**C*float, default=1.0***

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

**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’, ‘precomputed’ 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 matrix 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.

*Changed in version 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 [User Guide](https://scikit-learn.org/stable/modules/svm.html#shrinking-svm).

**probability*bool, default=False***

Whether to enable probability estimates. This must be enabled prior to calling fit, will slow down that method as it internally uses 5-fold cross-validation, and predict\_proba may be inconsistent with predict. Read more in the [User Guide](https://scikit-learn.org/stable/modules/svm.html#scores-probabilities).

**tol*float, default=1e-3***

Tolerance for stopping criterion.

**cache\_size*float, default=200***

Specify the size of the kernel cache (in MB).

**class\_weight*dict or ‘balanced’, default=None***

Set the parameter C of class i to class\_weight[i]\*C for SVC. If not given, all classes are supposed to have weight one. The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y)).

**verbose*bool, default=False***

Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

**max\_iter*int, default=-1***

Hard limit on iterations within solver, or -1 for no limit.

**decision\_function\_shape*{‘ovo’, ‘ovr’}, default=’ovr’***

Whether to return a one-vs-rest (‘ovr’) decision function of shape (n\_samples, n\_classes) as all other classifiers, or the original one-vs-one (‘ovo’) decision function of libsvm which has shape (n\_samples, n\_classes \* (n\_classes - 1) / 2). However, one-vs-one (‘ovo’) is always used as multi-class strategy. The parameter is ignored for binary classification.

*New in version 0.17: decision\_function\_shape=’ovr’* is recommended.

**break\_ties*bool, default=False***

If true, decision\_function\_shape='ovr', and number of classes > 2, [predict](https://scikit-learn.org/stable/glossary.html#term-predict) will break ties according to the confidence values of [decision\_function](https://scikit-learn.org/stable/glossary.html" \l "term-decision_function); otherwise the first class among the tied classes is returned. Please note that breaking ties comes at a relatively high computational cost compared to a simple predict.

*New in version 0.22.*

**random\_state*int, RandomState instance or None, default=None***

Controls the pseudo random number generation for shuffling the data for probability estimates. Ignored when probability is False. Pass an int for reproducible output across multiple function calls. See [Glossary](https://scikit-learn.org/stable/glossary.html#term-random_state).

a1a

Graphical user interface, text, application, email

Description automatically generated

Gamma =0.1 and kernel =’rbf’ acc =0.83

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

print("Loading Dataset...")

X,y = load\_svmlight\_file("a1a.txt")

print("Creating classifier object...")

clf = svm.SVC(kernel='rbf',gamma=0.1, C=1, random\_state= 42)

print("Training classifier with cross validation, k=5")

scores = cross\_val\_score(clf, X, y, cv=5,n\_jobs=4)

print("Training Complete!")

acc = scores.mean()

stdiv = scores.std()

print("Cross Validation Mean Accuracy = %0.2f" % acc )

print("Standard Deviation of the Mean Accuracy across all runs = %0.2f" % stdiv)

Graphical user interface, text, application, email

Description automatically generated

Gamma =0.01 and kernel =’rbf’ acc =0.82

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

print("Loading Dataset...")

X,y = load\_svmlight\_file("a1a.txt")

print("Creating classifier object...")

clf = svm.SVC(kernel='rbf',gamma=0.01, C=1, random\_state= 42)

print("Training classifier with cross validation, k=5")

scores = cross\_val\_score(clf, X, y, cv=5,n\_jobs=4)

print("Training Complete!")

acc = scores.mean()

stdiv = scores.std()

print("Cross Validation Mean Accuracy = %0.2f" % acc )

print("Standard Deviation of the Mean Accuracy across all runs = %0.2f" % stdiv)

Graphical user interface, text, application, email

Description automatically generated

Kernal =’poly’ , degree =2 acc =0.83

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

print("Loading Dataset...")

X,y = load\_svmlight\_file("a1a.txt")

print("Creating classifier object...")

clf = svm.SVC(kernel='poly', degree=2 , C=1, random\_state= 42)

print("Training classifier with cross validation, k=5")

scores = cross\_val\_score(clf, X, y, cv=5,n\_jobs=4)

print("Training Complete!")

acc = scores.mean()

stdiv = scores.std()

print("Cross Validation Mean Accuracy = %0.2f" % acc )

print("Standard Deviation of the Mean Accuracy across all runs = %0.2f" % stdiv)

Graphical user interface, text, application, email

Description automatically generated

a9a

Graphical user interface, text, application, email

Description automatically generated

Gama =0.1 kernal=’rbf’ acc =0.85

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

print("Loading Dataset...")

X,y = load\_svmlight\_file("a9a.txt")

print("Creating classifier object...")

clf = svm.SVC(kernel='rbf',gamma=0.1, C=1, random\_state= 42)

print("Training classifier with cross validation, k=5")

scores = cross\_val\_score(clf, X, y, cv=5,n\_jobs=4)

print("Training Complete!")

acc = scores.mean()

stdiv = scores.std()

print("Cross Validation Mean Accuracy = %0.2f" % acc )

print("Standard Deviation of the Mean Accuracy across all runs = %0.2f" % stdiv)

Graphical user interface, text, application, email

Description automatically generated

Gamma =0.01 ,kernel =’rbf’ acc =0.84

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

print("Loading Dataset...")

X,y = load\_svmlight\_file("a9a.txt")

print("Creating classifier object...")

clf = svm.SVC(kernel='rbf',gamma=0.01, C=1, random\_state= 42)

print("Training classifier with cross validation, k=5")

scores = cross\_val\_score(clf, X, y, cv=5,n\_jobs=4)

print("Training Complete!")

acc = scores.mean()

stdiv = scores.std()

print("Cross Validation Mean Accuracy = %0.2f" % acc )

print("Standard Deviation of the Mean Accuracy across all runs = %0.2f" % stdiv)

Graphical user interface, text, application, email

Description automatically generated

Kernel =’poly’ degree =2

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

print("Loading Dataset...")

X,y = load\_svmlight\_file("a9a.txt")

print("Creating classifier object...")

clf = svm.SVC(kernel='poly', C=1, random\_state= 42,degree=2)

print("Training classifier with cross validation, k=5")

scores = cross\_val\_score(clf, X, y, cv=5)

print("Training Complete!")

acc = scores.mean()

stdiv = scores.std()

print("Cross Validation Mean Accuracy = %0.2f" % acc )

print("Standard Deviation of the Mean Accuracy across all runs = %0.2f" % stdiv)

Graphical user interface, text, application, email

Description automatically generated

Part III: Parameter Fine Tuning

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

import pandas as pd

from sklearn.model\_selection import GridSearchCV

print("Loading Dataset...")

X,y = load\_svmlight\_file("a1a.txt")

print("Creating Parameter Grid...")

param\_grid = [

{'C': [1, 10], 'kernel': ['linear']},

{'C': [1, 10], 'gamma': [0.001, 0.01], 'kernel': ['rbf']},

]

print("Creating classifier object...")

svc = svm.SVC()

print("Creating a grid search cross validator object...")

clf = GridSearchCV(svc, param\_grid)

print("Fitting the models with different parameters...")

clf.fit(X, y)

print("Writing all fitting results...")

df = pd.DataFrame(clf.cv\_results\_)

df.to\_csv("Parameter\_Tuning\_Results.csv")

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

* What is printed in the output file?

Parameter tunning to avoid overfiting

* What does every column represent?

How the splits are done.

* What does every row represent?

Each try corresponds to split test and split test score after validation.

* Which run gave you the best outcome?

Row 0 , 0.834890965732087 . split2\_test\_score

* Modify the code to include testing with the polynomial kernel with degrees 2 & 3 along with regularization values C 0.01, 0.1, 1, 10. What are the obtained testing errors for these parameter sets?

from sklearn.datasets import load\_svmlight\_file

from sklearn import svm

import pandas as pd

from sklearn.model\_selection import GridSearchCV

print("Loading Dataset...")

X,y = load\_svmlight\_file("a1a.txt")

print("Creating Parameter Grid...")

param\_grid = [

{'C': [0.01,0.1,1,10], 'kernel': ['poly'] ,'degree':[2,3] },

]

print("Creating classifier object...")

svc = svm.SVC()

print("Creating a grid search cross validator object...")

clf = GridSearchCV(svc, param\_grid)

print("Fitting the models with different parameters...")

clf.fit(X, y)

print("Writing all fitting results...")

df = pd.DataFrame(clf.cv\_results\_)

df.to\_csv("Parameter\_Tuning\_Result.csv")

df.sort\_values("rank\_test\_score", axis = 0, ascending = True,

inplace = True, na\_position ='last')

for i in df.index :

print (df['rank\_test\_score'][i],df['params'][i], df['mean\_test\_score'][i], df['std\_test\_score'][i])

* Graphical user interface, text, application

  Description automatically generated

Text

Description automatically generated

Graphical user interface, application, table, Excel

Description automatically generated

1-0.8317757009345794

Testing Error = 0.16.