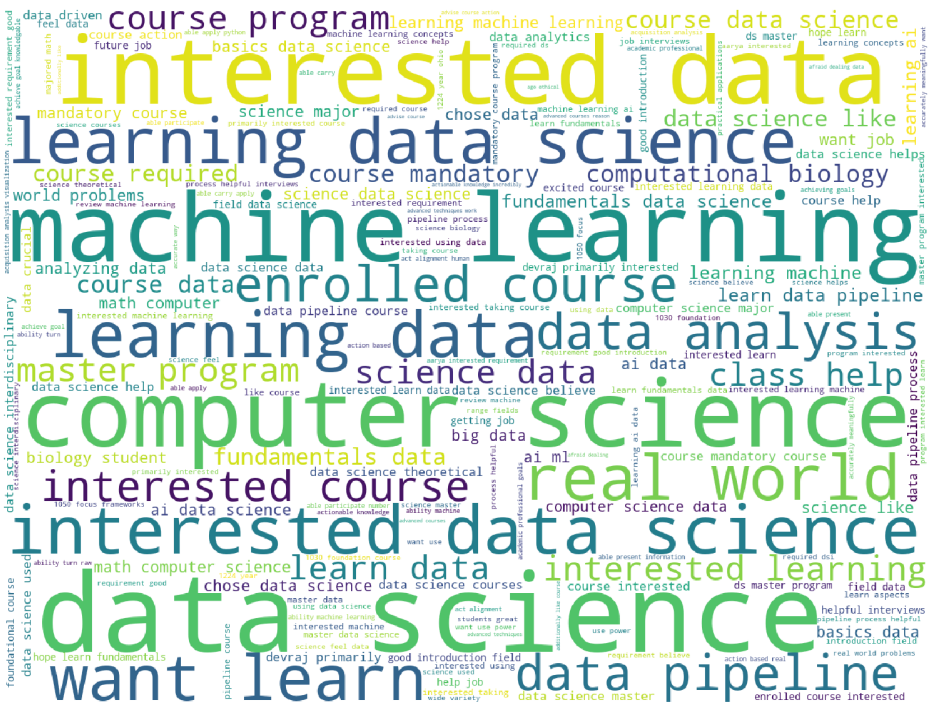
DATA1030: Hands-on Data Science

Intro to ML



Mud card

Admin

- **Will the project be a group work or we have to work on our own?**
  - It's an individual project.
- **any other requirement for the project dataset?**
  - Just the ones I listed on last lecture's slides.
- **When will it be best to begin working on the semester long project!**
  - start looking for datasets now!
  - the term flies by increibly fast and the grade weight of the final project is the same as the problem sets which means you should spend an equal amount of effort on it.
- **This was the first lecture, so I don't really have any muddy moments. I am a bit worried about what dataset I will choose for my project, though, because I have historically had issues finding solid datasets.**
  - Come to the office hours to discuss with me or the TAs.

Supervised ML

- **In CV problems, images are transformed into matrices of the same size. Can the image be considered as structured data at this point?**
  - images are not necessarily of the same size in computer vision but if you do decide to transform them into same size 3D matrices (x and y coordinates plus 3 color channels as the third matrix), you can consider it structured if you flatted the 3D array into a 1D vector.
- **Still a little confused on what the different outcomes when we have continuous or categorical target variables are. Would you be able to provide a simple example to help explain?**
- **When to use the different Y's (classification or regression) is a little muddy for me.**
  - Some examples of categorial target variables (classification) are yes-no problems like "Is the patient sick?", "Should we approve this customer for a loan?", etc. Sometimes there can be more than two categories which is called multi-class classification e.g., "What grade should the student receive for their essay?" with possible categories like A, B, C, NC.
  - Some examples of continuous target variables (regression) are for example "What will XYZ stocks sell tomorrow in USD?", "What will this house sell for in USD?", "What will the crop yield be this year?", etc. There are no distint categories in this case because the target variable is best expressed with a floating point number (which is called a real number in mathematics).
- **What is the best way to convert unstructured data (e.g. images, text, videos, music, etc.) into structured data? Are there certain structured archetypes that are most conducive for these less structured original forms?**
- **Not sure if this is an important topic in our class but the conversion between structured and unstructured data seems interesting!**
  - There is no one best way to go here. The technique depends on the data. I.e., text can be structured using bag of words or n-grams, various sized images can be converted to the same size and then the array can be flattened to a 1D vector, etc.

    - Generally speaking though, unstructured data is usually best left unstructured and you should select techniques that can deal with the data as is (e.g., neural networks).
- **How can you evaluate training data to figure out if it will work well for your supervised ML model?**
    - You cannot do this a priori. You need to train an ML model and measure its performance.
- **You mention there are ways to shift structured data to unstructured and vice versa. How do you determine when that change is necessary or beneficial? What kind of benefits/downsides do you look out for (when is using the "bag of words" better than approaching grading an essay with an ML model as unstructured data)?**
    - You develop ML models with both data representations and compare the model performance.
    - At the end of the day, one of the most important goals if an ML algorithm is to provide accurate predictions.
- **Regression modeling / data modeling vs data analysis!**
    - We have a statistician inthe audience! You'll see that data science is quite different from stats.
- **Just curious whether there are tasks for which machine learning performs better than deep learning, given the dataset is large enough.**
    - Yes, always. This is called the no free lunch theorem and we will discuss it later this term.
- **Codes examples helping us differentiate supervised and unsupervised data!**
    - no code is necessary for that, you only need to take a look at the data.
    - Supervised ML has a target variable Y.
    - Unspervised ML does not, it only has a feature matrix X.
- **I'm still abit confused about what unstructured data means. how can ML algorithm ma,e sense of something that it doesn't even know what to look for.**
    - that's the magic of deep learning!

Other ML areas

- **Maybe out of the scope of this course, but I am curious as to why recommender systems are considered to be independent from unsupervised ML, or is there some overlap in the methods used?**
    - the goals are very different. unsupervised ML tries to find clusters in the data while in recommender systems, as the name suggests, the goal is to recommend products to a customer.

Learning objectives

By the end of the lecture, you will be able to

- describe the main goals of the ML pipeline
- list the main steps of the ML pipeline
- explain the bias-variance trade off

Learning objectives

By the end of the lecture, you will be able to

- **describe the main goals of the ML pipeline**
- list the main steps of the ML pipeline
- explain the bias-variance trade off

An ML example

- let's assume you just moved to an island and you never had papayas before but it is common on the island
- what do you do?
- sample some papayas and collect some info
    - for each papaya you try, you collect color, firmness, and whether it tasted good or not
    - classification problem with two features
- once you have enough data, you can train a machine learning model to predict if a new previously unseen papaya is tasty or not based on its color and firmness

Let's define this problem from a statistician's point of view!

- **the learner's input**
    - Domain set $\mathcal{X}$ - a set of objects we wish to label. In the papaya example: the set of all papayas. $\mathcal{X}$ can be in infinite set or a set that's too large to handle on any computer (e.g., all possible 640x480 images with 3 color channels and 256 possible pixel values)
        - domain points are represented by a vector of features e.g., (color, firmness)
        - domain points are also called instances, and $\mathcal{X}$ is also called the instance space
    - Label set $\mathcal{Y}$ - a set of possible labels. In the papaya example: we restrict our label set to {0,1}, 0 meaning the papaya tastes bad, 1 meaning the papaya tastes good.
        - such a label set is categorical, i.e., we have a classification problem at hand

        - the label set can be continuous too, e.g., the real number between 0 and 1, meaning that 0.5 is an OK tasting papaya.
        - the label set can also be probabilistic
            - i.e., two papayas with the same color and firmness can sometimes be tasty and sometimes bad
            - this is quite normal, the features you collect usually do not uniquely determine the label
    - Training data $S = ((x_1, y_1), \ldots, (x_m, y_m))$ - a finite sequence of pairs from $\mathcal{X}$, $\mathcal{Y}$. This is what the learner has access to.
        - S is also called the training set, and examples in S are also called training examples
        - $X = (x_1, \ldots, x_m)$ is the feature matrix which is usually a 2D matrix, and $Y = (y_1, \ldots, y_m)$ is the target variable which is a vector.
- **the learner's output**
    - a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ - this is also called the predictor, a hypothesis, or in the papaya example a classifier. It would be a regressor if $\mathcal{Y}$ was continuous. In the papaya example, the predictor is the rule that our learner will employ to predict if a papaya will be tasty based on color and firmness as they examine it e.g., in the farmer's market or before picking the fruit from the tree.
    - this prediction rule is generated based on S so $h : X \to Y$ is more appropriate
    - once the prediction rule is determined, we can use it to predict the label to previously unseen data

How is S used?

