Mudcard

- **For the w-vector concept in the very end of the class, I'm wondering why does 2 vector of the same direction, different length, give 2 different 2 function values? For example, w = [2,-1.8] gives cost = 0.4, and w = [1,-0.9] gives cost = 0.43**
  - Great question!
  - The direction of the w vector determines where the decision boundary is because the decision boundary is perpendicular to w
  - The length of w determines the 'confidence' of the predictions
    - the predicted probability is determined by the distance from the decision bounday
    - the longer w is, the lest distance you need to go from the decision boundary to reach a p predicted probability
  - in the logloss metric, the cost doesn't just depend on the boundary (i.e., whether points are on the right or wrong side of the boundary)
  - the cost depends on the probabilities, which translates to the distance from the decision boundary
- **why the cost function of logistic regression has two scenarios: y_true = 0 and y_true = 1. The range of y is (0,1).Why it only has two options?**
  - that's how the true target variable is defined in binary classification.
  - I either find a papaya tasty (y_true = 1) or not (y_true = 0).
  - You can define y_true differently and we did already when we covered liner regression (y_true was continuous then)
- **Is there a reason why logistic regression is called 'regression' or was it arbitrary?**
  - regression back in the mid-20th century was defined more broadly than today, it meant predicting an outcome
  - the sigmoid function is also called the logistic function
  - these two bits combined explain why it is called logistic regression but it is used for classification
- **"I'm not sure what the two features of w is in the line 'w = [2,-2] # notice that w has two components and we have two features'. what does this mean?**
  - we calculate <w,X> so we have a w component per variable in our dataset
  - read more about the dot product and its meaning here
- **I'm wonder the logistic metric is telling the probability of what feature?**
  - not a feature, the target variable
- **Once I see the function sign, I get trouble - how much of the exact math do we need to know for the in-person final or should we focus on the application or**

code? I know how linear and logistic regression is applied but it gets muddy with the equations.

- **I feel that the class time is not enough for me to truly understand every line of code, including some functions that I'm not familiar with**
- **I am confused on whether the equations were just given to us for the purpose of understanding the background or if we need to know these equations in the back of our mind. I am confused about how to translate the math equations into the code we are working on.**
  - see my post on Ed

Lecture 8: Polynomial regression and regularization

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

- Describe why regularization is important and what are the two types of regularization
- Describe how regularized linear regression works
- Describe how regularized logistic regression works

Regularization

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

- **Describe why regularization is important and what are the two types of regularization**
- Describe how regularized linear regression works
- Describe how regularized logistic regression works

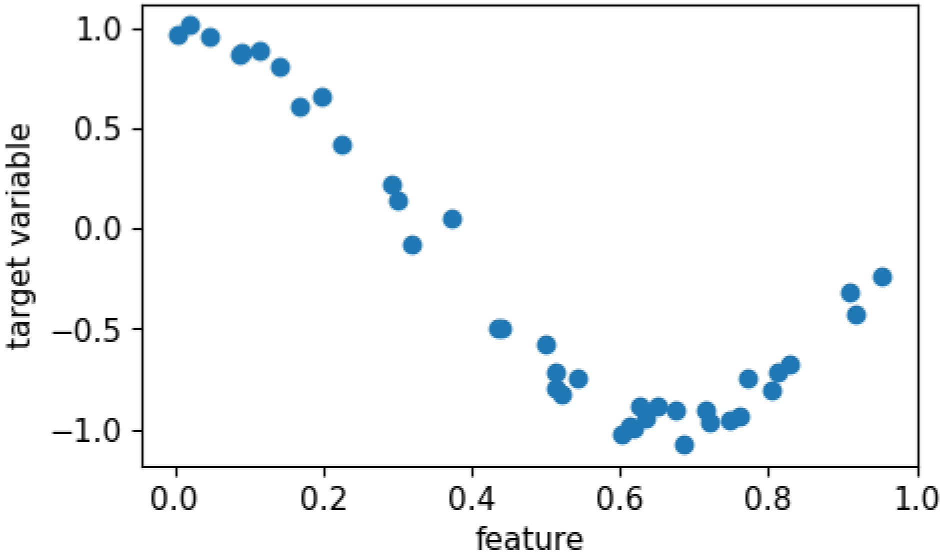
Polynomial regression

Let's work with a new example dataset

In [1]:
```
# load packages
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
import matplotlib
matplotlib.rcParams.update({'font.size': 11})

df = pd.read_csv('../data/regularization_example.csv')
X_ori = df['x0'].values.reshape(-1, 1)
```

```
y = df['y'].values
print(np.shape(X_ori))
print(np.shape(y))

# visualize the data
plt.figure(figsize=(5,3))
plt.scatter(X_ori,y)
plt.xlabel('feature')
plt.ylabel('target variable')
plt.show()
```

```
(40, 1)
(40,)
```



In [2]:
```
# lets generate more features because a linear model will obviously be insuf
pf = PolynomialFeatures(degree = 20,include_bias=False)
X = pf.fit_transform(X_ori)
print(np.shape(X))
print(pf.get_feature_names_out())
```

```
(40, 20)
['x0' 'x0^2' 'x0^3' 'x0^4' 'x0^5' 'x0^6' 'x0^7' 'x0^8' 'x0^9' 'x0^10'
 'x0^11' 'x0^12' 'x0^13' 'x0^14' 'x0^15' 'x0^16' 'x0^17' 'x0^18' 'x0^19'
 'x0^20']
```

We split data into train and validation!

