Mudcard

- **How could we tell that the learning rate is too large?**
  - let's go back to the previous lecture notes and increase the learning rate!
- **What's so special about the sigmoid function? Are there any other functions that are used to transform the output of a linear regression into a probability?**
  - it's derivative is quick to calculate :)
  - yes, there are other functions that might work for example tanh has a similar although not quite the same shape but its derivative is numerically costly to calculate
- **I have a question about gradient descent; How do we know when we should stop repeating until it converge?**
- **How to determine what number should we set for learning rate and iteration for gradient descent?**
  - you need to plot the cost function history to figure that out
  - generally speaking, the smaller the learning rate, the larger the iteration should be
- **can i get further explanation why gradient_descent is worse then sklearn other than just illustration!**
  - sklearn implements the algorithms in a version of c called cython which is faster than my naive python implementation
  - sklearn's version of gradient descent is also much more sophisticated than my naive algorithm
  - read up on stochastic gradient descent and adaptive learning rates
- **For R^2, one we can interpret it as the percentage of the variation that can be interpret by our model. In this case, what does -1 represent?**
  - in statistics, R2 is often referred to as the percentage of variance explained by the model
  - that holds more or less true in ML too if the dataset is IID
  - in time series data and if you deal with groups, R2 can be negative in ML
  - in statistics, the R2 is usually calculated using the whole dataset
  - in ML, we have a test set which can have different properties than the training set
- **Is there an easy way to "know" the shape of the loss function? I was a bit confused by the discussion at the end the lecture on local vs. absolute minima**
  - nope, it's difficult to visualize a loss function for more than two intrinsic parameters
- **So is it a good idea to try out¬† both logistic and linear regression models?**
  - NO!
  - use linear regression for regression problems (continuous target variable)
  - use logistic regression for classification problems (categorical target variable)
- **Sigmoid functions vs. linear regression model, are they the same and why do they have different names**
  - they are not the same so they have different names :)

- **can we use sigmoid transformation on standardscalared data?**
  - you don't need to do sigmoid transformation, sklearn will do it for you
- **Why was the grid implementation not able to find 1,32 as the minimum?**
  - print out the grid
  - the grid only contained 0 and 2 and no values in between, the grid is discreet
- **Can you only use logistic regression for classification, or can you use it for regression problems as well?**
  - only for classification
- **Mathematically, how is stochastic gradient descent different from gradient descent that allows us to find the global minimum?**
  - read more here
- **is there a case where the brute force method would be preferred?**
  - I cannot think of any
- **To what extent do we need to understand the mathematical models behind ML algorithms (gradient descent, logloss, etc.) for this class?**
  - you learn for yourself, not for me
  - the final exam is open-book so I obviously won't ask questions you can simply copy-paste from the notebooks if that's what you mean
  - you'll get questions on these ML concepts during the interview so getting a job without understanding them is difficult
- **What was the purpose of illustrating the gradient descent method in class, when sklearn is faster? Does the latter work more or less using principles similar to the gradient descent method?**
  - you need to understand what sklearn does but it's not easy to dissect sklearn's source code
- **What happens if a gradient descent there are two 'valleys' or bowls where one is lower than the other; therefore, the lower one would indicate the best set of parameters. How do we¬† make sure that our gradient descent reaches the lower 'valley'/bowl?**
  - that's the problem stochastic gradient descent solves
  - check the link above
- **Are the sigmoid functions for logistic regressions made with a cutoff that we choose? For example could we make it that probability >0.75 is Yes else No? or is the threshold usually 0.5. Can we produce confidence intervals for these measurements?**
  - the threshold is usually 50% but as you saw during the lectures on the evalution metric, sometimes p_crit is tuned