- in ML, you only use part of S to train the model
- you hold out some fraction of S to calculate what's called the **generalization error**
- it measures how well the model is expected to perform on previously unseen data
- it helps to avoid models that overfit or underfit
    - overfit: model is too complex, it performs very well on the training set but it doesn't generalize to previously unseen data
    - underfit: the model is too simple, it performs poorly on te training set and on previously unseen data as well

Recap the goals:

- use the training data (X and y) to develop a model which can accurately predict the target variable (y_new') for previously unseen data (X_new)
    - model performance or 'accuracy' is a metric you need to choose to measure model performance and objectively compare various models
- measure the generalization error: measure how well the model is expected to perform on previously unseen data

Quiz

Learning objectives

By the end of the lecture, you will be able to

- describe the main goals of the ML pipeline
- **list the main steps of the ML pipeline**
- explain the bias-variance trade off

The steps

1. Exploratory Data Analysis (EDA): you need to understand your data and verify that it doesn't contain errors

- do as much EDA as you can!

2. Split the data into different sets: most often the sets are train, validation, and test (or holdout)

- practitioners often make errors in this step!
- you can split the data randomly, based on groups, based on time, or any other non-standard way if necessary to answer your ML question

3. Preprocess the data: ML models only work if X and Y are numbers! Some ML models additionally require each feature to have 0 mean and 1 standard deviation (standardized features)

- often the original features you get contain strings (for example a gender feature would contain 'male', 'female', 'non-binary', 'unknown') which needs to be transformed into numbers
- often the features are not standardized (e.g., age is between 0 and 100) but it needs to be standardized

4. Choose an evaluation metric: depends on the priorities of the stakeholders

- often requires quite a bit of thinking and ethical considerations

5. Choose one or more ML techniques: it is highly recommended that you try multiple models

- start with simple models like linear or logistic regression
- try also more complex models like nearest neighbors, support vector machines, random forest, etc.

6. Tune the hyperparameters of your ML models (aka cross-validation)

- ML techniques have hyperparameters that you need to optimize to achieve best performance
- for each ML model, decide which parameters to tune and what values to try
- loop through each parameter combination
  - train one model for each parameter combination
  - evaluate how well the model performs on the validation set
- take the parameter combo that gives the best validation score
- evaluate that model on the test set to report how well the model is expected to perform on previously unseen data

7. Interpret your model: black boxes are often not useful

- check if your model uses features that make sense (excellent tool for debugging)
- often model predictions are not enough, you need to be able to explain how the model arrived to a particular prediction (e.g., in health care)

Quiz

Learning objectives

By the end of the lecture, you will be able to

- describe the main goals of the ML pipeline
- list the main steps of the ML pipeline
- **explain the bias-variance trade off**

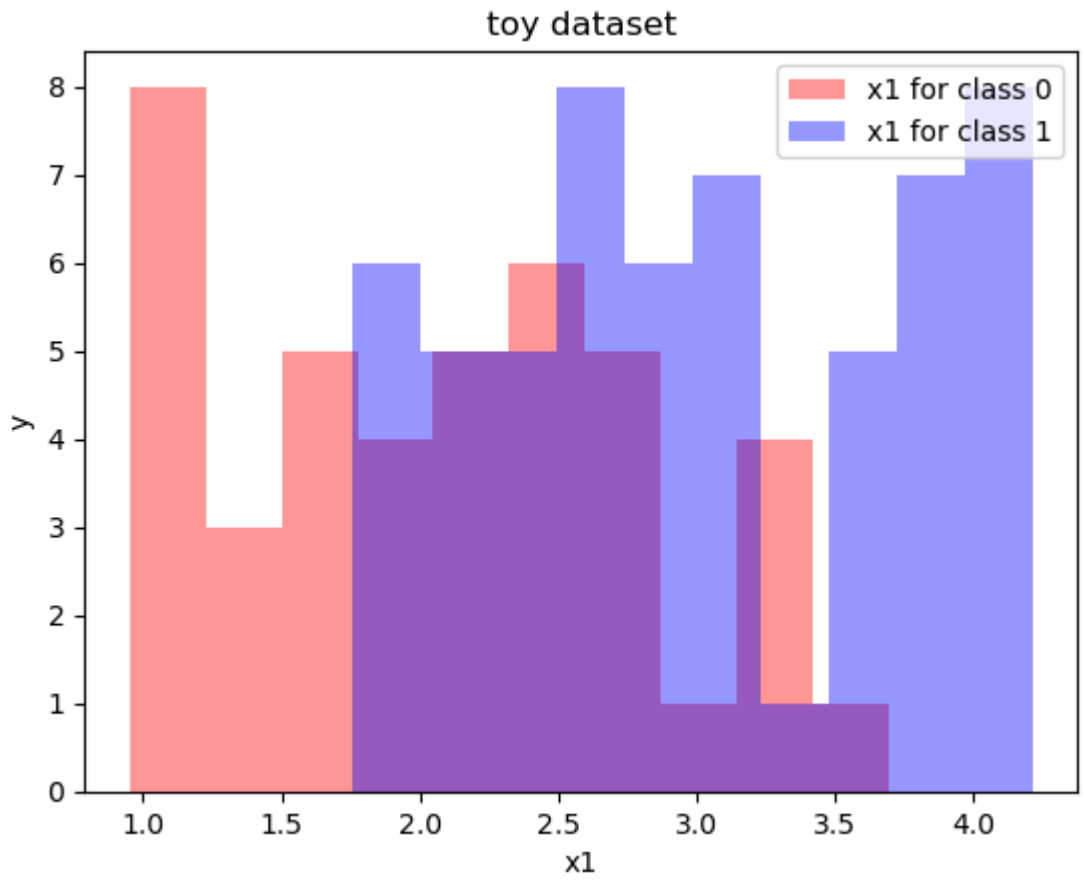
Bias-variance tradeoff illustrated through a simple ML pipeline

In [1]:
```
# import packages

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from matplotlib import pylab as plt
import matplotlib
from matplotlib.colors import ListedColormap
%matplotlib inline

# scikit-learn code is reproducable is the random seed is fixed.
np.random.seed(2)

# read in the data
# our toy dataset, we don't know how it was generated.
df = pd.read_csv('data/toy_data.csv')

X = df[['x1','x2']].values
y = df['y'].values

print(np.shape(X))
print(np.shape(y))
print(np.unique(y,return_counts=True))
```