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rando
print(np.shape(X_train),np.shape(y_train))
print(np.shape(X_val),np.shape(y_val))
```

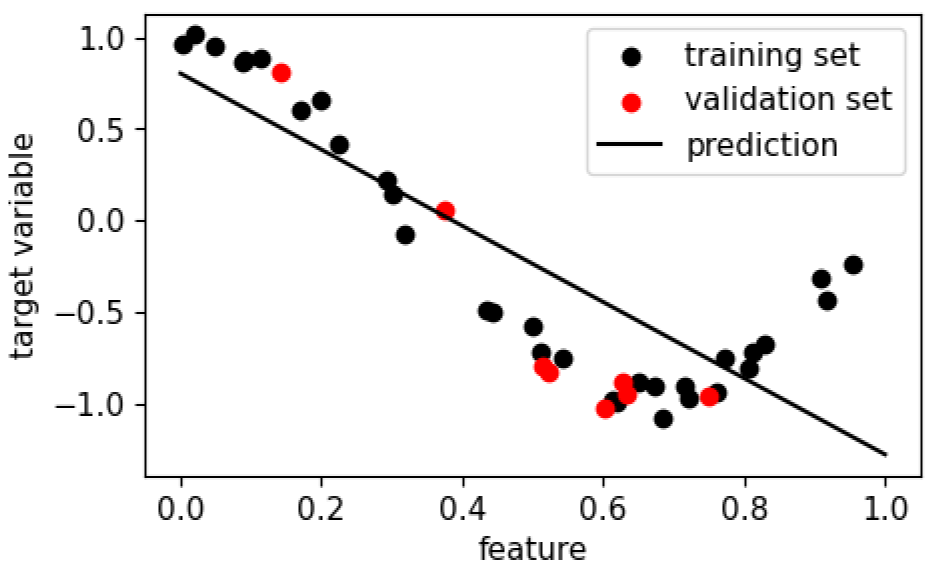
```
(32, 20) (32,)
(8, 20) (8,)
```

Let's train and validate some linear regression models

Use the first feature only

In [4]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# let's use only the first feature
linreg = LinearRegression(fit_intercept=True)
linreg.fit(X_train[:,:1], y_train)
print('intercept:',linreg.intercept_)
print('w:',linreg.coef_)

train_MSE = mean_squared_error(y_train,linreg.predict(X_train[:,:1]))
val_MSE = mean_squared_error(y_val,linreg.predict(X_val[:,:1]))
print('train MSE:',train_MSE)
print('val MSE:',val_MSE)


# let's visualuze the model
x_model = np.linspace(0,1,100)
plt.figure(figsize=(5,3))
plt.scatter(X_train[:,0],y_train,color='k',label='training set')
plt.scatter(X_val[:,0],y_val,color='r',label='validation set')
plt.plot(x_model,linreg.predict(x_model.reshape(-1,1)),color='k',label='prec
plt.xlabel('feature')
plt.ylabel('target variable')
plt.legend()
plt.show()
```

```
intercept: 0.8018842867499774
w: [-2.08151827]
train MSE: 0.13964692457239297
val MSE: 0.1714251606233729
```

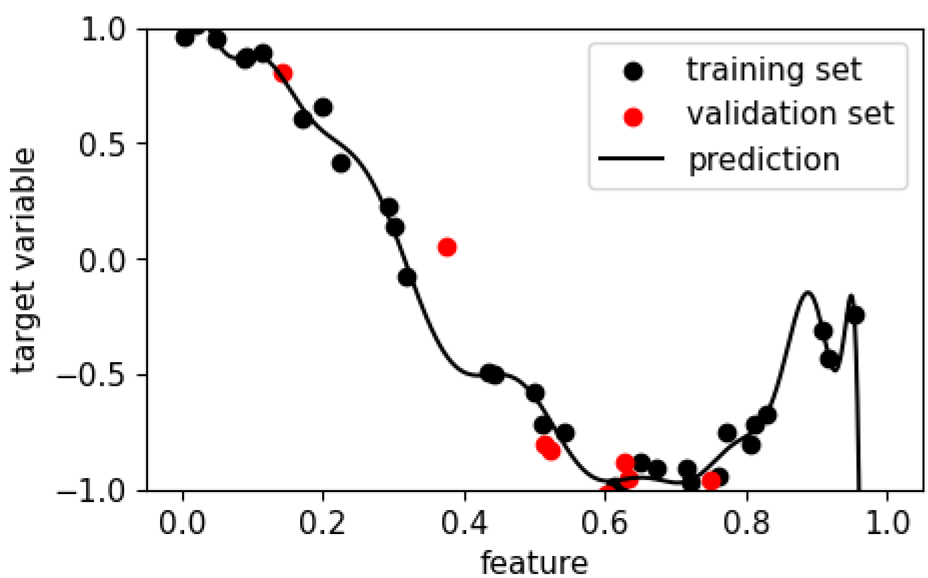


Use all features

In [5]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
# use all features
linreg = LinearRegression(fit_intercept=True)
linreg.fit(X_train, y_train)
print('intercept:',linreg.intercept_)
print('ws:',linreg.coef_)

train_MSE = mean_squared_error(y_train,linreg.predict(X_train))
val_MSE = mean_squared_error(y_val,linreg.predict(X_val))
print('train MSE:',train_MSE)
print('val MSE:',val_MSE)


