Support Vector Machines

Support vector machines (SVM) are a set of supervised learning methods used for *classification*, *regressions*, and *outliers detection*.

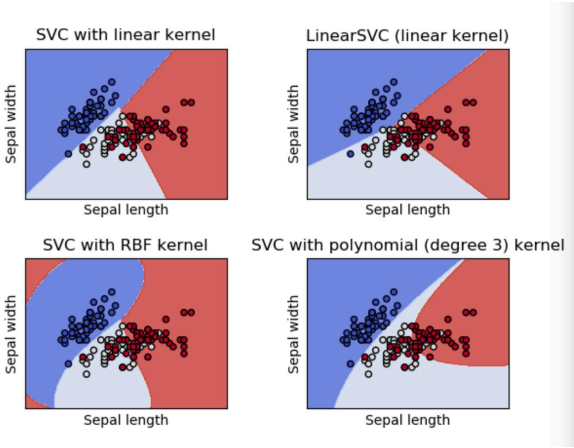
Given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples.

Advantage

- Effective in high dimensional spaces
- Effective in cases where # of dimensions is greater than # of samples
- Use a subset of training points in decision function (support vectors) so that can be memory efficient
- Versatile

Disadvantage

- Overfitting
- Do not provide probabilities estimates (calculated using fiiive-fold cross validation)



`SVC`, `NuSVC`, and `LinearSVC` can perform multi-class classification on a dataset.

- `SVC` and `NuSVC` are similar
- `LinearSVC` is another implementation for the case of a linear kernel (does not accept the keyword `kernel`, assumed to be linear)
- All take as input two arrays
  - An array `X` of size `[n_samples, n_features]` from training samples
  - An array `y` of class labels (strings or integers) of size `[n_samples]`

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0, decision_-
function_shape='ovr', degree=3, gam-
ma='auto', kernel='rbf', max_iter=-1,
probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False)
```

After being fitted, the model can then be used to predict new values:

```
>>> clf.predict([[2., 2.]])
array([1])
```

Multi-class Classification

- SVC and NuSVC implement the 'one-against-one' approach for multi-class classification
- If n_class is the # of class, then n_class * (n_class - 1) / 2 classifiers are constructed and each one trains data from two classes
- decision_function_shape options allows to aggregate the results of the 'one-against-one' classifiers to a decision function of shape (n_samples, n_classes)

```
>>> X = [[0], [1], [2], [3]]
>>> Y = [0, 1, 2, 3]
>>> clf = svm.SVC(decision_func-
tion_shape='ovo')
>>> clf.fit(X, Y)
SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0,
    decision_function_shape='ovo', de-
gree=3, gamma='auto', kernel='rbf', max_i-
ter=-1, probability=False, random_sta-
te=None, shrinking=True, tol=0.001, verbo-
se=False)
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes: 4*3/2 = 6
6
>>> clf.decision_function_shape = "ovr"
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes
4
```

- LinearSVC implements 'one-vs-the-rest' multi-class strategy (training n_class models)
- If there are two classes, only one class is trained

```
>>> lin_clf = svm.LinearSVC()
>>> lin_clf.fit(X, Y)
LinearSVC(C=1.0, class_weight=None,
dual=True, fit_intercept=True,
     intercept_scaling=1,
loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2',
random_state=None, tol=0.0001, verbose=0)
>>> dec = lin_clf.decision_function([[1]])
>>> dec.shape[1]
4
```

Scores and Probabilities

- SCV method decision_function gives per-class scores for each sample
- When the constructor option probability is set to True, class membership probability estimates are enabled

Unbalanced Problems

- Keywords class_weigth and sample_weight can be used in problems where it is desired to give more importance to certain classes or certain individual samples
- SVC implement a keyword class_weigth in the fit method (a dictionary of the form {class_label: value}, where the value is a floating point number > 0 that sets the parameter C of class_label to C * value
- SVC, NuSVC, SVR, NuSVR, and OneClassSVM implement also weights for individual samples in the method fit keyword sample_weight (parameter C for the i-th example to C * weight[i])

sklearn.svm.SVC

```
class sklearn.svm.SVC(C=1.0, kernel='rbf',
degree=3, gamma='auto', coef0=0.0, shrink-
ing=True, probability=False, tol=0.001,
cache_size=200, class_weight=None, ver-
bose=False, max_iter=-1, decision_func-
tion_shape='ovr', random_state=None)
```

Kernel: string, optional (default='rbf')

- One of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable

C: float, optional (default=1.0)

- Penalty parameter C of the error term
- Controls the tradeoff between smooth decision boundary and classifying training points correctly
- Straight forward may be a better choice

Large C value means getting more points correct

Gamma: float, optional (default='auto')

- Kernel coefficient for 'rbf', 'poly', and 'sigmoid'. If gamma is 'auto' then `1/n_features` will be used instead
- Defines how far the influence of a single training example reaches

Low gamma value means every point has a far reach, and low gamma value means every point has a close reach

Overfitting

- Stop overfitting (tuning parameters)
  - C
  - Gamma
  - Kernel