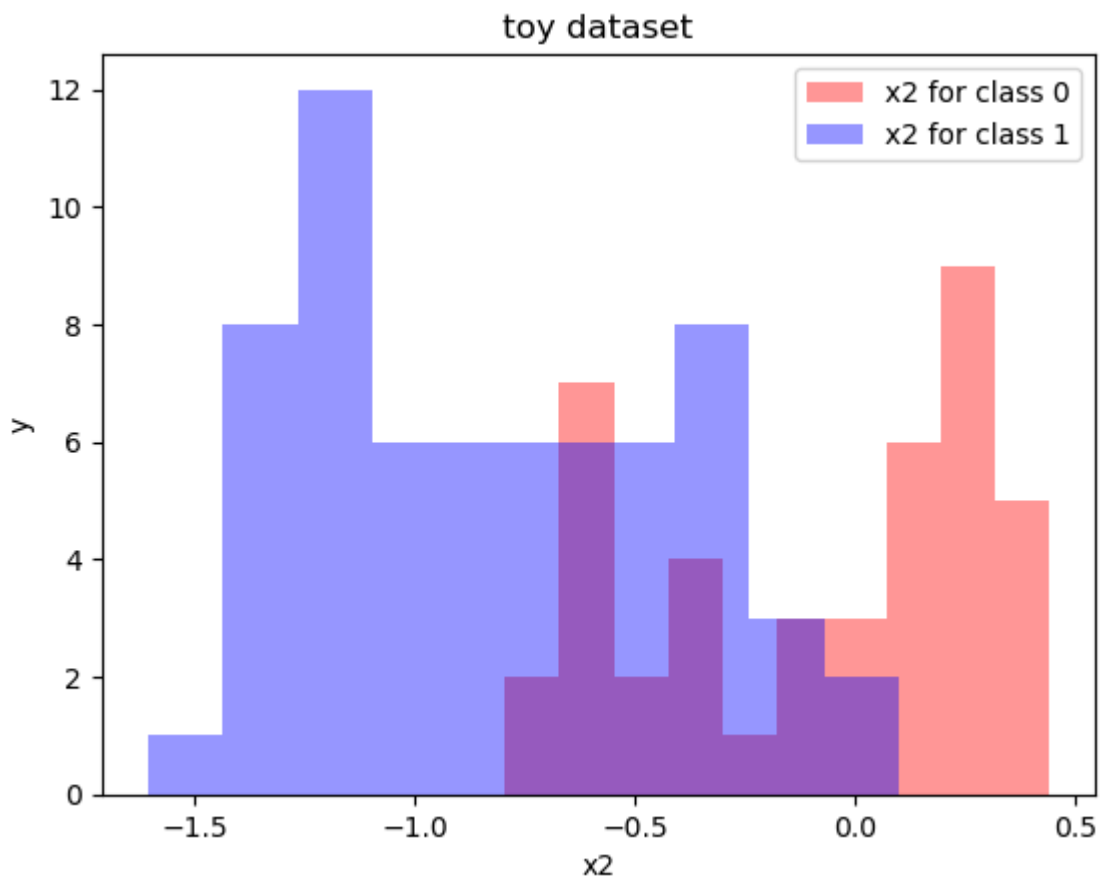
```
(100, 2)
(100,)
(array([0, 1]), array([42, 58]))
```

1. Exploratory Data Analysis (EDA)

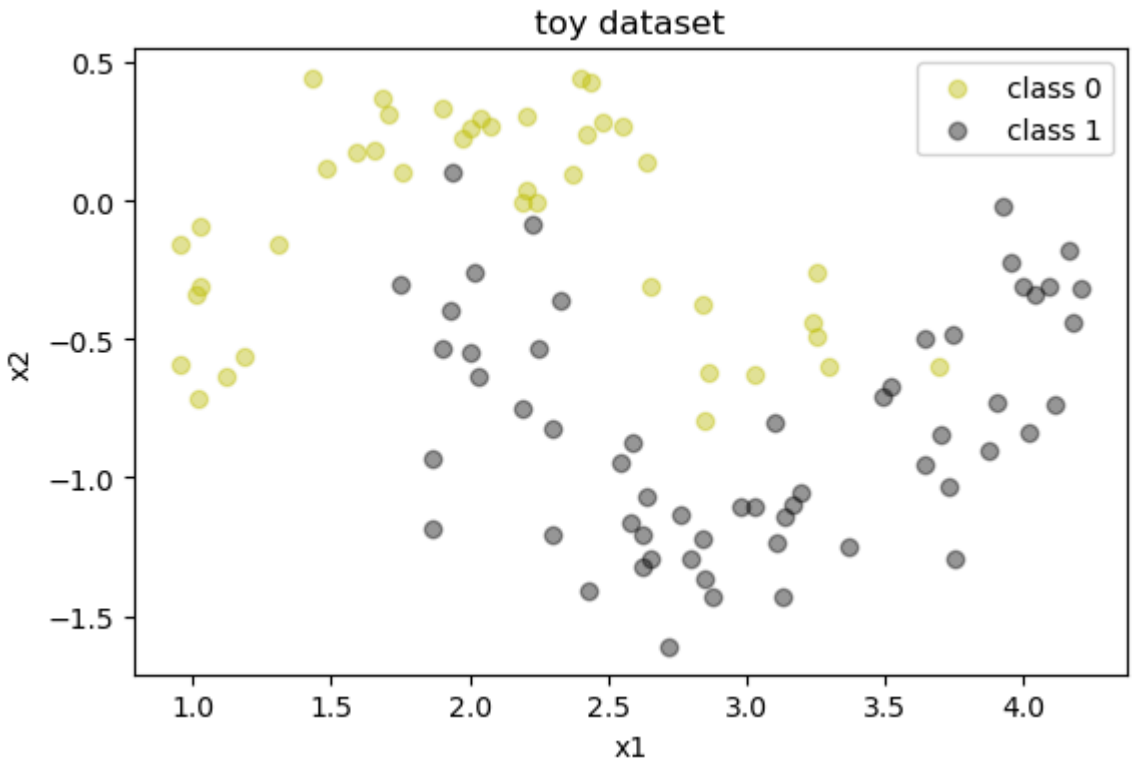
In [2]:
```
plt.hist(X[y==0,0],alpha=0.4,color='r',label='x1 for class 0')
plt.hist(X[y==1,0],alpha=0.4,color='b',label='x1 for class 1')
plt.xlabel('x1')
plt.ylabel('y')
plt.title('toy dataset')
plt.legend()
plt.show()
```



In [3]:
```
plt.hist(X[y==0,1],alpha=0.4,color='r',label='x2 for class 0')
plt.hist(X[y==1,1],alpha=0.4,color='b',label='x2 for class 1')
plt.xlabel('x2')
plt.ylabel('y')
plt.title('toy dataset')
plt.legend()
plt.show()
```



In [4]:
```
plt.scatter(X[y==0,0],X[y==0,1],color='y',label='class 0',alpha=0.4)
plt.scatter(X[y==1,0],X[y==1,1],color='k',label='class 1',alpha=0.4)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('toy dataset')
plt.gca().set_aspect('equal')
plt.legend()
plt.show()
```



2. Split the data into different sets

In [5]:
```
help(train_test_split)
```

```
Help on function train_test_split in module sklearn.model_selection._split:

train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
    Split arrays or matrices into random train and test subsets.

    Quick utility that wraps input validation,
    ``next(ShuffleSplit().split(X, y))``, and application to input data
    into a single call for splitting (and optionally subsampling) data into a
    one-liner.

    Read more in the :ref:`User Guide <cross_validation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same length / shape[0]
        Allowed inputs are lists, numpy arrays, scipy-sparse
        matrices or pandas dataframes.

    test_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the proportion
        of the dataset to include in the test split. If int, represents the
        absolute number of test samples. If None, the value is set to the
        complement of the train size. If ``train_size`` is also None, it will
        be set to 0.25.

    train_size : float or int, default=None
        If float, should be between 0.0 and 1.0 and represent the
        proportion of the dataset to include in the train split. If
        int, represents the absolute number of train samples. If None,
        the value is automatically set to the complement of the test size.

    random_state : int, RandomState instance or None, default=None
        Controls the shuffling applied to the data before applying the split.
        Pass an int for reproducible output across multiple function calls.
        See :term:`Glossary <random_state>`.

    shuffle : bool, default=True
        Whether or not to shuffle the data before splitting. If shuffle=False
        then stratify must be None.

    stratify : array-like, default=None
        If not None, data is split in a stratified fashion, using this as
        the class labels.
        Read more in the :ref:`User Guide <stratification>`.

    Returns
    -------
    splitting : list, length=2 * len(arrays)
        List containing train-test split of inputs.

        .. versionadded:: 0.16
            If the input is sparse, the output will be a
            ``scipy.sparse.csr_matrix``. Else, output type is the same as the
            input type.

    Examples
    --------
    >>> import numpy as np
    >>> from sklearn.model_selection import train_test_split
    >>> X, y = np.arange(10).reshape((5, 2)), range(5)
    >>> X
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7],
           [8, 9]])
    >>> list(y)
    [0, 1, 2, 3, 4]

    >>> X_train, X_test, y_train, y_test = train_test_split(
    ...     X, y, test_size=0.33, random_state=42)
    ...
    >>> X_train
    array([[4, 5],
           [0, 1],
           [6, 7]])
    >>> y_train
    [2, 0, 3]
    >>> X_test
    array([[2, 3],
           [8, 9]])
    >>> y_test
    [1, 4]

    >>> train_test_split(y, shuffle=False)
    [[0, 1, 2], [3, 4]]
```

In [6]:
```
X_train, X_other, y_train, y_other = train_test_split(X,y,test_size=0.4)
print(np.shape(X_other),np.shape(y_other))
print('train:',np.shape(X_train),np.shape(y_train))

X_val, X_test, y_val, y_test = train_test_split(X_other,y_other,test_size=0.5)
print('val:',np.shape(X_val),np.shape(y_val))
print('test:',np.shape(X_test),np.shape(y_test))
```

```
(40, 2) (40,)
train: (60, 2) (60,)
val: (20, 2) (20,)
test: (20, 2) (20,)
```

3. Preprocess the data

In [7]:
```
help(StandardScaler)
```

```
Help on class StandardScaler in module sklearn.preprocessing._data:

class StandardScaler(sklearn.base.OneToOneFeatureMixin, sklearn.base.TransformerMixin, sklearn.base.BaseEstimator)
 |  StandardScaler(*, copy=True, with_mean=True, with_std=True)
 |
 |  Standardize features by removing the mean and scaling to unit variance.
 |
 |  The standard score of a sample `x` is calculated as:
 |
 |      z = (x - u) / s
 |
 |  where `u` is the mean of the training samples or zero if `with_mean=False`,
 |  and `s` is the standard deviation of the training samples or one if
 |  `with_std=False`.
 |
 |  Centering and scaling happen independently on each feature by computing
 |  the relevant statistics on the samples in the training set. Mean and
 |  standard deviation are then stored to be used on later data using
 |  :meth:`transform`.
 |
 |  Standardization of a dataset is a common requirement for many
 |  machine learning estimators: they might behave badly if the
 |  individual features do not more or less look like standard normally
 |  distributed data (e.g. Gaussian with 0 mean and unit variance).
 |
 |  For instance many elements used in the objective function of
 |  a learning algorithm (such as the RBF kernel of Support Vector
 |  Machines or the L1 and L2 regularizers of linear models) assume that
 |  all features are centered around 0 and have variance in the same
 |  order. If a feature has a variance that is orders of magnitude larger
 |  than others, it might dominate the objective function and make the
 |  estimator unable to learn from other features correctly as expected.
 |
 |  `StandardScaler` is sensitive to outliers, and the features may scale
 |  differently from each other in the presence of outliers. For an example
 |  visualization, refer to :ref:`Compare StandardScaler with other scalers
 |  <plot_all_scaling_standard_scaler_section>`.
 |
 |  This scaler can also be applied to sparse CSR or CSC matrices by passing
 |  `with_mean=False` to avoid breaking the sparsity structure of the data.
 |
 |  Read more in the :ref:`User Guide <preprocessing_scaler>`.
 |
 |  Parameters
 |  ----------
 |  copy : bool, default=True
 |      If False, try to avoid a copy and do inplace scaling instead.
 |      This is not guaranteed to always work inplace; e.g. if the data is
 |      not a NumPy array or scipy.sparse CSR matrix, a copy may still be
 |      returned.
 |
 |  with_mean : bool, default=True
 |      If True, center the data before scaling.
 |      This does not work (and will raise an exception) when attempted on
 |      sparse matrices, because centering them entails building a dense
 |      matrix which in common use cases is likely to be too large to fit in
 |      memory.
 |
 |  with_std : bool, default=True
 |      If True, scale the data to unit variance (or equivalently,
 |      unit standard deviation).
 |
 |  Attributes
 |  ----------
 |  scale_ : ndarray of shape (n_features,) or None
 |      Per feature relative scaling of the data to achieve zero mean and unit
 |      variance. Generally this is calculated using `np.sqrt(var_)`. If a
 |      variance is zero, we can't achieve unit variance, and the data is left
 |      as-is, giving a scaling factor of 1. `scale_` is equal to `None`
 |      when `with_std=False`.
 |
 |      .. versionadded:: 0.17
 |         *scale_*
 |
 |  mean_ : ndarray of shape (n_features,) or None
 |      The mean value for each feature in the training set.
 |      Equal to ``None`` when ``with_mean=False`` and ``with_std=False``.
 |
 |  var_ : ndarray of shape (n_features,) or None
 |      The variance for each feature in the training set. Used to compute
 |      `scale_`. Equal to ``None`` when ``with_mean=False`` and
 |      ``with_std=False``.
 |
 |  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_samples_seen_ : int or ndarray of shape (n_features,)
|      The number of samples processed by the estimator for each feature.
|      If there are no missing samples, the ``n_samples_seen`` will be an
|      integer, otherwise it will be an array of dtype int. If
|      `sample_weights` are used it will be a float (if no missing data)
|      or an array of dtype float that sums the weights seen so far.
|      Will be reset on new calls to fit, but increments across
|      ``partial_fit`` calls.
|
|  See Also
|  --------
|  scale : Equivalent function without the estimator API.
|
|  :class:`~sklearn.decomposition.PCA` : Further removes the linear
|      correlation across features with 'whiten=True'.
|
|  Notes
|  -----
|  NaNs are treated as missing values: disregarded in fit, and maintained in
|  transform.
|
|  We use a biased estimator for the standard deviation, equivalent to
|  `numpy.std(x, ddof=0)`. Note that the choice of `ddof` is unlikely to
|  affect model performance.
|
|  Examples
|  --------
|  >>> from sklearn.preprocessing import StandardScaler
|  >>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
|  >>> scaler = StandardScaler()
|  >>> print(scaler.fit(data))
|  StandardScaler()
|  >>> print(scaler.mean_)
|  [0.5 0.5]
|  >>> print(scaler.transform(data))
|  [[-1. -1.]
|   [-1. -1.]
|   [ 1.  1.]
|   [ 1.  1.]]
|  >>> print(scaler.transform([[2, 2]]))
|  [[3. 3.]]
|
|  Method resolution order:
|      StandardScaler
|      sklearn.base.OneToOneFeatureMixin
|      sklearn.base.TransformerMixin
|      sklearn.utils._set_output._SetOutputMixin
|      sklearn.base.BaseEstimator
|      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
|      sklearn.utils._metadata_requests._MetadataRequester
|      builtins.object
|
|  Methods defined here:
|
|  __init__(self, *, copy=True, with_mean=True, with_std=True)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  fit(self, X, y=None, sample_weight=None)
|      Compute the mean and std to be used for later scaling.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The data used to compute the mean and standard deviation
|          used for later scaling along the features axis.
|
|      y : None
|          Ignored.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Individual weights for each sample.
|
|          .. versionadded:: 0.24
|             parameter *sample_weight* support to StandardScaler.
|
|      Returns
|      -------
|      self : object
|          Fitted scaler.
|
```

```
|  inverse_transform(self, X, copy=None)
|      Scale back the data to the original representation.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The data used to scale along the features axis.
|      copy : bool, default=None
|          Copy the input X or not.
|
|      Returns
|      -------
|      X_tr : {ndarray, sparse matrix} of shape (n_samples, n_features)
|          Transformed array.
|
|  partial_fit(self, X, y=None, sample_weight=None)
|      Online computation of mean and std on X for later scaling.
|
|      All of X is processed as a single batch. This is intended for cases
|      when :meth:`fit` is not feasible due to very large number of
|      `n_samples` or because X is read from a continuous stream.
|
|      The algorithm for incremental mean and std is given in Equation 1.5a,b
|      in Chan, Tony F., Gene H. Golub, and Randall J. LeVeque. "Algorithms
|      for computing the sample variance: Analysis and recommendations."
|      The American Statistician 37.3 (1983): 242-247:
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The data used to compute the mean and standard deviation
|          used for later scaling along the features axis.
|
|      y : None
|          Ignored.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Individual weights for each sample.
|
|          .. versionadded:: 0.24
|             parameter *sample_weight* support to StandardScaler.
|
|      Returns
|      -------
|      self : object
|          Fitted scaler.
|
|  set_fit_request(self: sklearn.preprocessing._data.StandardScaler, *, sample_weight: Union[bool, NoneType, str] =
'$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.RequestMethod.__g
et__.<locals>
|      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 nam
e.
|
|      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.UNCHAN
GED
|          Metadata routing for ``sample_weight`` parameter in ``fit``.
|
|      Returns
```

```
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_inverse_transform_request(self: sklearn.preprocessing._data.StandardScaler, *, copy: Union[bool, NoneType, s
tr] = '$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.RequestMeth
od.__get__.<locals>
 |      Request metadata passed to the ``inverse_transform`` 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 ``inverse_transform`` if provided. The request is ignored i
f metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not pass it to ``inverse_transform``.
 |
 |      - ``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 nam
e.
 |
 |      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
 |      ----------
 |      copy : str, True, False, or None,                     default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``copy`` parameter in ``inverse_transform``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_partial_fit_request(self: sklearn.preprocessing._data.StandardScaler, *, sample_weight: Union[bool, NoneTyp
e, str] = '$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.Request
Method.__get__.<locals>
 |      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 meta
data 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 nam
e.
 |
 |      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.UNCHAN
GED
 |          Metadata routing for ``sample_weight`` parameter in ``partial_fit``.
 |
 |      Returns
```

```
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_transform_request(self: sklearn.preprocessing._data.StandardScaler, *, copy: Union[bool, NoneType, str] =
'$UNCHANGED$') -> sklearn.preprocessing._data.StandardScaler from sklearn.utils._metadata_requests.RequestMethod.__g
et__.<locals>
 |      Request metadata passed to the ``transform`` 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 ``transform`` if provided. The request is ignored if metada
ta is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will not pass it to ``transform``.
 |
 |      - ``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 nam
e.
 |
 |      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
 |      ----------
 |      copy : str, True, False, or None,                     default=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``copy`` parameter in ``transform``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  transform(self, X, copy=None)
 |      Perform standardization by centering and scaling.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix of shape (n_samples, n_features)
 |          The data used to scale along the features axis.
 |      copy : bool, default=None
 |          Copy the input X or not.
 |
 |      Returns
 |      -------
 |      X_tr : {ndarray, sparse matrix} of shape (n_samples, n_features)
 |          Transformed array.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.OneToOneFeatureMixin:
 |
 |  get_feature_names_out(self, input_features=None)
 |      Get output feature names for transformation.
 |
 |      Parameters
 |      ----------
 |      input_features : array-like of str or None, default=None
 |          Input features.
 |
 |          - If `input_features` is `None`, then `feature_names_in_` is
 |            used as feature names in. If `feature_names_in_` is not defined,
 |            then the following input feature names are generated:
 |            `["x0", "x1", ..., "x(n_features_in_ - 1)"]`.
 |          - If `input_features` is an array-like, then `input_features` must
 |            match `feature_names_in_` if `feature_names_in_` is defined.
 |
 |      Returns
 |      -------
```

```
|      feature_names_out : ndarray of str objects
|          Same as input features.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.OneToOneFeatureMixin:
|
|  __dict__
|      dictionary for instance variables
|
|  __weakref__
|      list of weak references to the object
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.TransformerMixin:
|
|  fit_transform(self, X, y=None, **fit_params)
|      Fit to data, then transform it.
|
|      Fits transformer to `X` and `y` with optional parameters `fit_params`
|      and returns a transformed version of `X`.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Input samples.
|
|      y :  array-like of shape (n_samples,) or (n_samples, n_outputs),                default=None
|          Target values (None for unsupervised transformations).
|
|      **fit_params : dict
|          Additional fit parameters.
|
|      Returns
|      -------
|      X_new : ndarray array of shape (n_samples, n_features_new)
|          Transformed array.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.utils._set_output._SetOutputMixin:
|
|  set_output(self, *, transform=None)
|      Set output container.
|
|      See :ref:`sphx_glr_auto_examples_miscellaneous_plot_set_output.py`
|      for an example on how to use the API.
|
|      Parameters
|      ----------
|      transform : {"default", "pandas", "polars"}, default=None
|          Configure output of `transform` and `fit_transform`.
|
|          - `"default"`: Default output format of a transformer
|          - `"pandas"`: DataFrame output
|          - `"polars"`: Polars output
|          - `None`: Transform configuration is unchanged
|
|          .. versionadded:: 1.4
|              `"polars"` option was added.
|
|      Returns
|      -------
|      self : estimator instance
|          Estimator instance.
|
|  ----------------------------------------------------------------------
|  Class methods inherited from sklearn.utils._set_output._SetOutputMixin:
|
|  __init_subclass__(auto_wrap_output_keys=('transform',), **kwargs)
|      This method is called when a class is subclassed.
|
|      The default implementation does nothing. It may be
|      overridden to extend subclasses.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|      Helper for pickle.
|
|  __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.
```