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

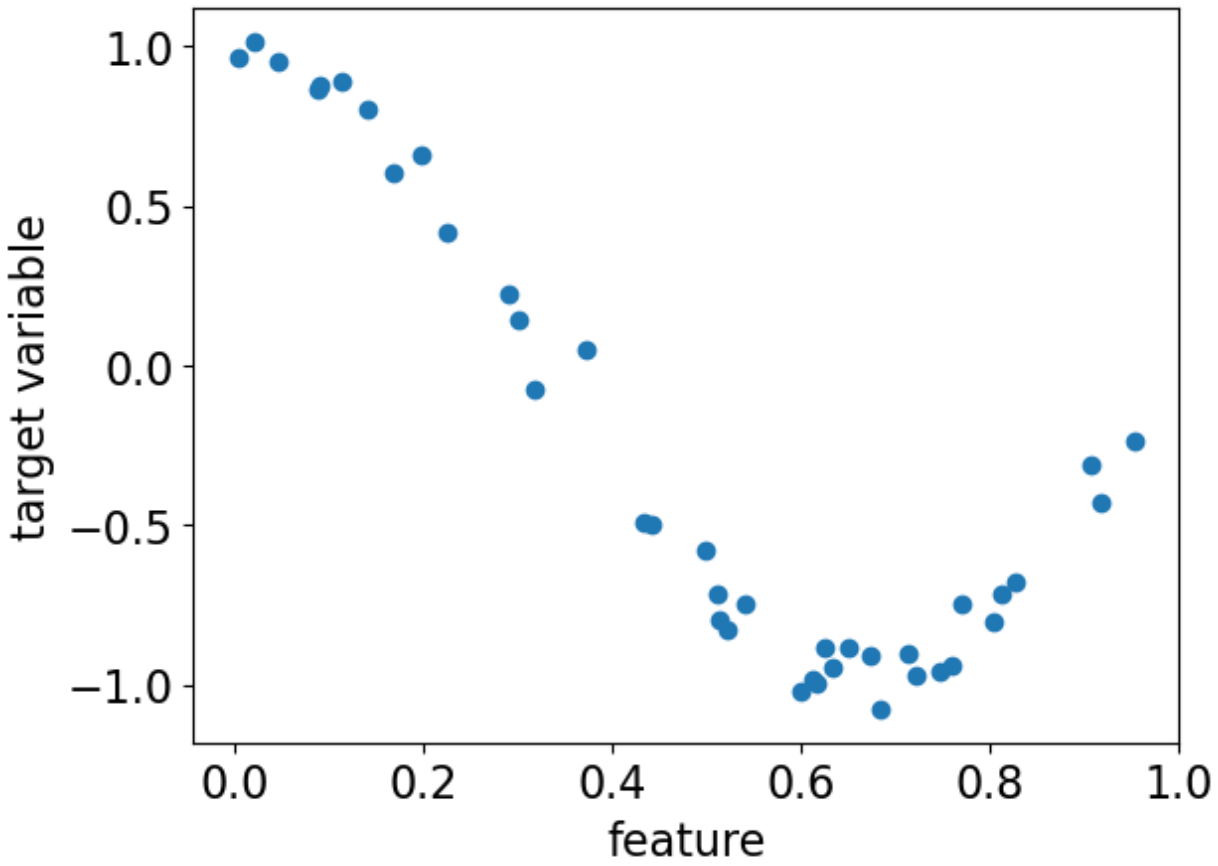
By the end of this lecture, you will be able to

- **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

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': 16})

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.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 insuffic
pf = PolynomialFeatures(degree = 20,include_bias=False)
X = pf.fit_transform(X_ori)
print(np.shape(X))
print(pf.get_feature_names())
```

```
(40, 20)
['x0', 'x0^2', 'x0^3', 'x0^4', 'x0^5', 'x0^6', 'x0^7', 'x0^8', 'x0^9', 'x0^1
0', 'x0^11', 'x0^12', 'x0^13', 'x0^14', 'x0^15', 'x0^16', 'x0^17', 'x0^18', 'x
0^19', 'x0^20']
```

```
/Users/azsom/opt/anaconda3/envs/data1030/lib/python3.10/site-packages/sklearn/
utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecat
ed; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please
use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