# let's visualuze the model
x_model = np.linspace(0,1,1000)
plt.figure(figsize=(5,3))
plt.scatter(X_train[:,0],y_train,color='k',label='training set')
plt.scatter(X_val[:,0],y_val,color='r',label='validation set')
plt.plot(x_model,linreg.predict(pf.transform(x_model.reshape(-1,1))),color='
plt.ylim([-1,1])
plt.xlabel('feature')
plt.ylabel('target variable')
plt.legend()
plt.show()
```

```
intercept: 1.0123143561425278
ws: [-1.96093294e+01  2.36513626e+03 -1.07393992e+05  2.50225090e+06
 -3.53253437e+07  3.31465492e+08 -2.18900095e+09  1.05456431e+10
 -3.78267222e+10  1.01733036e+11 -2.03422807e+11  2.92186021e+11
 -2.71104618e+11  9.03890007e+10  1.56380017e+11 -2.94809458e+11
  2.54968343e+11 -1.30001762e+11  3.76756544e+10 -4.82188511e+09]
train MSE: 0.002222765866268214
val MSE: 0.03332565989620885
```



What to do?

- the model is visibly performs poorly when only the original feature is used

- the model performs very good on the training set but poorly on the validation set when all features are used
  - the ws are huge!

Regulazation solves this problem!

Regularization

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

- Describe why regularization is important and what are the two types of regularization
- **Describe how regularized linear regression works**
- Describe how regularized logistic regression works

Regularization to the rescue!

- let's change the cost function and add a penalty term for large ws
- **Lasso regression**: regularize using the l1 norm of w:

$L(w) = \frac{1}{n}\sum_{i=1}^{n}[(w_0 + \sum_{j=1}^{m} w_j x_{ij}- y_i)^2] + \color{red}{ \alpha \sum_{j=0}^{m}|w_j|}$

- **Ridge regression**: regularize using the square of the l2 norm of w:

$L(w) = \frac{1}{n}\sum_{i=1}^{n}[(w_0 + \sum_{j=1}^{m} w_j x_{ij}- y_i)^2] + \color{red}{\alpha \sum_{j=0}^{m} w_j^2}$

- α is the regularization parameter (positive number), it describes how much we penalize large ws

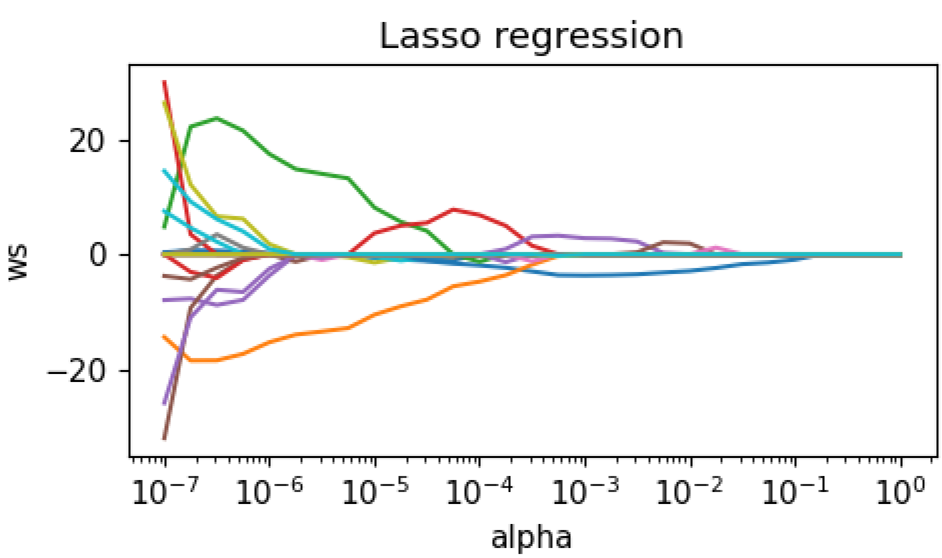
- With the cost function changed, the derivatives in gradient descent need to be updated too!

Feature selection with Lasso regularization

- Least Absolute Shrinkage and Selection Operator
- cost = MSE + α * l1 norm of w
  $L(w) = \frac{1}{n}\sum_{i=1}^{n}[(w_0 + \sum_{j=1}^{m} w_j x_{ij}- y_i)^2] + \color{red}{ \alpha \sum_{j=0}^{m}|w_j|}$
- ideal for feature selection
- as α increases, more and more feature weights are reduced to 0.

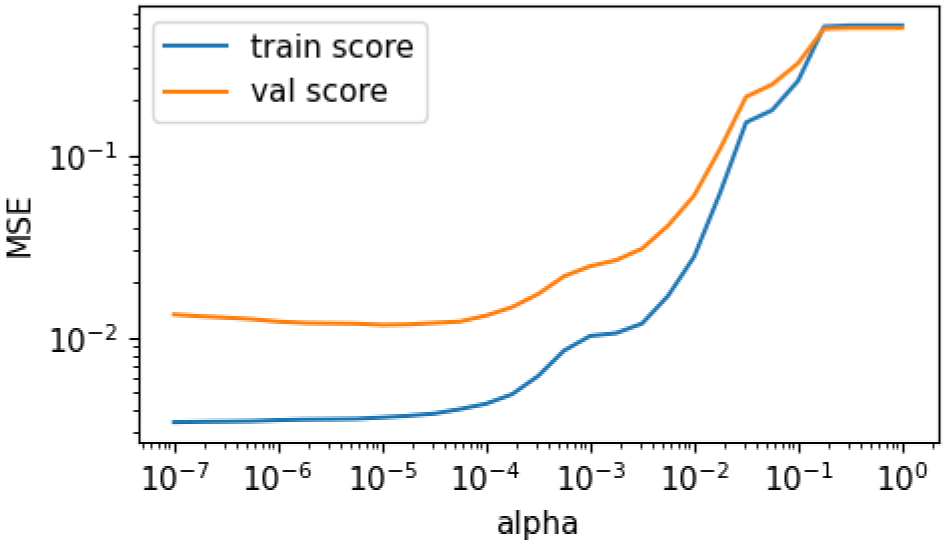
In [6]:
```
from  sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

alpha = np.logspace(-7,0,29)
ws = []
models = []
train_MSE = np.zeros(len(alpha))
val_MSE = np.zeros(len(alpha))