In [8]:
```
scaler = StandardScaler().fit(X_train)
# the scaler object contains the feature means and variations in the training set
print(scaler.mean_)
print(scaler.var_)

# the scaler is used to transform the sets
X_train_prep = scaler.transform(X_train)
X_val_prep = scaler.transform(X_val)
X_test_prep = scaler.transform(X_test)
```

```
[ 2.61729782 -0.55283401]
[0.74350517 0.32379089]
```

4. Choose an evaluation metric

In [9]:
```
help(accuracy_score)
```

```
Help on function accuracy_score in module sklearn.metrics._classification:

accuracy_score(y_true, y_pred, *, normalize=True, sample_weight=None)
    Accuracy classification score.

    In multilabel classification, this function computes subset accuracy:
    the set of labels predicted for a sample must *exactly* match the
    corresponding set of labels in y_true.

    Read more in the :ref:`User Guide <accuracy_score>`.

    Parameters
    ----------
    y_true : 1d array-like, or label indicator array / sparse matrix
        Ground truth (correct) labels.

    y_pred : 1d array-like, or label indicator array / sparse matrix
        Predicted labels, as returned by a classifier.

    normalize : bool, default=True
        If ``False``, return the number of correctly classified samples.
        Otherwise, return the fraction of correctly classified samples.

    sample_weight : array-like of shape (n_samples,), default=None
        Sample weights.

    Returns
    -------
    score : float or int
        If ``normalize == True``, return the fraction of correctly
        classified samples (float), else returns the number of correctly
        classified samples (int).

        The best performance is 1 with ``normalize == True`` and the number
        of samples with ``normalize == False``.

    See Also
    --------
    balanced_accuracy_score : Compute the balanced accuracy to deal with
        imbalanced datasets.
    jaccard_score : Compute the Jaccard similarity coefficient score.
    hamming_loss : Compute the average Hamming loss or Hamming distance between
        two sets of samples.
    zero_one_loss : Compute the Zero-one classification loss. By default, the
        function will return the percentage of imperfectly predicted subsets.

    Examples
    --------
    >>> from sklearn.metrics import accuracy_score
    >>> y_pred = [0, 2, 1, 3]
    >>> y_true = [0, 1, 2, 3]
    >>> accuracy_score(y_true, y_pred)
    0.5
    >>> accuracy_score(y_true, y_pred, normalize=False)
    2.0

    In the multilabel case with binary label indicators:

    >>> import numpy as np
    >>> accuracy_score(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))
    0.5
```

