> pca = PCA(random_state=1)
>>> pca.fit(X)
>>> skplt.decomposition.plot_pca_component_variance(pca)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```
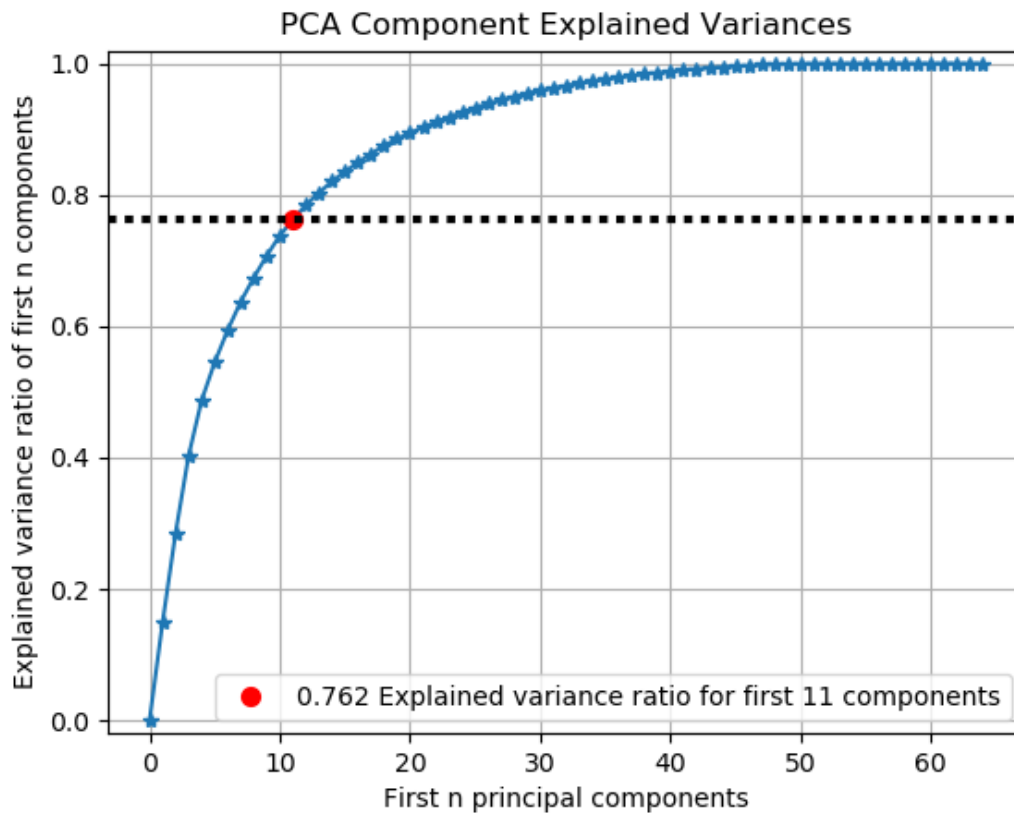
scikitplot.decomposition.**plot_pca_2d_projection**(*clf*, *X*, *y*, *title='PCA 2-D Pro-*
*jection'*, *biplot=False*, *fea-*
*ture_labels=None*, *ax=None*,
*figsize=None*, *cmap='Spectral'*,
*title_fontsize='large'*,
*text_fontsize='medium'*)

Plots the 2-dimensional projection of PCA on a given dataset.

### Parameters

- **clf** – Fitted PCA instance that can `transform` given data set into 2 dimensions.

- **X** (`array-like, shape (n_samples, n_features)`) – Feature set to project, where n_samples is the number of samples and n_features is the number of features.

- **y** (`array-like, shape (n_samples) or (n_samples, n_features)`) – Target relative to X for labeling.

- **title** (`string, optional`) – Title of the generated plot. Defaults to "PCA 2-D Projection"

- **biplot** (`bool, optional`) – If True, the function will generate and plot biplots. If false, the biplots are not generated.

- **feature_labels** (`array-like, shape (n_classes), optional`) – List of labels that represent each feature of X. Its index position must also be relative to the features. If `None` is given, then labels will be automatically generated for each feature. e.g. "variable1", "variable2", "variable3" . . .

---

- **ax** (matplotlib.axes.Axes, optional) – The axes upon which to plot the curve. If None, the plot is drawn on a new set of axes.

- **figsize** (*2-tuple, optional*) – Tuple denoting figure size of the plot e.g. (6, 6). Defaults to None.

- **cmap** (string or matplotlib.colors.Colormap instance, optional) – Colormap used for plotting the projection. View Matplotlib Colormap documentation for available options. https://matplotlib.org/users/colormaps.html

- **title_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "large".

- **text_fontsize** (*string or int, optional*) – Matplotlib-style fontsizes. Use e.g. "small", "medium", "large" or integer-values. Defaults to "medium".

**Returns**

> **The axes on which the plot was** drawn.

**Return type** ax (matplotlib.axes.Axes)

## Example

```
>>> import scikitplot as skplt
>>> pca = PCA(random_state=1)
>>> pca.fit(X)
>>> skplt.decomposition.plot_pca_2d_projection(pca, X, y)
<matplotlib.axes._subplots.AxesSubplot object at 0x7fe967d64490>
>>> plt.show()
```

# Python Module Index

## s

## P

## S