Experiment 5 – SVC Write-up

Name: Ashish Nanda

Roll No.: J041

class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)

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 LinearSVC or SGDClassifier instead.

The multiclass support is handled according to a one-vs-one scheme.

Parameters

C : *float, default=1.0*

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

kernel*{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).

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.

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.

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.

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. 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.

break_ties : ***bool, default=False***

If true, decision_function_shape='ovr', and number of classes > 2, predict will break ties according to the confidence values of decision_function; otherwise the first class among the tied classes is returned. Breaking ties comes at a relatively high computational cost compared to a simple predict.

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.

Attributes

1. **class_weight_ :** ndarray of shape (n_classes,) **:** Multipliers of parameter C for each class.
2. **classes_** : ndarray of shape (n_classes,) **:**The classes labels.
3. **coef_ :** ndarray of shape (n_classes * (n_classes - 1) / 2, n_features)
4. **dual_coef_** :ndarray of shape (n_classes -1, n_SV)
5. **fit_status_**: int **:** 0 if correctly fitted, 1 otherwise (will raise warning)
6. **intercept_ :** ndarray of shape (n_classes * (n_classes - 1) / 2,)
7. **support_ :** ndarray of shape (n_SV)
8. **support_vectors_ :** ndarray of shape (n_SV, n_features)
9. **n_support_** : ndarray of shape (n_classes,), dtype=int32
10. **probA_ :** ndarray of shape (n_classes * (n_classes - 1) / 2)
11. **probB_ :** ndarray of shape (n_classes * (n_classes - 1) / 2)
12. **shape_fit_ :** tuple of int of shape (n_dimensions_of_X,)

SVMs are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered numpy.ndarray (dense) or scipy.sparse.csr_matrix (sparse) with dtype=float64.