Quiz

5. Choose one or more ML techniques

In [10]:
```
help(SVC)
```

```
Help on class SVC in module sklearn.svm._classes:

class SVC(sklearn.svm._base.BaseSVC)
 |  SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, ca
che_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random
_state=None)
 |
 |  C-Support Vector Classification.
 |
 |  The implementation is based on libsvm. The fit time scales at least
 |  quadratically with the number of samples and may be impractical
 |  beyond tens of thousands of samples. For large datasets
 |  consider using :class:`~sklearn.svm.LinearSVC` or
 |  :class:`~sklearn.linear_model.SGDClassifier` instead, possibly after a
 |  :class:`~sklearn.kernel_approximation.Nystroem` transformer or
 |  other :ref:`kernel_approximation`.
 |
 |  The multiclass support is handled according to a one-vs-one scheme.
 |
 |  For details on the precise mathematical formulation of the provided
 |  kernel functions and how `gamma`, `coef0` and `degree` affect each
 |  other, see the corresponding section in the narrative documentation:
 |  :ref:`svm_kernels`.
 |
 |  To learn how to tune SVC's hyperparameters, see the following example:
 |  :ref:`sphx_glr_auto_examples_model_selection_plot_nested_cross_validation_iris.py`
 |
 |  Read more in the :ref:`User Guide <svm_classification>`.
 |
 |  Parameters
 |  ----------
 |  C : float, default=1.0
 |      Regularization parameter. The strength of the regularization is
 |      inversely proportional to C. Must be strictly positive. The penalty
 |      is a squared l2 penalty. For an intuitive visualization of the effects
 |      of scaling the regularization parameter C, see
 |      :ref:`sphx_glr_auto_examples_svm_plot_svm_scale_c.py`.
 |
 |  kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable,          default='rbf'
 |      Specifies the kernel type to be used in the algorithm. If
 |      none is given, 'rbf' will be used. If a callable is given it is used to
 |      pre-compute the kernel matrix from data matrices; that matrix should be
 |      an array of shape ``(n_samples, n_samples)``. For an intuitive
 |      visualization of different kernel types see
 |      :ref:`sphx_glr_auto_examples_svm_plot_svm_kernels.py`.
 |
 |  degree : int, default=3
 |      Degree of the polynomial kernel function ('poly').
 |      Must be non-negative. Ignored by all other kernels.
 |
 |  gamma : {'scale', 'auto'} or float, default='scale'
 |      Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
 |
 |      - if ``gamma='scale'`` (default) is passed then it uses
 |        1 / (n_features * X.var()) as value of gamma,
 |      - if 'auto', uses 1 / n_features
 |      - if float, must be non-negative.
 |
 |      .. versionchanged:: 0.22
 |         The default value of ``gamma`` changed from 'auto' to 'scale'.
 |
 |  coef0 : float, default=0.0
 |      Independent term in kernel function.
 |      It is only significant in 'poly' and 'sigmoid'.
 |
 |  shrinking : bool, default=True
 |      Whether to use the shrinking heuristic.
 |      See the :ref:`User Guide <shrinking_svm>`.
 |
 |  probability : bool, default=False
 |      Whether to enable probability estimates. This must be enabled prior
 |      to calling `fit`, will slow down that method as it internally uses
 |      5-fold cross-validation, and `predict_proba` may be inconsistent with
 |      `predict`. Read more in the :ref:`User Guide <scores_probabilities>`.
 |
 |  tol : float, default=1e-3
 |      Tolerance for stopping criterion.
 |
 |  cache_size : float, default=200
 |      Specify the size of the kernel cache (in MB).
 |
 |  class_weight : dict or 'balanced', default=None
 |      Set the parameter C of class i to class_weight[i]*C for
 |      SVC. 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))``.
|
|  verbose : bool, default=False
|      Enable verbose output. Note that this setting takes advantage of a
|      per-process runtime setting in libsvm that, if enabled, may not work
|      properly in a multithreaded context.
|
|  max_iter : int, default=-1
|      Hard limit on iterations within solver, or -1 for no limit.
|
|  decision_function_shape : {'ovo', 'ovr'}, default='ovr'
|      Whether to return a one-vs-rest ('ovr') decision function of shape
|      (n_samples, n_classes) as all other classifiers, or the original
|      one-vs-one ('ovo') decision function of libsvm which has shape
|      (n_samples, n_classes * (n_classes - 1) / 2). However, note that
|      internally, one-vs-one ('ovo') is always used as a multi-class strategy
|      to train models; an ovr matrix is only constructed from the ovo matrix.
|      The parameter is ignored for binary classification.
|
|      .. versionchanged:: 0.19
|          decision_function_shape is 'ovr' by default.
|
|      .. versionadded:: 0.17
|         *decision_function_shape='ovr'* is recommended.
|
|      .. versionchanged:: 0.17
|         Deprecated *decision_function_shape='ovo' and None*.
|
|  break_ties : bool, default=False
|      If true, ``decision_function_shape='ovr'``, and number of classes > 2,
|      :term:`predict` will break ties according to the confidence values of
|      :term:`decision_function`; otherwise the first class among the tied
|      classes is returned. Please note that breaking ties comes at a
|      relatively high computational cost compared to a simple predict.
|
|      .. versionadded:: 0.22
|
|  random_state : int, RandomState instance or None, default=None
|      Controls the pseudo random number generation for shuffling the data for
|      probability estimates. Ignored when `probability` is False.
|      Pass an int for reproducible output across multiple function calls.
|      See :term:`Glossary <random_state>`.
|
|  Attributes
|  ----------
|  class_weight_ : ndarray of shape (n_classes,)
|      Multipliers of parameter C for each class.
|      Computed based on the ``class_weight`` parameter.
|
|  classes_ : ndarray of shape (n_classes,)
|      The classes labels.
|
|  coef_ : ndarray of shape (n_classes * (n_classes - 1) / 2, n_features)
|      Weights assigned to the features (coefficients in the primal
|      problem). This is only available in the case of a linear kernel.
|
|      `coef_` is a readonly property derived from `dual_coef_` and
|      `support_vectors_`.
|
|  dual_coef_ : ndarray of shape (n_classes -1, n_SV)
|      Dual coefficients of the support vector in the decision
|      function (see :ref:`sgd_mathematical_formulation`), multiplied by
|      their targets.
|      For multiclass, coefficient for all 1-vs-1 classifiers.
|      The layout of the coefficients in the multiclass case is somewhat
|      non-trivial. See the :ref:`multi-class section of the User Guide
|      <svm_multi_class>` for details.
|
|  fit_status_ : int
|      0 if correctly fitted, 1 otherwise (will raise warning)
|
|  intercept_ : ndarray of shape (n_classes * (n_classes - 1) / 2,)
|      Constants in decision function.
|
|  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_ : ndarray of shape (n_classes * (n_classes - 1) // 2,)
|      Number of iterations run by the optimization routine to fit the model.
```

```
|      The shape of this attribute depends on the number of models optimized
|      which in turn depends on the number of classes.
|
|      .. versionadded:: 1.1
|
|  support_ : ndarray of shape (n_SV)
|      Indices of support vectors.
|
|  support_vectors_ : ndarray of shape (n_SV, n_features)
|      Support vectors. An empty array if kernel is precomputed.
|
|  n_support_ : ndarray of shape (n_classes,), dtype=int32
|      Number of support vectors for each class.
|
|  probA_ : ndarray of shape (n_classes * (n_classes - 1) / 2)
|  probB_ : ndarray of shape (n_classes * (n_classes - 1) / 2)
|      If `probability=True`, it corresponds to the parameters learned in
|      Platt scaling to produce probability estimates from decision values.
|      If `probability=False`, it's an empty array. Platt scaling uses the
|      logistic function
|      ``1 / (1 + exp(decision_value * probA_ + probB_))``
|      where ``probA_`` and ``probB_`` are learned from the dataset [2]_. For
|      more information on the multiclass case and training procedure see
|      section 8 of [1]_.
|
|  shape_fit_ : tuple of int of shape (n_dimensions_of_X,)
|      Array dimensions of training vector ``X``.
|
|  See Also
|  --------
|  SVR : Support Vector Machine for Regression implemented using libsvm.
|
|  LinearSVC : Scalable Linear Support Vector Machine for classification
|      implemented using liblinear. Check the See Also section of
|      LinearSVC for more comparison element.
|
|  References
|  ----------
|  .. [1] `LIBSVM: A Library for Support Vector Machines
|      <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>`_
|
|  .. [2] `Platt, John (1999). "Probabilistic Outputs for Support Vector
|      Machines and Comparisons to Regularized Likelihood Methods"
|      <https://citeseerx.ist.psu.edu/doc_view/pid/42e5ed832d4310ce4378c44d05570439df28a393>`_
|
|  Examples
|  --------
|  >>> import numpy as np
|  >>> from sklearn.pipeline import make_pipeline
|  >>> from sklearn.preprocessing import StandardScaler
|  >>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
|  >>> y = np.array([1, 1, 2, 2])
|  >>> from sklearn.svm import SVC
|  >>> clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
|  >>> clf.fit(X, y)
|  Pipeline(steps=[('standardscaler', StandardScaler()),
|                  ('svc', SVC(gamma='auto'))])
|
|  >>> print(clf.predict([[-0.8, -1]]))
|  [1]
|
|  Method resolution order:
|      SVC
|      sklearn.svm._base.BaseSVC
|      sklearn.base.ClassifierMixin
|      sklearn.svm._base.BaseLibSVM
|      sklearn.base.BaseEstimator
|      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
|      sklearn.utils._metadata_requests._MetadataRequester
|      builtins.object
|
|  Methods defined here:
|
|  __init__(self, *, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, to
l=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=Fa
lse, random_state=None)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  set_fit_request(self: sklearn.svm._classes.SVC, *, sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') ->
sklearn.svm._classes.SVC from sklearn.utils._metadata_requests.RequestMethod.__get__.<locals>
|      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 nam
e.
 |
 |      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.UNCHAN
GED
 |          Metadata routing for ``sample_weight`` parameter in ``fit``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_score_request(self: sklearn.svm._classes.SVC, *, sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$')
-> sklearn.svm._classes.SVC from sklearn.utils._metadata_requests.RequestMethod.__get__.<locals>
 |      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 i
s 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 nam
e.
 |
 |      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.UNCHAN
GED
 |          Metadata routing for ``sample_weight`` parameter in ``score``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  __annotations__ = {}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.svm._base.BaseSVC:
 |
```

```
|  decision_function(self, X)
|      Evaluate the decision function for the samples in X.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          The input samples.
|
|      Returns
|      -------
|      X : ndarray of shape (n_samples, n_classes * (n_classes-1) / 2)
|          Returns the decision function of the sample for each class
|          in the model.
|          If decision_function_shape='ovr', the shape is (n_samples,
|          n_classes).
|
|      Notes
|      -----
|      If decision_function_shape='ovo', the function values are proportional
|      to the distance of the samples X to the separating hyperplane. If the
|      exact distances are required, divide the function values by the norm of
|      the weight vector (``coef_``). See also `this question
|      <https://stats.stackexchange.com/questions/14876/
|      interpreting-distance-from-hyperplane-in-svm>`_ for further details.
|      If decision_function_shape='ovr', the decision function is a monotonic
|      transformation of ovo decision function.
|
|  predict(self, X)
|      Perform classification on samples in X.
|
|      For an one-class model, +1 or -1 is returned.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features) or                (n_samples_test, n_sampl
es_train)
|          For kernel="precomputed", the expected shape of X is
|          (n_samples_test, n_samples_train).
|
|      Returns
|      -------
|      y_pred : ndarray of shape (n_samples,)
|          Class labels for samples in X.
|
|  predict_log_proba(self, X)
|      Compute log probabilities of possible outcomes for samples in X.
|
|      The model need to have probability information computed at training
|      time: fit with attribute `probability` set to True.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features) or                (n_samples_test, n_samples_train)
|          For kernel="precomputed", the expected shape of X is
|          (n_samples_test, n_samples_train).
|
|      Returns
|      -------
|      T : ndarray of shape (n_samples, n_classes)
|          Returns the log-probabilities of the sample for each class in
|          the model. The columns correspond to the classes in sorted
|          order, as they appear in the attribute :term:`classes_`.
|
|      Notes
|      -----
|      The probability model is created using cross validation, so
|      the results can be slightly different than those obtained by
|      predict. Also, it will produce meaningless results on very small
|      datasets.
|
|  predict_proba(self, X)
|      Compute probabilities of possible outcomes for samples in X.
|
|      The model needs to have probability information computed at training
|      time: fit with attribute `probability` set to True.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          For kernel="precomputed", the expected shape of X is
|          (n_samples_test, n_samples_train).
|
|      Returns
|      -------
|      T : ndarray of shape (n_samples, n_classes)
|          Returns the probability of the sample for each class in
|          the model. The columns correspond to the classes in sorted
```

```
|          order, as they appear in the attribute :term:`classes_`.
|
|      Notes
|      -----
|      The probability model is created using cross validation, so
|      the results can be slightly different than those obtained by
|      predict. Also, it will produce meaningless results on very small
|      datasets.
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from sklearn.svm._base.BaseSVC:
|
|  probA_
|      Parameter learned in Platt scaling when `probability=True`.
|
|      Returns
|      -------
|      ndarray of shape  (n_classes * (n_classes - 1) / 2)
|
|  probB_
|      Parameter learned in Platt scaling when `probability=True`.
|
|      Returns
|      -------
|      ndarray of shape  (n_classes * (n_classes - 1) / 2)
|
|  ----------------------------------------------------------------------
|  Data and other attributes inherited from sklearn.svm._base.BaseSVC:
|
|  unused_param = 'nu'
|
|  ----------------------------------------------------------------------
|  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
|
|  __weakref__
|      list of weak references to the object
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.svm._base.BaseLibSVM:
|
|  fit(self, X, y, sample_weight=None)
|      Fit the SVM model according to the given training data.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)                or (n_samples, n_samples)
|          Training vectors, where `n_samples` is the number of samples
|          and `n_features` is the number of features.
|          For kernel="precomputed", the expected shape of X is
|          (n_samples, n_samples).
|
|      y : array-like of shape (n_samples,)
|          Target values (class labels in classification, real numbers in
|          regression).
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Per-sample weights. Rescale C per sample. Higher weights
|          force the classifier to put more emphasis on these points.
```

```
|
|      Returns
|      -------
|      self : object
|          Fitted estimator.
|
|      Notes
|      -----
|      If X and y are not C-ordered and contiguous arrays of np.float64 and
|      X is not a scipy.sparse.csr_matrix, X and/or y may be copied.
|
|      If X is a dense array, then the other methods will not support sparse
|      matrices as input.
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from sklearn.svm._base.BaseLibSVM:
|
|  coef_
|      Weights assigned to the features when `kernel="linear"`.
|
|      Returns
|      -------
|      ndarray of shape (n_features, n_classes)
|
|  n_support_
|      Number of support vectors for each class.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|      Helper for pickle.
|
|  __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)
|      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
```

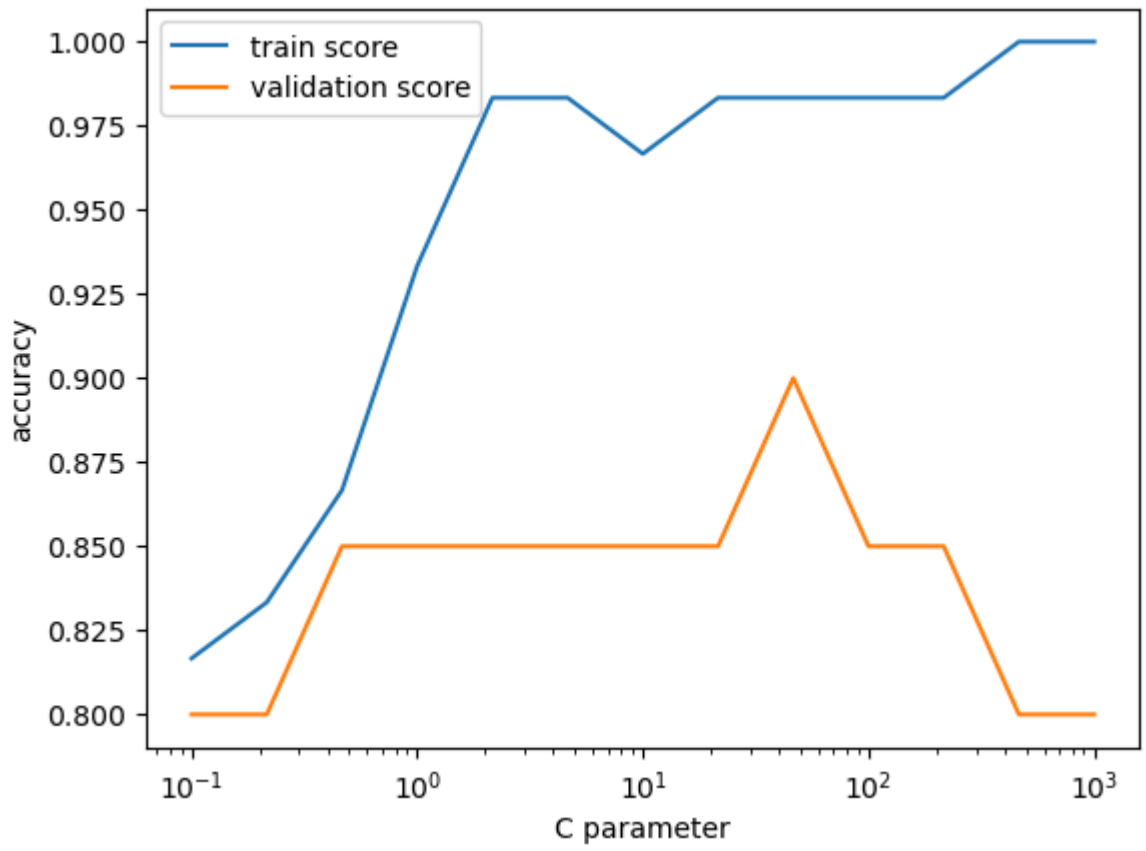
6. Tune the hyperparameters of your ML models (aka cross-validation)

In [11]:
```
Cs = np.logspace(-1,3,13)
print(Cs)
train_scores = []
validation_scores = []
models = []
for C in Cs:
    classifier = SVC(kernel='rbf',C = C, probability=True) # this is our classifier
    classifier.fit(X_train_prep,y_train) # the model is fitted to the training data

    y_train_pred = classifier.predict(X_train_prep)
    train_accuracy = accuracy_score(y_train,y_train_pred) # calculate the validation accuracy
    train_scores.append(train_accuracy)

    y_val_pred = classifier.predict(X_val_prep) # predict the validation set
    validation_accuracy = accuracy_score(y_val,y_val_pred) # calculate the validation accuracy
    validation_scores.append(validation_accuracy)

    models.append(classifier)
    print(C, train_accuracy, validation_accuracy)
```

```
[1.00000000e-01 2.15443469e-01 4.64158883e-01 1.00000000e+00
 2.15443469e+00 4.64158883e+00 1.00000000e+01 2.15443469e+01
 4.64158883e+01 1.00000000e+02 2.15443469e+02 4.64158883e+02
 1.00000000e+03]
0.1 0.8166666666666667 0.8
0.21544346900318834 0.8333333333333334 0.8
0.46415888336127786 0.8666666666666667 0.85
1.0 0.9333333333333333 0.85
2.1544346900318834 0.9833333333333333 0.85
4.6415888336127775 0.9833333333333333 0.85
10.0 0.9666666666666667 0.85
21.54434690031882 0.9833333333333333 0.85
46.41588833612777 0.9833333333333333 0.9
100.0 0.9833333333333333 0.85
215.44346900318823 0.9833333333333333 0.85
464.15888336127773 1.0 0.8
1000.0 1.0 0.8
```

The bias - variance tradeoff

In [12]:
```
plt.plot(Cs,train_scores,label='train score')
plt.plot(Cs,validation_scores,label='validation score')
plt.semilogx()
plt.legend()
plt.xlabel('C parameter')
plt.ylabel('accuracy')
plt.show()
```



- **high bias model** (aka underfitting)
  - it performs poorly on the train and validation sets
  - small C values in the example above
- **high variance model** (aka overfitting)
  - it performs very well on the training set but it performs poorly on the validation set
  - high C
- the goal of the parameter tuning is to find the balance between bias and variance
  - usually the best model is the one with the best validation score
  - C = 46 in our case

How does the best model perform on the test set?

- this score tells us how well the model generalizes to previously unseen data because the test set was not touched before
- usually it is close to the best validation score

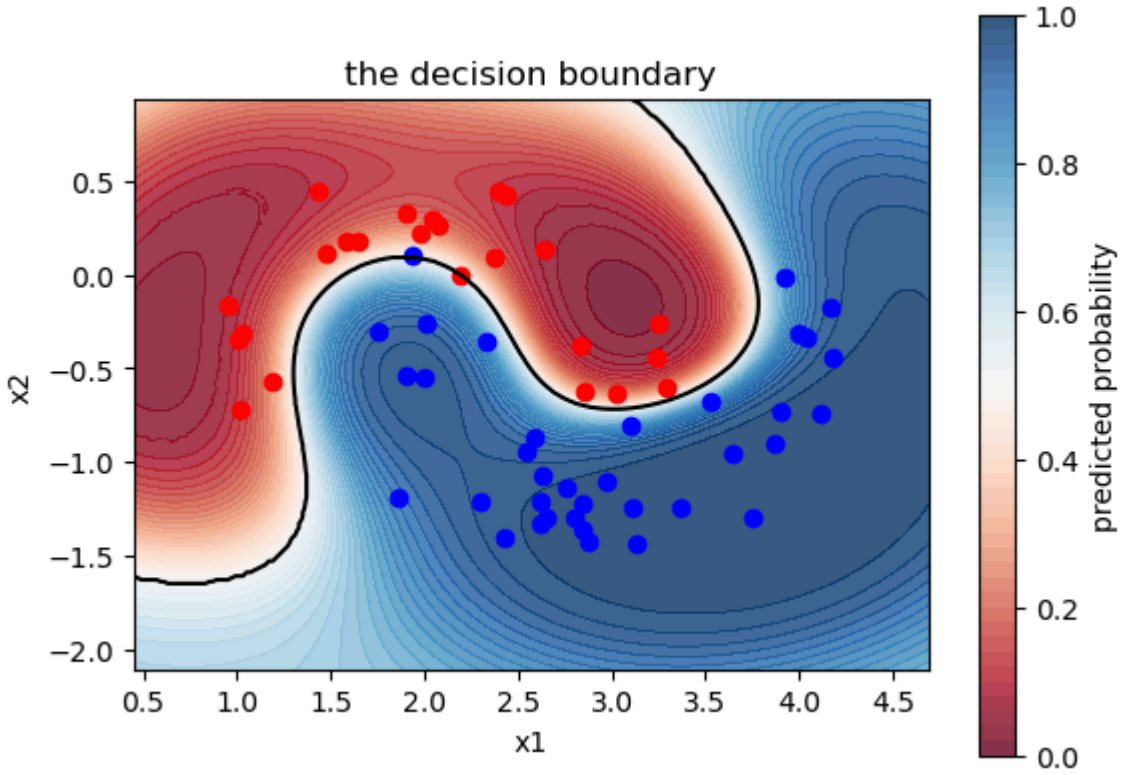
In [13]:
```
y_test_pred = models[-5].predict(X_test_prep)
print(accuracy_score(y_test, y_test_pred))
```

```
0.95
```

7. Interpret your model

- with two features, this is easy
- plot the decision boundary and probabilities

In [14]:
```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, m_max]x[y_min, y_max].

cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# use the best model with C = 46
classifier = models[-5]
# scale the data before predicting! this is very important!
Z = classifier.predict_proba(scaler.transform(np.c_[xx.ravel(), yy.ravel()]))[:, 1]

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z,vmin=0,vmax=1,levels=[0.5],colors=['k'])
plt.contourf(xx, yy, Z, cmap=cm, alpha=.8,vmin=0,vmax=1,levels=np.arange(0,1.02,0.02))
plt.colorbar(ticks=[0,0.2,0.4,0.6,0.8,1],label='predicted probability')
plt.scatter(X_train[y_train==0,0],X_train[y_train==0,1],color='r',label='class 0')
plt.scatter(X_train[y_train==1,0],X_train[y_train==1,1],color='b',label='class 1')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('the decision boundary')
plt.gca().set_aspect('equal')
plt.savefig('figures/decision_boundary.jpg',dpi=150)
plt.show()
```



Learning objectives

Hopefully you can now

- describe the main goals of the ML pipeline
- list the main steps of the ML pipeline
- explain the bias-variance trade off

Mud card

In [ ]: