Data Analysis and Machine Learning: Support Vector Machines

Morten Hjorth-Jensen[1,2]

Department of Physics, University of Oslo[1]

Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University[2]

Nov 1, 2018

Support Vector Machines, overarching aims

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. It is one of the most popular models in Machine Learning, and anyone interested in Machine Learning should have it in their toolbox. SVMs are particularly well suited for classification of complex but small-sized or medium-sized datasets.

The basic mathematics relies on the definition of hyperplanes and the definition of a **margin** which separates classes (in case of classification problems) of variables. It is also used for regression problems.

With SVMs we distinguish between hard margin and soft margins. The latter introduces a so-called softening parameter to be discussed below. We distringuish also between linearn and non-linear approaches.

These notes will be updated shortly with more material

SVMs, basics with scikit-learn

The following Scikit-Learn code loads the iris dataset, scales the features, and then trains a linear SVM model (using the LinearSVC class with C = 0.1 and the hinge loss function, described shortly) to detect Iris-Virginica flowers.

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
iris = datasets.load_iris()
X = iris["data"][:, (2, 3)] # petal length, petal width
y = (iris["target"] == 2).astype(np.float64) # Iris-Virginica
svm_clf = Pipeline((
("scaler", StandardScaler()),
("linear_svc", LinearSVC(C=1, loss="hinge")),
))
svm_clf.fit(X_scaled, y)
```

Alternatively, you could use the SVC class, using **SVC(kernel="linear", C=1)**, but it is much slower, especially with large training sets, so it is not recommended. Another option is to use the SGDClassifier class, with **SGDClassifier(loss="hinge", alpha=1/(m*C))**. This applies

SVMs, adding polynomial features

Although linear SVM classifiers are efficient and work surprisingly well in many cases, many datasets are not even close to being linearly separable. One approach to handling nonlinear datasets is to add more features, such as polynomial features. In some cases this can result in a linearly separable dataset.

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

SVMs, polynomials and kernels

Adding polynomial features is simple to implement and can work great with all sorts of Machine Learning algorithms (not just SVMs), but at a low polynomial degree it cannot deal with very complex datasets, and with a high polynomial degree it creates a huge number of features, making the model too slow. Fortunately, when using SVMs you can apply an almost miraculous mathematical technique called the kernel trick discussed during the lectures. It makes it possible to get the same result as if you added many polynomial features, even with very high-degree polynomials, without actually having to add them. So there is no combinatorial explosion of the number of features since you don't actually add any features. This trick is implemented by the SVC class.

```
from sklearn.svm import SVC
poly_kernel_svm_clf = Pipeline((
("scaler", StandardScaler()),
("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
))
poly_kernel_svm_clf.fit(X, y)
```

This code trains an SVM classifier using a 3rd-degree polynomial

SVMs, regression

You can use Scikit-Learn's LinearSVR class to perform linear SVM Regression. The following code shows how to use the regression option (more material to come)

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

To tackle nonlinear regression tasks, you can use a kernelized SVM model