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, random_s
print(np.shape(X_train),np.shape(y_train))
print(np.shape(X_val),np.shape(y_val))
```

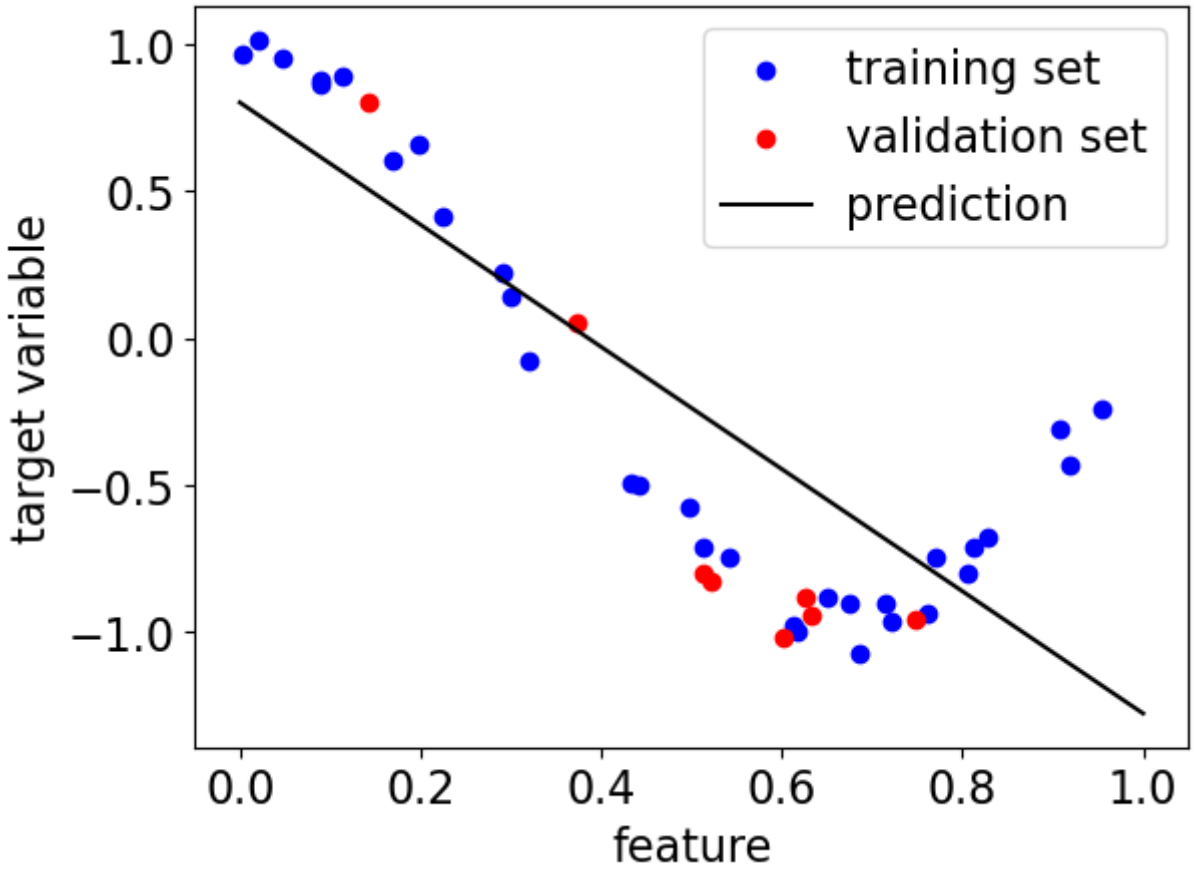
```
(32, 20) (32,)
(8, 20) (8,)
```

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.scatter(X_train[:,0],y_train,color='b',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='predict

plt.xlabel('feature')
plt.ylabel('target variable')
plt.legend()
plt.show()
```

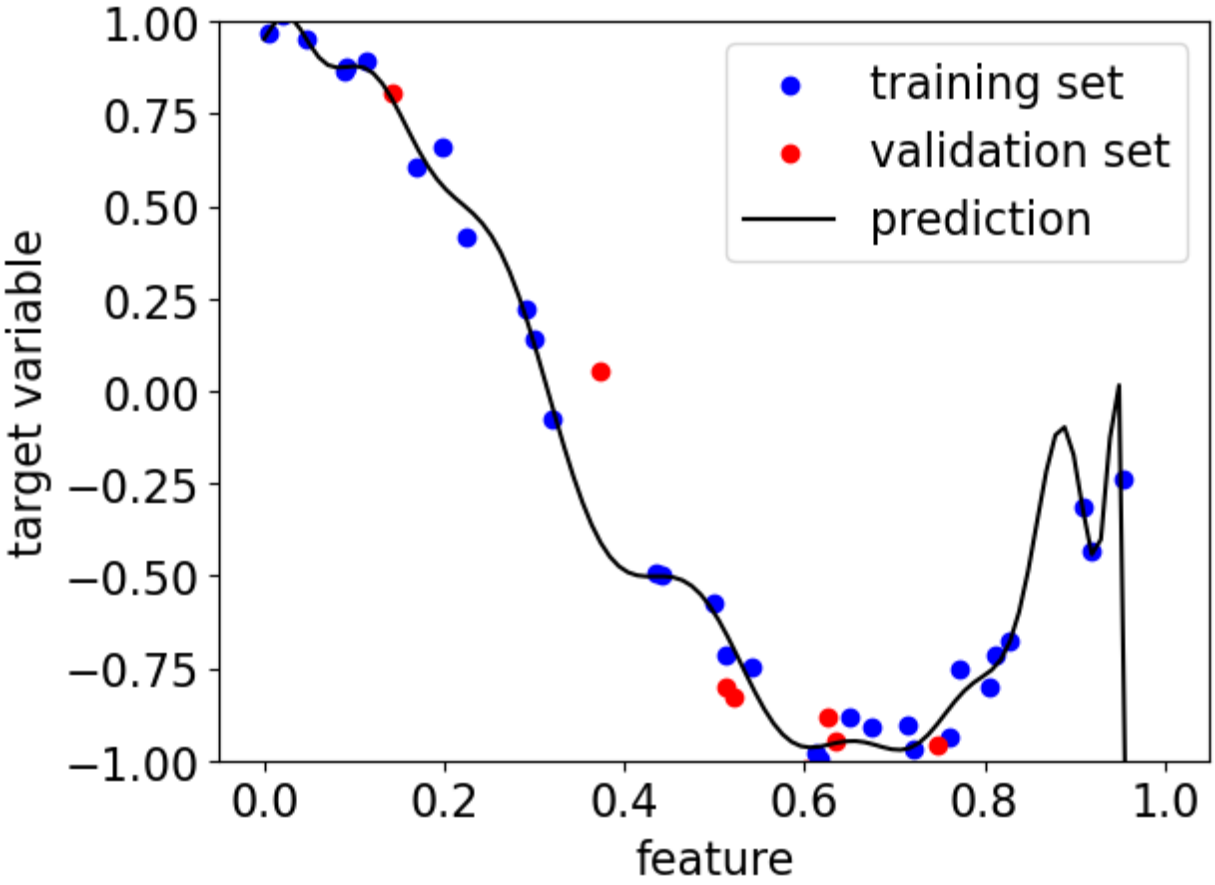
```
intercept: 0.8018842867499771
w: [-2.08151827]
train MSE: 0.13964692457239292
val MSE: 0.17142516062337293
```



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(0,1,100)
plt.scatter(X_train[:,0],y_train,color='b',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='k',
plt.ylim([-1,1])
plt.xlabel('feature')
plt.ylabel('target variable')
plt.legend()
plt.show()
```

```
intercept: 0.9520757590563689
ws: [ 2.94556467e+00  1.78575224e+02 -1.07852987e+04  7.71588160e+04
  3.57083482e+06 -9.66895422e+07  1.20126203e+09 -9.39908662e+09
  5.13303355e+10 -2.05802411e+11  6.23129909e+11 -1.44706005e+12
  2.59415710e+12 -3.58595231e+12  3.78788495e+12 -3.00091345e+12
  1.72536419e+12 -6.79474340e+11  1.63872612e+11 -1.82456642e+10]
train MSE: 0.002223656221532701
val MSE: 0.032878428680030686
```



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:

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

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

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

- 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

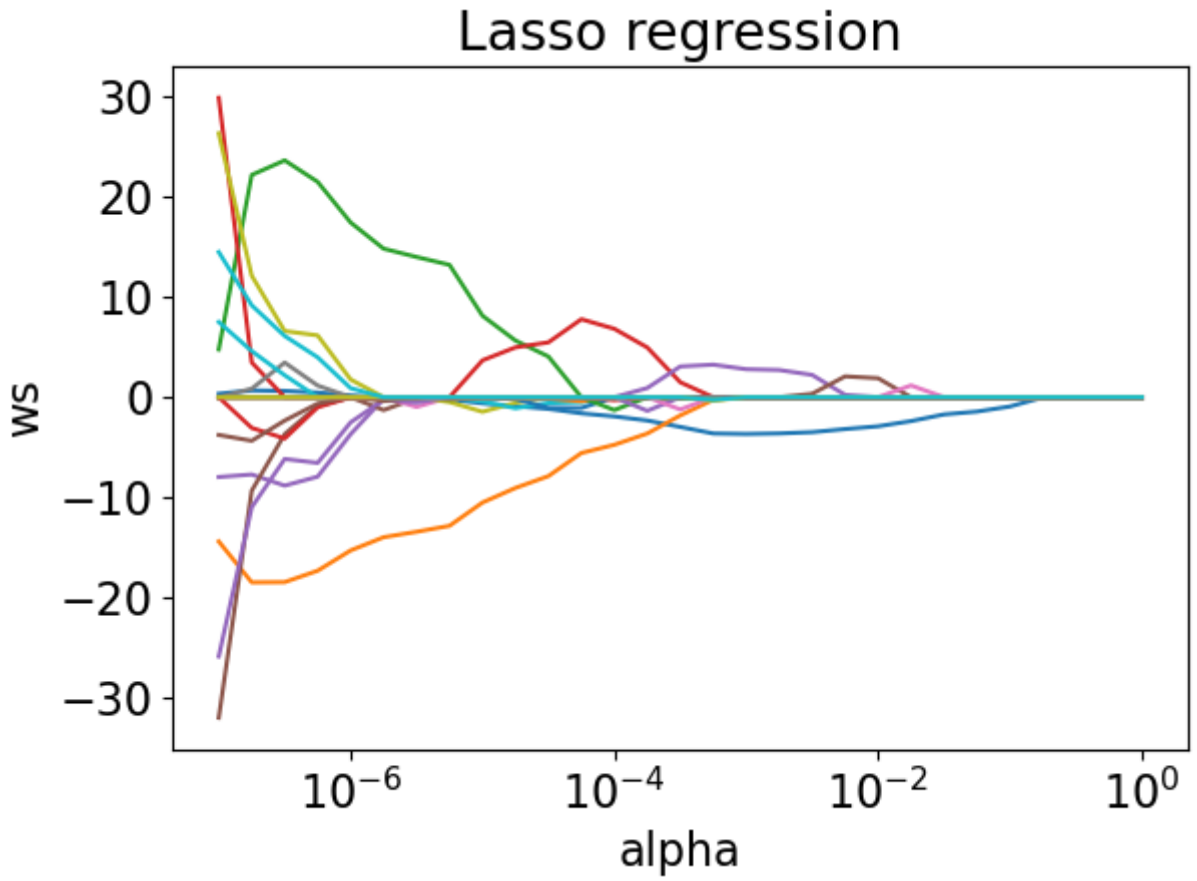
$$L(w) = \frac{1}{n}\sum_{i=1}^{n}[(w_0 + \sum_{j=1}^{m} w_j x_{ij} - y_i)^2] + \frac{\alpha}{m}\sum_{j=0}^{m}|w_j|$$

- ideal for feature selection - as α increases, more and more feature weights are reduced to 0.

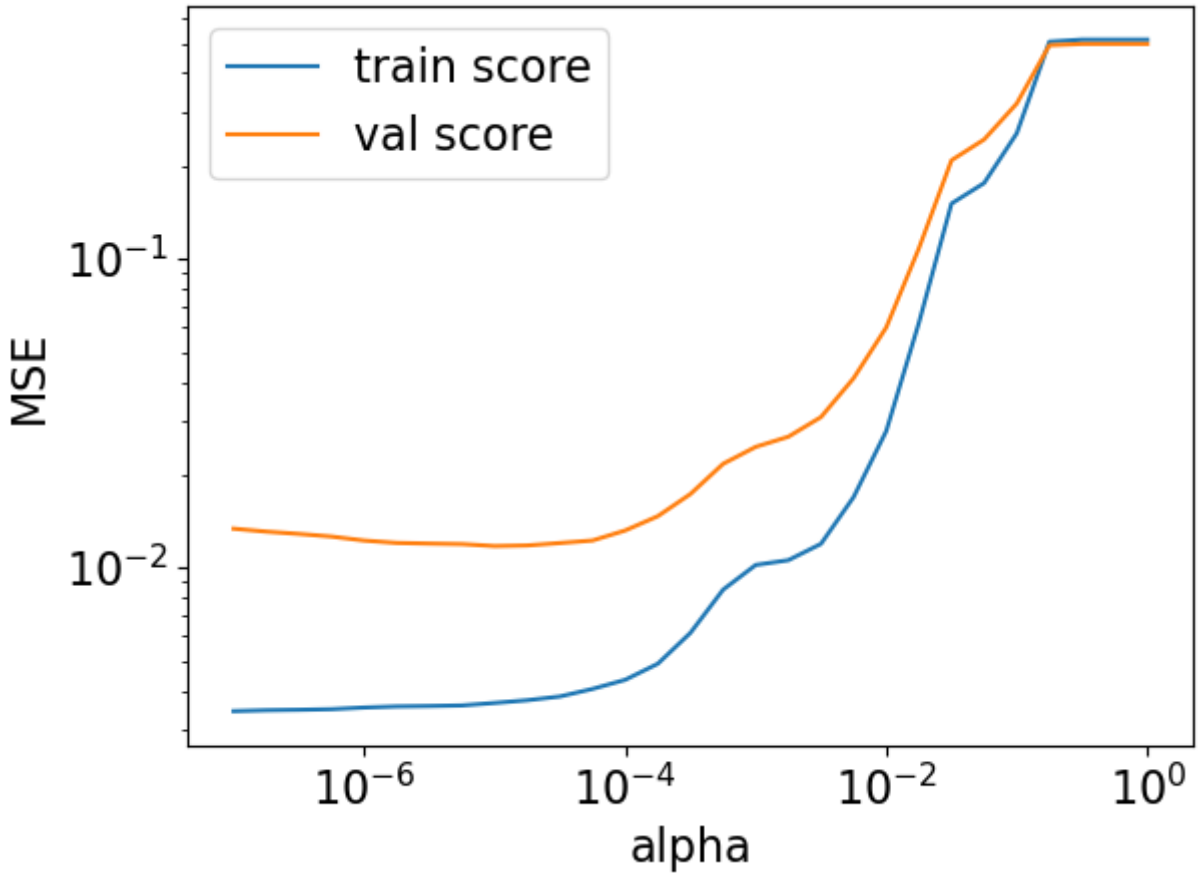
In [6]:
```
from  sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

