Support Vector Machines

A Support Vector Machine (SVM) is a very powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection. SVMs are particularly well suited for classification of complex but small- or medium-sized datasets that is not linearly separable

Linearly Separable data

• We can define a linear function as the decision boundary and all data points get perfectly separated

Linear SVM Classification
SVM classifier as fitting the widest possible street (represented by the parallel dashed lines) between the classes. This is called **large margin classification.**
Support Vectors
Adding more training instances “off the street” will not affect the decision boundary at all: it is fully determined (or “supported”) by the instances located on the edge of the street. These instances are called the support vectors
Note
• SVMs are sensitive to the feature scales. • Unlike Logistic Regression classifiers, SVM classifiers do not out put probabilities for each class.
Computational Complexity
The LinearSVC class is based on the liblinear library, which implements an optimized algorithm for linear SVMs.1 It does not support the kernel trick, but it scales almost linearly with the number of training instances and the number of features: its training time complexity is roughly O(m × n).

Soft Margin Classification:

If we strictly impose that all instances be off the street and on the right side, this iscalled hard margin classification There are two main issues with hard margin classification.First, it only works if the data is linearly separable, and second it is quite sensitive to outliers.

- The objective is to find a good balance between keeping the street as large as possible and limiting the margin violations (i.e., instances that end up in the middle of the street or even on the wrong side). This is called soft margin classification
- In Scikit-Learn's SVM classes, you can control this balance using the hyperparameter C: a smaller C value leads to a wider street but more margin violations
- If your SVM model is overfitting, you can try regularizing it by reducing C.

Notes

- Instead of LinearSVC ,you could use the SVC class, using SVC(kernel="linear", C =1), but it is much slower, especially with large training sets
- The SVC class is based on the libsvm library, which implements an algorithm that supports the kernel trick. The training time complexity is usually between O(m2 × n) and O(m3 × n). Unfortunately, this means that it gets dreadfully slow when the number of training instances gets large
- However, it scales well with the number of features, especially with sparse features (i.e., when each instance has few nonzero features). In this case, the algorithm scales roughly with the average number of nonzero features per instance
- The LinearSVC class regularizes the bias term, so you should center the training set first by subtracting its mean. This is automatic if you scale the data using the StandardScaler. Moreover, make sure you set the hyperparameter to "hinge", as it is not the default loss "value. Finally, for better performance you should set the dual hyperparameter to False, unless there are more features than training instances

• **Pipeline** can be used to chain multiple estimators into one.This is useful as there is often a fixed sequence of steps in processing the data, e.g., feature selection, normalization and classification. • Calling fit on the pipeline is the same as calling fit on each estimator in turn, transform the input and pass it on to the next step.The Pipeline is built using a list of (key, value) pairs • key is a string containing the name you want to give this step and value is an estimator object

- SGDClassifier class, with SGDClassifier(loss="hinge",alpha=1/(m*C)). This applies regular Stochastic Gradient Descent to train a linear SVM classifier. It does not converge as fast as the LinearSVC class, but it can be useful to handle huge datasets that do not fit in memory (out-of-core training),or to handle online classification tasks.

Comparison of Scikit-Learn classes for SVM classification

Class
Time complexity
Out-of-core support
Scaling required
Kernel trick
LinearSVC
O(m × n)
No
Yes
No
SGDClassifier
O(m × n)
Yes
Yes
No
No
SVC
$O(m^2 \times n)$ to $O(m^3 \times n)$
No
Yes
Yes

LinearSVC implemented via the LIBLINEAR library

• Uses coordinate descent, suitable for large datasets, but NO KERNEL TRICK! • Best for binary classification, • For multiclass uses One vs The Rest • The underlying C implementation uses a random number generator to select features when fitting the model. • You may get slightly different results for the same input data. • Try with a smaller tol parameter if you get different results. • LinearSVC Regularizes the bias term, so you must always scale the data • Not suitable for the kernel trick •

SVC implemented via the LIBSVM library

• Uses Sequential Minimum Optimization (SMO) to solve the QP associated with Support Vector Machines • Can use nonlinear Kernel, but does not scale well to large data

---

Nonlinear SVM Classification

One approach to handling nonlinear datasets is to add more features, such as polynomial features

Polynomial Kernel

```
• from sklearn.datasets import make_moons
• from sklearn.pipeline import Pipeline
• from sklearn.preprocessing import PolynomialFeatures
• X, y = make_moons(n_samples=100, noise=0.15)
• polynomial_svm_clf = Pipeline([
("poly_features", PolynomialFeatures(degree=3)),
("scaler", StandardScaler()),
("svm_clf", LinearSVC(C=10, loss="hinge"))
])
• polynomial_svm_clf.fit(X, y)
```

Kernel trick

When using Polynomial transformation with large dimensional polynomials, SVC is slow bcz of the combinatorial explosion of the number of features • Fortunately, SVM class has a kernel trick

- The Kernel Trick comes from viewing the SVM problem as the dual of the Quadratic Program of soft margin classifier

How to find the right hyperparameter? – Use Grid Search

• First, do a very coarse grid search,

• Then, do a finer grid search around the best values found above

Action Items
Implementation
Parameters

Pipeline([("scaler", StandardScaler()), ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))]) • poly_kernel_svm_clf.fit(X, y)	• Degree – Degree of the poynomial kernel • Coef0 – Controls how much the model is influenced by high degree vs low degree • It is the r in the Kernel Function • C – regularize the parameters with a l2 penalty

Adding Similarity Features

Another technique to tackle nonlinear problems is to add features computed using a similarity function that measures how much each instance resembles a particular landmark., ℓ is the landmark (which can be another datapoint)

Gaussian RBF

$\phi_\gamma(x, \ell) = \exp(-\gamma \| x - \ell \|^2)$

It is a bell-shaped function varying from 0 (very far away from the landmark) to 1 (at the landmark).

Gaussian RBF Kernel

```
rbf_kernel_svm_clf = Pipeline([
("scaler", StandardScaler()),
("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
])
rbf_kernel_svm_clf.fit(X, y)
```

• $\phi\gamma\ \boldsymbol{x}, \ell = \exp -\gamma \| \boldsymbol{x} - \ell \| 2$

• γ acts like a regularizer

• Imagine a Gaussian bump around each data point

• The larger you make γ the more individual centric the decision boundaries become

---

SVM Regression

SVM Regression tries to fit as many instances as possible on the street while limiting margin violations (i.e., instances off the street). The width of the street is controlled by a hyperparameter ϵ.Adding more training instances within the margin does not affect the model's predictions; thus, the model is said to be .ϵ-insensitive.

.

```
from sklearn.svm import LinearSVR
svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

Scikit-Learn's SVR class (which supports the kernel trick). The SVR class is the regression equivalent of the SVC class, and the LinearSVR class is the regression equivalent of the LinearSVC class.The LinearSVR class scales linearly with the size of the training set (just like the LinearSVC class), while the SVR class gets much too slow when the training set grows large (just like the SVC
class).

```
from sklearn.svm import SVR
svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```

The bias term will be called b and the feature weights vector will be called w. No bias feature will be added to the input feature vectors

Decision Function and Predictions

Linear SVM classifier prediction

y = {0 if wTx + b < 0,
1 if wTx + b ≥ 0}

The decision boundary is the set of points where the decision function is equal to 0: it is the intersection of two planes, which is a straight line. The dashed lines represent the points where the decision function is equal to 1 or –1: they are parallel and at equal distance to the decision boundary, forming a margin around it

- Training a linear SVM classifier means finding the value of w and b that make this margin as wide as possible while avoiding margin violations (hard margin) or limiting them (soft margin).

Training Objective

the slope of the decision function: it is equal to the norm of the weight vector,.||| w|| The smaller the weight vector w, the larger the margin

Hard margin linear SVM classifier objective

minimize $_{w, b}$= 1/2(wTw)

subject to t^{i} (wTx i + b) ≥ 1 for i = 1, 2, ⋯,m

Quadratic Programming

The hard margin and soft margin problems are both convex quadratic optimization problems with linear constraints. Such problems are known as Quadratic Programming
(QP) problems.

Quadratic Programming problem