APSSDC

Andhra Pradesh State Skill Development Corporation

Day05 Machine Learning Using Python

Day05 Objectives Classification models - 1

- Logistic regression
- Support Vector Machines

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[Titanic Dataset (https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/Datasets/master/Classification/titanic.csv)](https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/Datasets/master/Classification/titanic.csv)

[Mushroom Dataset (https://github.com/AP-State-Skill-Development-Corporation/Datasets/blob/master/Classification/mushrooms.csv)](https://github.com/AP-State-Skill-Development-Corporation/Datasets/blob/master/Classification/mushrooms.csv)

$$Y = 1/1 + e^{-z}$$
$$Y = e^{z}/e^{z} + 1$$
$$log(P[1|0]/1 + P[1|0]) = z$$

In [2]:

```
import pandas as pd
```

Logistic Regression (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.ht

In [3]:

```
df = pd.read_csv('https://raw.githubusercontent.com/AP-State-Skill-Development-Corporat
ion/Datasets/master/Classification/titanic.csv')

df.head()
```

Out[3]:

	survived	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embai
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [4]:

```
df.shape
```

Out[4]:

```
(891, 11)
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['survived', 'pclass', 'name', 'sex', 'age', 'sibsp', 'parch', 'tick
et',
       'fare', 'cabin', 'embarked'],
      dtype='object')
```

In [6]:

```
df['pclass'].value_counts()
```

Out[6]:

```
3    491
1    216
2    184
Name: pclass, dtype: int64
```

In [7]:

```
df['age'].min()
```

Out[7]:

```
0.42
```

In [8]:

```
df['age'].max()
```

Out[8]:

```
80.0
```

In [9]:

```
df['parch'].value_counts()
```

Out[9]:

```
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: parch, dtype: int64
```

In [10]:

```
df['fare'].min(), df['fare'].max()
```

Out[10]:

```
(0.0, 512.3292)
```

In [11]:

```
df['embarked'].value_counts()
```

Out[11]:

```
S    644
C    168
Q     77
Name: embarked, dtype: int64
```

In [12]:

```
df['sex'].value_counts()
```

Out[12]:

```
male      577
female    314
Name: sex, dtype: int64
```

In [13]:

```
df.groupby('sex').sum()
```

Out[13]:

	survived	pclass	age	sibsp	parch	fare
sex						
female	233	678	7286.00	218	204	13966.6628
male	109	1379	13919.17	248	136	14727.2865

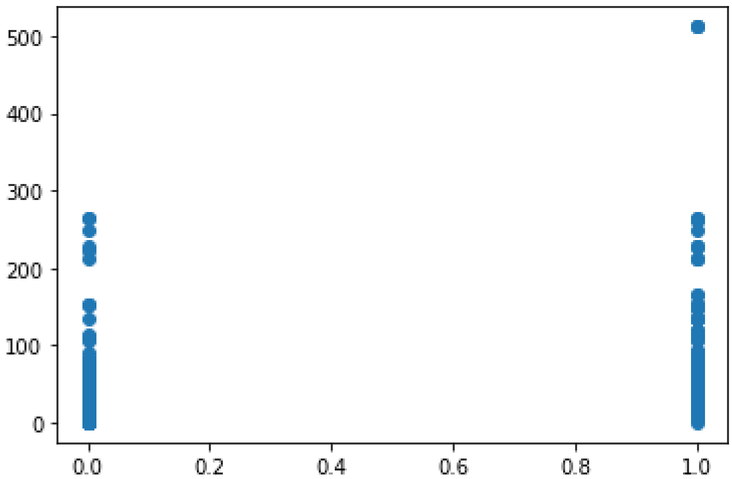
In [14]:

```
import matplotlib.pyplot as plt

plt.plot(df['survived'], df['fare'], 'o')
```

Out[14]:

```
[<matplotlib.lines.Line2D at 0x25643bc38e0>]
```



In [15]:

```
df.isnull().sum()
```

Out[15]:

```
survived      0
pclass        0
name          0
sex           0
age         177
sibsp         0
parch         0
ticket        0
fare          0
cabin       687
embarked      2
dtype: int64
```

- mean - not outliers
- median - outliers
- mode/most frequent value
- constant value
- bfill
- ffill

- fillna()

In [16]:

```
df['age'].fillna(df['age'].mean(), inplace = True)
df['embarked'].fillna(df['embarked'].mode().values[0], inplace = True)

df.isnull().sum()
```

Out[16]:

```
survived      0
pclass        0
name          0
sex           0
age           0
sibsp         0
parch         0
ticket        0
fare          0
cabin       687
embarked      0
dtype: int64
```

In [17]:

```
df.head()
```

Out[17]:

	survived	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embai
0	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [18]:

```
from sklearn.preprocessing import LabelEncoder


lbc = LabelEncoder()
```

In [19]:

```
y = df['survived']

x = df.drop(['survived', 'name', 'ticket','cabin'], axis = 'columns')
```

In [20]:

```
x.head()
```

Out[20]:

	pclass	sex	age	sibsp	parch	fare	embarked
0	3	male	22.0	1	0	7.2500	S
1	1	female	38.0	1	0	71.2833	C
2	3	female	26.0	0	0	7.9250	S
3	1	female	35.0	1	0	53.1000	S
4	3	male	35.0	0	0	8.0500	S

In [21]:

```
x['sex'] = lbc.fit_transform(x['sex'])
x['embarked'] = lbc.fit_transform(x['embarked'])
```

In [22]:

```
x.head()
```

Out[22]:

	pclass	sex	age	sibsp	parch	fare	embarked
0	3	1	22.0	1	0	7.2500	2
1	1	0	38.0	1	0	71.2833	0
2	3	0	26.0	0	0	7.9250	2
3	1	0	35.0	1	0	53.1000	2
4	3	1	35.0	0	0	8.0500	2

In [23]:

```
from sklearn.model_selection import train_test_split

xtr, xtt, ytr, ytt = train_test_split(x, y, test_size = 0.25, random_state = 42)
```

In [24]:

```
from sklearn.linear_model import LogisticRegression


model = LogisticRegression()
```

In [25]:

```
model.fit(xtr, ytr)
```

```
C:\Users\Jesus\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  n_iter_i = _check_optimize_result(
```

Out[25]:

```
LogisticRegression()
```

In [26]:

```
ytt.head(1)
```

Out[26]:

```
709    1
Name: survived, dtype: int64
```

In [27]:

```
xtt.head(1)
```

Out[27]:

	pclass	sex	age	sibsp	parch	fare	embarked
709	3	1	29.699118	1	1	15.2458	0

In [28]:

```
print(ytt)
```

```
709    1
439    0
840    0
720    1
39     1
      ..
880    1
425    0
101    0
199    0
424    0
Name: survived, Length: 223, dtype: int64
```

In [30]:

```
model.predict(xtt.head(1).values)
```

Out[30]:

```
array([0], dtype=int64)
```

In [31]:

```
model.predict_proba(xtt.head(1).values)
```

Out[31]:

```
array([[0.8848125, 0.1151875]])
```

In [32]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [33]:

```
pred = model.predict(xtt)
```

In [34]:

```
pred
```

Out[34]:

```
array([0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0], dtype=int64)
```

In [35]:

```
confusion_matrix(ytt, pred), confusion_matrix(ytt, pred).sum()
```

Out[35]:

```
(array([[115,  19],
        [ 24,  65]], dtype=int64),
 223)
```

In [36]:

```
accuracy_score(ytt, pred)
```

Out[36]:

```
0.8071748878923767
```

[SVM (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) --> Support Vector Machine

Content

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy.

About this file:

- `Attribute Information:` (classes: edible=e, poisonous=p)
- **cap-shape:** bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- **cap-surface:** fibrous=f,grooves=g,scaly=y,smooth=s
- **cap-color:** brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
- **bruises:** bruises=t,no=f
- **odor:** almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
- **gill-attachment:** attached=a,descending=d,free=f,notched=n
- **gill-spacing:** close=c,crowded=w,distant=d
- **gill-size:** broad=b,narrow=n
- **gill-color:** black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
- **stalk-shape:** enlarging=e,tapering=t
- **stalk-root:** bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- **stalk-surface-above-ring:** fibrous=f,scaly=y,silky=k,smooth=s
- **stalk-surface-below-ring:** fibrous=f,scaly=y,silky=k,smooth=s
- **stalk-color-above-ring:** brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- **stalk-color-below-ring:** brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- **veil-type:** partial=p,universal=u
- **veil-color:** brown=n,orange=o,white=w,yellow=y
- **ring-number:** none=n,one=o,two=t
- **ring-type:** cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- **spore-print-color:** black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
- **population:** abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- **habitat:** grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d-

In [37]:

```
df1 = pd.read_csv('https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/Datasets/master/Classification/mushrooms.csv')
```

In [38]:

```
df1.head()
```

Out[38]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk surface below rin
0	p	x	s	n	t	p	f	c	n	k	...	
1	e	x	s	y	t	a	f	c	b	k	...	
2	e	b	s	w	t	l	f	c	b	n	...	
3	p	x	y	w	t	p	f	c	n	n	...	
4	e	x	s	g	f	n	f	w	b	k	...	

5 rows × 23 columns

In [39]:

```
df1.head(1).values
```

Out[39]:

```
array([['p', 'x', 's', 'n', 't', 'p', 'f', 'c', 'n', 'k', 'e', 'e', 's',
        's', 'w', 'w', 'p', 'w', 'o', 'p', 'k', 's', 'u']], dtype=object)
```

In [40]:

```
df1.shape
```

Out[40]:

```
(8124, 23)
```

In [41]:

```
from sklearn.preprocessing import LabelEncoder
lbc = LabelEncoder()
```

In [42]:

```
for col in df1.columns:
    df1[col] = lbc.fit_transform(df1[col])

df1.head()
```

Out[42]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk surface below rin
0	1	5	2	4	1	6	1	0	1	4	...	
1	0	5	2	9	1	0	1	0	0	4	...	
2	0	0	2	8	1	3	1	0	0	5	...	
3	1	5	3	8	1	6	1	0	1	5	...	
4	0	5	2	3	0	5	1	1	0	4	...	

5 rows × 23 columns

In [43]:

```
x = df1.drop('class', axis = 'columns')
```

In [44]:

```
y = df1['class']
```

In [45]:

```
xtr, xtt, ytr, ytt = train_test_split(x, y, test_size = 0.3, random_state = 42)
```

In [46]:

```
from sklearn.svm import SVC
```

In [47]:

```
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, shrink
ing=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=Fals
e, 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.
 |
 |  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`.
 |
 |  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 penalt
y
 |      is a squared l2 penalty.
 |
 |  kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default
='rbf'
 |      Specifies the kernel type to be used in the algorithm.
 |      It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precompute
d' or
 |      a callable.
 |      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 mat
rix
 |      should be an array of shape ``(n_samples, n_samples)``.
 |
 |  degree : int, default=3
 |      Degree of the polynomial kernel function ('poly').
 |      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.
 |
 |      .. 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 prio
r
 |      to calling `fit`, will slow down that method as it internally uses
 |      5-fold cross-validation, and `predict_proba` may be inconsistent w
ith
 |      `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 d
ata
 |      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 wo
rk
 |      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, one-vs-one
 |      ('ovo') is always used as multi-class strategy. 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 tie
d
 |      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 or RandomState instance, default=None
 |      Controls the pseudo random number generation for shuffling the dat
a for
 |      probability estimates. Ignored when `probability` is False.
 |      Pass an int for reproducible output across multiple function call
s.
 |      See :term:`Glossary <random_state>`.
 |
 |  Attributes
 |  ----------
 |  support_ : ndarray of shape (n_SV,)
 |      Indices of support vectors.
 |
 |  support_vectors_ : ndarray of shape (n_SV, n_features)
 |      Support vectors.
 |
 |  n_support_ : ndarray of shape (n_class,), dtype=int32
 |      Number of support vectors for each class.
 |
 |  dual_coef_ : ndarray of shape (n_class-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.
 |
 |  coef_ : ndarray of shape (n_class * (n_class-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_`.
 |
 |  intercept_ : ndarray of shape (n_class * (n_class-1) / 2,)
 |      Constants in decision function.
 |
 |  fit_status_ : int
 |      0 if correctly fitted, 1 otherwise (will raise warning)
 |
 |  classes_ : ndarray of shape (n_classes,)
 |      The classes labels.
 |
 |  probA_ : ndarray of shape (n_class * (n_class-1) / 2)
 |  probB_ : ndarray of shape (n_class * (n_class-1) / 2)
 |      If `probability=True`, it corresponds to the parameters learned in
 |      Platt scaling to produce probability estimates from decision value
s.
 |      If `probability=False`, it's an empty array. Platt scaling uses th
e
 |      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]_.
 |
 |  class_weight_ : ndarray of shape (n_class,)
 |      Multipliers of parameter C for each class.
 |      Computed based on the ``class_weight`` parameter.
 |
 |  shape_fit_ : tuple of int of shape (n_dimensions_of_X,)
 |      Array dimensions of training vector ``X``.
 |
 |  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]
 |
 |  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 comparison to regularizedlikelihood methods."
 |      <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1639>`_
 |
 |  Method resolution order:
 |      SVC
 |      sklearn.svm._base.BaseSVC
 |      sklearn.base.ClassifierMixin
 |      sklearn.svm._base.BaseLibSVM
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=
0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_w
eight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', bre
ak_ties=False, random_state=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.svm._base.BaseSVC:
 |
 |  decision_function(self, X)
 |      Evaluates the decision function for the samples in X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |
 |      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 proporti
onal
 |      to the distance of the samples X to the separating hyperplane. If
the
 |      exact distances are required, divide the function values by the no
rm of
 |      the weight vector (``coef_``). See also `this question
 |      <https://stats.stackexchange.com/questions/14876/
 |      interpreting-distance-from-hyperplane-in-svm>`_ for further detail
s.
 |      If decision_function_shape='ovr', the decision function is a monot
onic
 |      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) o
r                 (n_samples_test, n_samples_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.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from sklearn.svm._base.BaseSVC:
 |
 |  predict_log_proba
```

```
 |      Compute log probabilities of possible outcomes for samples in X.
 |
 |      The model need to have probability information computed at trainin
g
 |      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
 |      Compute probabilities of possible outcomes for samples in X.
 |
 |      The model need to have probability information computed at trainin
g
 |      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.
 |
 |  probA_
 |
 |  probB_
 |
 |  ----------------------------------------------------------------------
 |  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.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
 |
 |      Notes
 |      -----
```

```
 |      If X and y are not C-ordered and contiguous arrays of np.float64 a
nd
 |      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 spa
rse
 |      matrices as input.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from sklearn.svm._base.BaseLibSVM:
 |
 |  coef_
 |
 |  n_support_
 |
 |  ----------------------------------------------------------------------
 |  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 : mapping of string to any
 |          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 pipelines). 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 : object
 |          Estimator instance.
 |
```

In [48]:

```
svc = SVC(kernel = 'linear')
```

In [49]:

```
svc.fit(xtr, ytr)
```

Out[49]:

```
SVC(kernel='linear')
```

In [50]:

```
pred = svc.predict(xtt)
```

In [51]:

```
confusion_matrix(ytt, pred)
```

Out[51]:

```
array([[1207,   50],
       [  45, 1136]], dtype=int64)
```

In [52]:

```
accuracy_score(ytt, pred)
```

Out[52]:

```
0.961033634126333
```

In [53]:

```
ytt.shape
```

Out[53]:

```
(2438,)
```

In [55]:

```
from sklearn.model_selection import GridSearchCV

parameters = [{'C':[1,10,100,1000],'kernel':['linear']},
             {'C':[1,10,100,1000],'kernel':['rbf'],
             'gamma':[0.1,0.2,0.4,0.6,0.8]}]

grid_search = GridSearchCV(estimator = svc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search.fit(xtt,ytt)
best_accuracy=grid_search.best_score_
best_parameters = grid_search.best_params_
```