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.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.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 that model!

In [ ]:

The bias-variance tradeoff with Ridge regularization

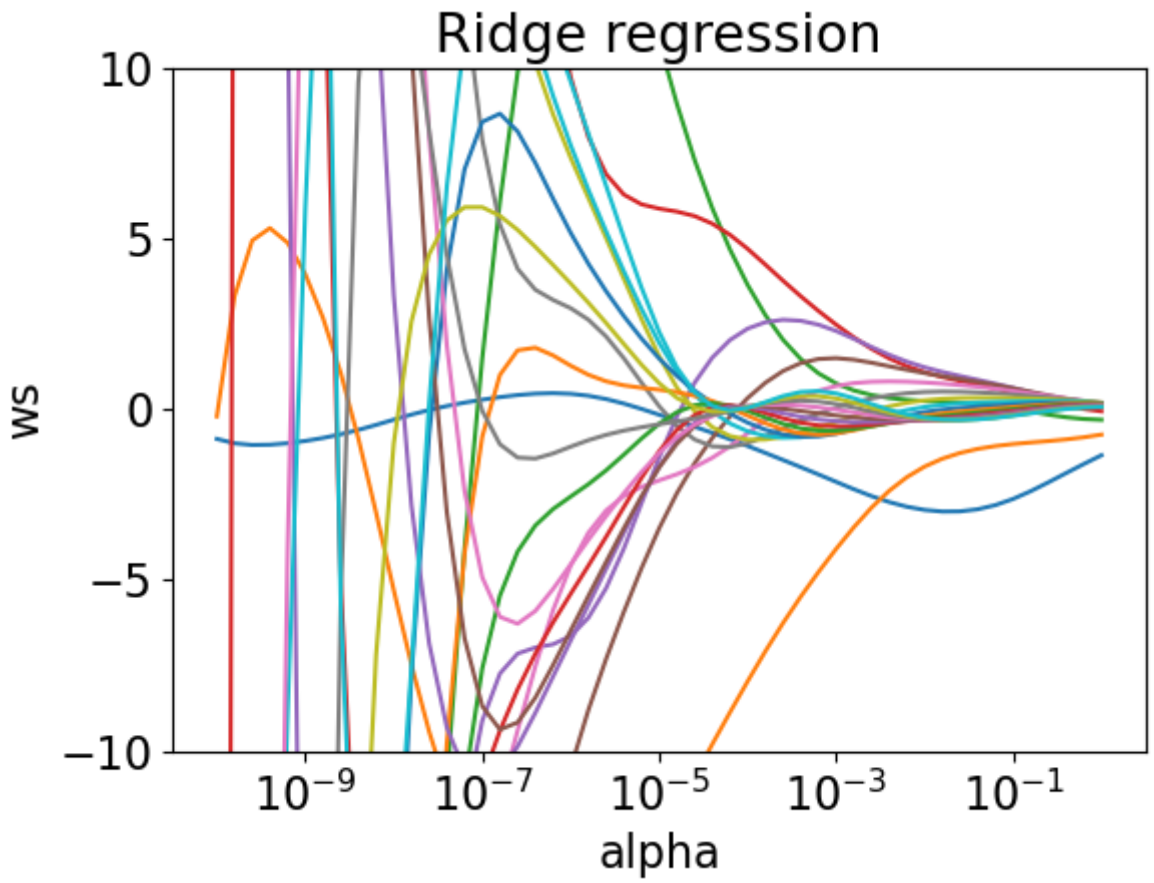
- cost = MSE + α * l2 norm of w

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

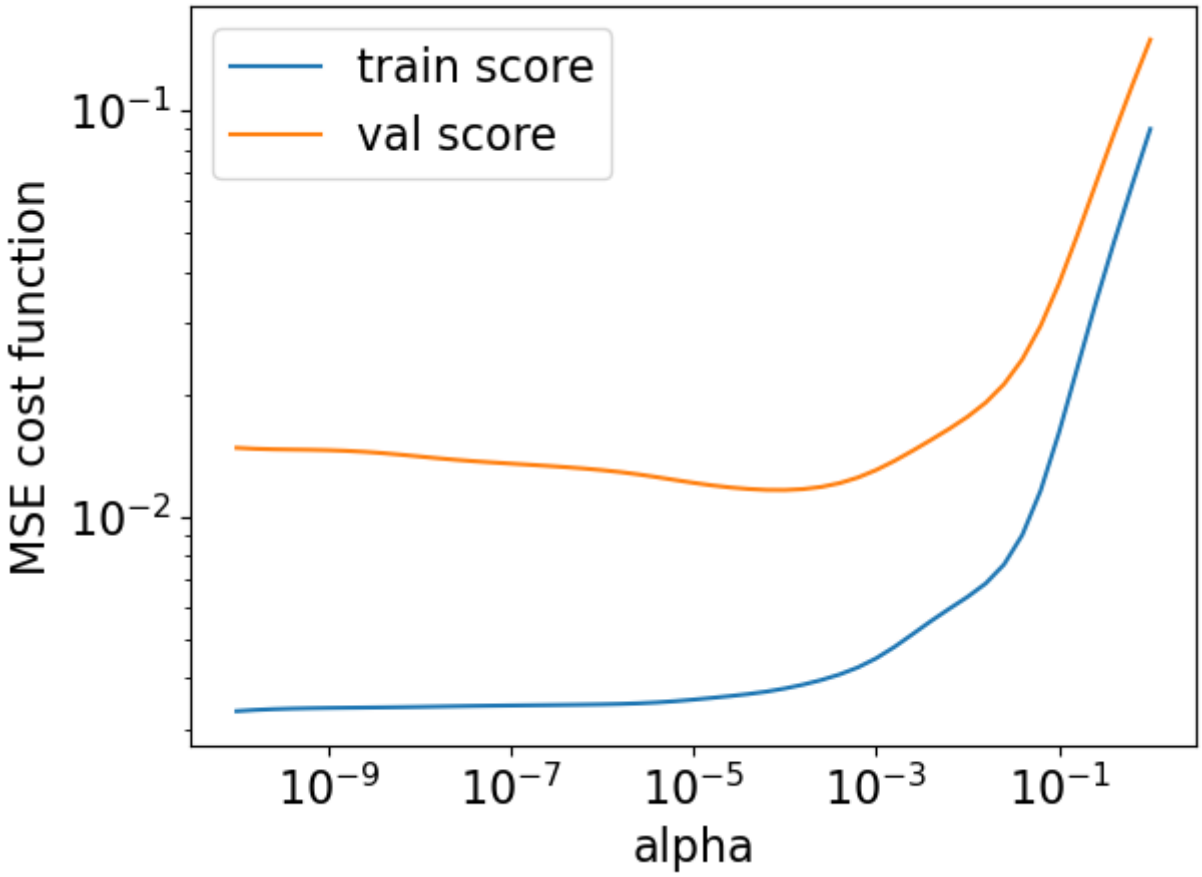
- as α approaches 0, we reproduce the linear regression weights
- small α creates high variance
- large α creates high bias

In [9]:
```
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 = []

# 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)
    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 [10]:
```
plt.plot(alpha, ws)
plt.semilogx()
plt.ylim([-1e1,1e1])
plt.xlabel('alpha')
plt.ylabel('ws')
plt.title('Ridge regression')
plt.tight_layout()
plt.savefig('figures/ridge_coefs.png',dpi=300)
plt.show()
```



In [11]:
```
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 [ ]:

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

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

- the logloss metric with l2 regularization

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

Logistic regression in sklearn

In [12]:
```
from sklearn.linear_model import LogisticRegression

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

In [13]:
```
help(LogisticRegression)
```

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

class LogisticRegression(sklearn.linear_model._base.LinearClassifierMixin, skl
earn.linear_model._base.SparseCoefMixin, sklearn.base.BaseEstimator)
 |  LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_int
ercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver
='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_job
s=None, l1_ratio=None)
 |
 |  Logistic Regression (aka logit, MaxEnt) classifier.
 |
 |  In the multiclass case, the training algorithm uses the one-vs-rest (OvR)
 |  scheme if the 'multi_class' option is set to 'ovr', and uses the
 |  cross-entropy loss if the 'multi_class' option is set to 'multinomial'.
 |  (Currently the 'multinomial' option is supported only by the 'lbfgs',
 |  'sag', 'saga' and 'newton-cg' solvers.)
 |
 |  This class implements regularized logistic regression using the
 |  'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Not
e
 |  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 regularization
 |  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.
 |
 |  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 or primal formulation. Dual formulation is only implemented for
 |      l2 penalty with liblinear solver. Prefer dual=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 regularization
  |      as all other features.
  |      To lessen the effect of regularization on synthetic feature weight
  |      (and therefore on the intercept) intercept_scaling has to be increase
d.
  |
  |  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 data
  |      as ``n_samples / (n_classes * np.bincount(y))``.
  |
  |      Note that these weights will be multiplied with sample_weight (passed
  |      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 : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'},             d
efault='lbfgs'
  |
  |      Algorithm to use in the optimization problem. Default is 'lbfgs'.
  |      To choose a solver, you might want to consider the following aspects:
  |
  |          - For small datasets, 'liblinear' is a good choice, whereas 'sag'
  |            and 'saga' are faster for large ones;
  |          - For multiclass problems, only 'newton-cg', 'sag', 'saga' and
  |            'lbfgs' handle multinomial loss;
  |          - 'liblinear' is limited to one-versus-rest schemes.
  |
  |      .. warning::
```

```
 |        The choice of the algorithm depends on the penalty chosen:
 |        Supported penalties by solver:
 |
 |        - 'newton-cg'   -   ['l2', 'none']
 |        - 'lbfgs'       -   ['l2', 'none']
 |        - 'liblinear'   -   ['l1', 'l2']
 |        - 'sag'         -   ['l2', 'none']
 |        - 'saga'        -   ['elasticnet', 'l1', 'l2', 'none']
 |
 |      .. note::
 |         'sag' and 'saga' fast convergence is only guaranteed on
 |         features with approximately the same scale. You can
 |         preprocess the data with a scaler from :mod:`sklearn.preprocessing
`.
 |
 |      .. seealso::
 |         Refer to the User Guide for more information regarding
 |         :class:`LogisticRegression` and more specifically the
 |         `Table <https://scikit-learn.org/dev/modules/linear_model.html#logi
stic-regression>`_
 |         summarazing 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.
 |
 |  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 fi
t
 |      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='liblinear',
 |      and otherwise selects 'multinomial'.
 |
 |      .. versionadded:: 0.18
 |         Stochastic Average Gradient descent solver for 'multinomial' case.
 |      .. versionchanged:: 0.22
 |          Default changed from 'ovr' to 'auto' in 0.22.
 |
 |  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* solver
```

```
s.
 |
 |  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 equivalent
 |      to using ``penalty='l2'``, while setting ``l1_ratio=1`` is equivalent
 |      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 binary.
 |      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 multinomial,
 |      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"``).
|  LogisticRegressionCV : Logistic regression with built-in cross validation.
|
|  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 descent
|      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...
|
```

```
 |  Method resolution order:
 |      LogisticRegression
 |      sklearn.linear_model._base.LinearClassifierMixin
 |      sklearn.base.ClassifierMixin
 |      sklearn.linear_model._base.SparseCoefMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_interce
pt=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lb
fgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=No
ne, l1_ratio=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  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 an
d
 |          `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 an
d
 |          `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 model,
 |          where classes are ordered as they are in ``self.classes_``.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.linear_model._base.LinearClassifierMixin:
 |
 |  decision_function(self, X)
 |      Predict confidence scores for samples.
 |
 |      The confidence score for a sample is proportional to the signed
 |      distance of that sample to the hyperplane.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The data matrix for which we want to get the confidence scores.
 |
 |      Returns
 |      -------
 |      scores : ndarray of shape (n_samples,) or (n_samples, n_classes)
 |          Confidence scores per `(n_samples, n_classes)` combination. In the
 |          binary case, confidence score for `self.classes_[1]` where >0 mean
s
 |          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)`` wrt. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  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 thi
s
 |      to provide significant benefits.
 |
 |      After calling this method, further fitting with the partial_fit
 |      method (if any) will not work until you call densify.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  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.
```

In [ ]: