Chapter 5: Support Vector Machines

SVM is capable of performing linear or nonlinear classification, regression and outlier detection. It is well suited for classification of complex small, or medium sizd datasets

1. Linear SVM Classification

SVM classifier can be thought of as Large margin classification - fitting the widest possible street. Adding more training instances "off the street" will not affect the decision boundary at all as it is fully determined or supported by instances located on the edge of the street (support vectors)

Tries to fit the largest possible street between two classes while limiting the margin violations.

SVMs are sensitive to the feature scales

LinearSVM classifier model predicts the class of a new instance x by computing the decision function $w^T x + b = w_1 x_1 + \ldots + w_n x_n + b$. If result is positive, the predicted class $\hat{y}$ is the positive class and otherwise it is the negative class.

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

1.1 Hard Margin Classification

We strictly impose that all instances must be off the street. Meaning that there should be no outliers. But there are two main issues here:

- Only works if data is linearly separable
- it is sensitive to outliers

1.2 Soft Margin Classification

To avoid these issues, we use a more flexible model. The objective is to find a good balance between keeping the street as large as possible and limiting the margin violations. This means that instances end up in the middle of the street or even on the wrong side.

This margin is controlled by the hyperparameter C (how soft margin is)

- Low C: Margin is very soft --> Allow more misclassification
- High C: Margin is harder --> Less misclassification

In [ ]:
```
import numpy as np
import os
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
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", random_state=42)),
    ])

svm_clf.fit(X, y)
```

Out[ ]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('linear_svc', LinearSVC(C=1, loss='hinge', random_state=42))])
```

In [ ]:
```
svm_clf.predict([[5.5, 1.7]])
```

Out[ ]:
```
array([1.])
```

Unlike logistic regression classifiers, SVM classifiers do not output probabilties for each class

LinearSVC class regularizes the bias term so we should center the training set first by subtracting its mean.

Instead of LinearSVD, we could also use SVC(kernel="linear", C=1). We could also use SGDClassifier() and this applies the regular Stochastic Gradient Descent to train linear SVM classifier

2. Nonlinear SVM Classifier

One approach to handling nonlinear datasets is to add more features such as polynomial features. This can result in a linearly separable dataset. Example: with just one feature the dataset is not linearly separable. But if we add $feature2 = (feature1)^2$, we can get a linearly serarable dataset

In [ ]:
```
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)
```

Out[ ]:
```
Pipeline(steps=[('poly_features', PolynomialFeatures(degree=3)),
                ('scaler', StandardScaler()),
                ('svm_clf', LinearSVC(C=10, loss='hinge'))])
```

2.1 Polynomial Kernel

With low polynomial degree, we cannot deal with very complex datasets. With high polynomial degree it can create a huge number of features, making the model too slow.

When using SVMs, we can apply the kernel trick technique. The kernel trick makes it possible to get the same result as if you had added many polynomial features, even with very high-degree polynomials without actually having to add them. Hence, there is not combinatorial explosion of the number of features because you don't actually add any features.

In [ ]:
```
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)
```

Out[ ]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('svm_clf', SVC(C=5, coef0=1, kernel='poly'))])
```

If model is overfitting, we can reduce the polynomial degree. If model is underfitting, we can increase the polynomial degree.

A common approach to finding the right hyperparameter values is to use grid search. It is often faster to first do a very coarse grid search, then a finer grid search around the best values found. Having a good sense of what each hyperparameter actually does can help you search in the right part of the hyperparameter space.

2.2 Similarity Features

Another technique to tackle nonlinear problems is to add features computed using a similarity function, which measures how much each instance resembles a particular landmark.

Just like the polynomial features method, the similarity features method can be useful but it may be computationally expensive to compute all the additional features. Kernel trick makes it possible to obtain similar result as if you have added many similarity features.

In [ ]:
```
rbf_kernel_svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
])

rbf_kernel_svm_clf.fit(X, y)
```

Out[ ]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('svm_clf', SVC(C=0.001, gamma=5))])
```

- A big γ makes the bell-shaped curve narrower. As a result, each instance's range of influence is smaller. The decision boundary ends up being more irregular.
- A small γ makes the bell-shaped curve wider, hence instances have a larger range of influence and the decision boundary ends up smoother.

γ acts like a regularization hyperparameter. If model is overfitting, reduce it and if model is underfitting, increase it.

3. SVM Regression

- **SVM Regression**: tries to fit as many instances as possible on the street while limiting margin violations. The width of the street is controlled by hyperparameter ϵ
- **SVM Classificaiton**: tries to fit the largest possible street between two classes while limiting margin violations

In [ ]:
```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

Out[ ]:
```
LinearSVR(epsilon=1.5)
```

In [ ]:
```
from sklearn.svm import SVR

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

Out[ ]:
```
SVR(C=100, degree=2, kernel='poly')
```

LinearSVR class scales linearly with the size of the training set, while SVR class gets much too slow when the training set grows large.

Exercises

1. What is the fundamental idea behind Support Vector Machines?

You fit the widest possible "street" between the classes. In other words, the goal is to have the largest possible margin between the decision boundary that separates the two classes and the training instances. When performing soft margin classification, the SVM searches for a compromise between perfectly separating the two classes and having the widest possible street. Another key idea is to use kernels when training on nonlinear datasets.

2. What is a support vector?

After training an SVM, the support vector is any instance locatied on the "street" including its border. The decision boundary is entirely determined by the support vectors. Any instance is not a support vector has no influence. We could remove them, add more instances or move them around and as long as they stay off the street, they won't affect the decision boundary. Computing the predictions only involves the support vectors and not the whole training set.

3. Why is it important to scale the inputs when using SVMs?

SVMs try to fit the largest possible "street" between classes, so if the training set is is not scaled, the SVM will tend to neglect small features.

4. Can an SVM classifier output a confidence score when it classifies an isntance? What about a probability?

An SVM classifier can output the distance between the test instance and the decision boundary and you can use this as a confidence score. However, this score cannot be directly converted into an estimation of the class probability. If we set probability=True when creating an SVM instance, after training it will calibrate the probabilties using Logistic Regression on the SVM's scores.

6. Say you've trained an SVM classifier with an RBF kernel, but it seems to underfit the training set. Should you increase or decrease γ? What about C?

If underfitting, there might be too much regularization. To decrease it, we can increase gamma or C or both