# do the fit
for i in range(len(alpha)):
    # load the linear regression model
    lin_reg = Lasso(alpha=alpha[i],max_iter=100000000)
    lin_reg.fit(X_train, y_train)
    ws.append(lin_reg.coef_)
    models.append(lin_reg)
    train_MSE[i] = mean_squared_error(y_train,lin_reg.predict(X_train))
    val_MSE[i] = mean_squared_error(y_val,lin_reg.predict(X_val))
```

In [7]:
```
plt.figure(figsize=(5,3))
plt.plot(alpha, ws)
plt.semilogx()
plt.xlabel('alpha')
plt.ylabel('ws')
plt.title('Lasso regression')
plt.tight_layout()
plt.savefig('../figures/lasso_coefs.png',dpi=300)
plt.show()
```



In [8]:
```
plt.figure(figsize=(5,3))
plt.plot(alpha,train_MSE,label='train score')
plt.plot(alpha,val_MSE,label='val score')
plt.semilogy()
plt.semilogx()
plt.xlabel('alpha')
plt.ylabel('MSE')
plt.legend()
```

```
plt.tight_layout()
plt.savefig('../figures/train_val_MSE_lasso.png',dpi=300)
plt.show()
```



Bias vs variance

- Bias: the model performs poorly on both the train and validation sets
  - high alpha in our example
- the model performs very well on the training set but it performs poorly on the validation set
  - low alpha in our example
  - lowering the alpha further would improve the train score but the validation score would increase
  - we don't do it because of convergence issues

The bias-variance trade off

- the curve of the validation score as a function of a hyper-parameter usually has a U shape if evaluation metric needs to be minimized, or an inverted U if the metric needs to be maximized
- choose the hyper-parameter value that gives you the best validation score

Quiz

Which alpha value gives the best validation score? Visualize the corresponding model!

In [9]:
```
# your code here
```

The bias-variance tradeoff with Ridge regularization

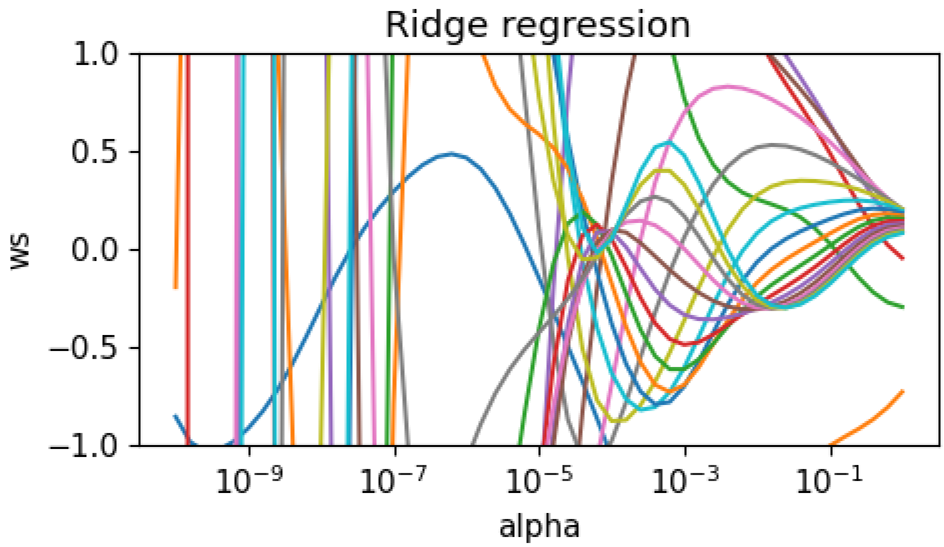
- cost = MSE + α * (l2 norm of w)^2

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}[(w_0 + \sum_{j=1}^{m} w_j x_{ij}- y_i)^2] + \color{red}{\alpha \sum_{j=0}^{m} w_j^2}$$

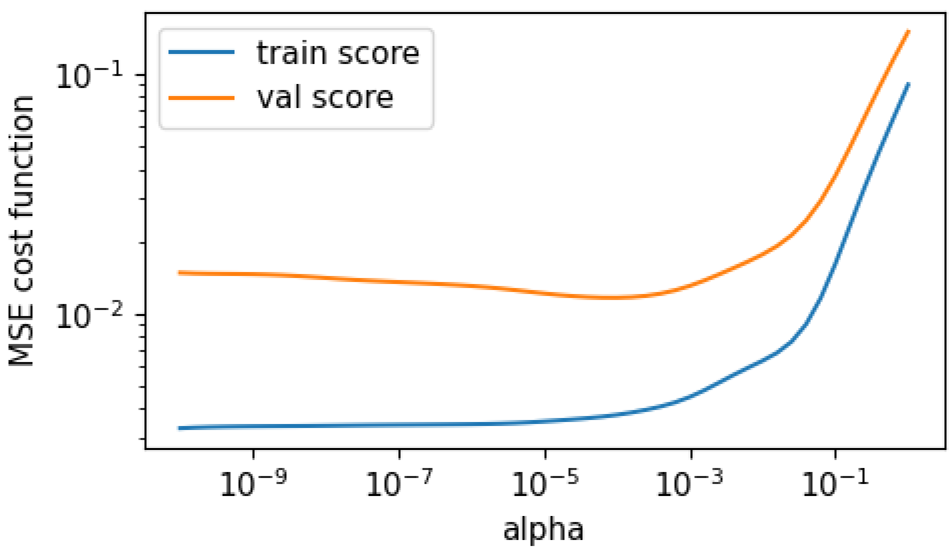
- as α approaches 0, we reproduce the linear regression weights
- small α creates high variance
- large α creates high bias

In [10]:
```
from  sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

alpha = np.logspace(-10,0,51)

# arrays to save train and test MSE scores
train_MSE = np.zeros(len(alpha))
val_MSE = np.zeros(len(alpha))

ws = []
models = []
# do the fit
for i in range(len(alpha)):
    # load the linear regression model
    lin_reg = Ridge(alpha=alpha[i])
    lin_reg.fit(X_train, y_train)
    models.append(lin_reg)
    ws.append(lin_reg.coef_)
    # train and test scores
    train_MSE[i] = mean_squared_error(y_train,lin_reg.predict(X_train))
    val_MSE[i] = mean_squared_error(y_val,lin_reg.predict(X_val))
```

In [11]:
```
plt.figure(figsize=(5,3))
plt.plot(alpha, ws)
plt.semilogx()
plt.ylim([-1e0,1e0])
plt.xlabel('alpha')
plt.ylabel('ws')
plt.title('Ridge regression')
plt.tight_layout()
plt.savefig('../figures/ridge_coefs.png',dpi=300)
plt.show()
```



In [12]:
```
plt.figure(figsize=(5,3))
plt.plot(alpha,train_MSE,label='train score')
plt.plot(alpha,val_MSE,label='val score')
plt.semilogy()
plt.semilogx()
plt.xlabel('alpha')
plt.ylabel('MSE cost function')
plt.legend()
plt.tight_layout()
plt.savefig('../figures/train_val_MSE_ridge.png',dpi=300)
plt.show()
```



Quiz

Which α gives us the best tradeoff between bias and variance?

In [13]:
```
# your code here:
```

Regularization

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

- Describe why regularization is important and what are the two types of regularization
- Describe how regularized linear regression works
- **Describe how regularized logistic regression works**

Logistic regression

- Recap: the logloss metric is the cost function

$L(w) = - \frac{1}{N}\sum_{i=1}^{n} [y_i\ln(y_i') + (1-y_i)\ln(1-y_i')]$
$L(w) = - \frac{1}{N}\sum_{i=1}^{n} [y_i\ln(\frac{1}{1+e^{-w_0 + \sum_{j=1}^{m} w_j x_{ij}}}) + (1-y_i)\ln(1-\frac{1}{1+e^{-w_0 + \sum_{j=1}^{m} w_j x_{ij}}})]$

- the logloss metric with l1 regularization

$L(w) = - \frac{1}{N}\sum_{i=1}^{n} [y_i\ln(\frac{1}{1+e^{-w_0 + \sum_{j=1}^{m} w_j x_{ij}}}) + (1-y_i)\ln(1-\frac{1}{1+e^{-w_0 + \sum_{j=1}^{m} w_j x_{ij}}}))] + \color{red}{ \alpha \sum_{j=0}^{m}|w_j|}$

- the logloss metric with l2 regularization

$L(w) = - \frac{1}{N}\sum_{i=1}^{n} [y_i\ln(\frac{1}{1+e^{-w_0 + \sum_{j=1}^{m} w_j x_{ij}}}) + (1-y_i)\ln(1-\frac{1}{1+e^{-w_0 + \sum_{j=1}^{m} w_j x_{ij}}}))] + \color{red}{\alpha \sum_{j=0}^{m} w_j^2}$

Logistic regression in sklearn

In [14]:
```
from sklearn.linear_model import LogisticRegression

log_reg_l1 = LogisticRegression(penalty='l1', C = 1/alpha) # C is the invers
log_reg_l2 = LogisticRegression(penalty='l2', C = 1/alpha)
# fit, predict, predict_proba are available
# log_reg.coef_ returns the w values
```

In [15]:
```
help(LogisticRegression)
```

```
Help on class LogisticRegression in module sklearn.linear_model._logistic:

class LogisticRegression(sklearn.linear_model._base.LinearClassifierMixin, s
klearn.linear_model._base.SparseCoefMixin, sklearn.base.BaseEstimator)
 |  LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_i
ntercept=True, intercept_scaling=1, class_weight=None, random_state=None, so
lver='lbfgs', max_iter=100, multi_class='deprecated', verbose=0, warm_start=
False, n_jobs=None, l1_ratio=None)
 |
 |  Logistic Regression (aka logit, MaxEnt) classifier.
 |
 |  This class implements regularized logistic regression using the
 |  'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **N
ote
 |  that regularization is applied by default**. It can handle both dense
 |  and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit
 |  floats for optimal performance; any other input format will be converted
 |  (and copied).
 |
 |  The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularizati
on
 |  with primal formulation, or no regularization. The 'liblinear' solver
 |  supports both L1 and L2 regularization, with a dual formulation only for
 |  the L2 penalty. The Elastic-Net regularization is only supported by the
 |  'saga' solver.
 |
 |  For :term:`multiclass` problems, only 'newton-cg', 'sag', 'saga' and 'lb
fgs'
 |  handle multinomial loss. 'liblinear' and 'newton-cholesky' only handle b
inary
 |  classification but can be extended to handle multiclass by using
 |  :class:`~sklearn.multiclass.OneVsRestClassifier`.
 |
 |  Read more in the :ref:`User Guide <logistic_regression>`.
 |
 |  Parameters
 |  ----------
 |  penalty : {'l1', 'l2', 'elasticnet', None}, default='l2'
 |      Specify the norm of the penalty:
 |
 |      - `None`: no penalty is added;
 |      - `'l2'`: add a L2 penalty term and it is the default choice;
 |      - `'l1'`: add a L1 penalty term;
 |      - `'elasticnet'`: both L1 and L2 penalty terms are added.
 |
 |      .. warning::
 |         Some penalties may not work with some solvers. See the parameter
 |         `solver` below, to know the compatibility between the penalty and
 |         solver.
 |
 |      .. versionadded:: 0.19
 |         l1 penalty with SAGA solver (allowing 'multinomial' + L1)
 |
 |  dual : bool, default=False
 |      Dual (constrained) or primal (regularized, see also
 |      :ref:`this equation <regularized-logistic-loss>`) formulation. Dual
```

```
formulation
 |      is only implemented for l2 penalty with liblinear solver. Prefer dua
l=False when
 |      n_samples > n_features.
 |
 |  tol : float, default=1e-4
 |      Tolerance for stopping criteria.
 |
 |  C : float, default=1.0
 |      Inverse of regularization strength; must be a positive float.
 |      Like in support vector machines, smaller values specify stronger
 |      regularization.
 |
 |  fit_intercept : bool, default=True
 |      Specifies if a constant (a.k.a. bias or intercept) should be
 |      added to the decision function.
 |
 |  intercept_scaling : float, default=1
 |      Useful only when the solver 'liblinear' is used
 |      and self.fit_intercept is set to True. In this case, x becomes
 |      [x, self.intercept_scaling],
 |      i.e. a "synthetic" feature with constant value equal to
 |      intercept_scaling is appended to the instance vector.
 |      The intercept becomes ``intercept_scaling * synthetic_feature_weight
``.
 |
 |      Note! the synthetic feature weight is subject to l1/l2 regularizatio
n
 |      as all other features.
 |      To lessen the effect of regularization on synthetic feature weight
 |      (and therefore on the intercept) intercept_scaling has to be increas
ed.
 |
 |  class_weight : dict or 'balanced', default=None
 |      Weights associated with classes in the form ``{class_label: weight}`
`.
 |      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 dat
a
 |      as ``n_samples / (n_classes * np.bincount(y))``.
 |
 |      Note that these weights will be multiplied with sample_weight (passe
d
 |      through the fit method) if sample_weight is specified.
 |
 |      .. versionadded:: 0.17
 |         *class_weight='balanced'*
 |
 |  random_state : int, RandomState instance, default=None
 |      Used when ``solver`` == 'sag', 'saga' or 'liblinear' to shuffle the
 |      data. See :term:`Glossary <random_state>` for details.
 |
 |  solver : {'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag',
'saga'},             default='lbfgs'
```

```
 |
 |      Algorithm to use in the optimization problem. Default is 'lbfgs'.
 |      To choose a solver, you might want to consider the following aspect
s:
 |
 |      - For small datasets, 'liblinear' is a good choice, whereas 'sag'
 |        and 'saga' are faster for large ones;
 |      - For :term:`multiclass` problems, all solvers except 'liblinear' mi
nimize the
 |        full multinomial loss;
 |      - 'liblinear' can only handle binary classification by default. To a
pply a
 |        one-versus-rest scheme for the multiclass setting one can wrap it
with the
 |        :class:`~sklearn.multiclass.OneVsRestClassifier`.
 |      - 'newton-cholesky' is a good choice for
 |        `n_samples` >> `n_features * n_classes`, especially with one-hot e
ncoded
 |        categorical features with rare categories. Be aware that the memor
y usage
 |        of this solver has a quadratic dependency on `n_features * n_class
es`
 |        because it explicitly computes the full Hessian matrix.
 |
 |      .. warning::
 |         The choice of the algorithm depends on the penalty chosen and on
 |         (multinomial) multiclass support:
 |
 |         ================= ============================== ================
======
 |         solver            penalty                        multinomial mult
iclass
 |         ================= ============================== ================
======
 |         'lbfgs'           'l2', None                     yes
 |         'liblinear'       'l1', 'l2'                     no
 |         'newton-cg'       'l2', None                     yes
 |         'newton-cholesky' 'l2', None                     no
 |         'sag'             'l2', None                     yes
 |         'saga'            'elasticnet', 'l1', 'l2', None yes
 |         ================= ============================== ================
======
 |
 |      .. note::
 |         'sag' and 'saga' fast convergence is only guaranteed on features
 |         with approximately the same scale. You can preprocess the data wi
th
 |         a scaler from :mod:`sklearn.preprocessing`.
 |
 |      .. seealso::
 |         Refer to the :ref:`User Guide <Logistic_regression>` for more
 |         information regarding :class:`LogisticRegression` and more specif
ically the
 |         :ref:`Table <logistic_regression_solvers>`
 |         summarizing solver/penalty supports.
 |
```

```
 |      .. versionadded:: 0.17
 |         Stochastic Average Gradient descent solver.
 |      .. versionadded:: 0.19
 |         SAGA solver.
 |      .. versionchanged:: 0.22
 |          The default solver changed from 'liblinear' to 'lbfgs' in 0.22.
 |      .. versionadded:: 1.2
 |         newton-cholesky solver.
 |
 |  max_iter : int, default=100
 |      Maximum number of iterations taken for the solvers to converge.
 |
 |  multi_class : {'auto', 'ovr', 'multinomial'}, default='auto'
 |      If the option chosen is 'ovr', then a binary problem is fit for each
 |      label. For 'multinomial' the loss minimised is the multinomial loss
fit
 |      across the entire probability distribution, *even when the data is
 |      binary*. 'multinomial' is unavailable when solver='liblinear'.
 |      'auto' selects 'ovr' if the data is binary, or if solver='liblinea
r',
 |      and otherwise selects 'multinomial'.
 |
 |      .. versionadded:: 0.18
 |         Stochastic Average Gradient descent solver for 'multinomial' cas
e.
 |      .. versionchanged:: 0.22
 |          Default changed from 'ovr' to 'auto' in 0.22.
 |      .. deprecated:: 1.5
 |         ``multi_class`` was deprecated in version 1.5 and will be removed
in 1.7.
 |         From then on, the recommended 'multinomial' will always be used f
or
 |         `n_classes >= 3`.
 |         Solvers that do not support 'multinomial' will raise an error.
 |         Use `sklearn.multiclass.OneVsRestClassifier(LogisticRegression())
` if you
 |         still want to use OvR.
 |
 |  verbose : int, default=0
 |      For the liblinear and lbfgs solvers set verbose to any positive
 |      number for verbosity.
 |
 |  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.
 |      Useless for liblinear solver. See :term:`the Glossary <warm_start>`.
 |
 |      .. versionadded:: 0.17
 |         *warm_start* to support *lbfgs*, *newton-cg*, *sag*, *saga* solve
rs.
 |
 |  n_jobs : int, default=None
 |      Number of CPU cores used when parallelizing over classes if
 |      multi_class='ovr'". This parameter is ignored when the ``solver`` is
 |      set to 'liblinear' regardless of whether 'multi_class' is specified
or
```

```
 |      not. ``None`` means 1 unless in a :obj:`joblib.parallel_backend`
 |      context. ``-1`` means using all processors.
 |      See :term:`Glossary <n_jobs>` for more details.
 |
 |  l1_ratio : float, default=None
 |      The Elastic-Net mixing parameter, with ``0 <= l1_ratio <= 1``. Only
 |      used if ``penalty='elasticnet'``. Setting ``l1_ratio=0`` is equivale
nt
 |      to using ``penalty='l2'``, while setting ``l1_ratio=1`` is equivalen
t
 |      to using ``penalty='l1'``. For ``0 < l1_ratio <1``, the penalty is a
 |      combination of L1 and L2.
 |
 |  Attributes
 |  ----------
 |
 |  classes_ : ndarray of shape (n_classes, )
 |      A list of class labels known to the classifier.
 |
 |  coef_ : ndarray of shape (1, n_features) or (n_classes, n_features)
 |      Coefficient of the features in the decision function.
 |
 |      `coef_` is of shape (1, n_features) when the given problem is binar
y.
 |      In particular, when `multi_class='multinomial'`, `coef_` corresponds
 |      to outcome 1 (True) and `-coef_` corresponds to outcome 0 (False).
 |
 |  intercept_ : ndarray of shape (1,) or (n_classes,)
 |      Intercept (a.k.a. bias) added to the decision function.
 |
 |      If `fit_intercept` is set to False, the intercept is set to zero.
 |      `intercept_` is of shape (1,) when the given problem is binary.
 |      In particular, when `multi_class='multinomial'`, `intercept_`
 |      corresponds to outcome 1 (True) and `-intercept_` corresponds to
 |      outcome 0 (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_iter_ : ndarray of shape (n_classes,) or (1, )
 |      Actual number of iterations for all classes. If binary or multinomia
l,
 |      it returns only 1 element. For liblinear solver, only the maximum
 |      number of iteration across all classes is given.
 |
 |      .. versionchanged:: 0.20
 |
 |          In SciPy <= 1.0.0 the number of lbfgs iterations may exceed
```

```
 |          ``max_iter``. ``n_iter_`` will now report at most ``max_iter``.
 |
 |  See Also
 |  --------
 |  SGDClassifier : Incrementally trained logistic regression (when given
 |      the parameter ``loss="log_loss"``).
 |  LogisticRegressionCV : Logistic regression with built-in cross validatio
n.
 |
 |  Notes
 |  -----
 |  The underlying C implementation uses a random number generator to
 |  select features when fitting the model. It is thus not uncommon,
 |  to have slightly different results for the same input data. If
 |  that happens, try with a smaller tol parameter.
 |
 |  Predict output may not match that of standalone liblinear in certain
 |  cases. See :ref:`differences from liblinear <liblinear_differences>`
 |  in the narrative documentation.
 |
 |  References
 |  ----------
 |
 |  L-BFGS-B -- Software for Large-scale Bound-constrained Optimization
 |      Ciyou Zhu, Richard Byrd, Jorge Nocedal and Jose Luis Morales.
 |      http://users.iems.northwestern.edu/~nocedal/lbfgsb.html
 |
 |  LIBLINEAR -- A Library for Large Linear Classification
 |      https://www.csie.ntu.edu.tw/~cjlin/liblinear/
 |
 |  SAG -- Mark Schmidt, Nicolas Le Roux, and Francis Bach
 |      Minimizing Finite Sums with the Stochastic Average Gradient
 |      https://hal.inria.fr/hal-00860051/document
 |
 |  SAGA -- Defazio, A., Bach F. & Lacoste-Julien S. (2014).
 |          :arxiv:`"SAGA: A Fast Incremental Gradient Method With Support
 |          for Non-Strongly Convex Composite Objectives" <1407.0202>`
 |
 |  Hsiang-Fu Yu, Fang-Lan Huang, Chih-Jen Lin (2011). Dual coordinate desce
nt
 |      methods for logistic regression and maximum entropy models.
 |      Machine Learning 85(1-2):41-75.
 |      https://www.csie.ntu.edu.tw/~cjlin/papers/maxent_dual.pdf
 |
 |  Examples
 |  --------
 |  >>> from sklearn.datasets import load_iris
 |  >>> from sklearn.linear_model import LogisticRegression
 |  >>> X, y = load_iris(return_X_y=True)
 |  >>> clf = LogisticRegression(random_state=0).fit(X, y)
 |  >>> clf.predict(X[:2, :])
 |  array([0, 0])
 |  >>> clf.predict_proba(X[:2, :])
 |  array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
 |         [9.7...e-01, 2.8...e-02, ...e-08]])
 |  >>> clf.score(X, y)
```

```
 |  0.97...
 |  
 |  For a comaprison of the LogisticRegression with other classifiers see:
 |  :ref:`sphx_glr_auto_examples_classification_plot_classification_probabil
ity.py`.
 |  
 |  Method resolution order:
 |      LogisticRegression
 |      sklearn.linear_model._base.LinearClassifierMixin
 |      sklearn.base.ClassifierMixin
 |      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='l2', *, dual=False, tol=0.0001, C=1.0, fit_inter
cept=True, intercept_scaling=1, class_weight=None, random_state=None, solver
='lbfgs', max_iter=100, multi_class='deprecated', verbose=0, warm_start=Fals
e, n_jobs=None, l1_ratio=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  __sklearn_tags__(self)
 |  
 |  fit(self, X, y, sample_weight=None)
 |      Fit the model according to the given training data.
 |      
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          Training vector, where `n_samples` is the number of samples and
 |          `n_features` is the number of features.
 |      
 |      y : array-like of shape (n_samples,)
 |          Target vector relative to X.
 |      
 |      sample_weight : array-like of shape (n_samples,) default=None
 |          Array of weights that are assigned to individual samples.
 |          If not provided, then each sample is given unit weight.
 |      
 |          .. versionadded:: 0.17
 |             *sample_weight* support to LogisticRegression.
 |      
 |      Returns
 |      -------
 |      self
 |          Fitted estimator.
 |      
 |      Notes
 |      -----
 |      The SAGA solver supports both float64 and float32 bit arrays.
 |  
 |  predict_log_proba(self, X)
 |      Predict logarithm of probability estimates.
```

```
 |
 |      The returned estimates for all classes are ordered by the
 |      label of classes.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Vector to be scored, where `n_samples` is the number of samples
and
 |          `n_features` is the number of features.
 |
 |      Returns
 |      -------
 |      T : array-like of shape (n_samples, n_classes)
 |          Returns the log-probability of the sample for each class in the
 |          model, where classes are ordered as they are in ``self.classes_`
`.
 |
 |  predict_proba(self, X)
 |      Probability estimates.
 |
 |      The returned estimates for all classes are ordered by the
 |      label of classes.
 |
 |      For a multi_class problem, if multi_class is set to be "multinomial"
 |      the softmax function is used to find the predicted probability of
 |      each class.
 |      Else use a one-vs-rest approach, i.e. calculate the probability
 |      of each class assuming it to be positive using the logistic function
 |      and normalize these values across all the classes.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Vector to be scored, where `n_samples` is the number of samples
and
 |          `n_features` is the number of features.
 |
 |      Returns
 |      -------
 |      T : array-like of shape (n_samples, n_classes)
 |          Returns the probability of the sample for each class in the mode
l,
 |          where classes are ordered as they are in ``self.classes_``.
 |
 |  set_fit_request(self: sklearn.linear_model._logistic.LogisticRegression,
*, sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.lin
ear_model._logistic.LogisticRegression from sklearn.utils._metadata_request
s.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 provide
d. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``fit``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise 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 t
he
 |      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,                    defau
lt=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``fit``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_score_request(self: sklearn.linear_model._logistic.LogisticRegressio
n, *, sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.
linear_model._logistic.LogisticRegression from sklearn.utils._metadata_reque
sts.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 provid
ed. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``score``.
 |
```

```
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise 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 t
he
 |      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,                     defau
lt=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:
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  ----------------------------------------------------------------------
 |  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 t
he
 |          binary case, confidence score for `self.classes_[1]` where >0 me
ans
 |          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
 |
 |  __weakref__
 |      list of weak references to the object
 |
 |  ----------------------------------------------------------------------
 |  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 t
his
 |      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)
 |      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._MetadataRequest
er:
 |
 |  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` encap
sulating
 |          routing information.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._metadata_requests._MetadataR
equester:
 |
 |  __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 metho
d
 |      does not explicitly accept a metadata through its arguments or if th
e
 |      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
```

Mudcard

In [ ]: