Chapter_3_A_Tour_of_Machine_Learning_Classifiers_Using_Scikit-Learn

March 17, 2024

0.1 Implementing Perceptron using scikit-learn on iris dataset

[34]:
```
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
print('Class labels:', np.unique(y))
```

```
Class labels: [0 1 2]
```

[35]:
```
# iris
```

[36]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=1, stratify=y)
```

[37]:
```
print('Labels counts in y:', np.bincount(y))
print('Labels counts in y_train:', np.bincount(y_train))
print('Labels counts in y_test:', np.bincount(y_test))
```

```
Labels counts in y: [50 50 50]
Labels counts in y_train: [35 35 35]
Labels counts in y_test: [15 15 15]
```

[38]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

[39]:
```
from sklearn.linear_model import Perceptron
ppn = Perceptron(max_iter=40, eta0=0.1, random_state=1)
ppn.fit(X_train_std, y_train)
```

[39]:
```
Perceptron(eta0=0.1, max_iter=40, random_state=1)
```

[40]:
```
y_pred = ppn.predict(X_test_std)
print('Misclassified samples: %d' % (y_test != y_pred).sum())
```

```
Misclassified samples: 1
```

[41]:
```
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.98
```

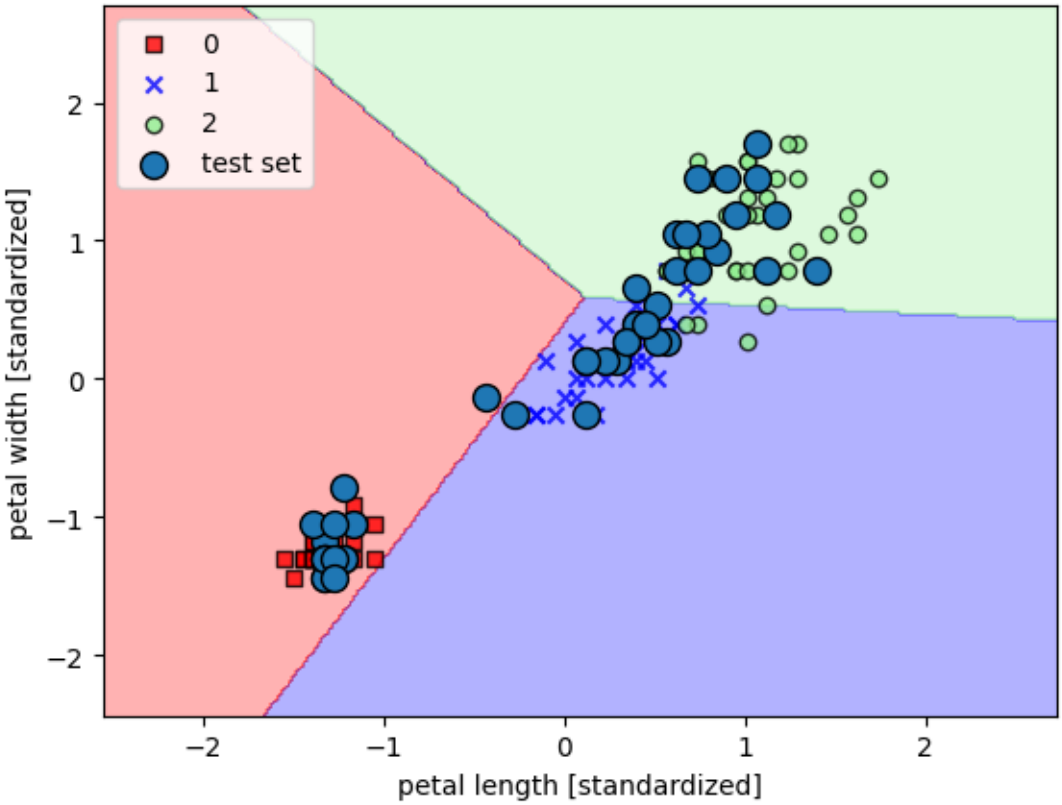
[42]:
```
print('Accuracy: %.2f' % ppn.score(X_test_std, y_test))
```

```
Accuracy: 0.98
```

[43]:
```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
def plot_decision_regions(X, y, classifier, test_idx=None,resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,␣
 ↪c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
    # highlight test samples
    if test_idx:
        # plot all samples
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1], edgecolor='black', alpha=1.
 ↪0,linewidth=1, marker='o',s=100, label='test set')
```

[44]:
```
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X=X_combined_std,y=y_combined,classifier=ppn,test_idx=range(105,␣
 ↪150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



[45]:
```
## decision boundary is not perfectly separable because data classes are not␣
↪linearly separable. So, use logistic algorithm
```

[46]:
```
help(Perceptron)
```

```
Help on class Perceptron in module sklearn.linear_model._perceptron:

class Perceptron(sklearn.linear_model._stochastic_gradient.BaseSGDClassifier)
 |  Perceptron(*, penalty=None, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None,
random_state=0, early_stopping=False, validation_fraction=0.1,
n_iter_no_change=5, class_weight=None, warm_start=False)
 |
 |  Linear perceptron classifier.
 |
```

```
 |  The implementation is a wrapper around
:class:`~sklearn.linear_model.SGDClassifier`
 |  by fixing the `loss` and `learning_rate` parameters as::
 |
 |      SGDClassifier(loss="perceptron", learning_rate="constant")
 |
 |  Other available parameters are described below and are forwarded to
 |  :class:`~sklearn.linear_model.SGDClassifier`.
 |
 |  Read more in the :ref:`User Guide <perceptron>`.
 |
 |  Parameters
 |  ----------
 |
 |  penalty : {'l2','l1','elasticnet'}, default=None
 |      The penalty (aka regularization term) to be used.
 |
 |  alpha : float, default=0.0001
 |      Constant that multiplies the regularization term if regularization is
 |      used.
 |
 |  l1_ratio : float, default=0.15
 |      The Elastic Net mixing parameter, with `0 <= l1_ratio <= 1`.
 |      `l1_ratio=0` corresponds to L2 penalty, `l1_ratio=1` to L1.
 |      Only used if `penalty='elasticnet'`.
 |
 |      .. versionadded:: 0.24
 |
 |  fit_intercept : bool, default=True
 |      Whether the intercept should be estimated or not. If False, the
 |      data is assumed to be already centered.
 |
 |  max_iter : int, default=1000
 |      The maximum number of passes over the training data (aka epochs).
 |      It only impacts the behavior in the ``fit`` method, and not the
 |      :meth:`partial_fit` method.
 |
 |      .. versionadded:: 0.19
 |
 |  tol : float or None, default=1e-3
 |      The stopping criterion. If it is not None, the iterations will stop
 |      when (loss > previous_loss - tol).
 |
 |      .. versionadded:: 0.19
 |
 |  shuffle : bool, default=True
 |      Whether or not the training data should be shuffled after each epoch.
 |
```

```
|  verbose : int, default=0
|      The verbosity level.
|
|  eta0 : float, default=1
|      Constant by which the updates are multiplied.
|
|  n_jobs : int, default=None
|      The number of CPUs to use to do the OVA (One Versus All, for
|      multi-class problems) computation.
|      ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
|      ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
|      for more details.
|
|  random_state : int, RandomState instance or None, default=0
|      Used to shuffle the training data, when ``shuffle`` is set to
|      ``True``. Pass an int for reproducible output across multiple
|      function calls.
|      See :term:`Glossary <random_state>`.
|
|  early_stopping : bool, default=False
|      Whether to use early stopping to terminate training when validation
|      score is not improving. If set to True, it will automatically set aside
|      a stratified fraction of training data as validation and terminate
|      training when validation score is not improving by at least `tol` for
|      `n_iter_no_change` consecutive epochs.
|
|      .. versionadded:: 0.20
|
|  validation_fraction : float, default=0.1
|      The proportion of training data to set aside as validation set for
|      early stopping. Must be between 0 and 1.
|      Only used if early_stopping is True.
|
|      .. versionadded:: 0.20
|
|  n_iter_no_change : int, default=5
|      Number of iterations with no improvement to wait before early stopping.
|
|      .. versionadded:: 0.20
|
|  class_weight : dict, {class_label: weight} or "balanced", default=None
|      Preset for the class_weight fit parameter.
|
|      Weights associated with classes. If not given, all classes
|      are supposed to have weight one.
|
|      The "balanced" mode uses the values of y to automatically adjust
|      weights inversely proportional to class frequencies in the input data
```

```
|      as ``n_samples / (n_classes * np.bincount(y))``.
|
|  warm_start : bool, default=False
|      When set to True, reuse the solution of the previous call to fit as
|      initialization, otherwise, just erase the previous solution. See
|      :term:`the Glossary <warm_start>`.
|
|  Attributes
|  ----------
|  classes_ : ndarray of shape (n_classes,)
|      The unique classes labels.
|
|  coef_ : ndarray of shape (1, n_features) if n_classes == 2 else
(n_classes, n_features)
|      Weights assigned to the features.
|
|  intercept_ : ndarray of shape (1,) if n_classes == 2 else (n_classes,)
|      Constants in decision function.
|
|  loss_function_ : concrete LossFunction
|      The function that determines the loss, or difference between the
|      output of the algorithm and the target values.
|
|  n_features_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 0.24
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
|
|  n_iter_ : int
|      The actual number of iterations to reach the stopping criterion.
|      For multiclass fits, it is the maximum over every binary fit.
|
|  t_ : int
|      Number of weight updates performed during training.
|      Same as ``(n_iter_ * n_samples + 1)``.
|
|  See Also
|  --------
|  sklearn.linear_model.SGDClassifier : Linear classifiers
|      (SVM, logistic regression, etc.) with SGD training.
|
|  Notes
```

```
 |  -----
 |  ``Perceptron`` is a classification algorithm which shares the same
 |  underlying implementation with ``SGDClassifier``. In fact,
 |  ``Perceptron()`` is equivalent to `SGDClassifier(loss="perceptron",
 |  eta0=1, learning_rate="constant", penalty=None)`.
 |
 |  References
 |  ----------
 |  https://en.wikipedia.org/wiki/Perceptron and references therein.
 |
 |  Examples
 |  --------
 |  >>> from sklearn.datasets import load_digits
 |  >>> from sklearn.linear_model import Perceptron
 |  >>> X, y = load_digits(return_X_y=True)
 |  >>> clf = Perceptron(tol=1e-3, random_state=0)
 |  >>> clf.fit(X, y)
 |  Perceptron()
 |  >>> clf.score(X, y)
 |  0.939…
 |
 |  Method resolution order:
 |      Perceptron
 |      sklearn.linear_model._stochastic_gradient.BaseSGDClassifier
 |      sklearn.linear_model._base.LinearClassifierMixin
 |      sklearn.base.ClassifierMixin
 |      sklearn.linear_model._stochastic_gradient.BaseSGD
 |      sklearn.linear_model._base.SparseCoefMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, penalty=None, alpha=0.0001, l1_ratio=0.15,
fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, eta0=1.0,
n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1,
n_iter_no_change=5, class_weight=None, warm_start=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  set_fit_request(self: sklearn.linear_model._perceptron.Perceptron, *,
coef_init: Union[bool, NoneType, str] = '$UNCHANGED$', intercept_init:
Union[bool, NoneType, str] = '$UNCHANGED$', sample_weight: Union[bool, NoneType,
str] = '$UNCHANGED$') -> sklearn.linear_model._perceptron.Perceptron
 |      Request metadata passed to the ``fit`` method.
 |
 |      Note that this method is only relevant if
```

```
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``fit`` if provided.
The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not
pass it to ``fit``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will raise
an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains the
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      coef_init : str, True, False, or None,
default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``coef_init`` parameter in ``fit``.
 |
 |      intercept_init : str, True, False, or None,
default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``intercept_init`` parameter in ``fit``.
 |
 |      sample_weight : str, True, False, or None,
default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``fit``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
```

```
 |  set_partial_fit_request(self: sklearn.linear_model._perceptron.Perceptron,
*, classes: Union[bool, NoneType, str] = '$UNCHANGED$', sample_weight:
Union[bool, NoneType, str] = '$UNCHANGED$') ->
sklearn.linear_model._perceptron.Perceptron
 |      Request metadata passed to the ``partial_fit`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``partial_fit`` if
provided. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not
pass it to ``partial_fit``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will raise
an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains the
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      classes : str, True, False, or None,
default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``classes`` parameter in ``partial_fit``.
 |
 |      sample_weight : str, True, False, or None,
default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``partial_fit``.
 |
 |      Returns
 |      -------
```

```
 |      self : object
 |          The updated object.
 |
 |  set_score_request(self: sklearn.linear_model._perceptron.Perceptron, *,
sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') ->
sklearn.linear_model._perceptron.Perceptron
 |      Request metadata passed to the ``score`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``score`` if provided.
The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not
pass it to ``score``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will raise
an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains the
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      sample_weight : str, True, False, or None,
default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``score``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from
sklearn.linear_model._stochastic_gradient.BaseSGDClassifier:
 |
 |  fit(self, X, y, coef_init=None, intercept_init=None, sample_weight=None)
 |      Fit linear model with Stochastic Gradient Descent.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          Training data.
 |
 |      y : ndarray of shape (n_samples,)
 |          Target values.
 |
 |      coef_init : ndarray of shape (n_classes, n_features), default=None
 |          The initial coefficients to warm-start the optimization.
 |
 |      intercept_init : ndarray of shape (n_classes,), default=None
 |          The initial intercept to warm-start the optimization.
 |
 |      sample_weight : array-like, shape (n_samples,), default=None
 |          Weights applied to individual samples.
 |          If not provided, uniform weights are assumed. These weights will
 |          be multiplied with class_weight (passed through the
 |          constructor) if class_weight is specified.
 |
 |      Returns
 |      -------
 |      self : object
 |          Returns an instance of self.
 |
 |  partial_fit(self, X, y, classes=None, sample_weight=None)
 |      Perform one epoch of stochastic gradient descent on given samples.
 |
 |      Internally, this method uses ``max_iter = 1``. Therefore, it is not
 |      guaranteed that a minimum of the cost function is reached after calling
 |      it once. Matters such as objective convergence, early stopping, and
 |      learning rate adjustments should be handled by the user.
 |
```

```
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix}, shape (n_samples, n_features)
 |          Subset of the training data.
 |
 |      y : ndarray of shape (n_samples,)
 |          Subset of the target values.
 |
 |      classes : ndarray of shape (n_classes,), default=None
 |          Classes across all calls to partial_fit.
 |          Can be obtained by via `np.unique(y_all)`, where y_all is the
 |          target vector of the entire dataset.
 |          This argument is required for the first call to partial_fit
 |          and can be omitted in the subsequent calls.
 |          Note that y doesn't need to contain all labels in `classes`.
 |
 |      sample_weight : array-like, shape (n_samples,), default=None
 |          Weights applied to individual samples.
 |          If not provided, uniform weights are assumed.
 |
 |      Returns
 |      -------
 |      self : object
 |          Returns an instance of self.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from
sklearn.linear_model._stochastic_gradient.BaseSGDClassifier:
 |
 |  loss_functions = {'epsilon_insensitive': (<class 'sklearn.linear_model…
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.linear_model._base.LinearClassifierMixin:
 |
 |  decision_function(self, X)
 |      Predict confidence scores for samples.
 |
 |      The confidence score for a sample is proportional to the signed
 |      distance of that sample to the hyperplane.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The data matrix for which we want to get the confidence scores.
 |
 |      Returns
 |      -------
 |      scores : ndarray of shape (n_samples,) or (n_samples, n_classes)
```

```
|          Confidence scores per `(n_samples, n_classes)` combination. In the
|          binary case, confidence score for `self.classes_[1]` where >0 means
|          this class would be predicted.
|
|  predict(self, X)
|      Predict class labels for samples in X.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The data matrix for which we want to get the predictions.
|
|      Returns
|      -------
|      y_pred : ndarray of shape (n_samples,)
|          Vector containing the class labels for each sample.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return the mean accuracy on the given test data and labels.
|
|      In multi-label classification, this is the subset accuracy
|      which is a harsh metric since you require for each sample that
|      each label set be correctly predicted.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` w.r.t. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
```

```
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from
sklearn.linear_model._stochastic_gradient.BaseSGD:
 |
 |  loss_function_
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.linear_model._base.SparseCoefMixin:
 |
 |  densify(self)
 |      Convert coefficient matrix to dense array format.
 |
 |      Converts the ``coef_`` member (back) to a numpy.ndarray. This is the
 |      default format of ``coef_`` and is required for fitting, so calling
 |      this method is only required on models that have previously been
 |      sparsified; otherwise, it is a no-op.
 |
 |      Returns
 |      -------
 |      self
 |          Fitted estimator.
 |
 |  sparsify(self)
 |      Convert coefficient matrix to sparse format.
 |
 |      Converts the ``coef_`` member to a scipy.sparse matrix, which for
 |      L1-regularized models can be much more memory- and storage-efficient
 |      than the usual numpy.ndarray representation.
 |
 |      The ``intercept_`` member is not converted.
 |
 |      Returns
 |      -------
 |      self
 |          Fitted estimator.
 |
 |      Notes
 |      -----
 |      For non-sparse models, i.e. when there are not many zeros in ``coef_``,
 |      this may actually *increase* memory usage, so use this method with
 |      care. A rule of thumb is that the number of zero elements, which can
 |      be computed with ``(coef_ == 0).sum()``, must be more than 50% for this
 |      to provide significant benefits.
 |
```

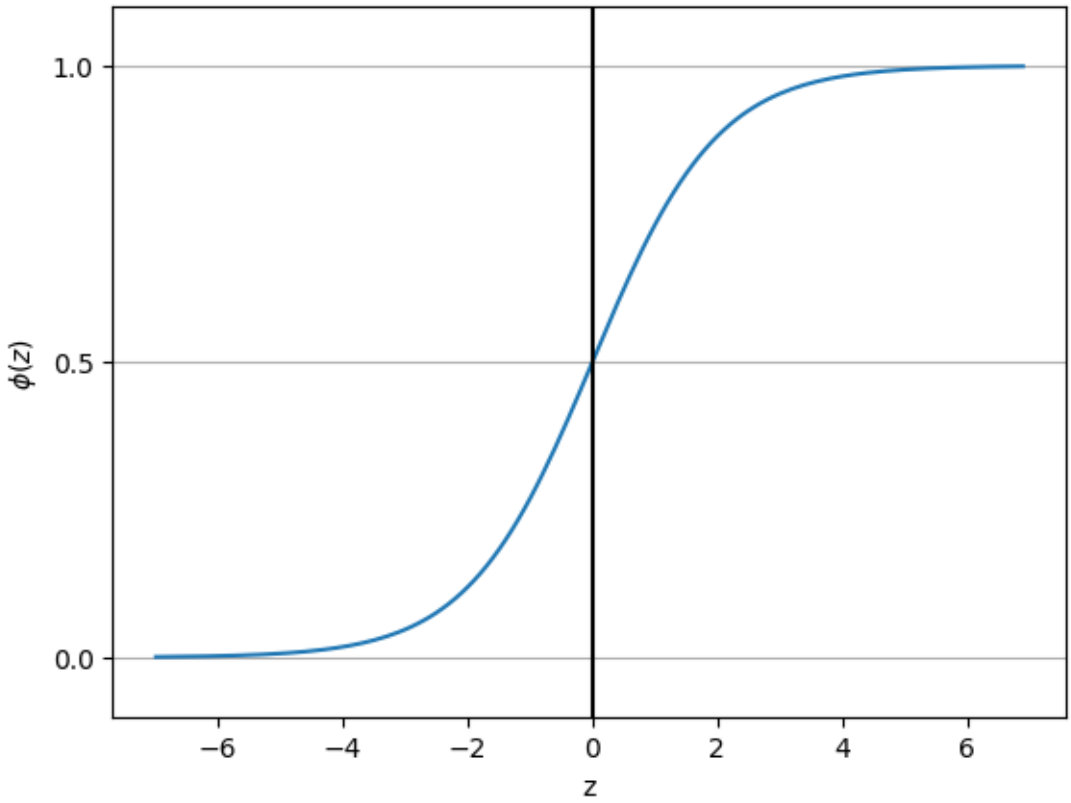
```
|      After calling this method, further fitting with the partial_fit
|      method (if any) will not work until you call densify.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  __sklearn_clone__(self)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          Parameter names mapped to their values.
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
|
|      Returns
|      -------
|      self : estimator instance
|          Estimator instance.
|
|  ----------------------------------------------------------------------
```

```
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequester:
 |
 |  get_metadata_routing(self)
 |      Get metadata routing of this object.
 |
 |      Please check :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      Returns
 |      -------
 |      routing : MetadataRequest
 |          A :class:`~sklearn.utils.metadata_routing.MetadataRequest`
encapsulating
 |          routing information.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from
sklearn.utils._metadata_requests._MetadataRequester:
 |
 |  __init_subclass__(**kwargs) from abc.ABCMeta
 |      Set the ``set_{method}_request`` methods.
 |
 |      This uses PEP-487 [1]_ to set the ``set_{method}_request`` methods. It
 |      looks for the information available in the set default values which are
 |      set using ``__metadata_request__*`` class attributes, or inferred
 |      from method signatures.
 |
 |      The ``__metadata_request__*`` class attributes are used when a method
 |      does not explicitly accept a metadata through its arguments or if the
 |      developer would like to specify a request value for those metadata
 |      which are different from the default ``None``.
 |
 |      References
 |      ----------
 |      .. [1] https://www.python.org/dev/peps/pep-0487
```

0.2 Implementing Logistic Regression using scikit-learn on iris dataset

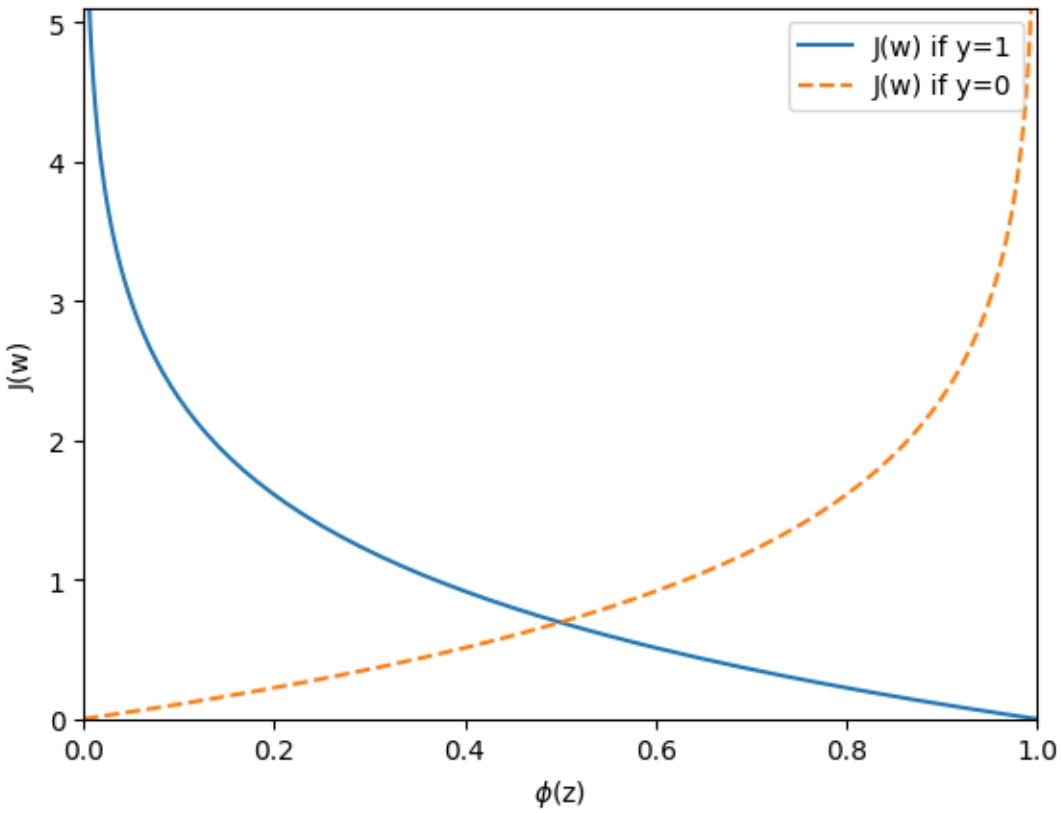
[47]:
```
#Now let us simply plot the sigmoid function for some values in the range -7 to␣
 ↪7 to
#see how it looks:
import matplotlib.pyplot as plt
import numpy as np
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))
z = np.arange(-7, 7, 0.1)
```

```
phi_z = sigmoid(z)
plt.plot(z, phi_z)
plt.axvline(0.0, color='k')
plt.ylim(-0.1, 1.1)
plt.xlabel('z')
plt.ylabel('$\phi (z)$')
# y axis ticks and gridline
plt.yticks([0.0, 0.5, 1.0])
ax = plt.gca()
ax.yaxis.grid(True)
plt.show()
```



[48]:
```
def cost_1(z):
    return - np.log(sigmoid(z))
def cost_0(z):
    return - np.log(1 - sigmoid(z))
z = np.arange(-10, 10, 0.1)
phi_z = sigmoid(z)
c1 = [cost_1(x) for x in z]
plt.plot(phi_z, c1, label='J(w) if y=1')
c0 = [cost_0(x) for x in z]
```

```
plt.plot(phi_z, c0, linestyle='--', label='J(w) if y=0')
plt.ylim(0.0, 5.1)
plt.xlim([0, 1])
plt.xlabel('$\phi$(z)')
plt.ylabel('J(w)')
plt.legend(loc='best')
plt.show()
```



0.2.1 Implementing Logistic Regression by modifying Adaline

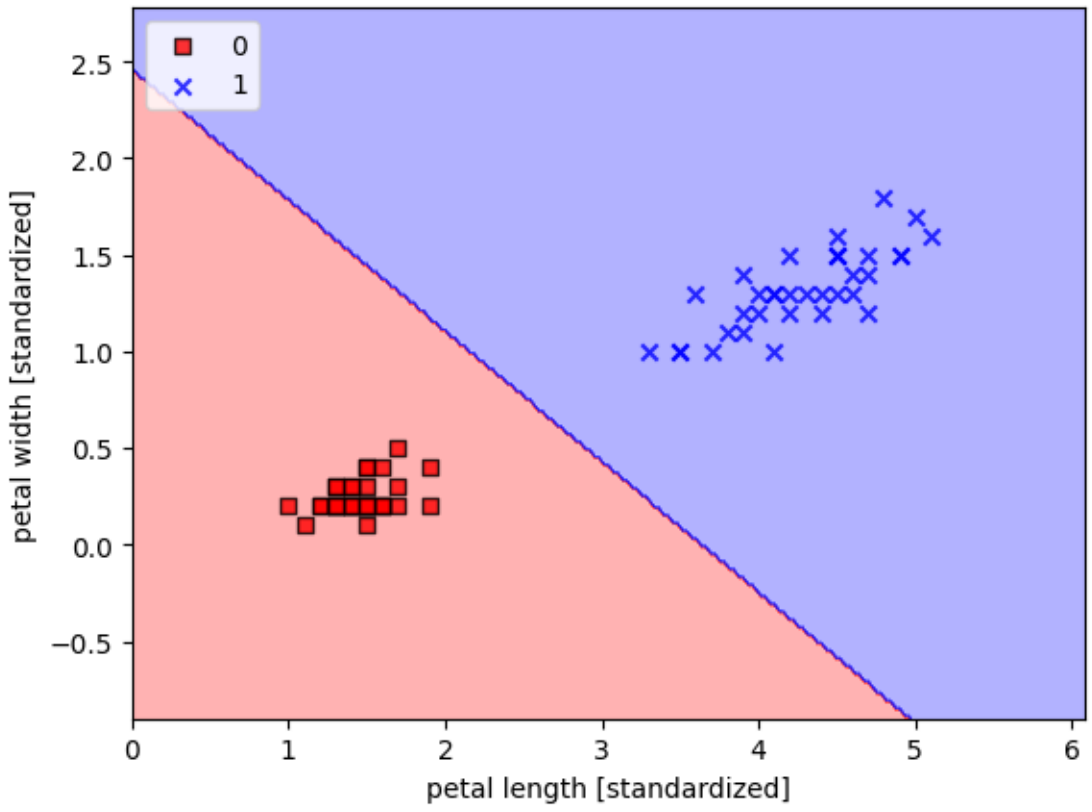
[49]:
```
class LogisticRegressionGD(object):
    """Logistic Regression Classifier using gradient descent.
    Parameters
    ------------
    eta : float
    Learning rate (between 0.0 and 1.0)
    n_iter : int
    Passes over the training dataset.
    random_state : int
    Random number generator seed for random weight
```

```
    initialization.
    Attributes
    -----------
    w_ : 1d-array
    Weights after fitting.
    cost_ : list
    Sum-of-squares cost function value in each epoch.
    """
    def __init__(self, eta=0.05, n_iter=100, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state
    def fit(self, X, y):
        """ Fit training data.
        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
        Training vectors, where n_samples is the number of
        samples and
        n_features is the number of features.
        y : array-like, shape = [n_samples]
        Target values.
        Returns
        -------
        self : object
        """
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01,size=1 + X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            net_input = self.net_input(X)
            output = self.activation(net_input)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            # note that we compute the logistic `cost` now
            # instead of the sum of squared errors cost
            cost = (-y.dot(np.log(output)) -
            ((1 - y).dot(np.log(1 - output))))
            self.cost_.append(cost)
        return self
    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]
    def activation(self, z):
        """Compute logistic sigmoid activation"""
        return 1. / (1. + np.exp(-np.clip(z, -250, 250)))
```

```
    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, 0)
        # equivalent to:
        # return np.where(self.activation(self.net_input(X))
        # >= 0.5, 1, 0)
```

[50]:
```
X_train_01_subset = X_train[(y_train == 0) | (y_train == 1)]
y_train_01_subset = y_train[(y_train == 0) | (y_train == 1)]
lrgd = LogisticRegressionGD(eta=0.05,n_iter=1000,random_state=1)
lrgd.fit(X_train_01_subset,y_train_01_subset)
plot_decision_regions(X=X_train_01_subset,y=y_train_01_subset,classifier=lrgd)
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

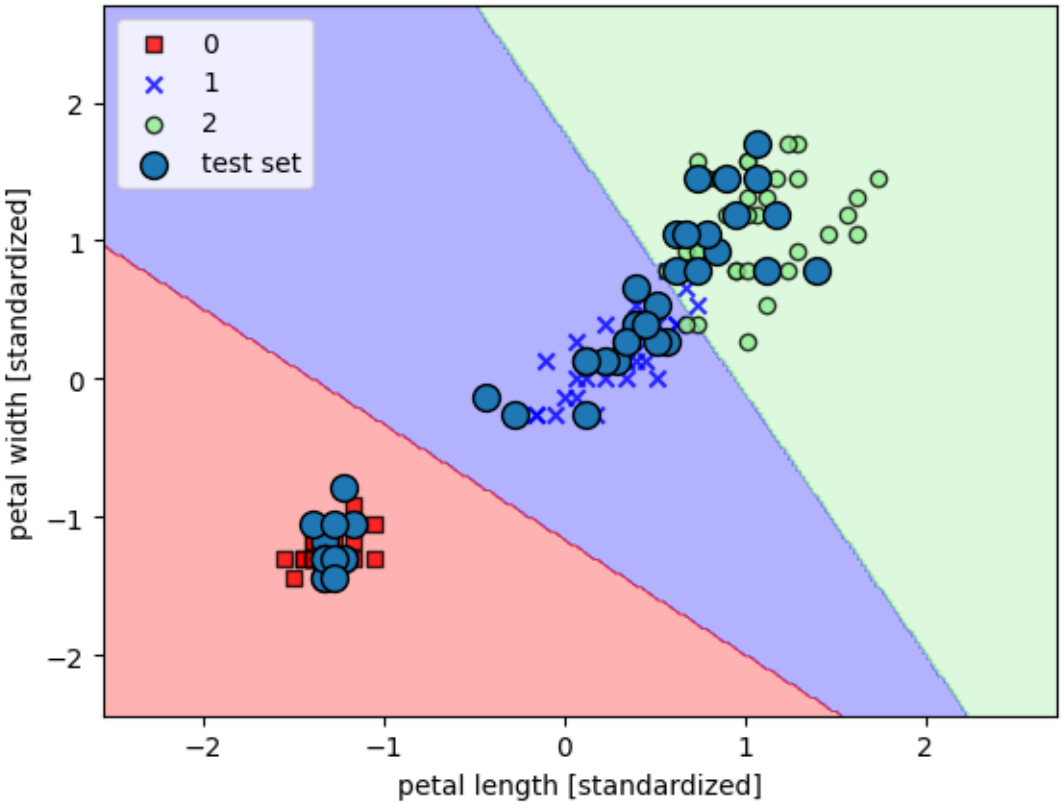
```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



0.3 Training a logistic regression model with scikit-learn

[51]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=100.0, random_state=1)
lr.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std,y_combined,classifier=lr,test_idx=range(105,
↪150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



[52]:
```
lr.predict_proba(X_test_std[:3, :])
```

[52]:
```
array([[9.37368464e-13, 3.91458193e-04, 9.99608542e-01],
       [9.93631074e-01, 6.36892585e-03, 1.20730798e-15],
       [9.98707332e-01, 1.29266792e-03, 1.82177043e-17]])
```

[53]:
```
lr.predict_proba(X_test_std[:3, :]).sum(axis=1)
```

[53]:
```
array([1., 1., 1.])
```

[54]:
```
lr.predict_proba(X_test_std[:3, :]).argmax(axis=1)
```

[54]:
```
array([2, 0, 0], dtype=int64)
```

[55]:
```
lr.predict(X_test_std[:3, :])
```

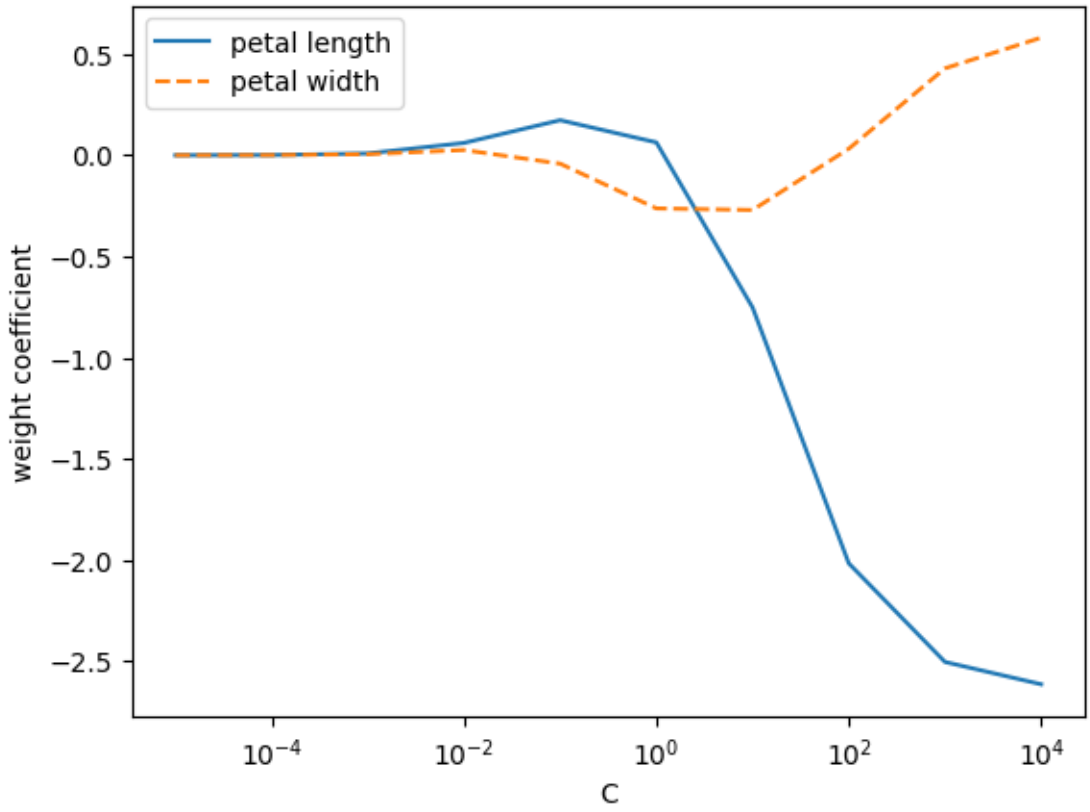
[55]:
```
array([2, 0, 0])
```

[56]:
```
lr.predict(X_test_std[0, :].reshape(1, -1))
```

[56]:
```
array([2])
```

0.4 Tackling Overfitting and Regularization

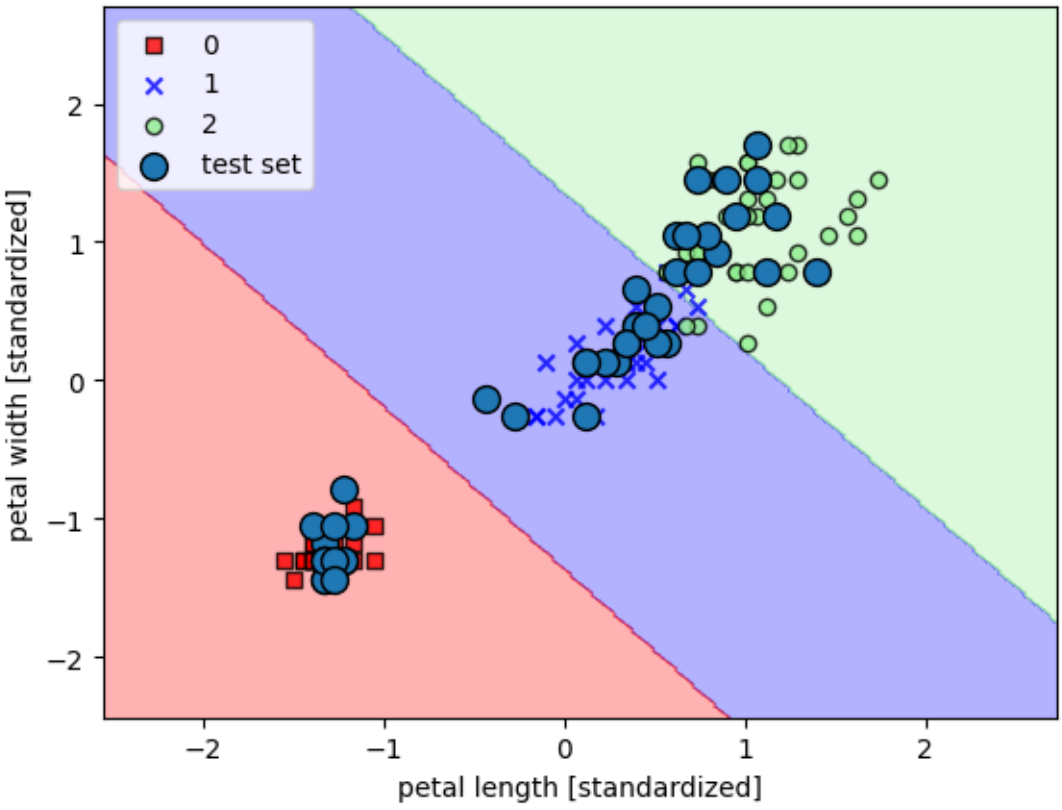
[57]:
```
weights, params = [], []
for c in np.arange(-5, 5):
    lr = LogisticRegression(C=10.**c, random_state=1)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10.**c)
weights = np.array(weights)
plt.plot(params, weights[:, 0],label='petal length')
plt.plot(params, weights[:, 1], linestyle='--',label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='upper left')
plt.xscale('log')
plt.show()
```



0.5 support vector machines

[58]:
```
from sklearn.svm import SVC
svm = SVC(kernel='linear', C=1.0, random_state=1)
svm.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std,y_combined,classifier=svm,test_idx=range(105,␣
↪150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



0.6 Alternative implementations in scikit-learn

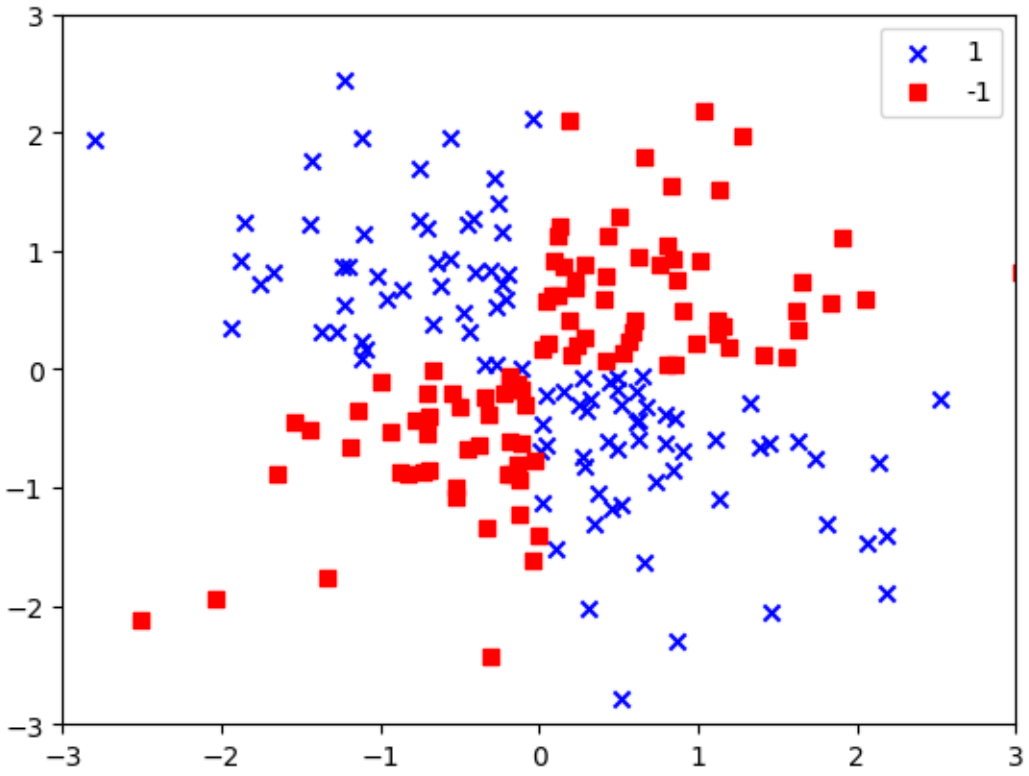
[59]:
```
# for large datasets

from sklearn.linear_model import SGDClassifier
ppn = SGDClassifier(loss='perceptron')
lr = SGDClassifier(loss='log')
svm = SGDClassifier(loss='hinge')
```

0.7 Solving nonlinear problems using a kernel SVM

0.7.1 Kernel methods for linearly inseparable data

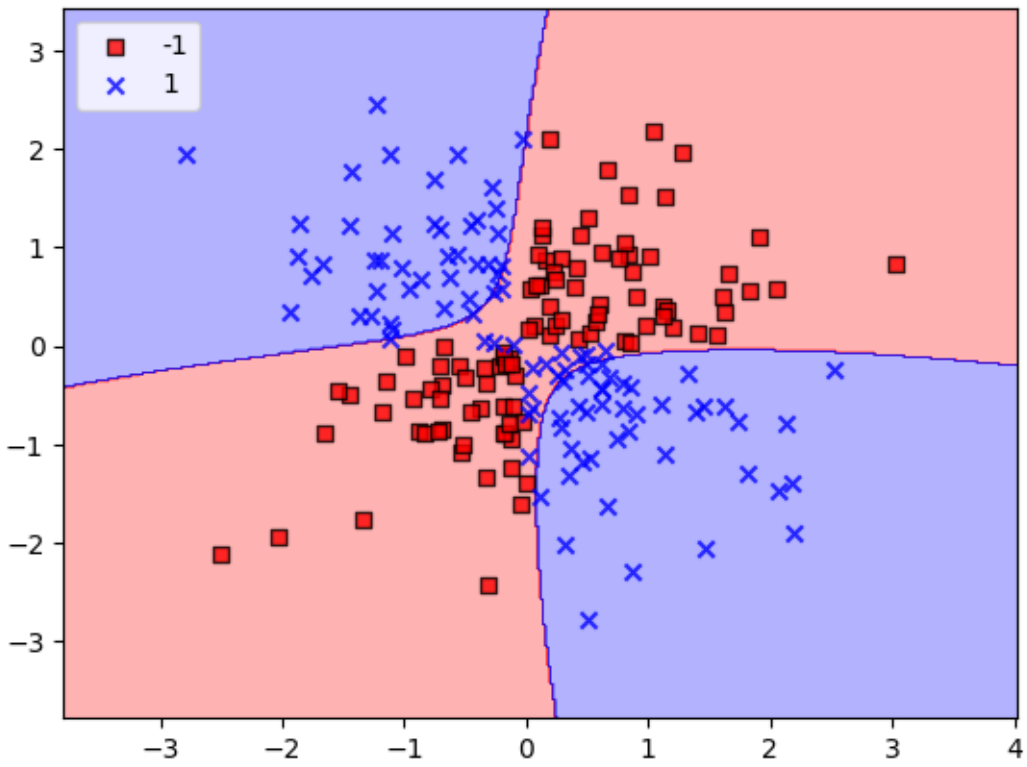
[60]:
```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(1)
X_xor = np.random.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0,X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)
plt.scatter(X_xor[y_xor == 1, 0],X_xor[y_xor == 1, 1],c='b',␣
 ↪marker='x',label='1')
```

```
plt.scatter(X_xor[y_xor == -1, 0],X_xor[y_xor == -1,␣
 ↪1],c='r',marker='s',label='-1')
plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.show()
```



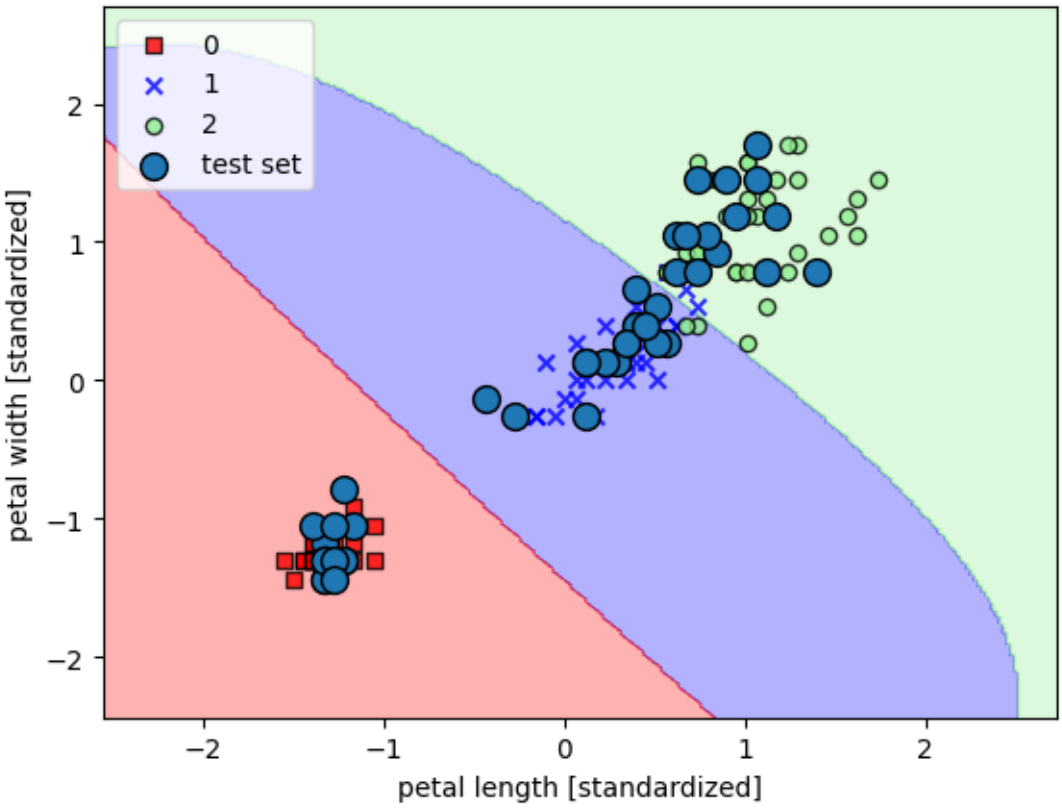
[61]:
```
svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=10.0)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor, classifier=svm)
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



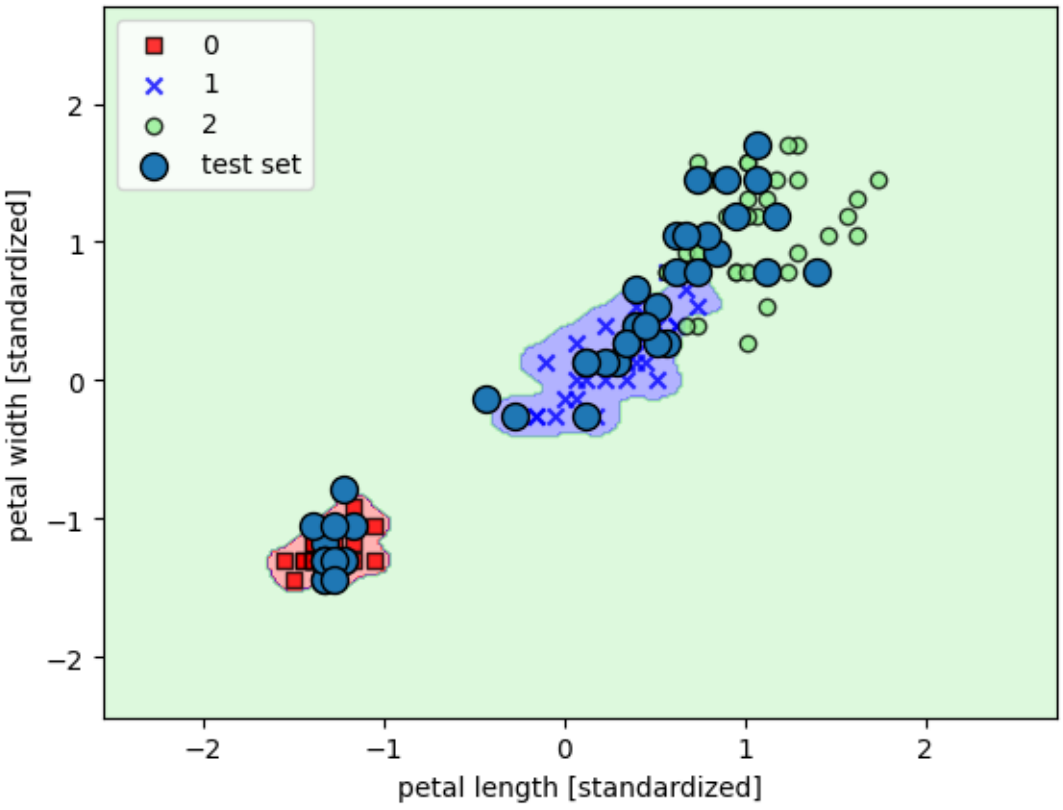
[62]:
```
svm = SVC(kernel='rbf', random_state=1, gamma=0.2, C=1.0)
svm.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std,y_combined,␣
 ↪classifier=svm,test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



[63]:
```
svm = SVC(kernel='rbf', random_state=1, gamma=100.0, C=1.0)
svm.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std,y_combined,␣
 ↪classifier=svm,test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

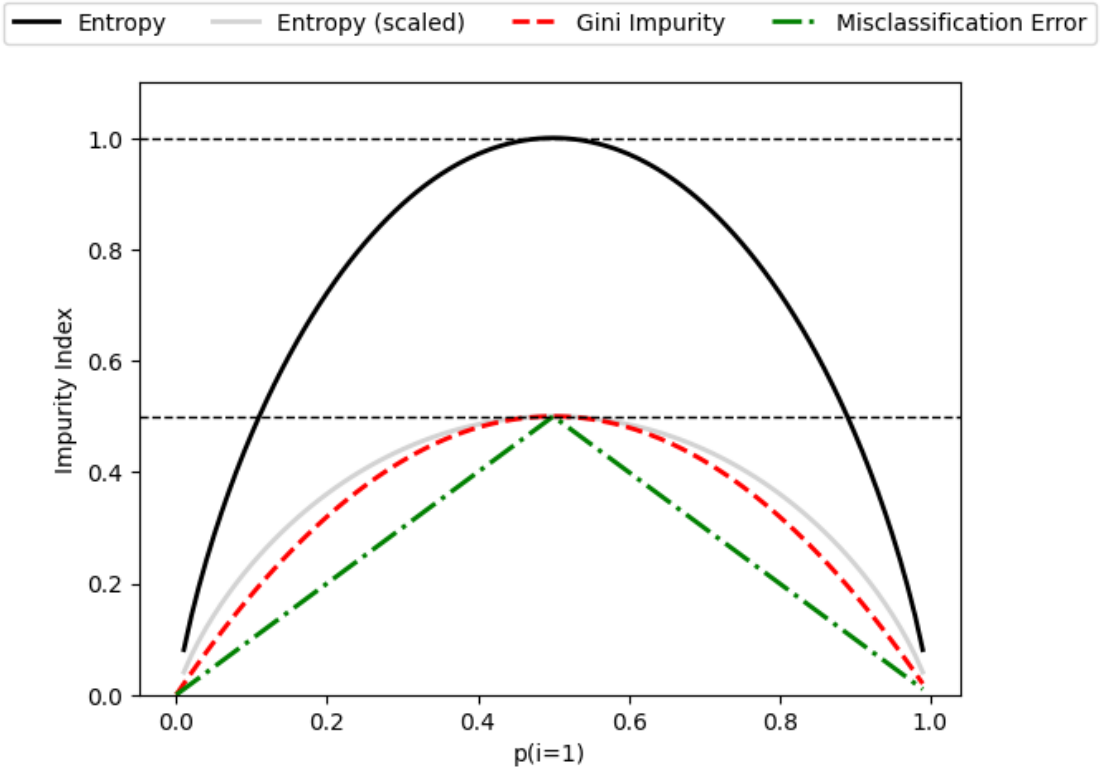
```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



0.8 Decision tree learning

[64]:
```
import matplotlib.pyplot as plt
import numpy as np
def gini(p):
    return (p)*(1 - (p)) + (1 - p)*(1 - (1-p))
def entropy(p):
    return - p*np.log2(p) - (1 - p)*np.log2((1 - p))
def error(p):
    return 1 - np.max([p, 1 - p])
x = np.arange(0.0, 1.0, 0.01)
ent = [entropy(p) if p != 0 else None for p in x]
sc_ent = [e*0.5 if e else None for e in ent]
err = [error(i) for i in x]
fig = plt.figure()
ax = plt.subplot(111)
for i, lab, ls, c, in zip([ent, sc_ent, gini(x), err],['Entropy', 'Entropy␣
 ↪(scaled)','Gini Impurity','Misclassification Error'],['-', '-', '--', '-.
 ↪'],['black', 'lightgray','red', 'green', 'cyan']):
    line = ax.plot(x, i, label=lab,linestyle=ls, lw=2, color=c)
```

```
ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15),ncol=5, fancybox=True,␣
 ↪shadow=False)
ax.axhline(y=0.5, linewidth=1, color='k', linestyle='--')
ax.axhline(y=1.0, linewidth=1, color='k', linestyle='--')
plt.ylim([0, 1.1])
plt.xlabel('p(i=1)')
plt.ylabel('Impurity Index')
plt.show()
```

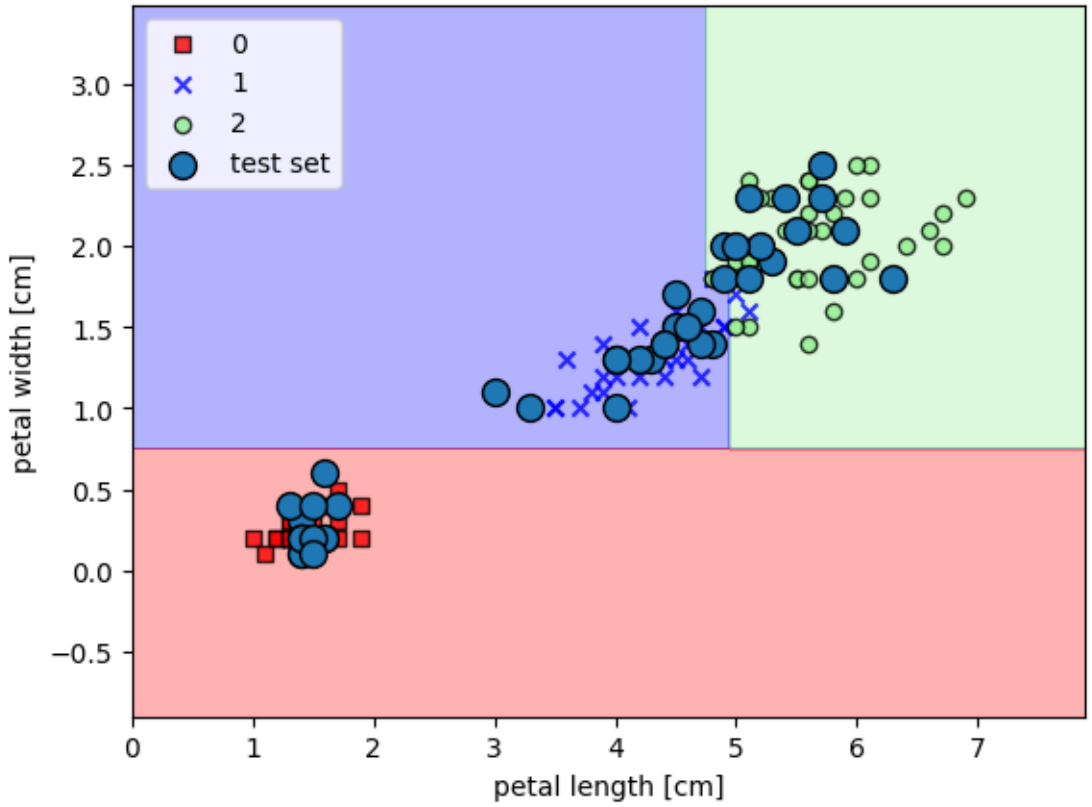


0.9 Building a decision tree

[65]:
```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion='gini',max_depth=4,random_state=1)
tree.fit(X_train, y_train)
X_combined = np.vstack((X_train, X_test))
y_combined = np.hstack((y_train, y_test))
plot_decision_regions(X_combined,y_combined,classifier=tree,test_idx=range(105,␣
 ↪150))
plt.xlabel('petal length [cm]')
plt.ylabel('petal width [cm]')
plt.legend(loc='upper left')
```

```
plt.show()
```

```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



[66]:
```
# ! pip3 install pydotplus
# ! pip3 install graphviz
# ! pip3 install pyparsing
```

[68]:
```
# first download executable from https://graphviz.org/download/ and then add␣
 ↪into the path
import os
os.environ["PATH"] += os.pathsep + 'C:\Program Files (x86)\Graphviz-10.0.
 ↪1-win64/bin/'
```

[69]:
```
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
```

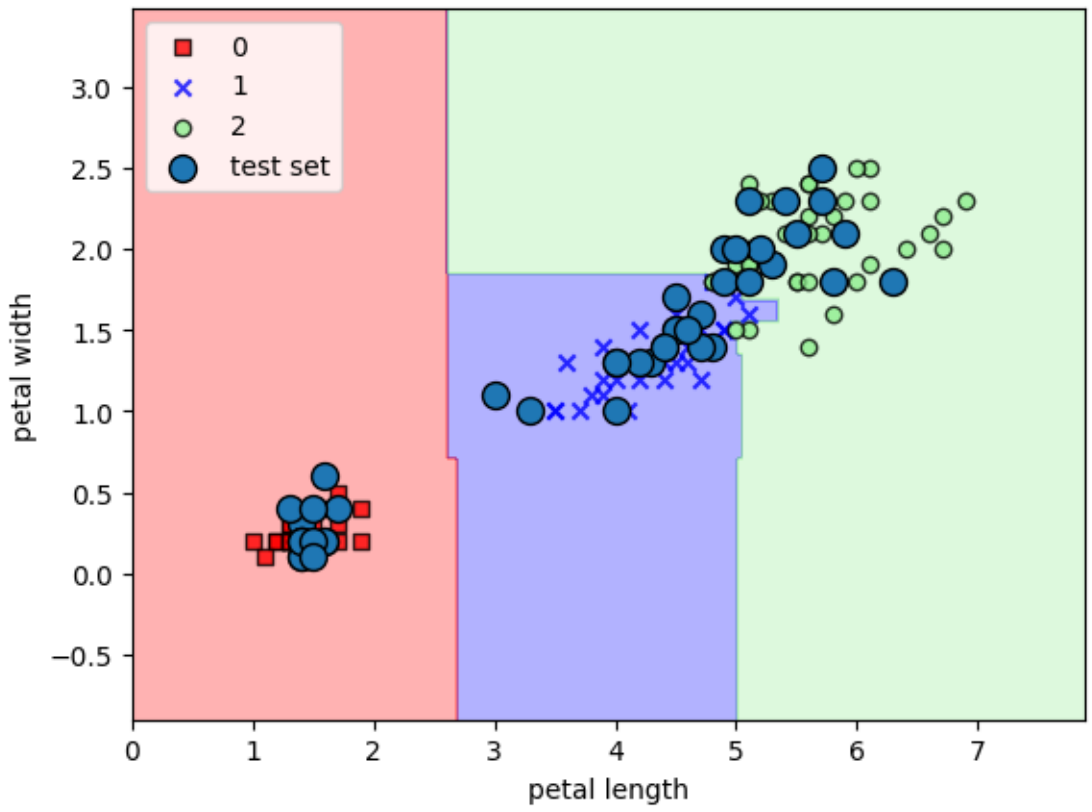
```
dot_data =␣
 ↪export_graphviz(tree,filled=True,rounded=True,class_names=['Setosa','Versicolor','Virginica
 ↪length','petal width'],out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png('tree.png')
```

[69]: True

0.10 Combining multiple decision trees via random forests

[70]:
```
from sklearn.ensemble import RandomForestClassifier
forest =␣
 ↪RandomForestClassifier(criterion='gini',n_estimators=25,random_state=1,n_jobs=2)
forest.fit(X_train, y_train)
plot_decision_regions(X_combined, y_combined,classifier=forest,␣
 ↪test_idx=range(105,150))
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend(loc='upper left')
plt.show()
```

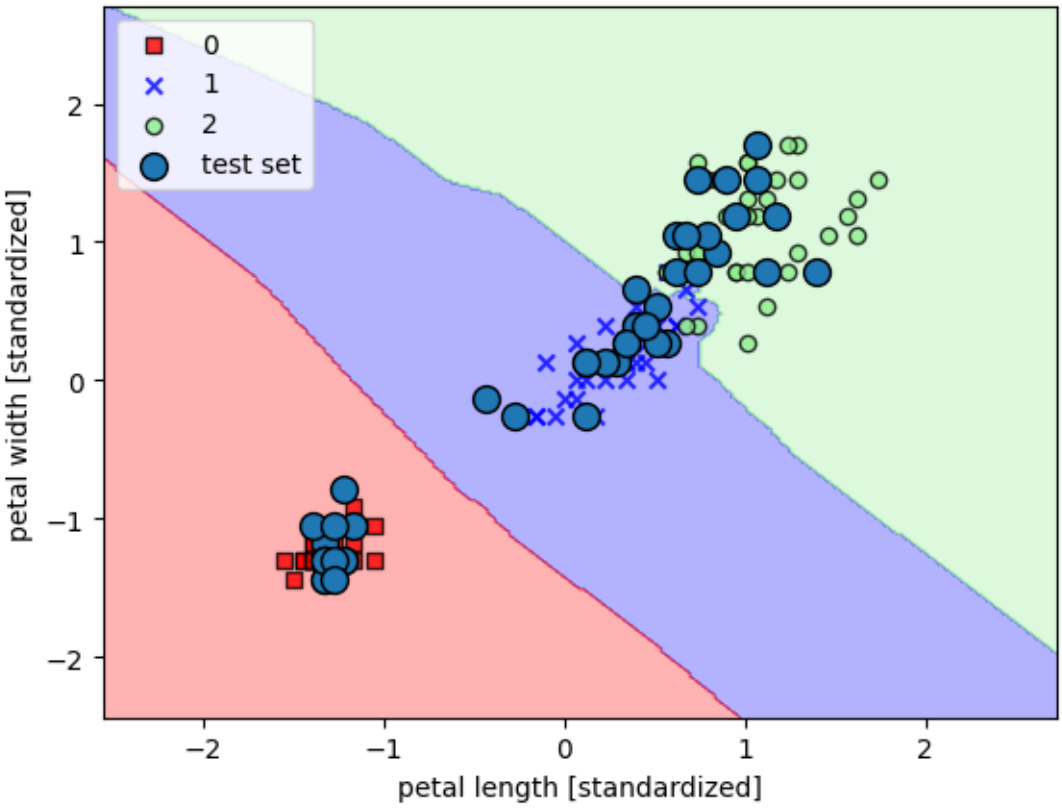
```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```



0.11 KNN - A Lazy Learning Algorithm

[71]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5, p=2,metric='minkowski')
knn.fit(X_train_std, y_train)
plot_decision_regions(X_combined_std, y_combined,classifier=knn,␣
 ↪test_idx=range(105,150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\ankit19.gupta\AppData\Local\Temp\ipykernel_15304\1174551119.py:19:
UserWarning: You passed a edgecolor/edgecolors ('black') for an unfilled marker
('x').  Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.8,
c=colors[idx],marker=markers[idx], label=cl,edgecolor='black')
```


0
1
2
test set
petal width [standardized]
petal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2

[ ]: