# CME 216, ME 343 - Winter 2020

Eric Darve, ICME

Stanford University

# Training and validation sets

This leads us to the general question of how we should pick $C$.

This is a problem that we will explore again in the future for other methods.

In machine learning, errors may arise from:

- Error in the data; noise in the data may prevent us from finding an exact answer.
- The model may be overly sensitive to small changes in the data (e.g., large conditioning) and needs to be "stabilized."

Because of this, it is common to use a regularization strategy.

In our case, this was done by varying $C$.

A small $C$ corresponds to a lot of regularization.

This leads to a small vector $w$ and if $C$ is too small, the vector $w$ and scalar $b$ may become significantly wrong.

A large $C$ corresponds to minimal regularization.

In that case, we assume the data is accurate and look for a hyperplane that optimally separates the data (e.g., the hyperplane maximizes the distance to all points).

A typical strategy consists of the following.

We define two sets of points, called the *training* set and the *validation* set.

- Training set: this set is used to fit the model. In our case, this is used to calculate $(w, b, \xi)$.
- Validation set: the set is used to tune some of the model parameters, in our case $C$. It is usually used to control over- or under-fitting.

The optimization may be based on an iterative loop where we perform in turn

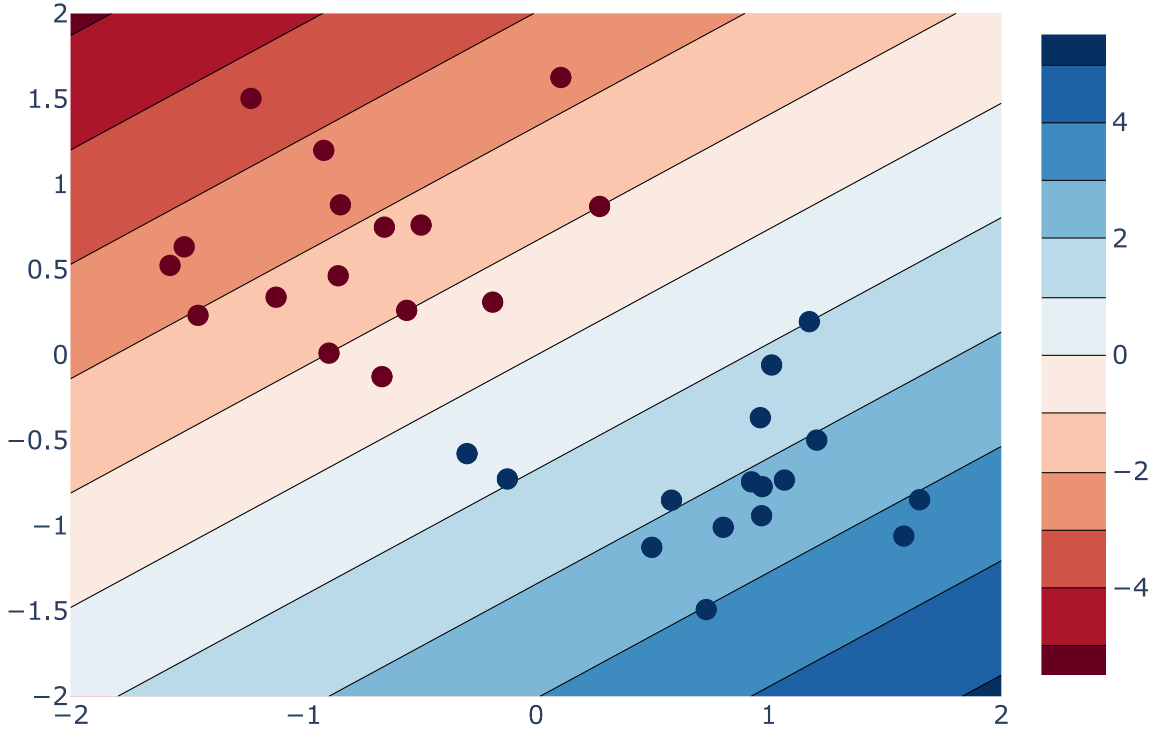
- $(w, b, \xi)$: optimized based on training set
- parameter $C$: optimized based on validation set

Let's give an example to illustrate how this may work in practice.

We consider our previous test case where the input data is randomly perturbed.

```
# randomly perturb the points X
X0 = 0.5 * np.random.randn(16, 2) + 0.8 * np.array([-1., 1.])
X1 = 0.5 * np.random.randn(16, 2) + 0.8 * np.array([1., -1.])
X = np.vstack((X0, X1))
for i in range(0,X.shape[0]):
X[i,:] = X[i,:] + np.random.randn(1, 2) / 6
```

## Decision function



scikitlearn provides a few functionalities that can be used to simplify the process.

Let's start by splitting the input data into a training and validation set.

```python
from sklearn.model_selection import train_test_split
X_train, X_valid, y_train, y_valid=train_test_split(X, y, test_size=0.4)
```

`test_size` is the fraction of the input data that will be used for the validation set.

The word `test` is used because this function is usually used to split the data into a training set and a test set.

But, for our application, we will use the same function to split the data into a training set and a validation set to control over-fitting.

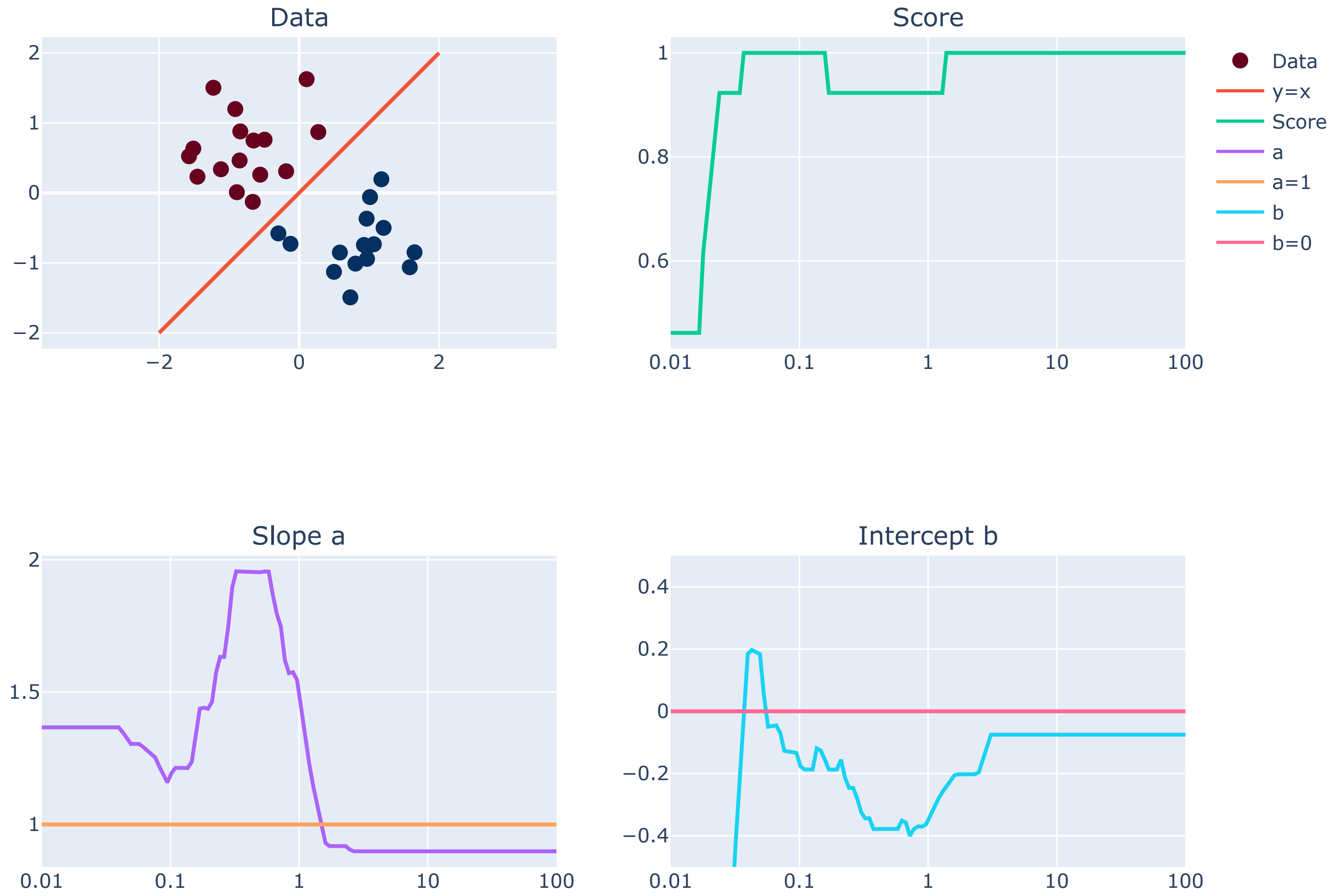
We can verify that the sets have the correct sizes (60% and 40%):

```
print('Size of X_train: ',X_train.shape[0])
print(' Size of X_test: ',X_valid.shape[0])
Size of X_train: 19
Size of X_valid: 13
```

The `clf.score()` function can be used to evaluate the accuracy of our SVM prediction using the test set.

The goal is then to find the value of $C$ that gives us the highest score.

We obtain the following results.


Data
Score
Slope a
Intercept b
● Data
y=x
Score
a
a=1
b
b=0
2
1
0
−1
−2
−2
0
2
1
0.8
0.6
0.01
0.1
1
10
100
2
1.5
1
0.4
0.2
0
−0.2
−0.4

$a$ is the slope of the predicted decision line.

The exact solution is $y = x$ and so $a = 1$ is the exact solution.

$b$ is the predicted intercept. The exact solution is $b = 0$.

We see that values of $C$ below 1 lead to under-fitting (too much regularization).

Larger values, with $C > 5$, give consistently good results.

## Cross-validation

We have seen a simple way to decompose the set into a training set and validation set.

However, this may lead to some issues.

In particular, if we make a unique choice of training and validation set it is quite possible that our prediction becomes biased by our choice of validation set.

Indeed, only the validation set is used to estimate how good $C$ is and whether it should be adjusted.

So to overcome this, it is preferable to optimize our model for many training sets and evaluate its performance on many validation sets.

But how can this be done? This would require a tremendous amount of data.

In practice, the method of cross-validation is used.

The dataset at our disposal is split into $K$ groups.

Then we run a series of experiments in which we use $K - 1$ groups of data for optimizing the model, and the remaining group for validation (i.e., to score our model).

By "reusing" our dataset in this fashion, we are able to generate many scores with limited data.

These scores can then be averaged to estimate how good $C$ is.

Based on this result, $C$ can be optimized.

The advantage of this approach is that we minimize a possible bias due to our selection of training and validation sets, without requiring a tremendous amount of data.

For more information on this, please see scikit-learn cross-validation.

There are many variants on this idea.

The cross-validation section in scikit-learn and the wikipedia page on training and validation sets introduce as well the concept of *test set.*

Here is a summary of what each one of these sets is used for.

- Training set: this set is used to optimize the model, in our case $w$ and $b$.
- Validation set (also called development set) is used to optimize hyper-parameters in the method, for example the parameter $C$ in our example. This can be used to control overfitting and for other optimization of parameters that is not part of step 1 with the training set.

- Once the model has been computed and all parameters have been optimized based on the available data, we use a yet-unseen dataset, the test set, to evaluate the accuracy of our model.

By definition, the test set is applied to the final model.

No further changes are made to the model during that stage.

See scikit-learn cross_validation and training, validation, and test sets.