ML 2016/17
Exercise 3: SVM

14/11/16

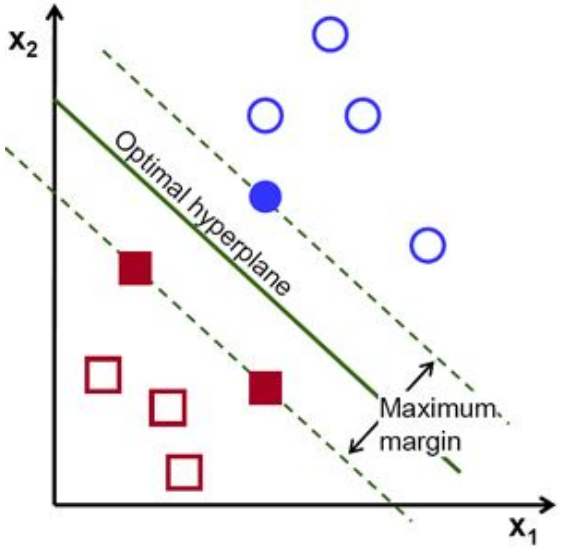
General information

- The assignments are not graded on a scale: it's simply pass/no pass
  - If one homework is not sufficient you can simply redo it
- All assignments must be delivered one month before **you** take the exam
- Submission through email: send to fabiom.carlucci@dis.uniroma1.it
- Questions can be written to same email address.
- Office hours to meet in person: **Wednesday** at B004 (Via Ariosto, the door in front of library), 10AM-12PM.
- Python recommended: https://www.continuum.io/downloads
- *There is no need to replicate exactly the images I show!*

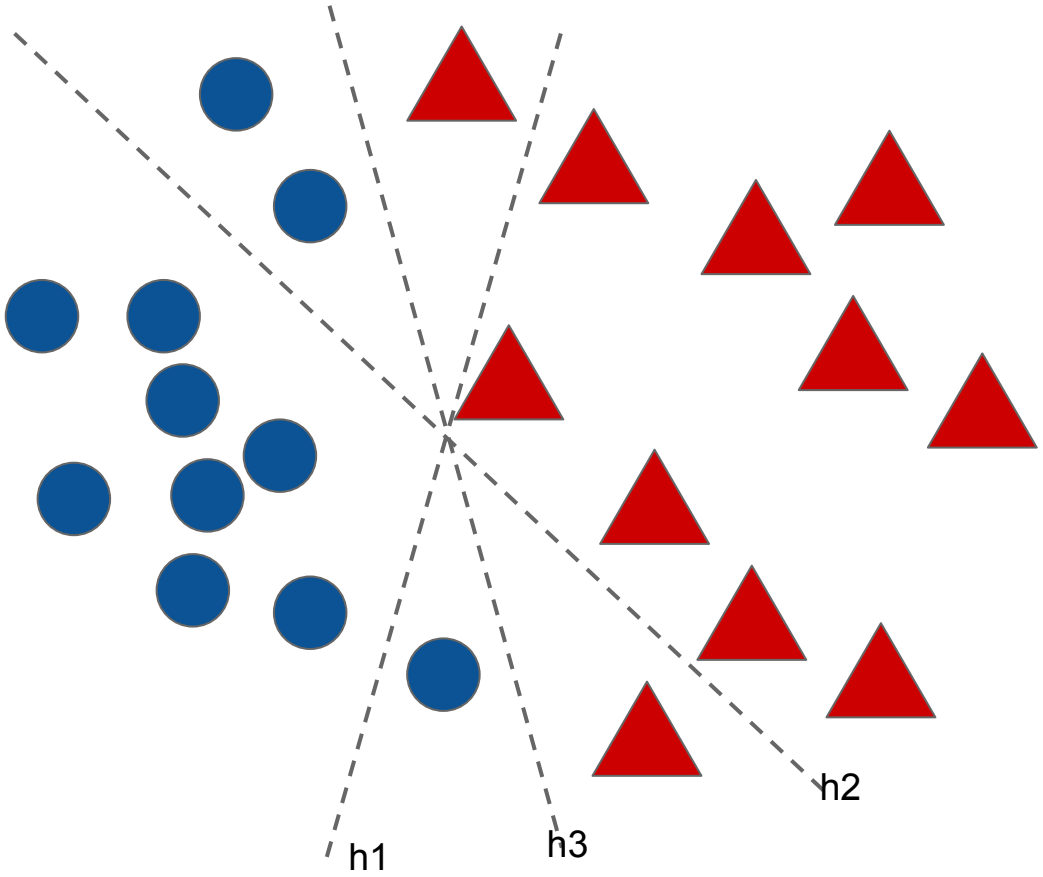
HW3: SVM

- Familiarize with the basic concepts and options of SVMs
- Validation, cross validation and grid search
- Multi class SVM

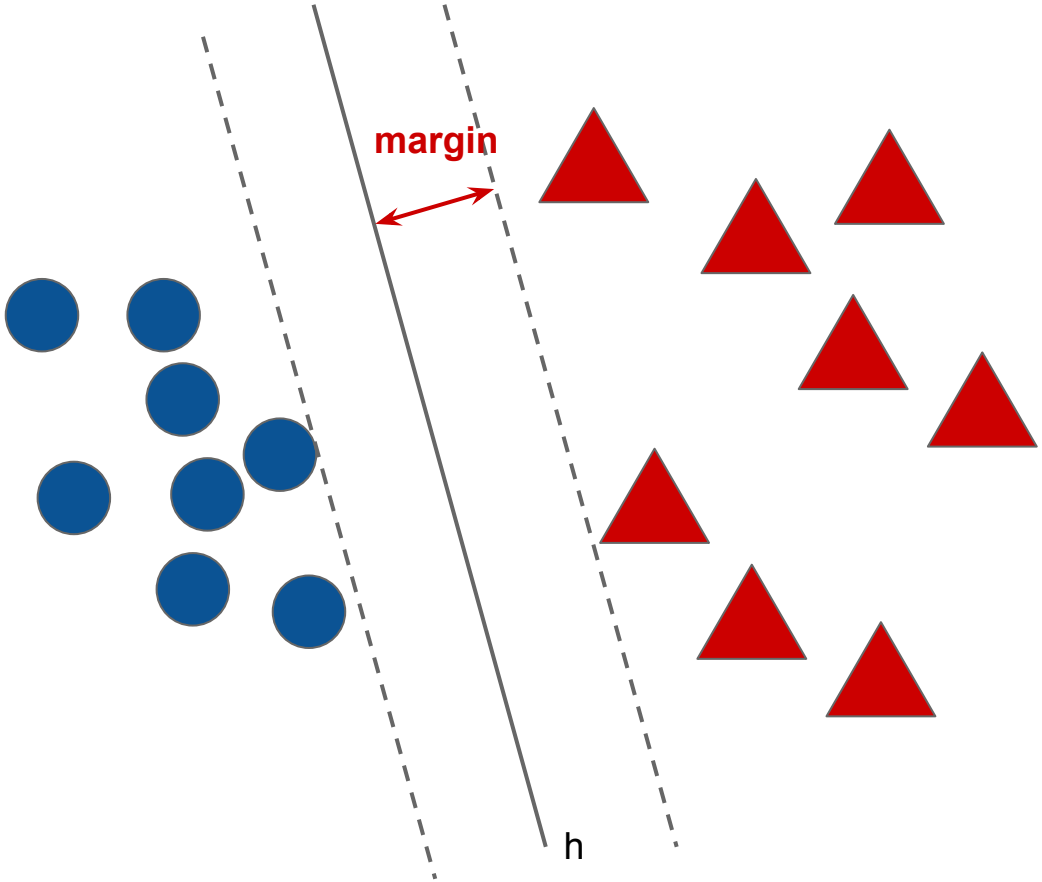
Once you complete the experience send the report to fabiom.carlucci@dis.uniroma1.it with subject **“[ML1617] SVM report”**



How does classification work in SVM?



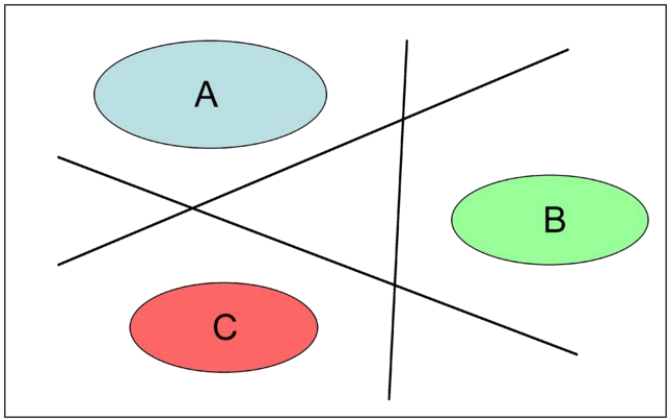
How does classification work in SVM?



How to deal with multiple classes?

The SVM classifier is defined as a binary classifier. How to deal with multiple classes?

- One-vs-all
- One-vs-one



In practice the SVM libraries will handle this for you, just keep in mind this is what is happening behind the scenes

What hyper parameters for SVM?

Linear: **C = 1/λ**

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w}\cdot\vec{x_i} - b)\right)\right] + \lambda\|\vec{w}\|^2,$$

Polygon: **C, degree** and **coef0**

RBF: **C** and **gamma**

- other kernels are available

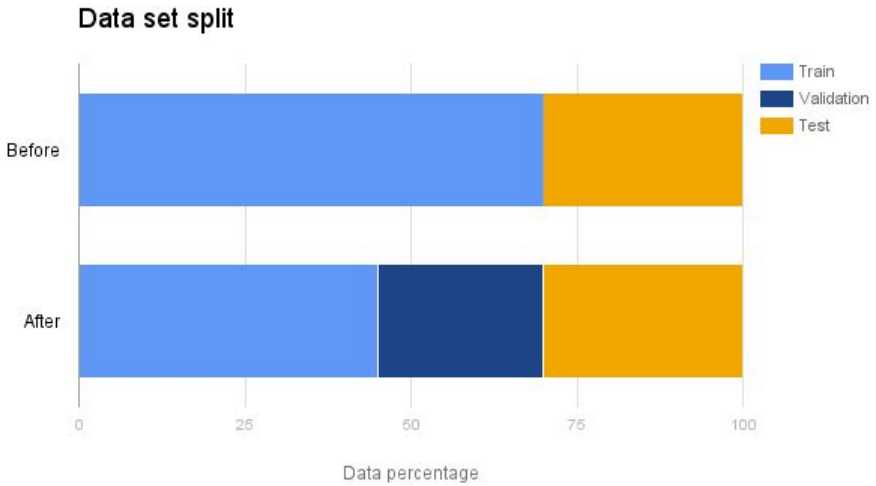
Check the docs:

- http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- http://scikit-learn.org/stable/modules/svm.html#svm-kernels

How to choose the parameters?

You cannot choose the optimal values based on your test set; in real problems you won't have access to it!

The proper way to evaluate a method is to split the data into 3 parts



How to choose the parameters?

You choose some parameters and **train** your model on the **training set**.

You then evaluate your performances on the **validation set**.

Once you find the parameters which work best on the **validation set**, you apply the **same** model on the **test set**. This is the correct estimate of the accuracy you will get on unseen data.

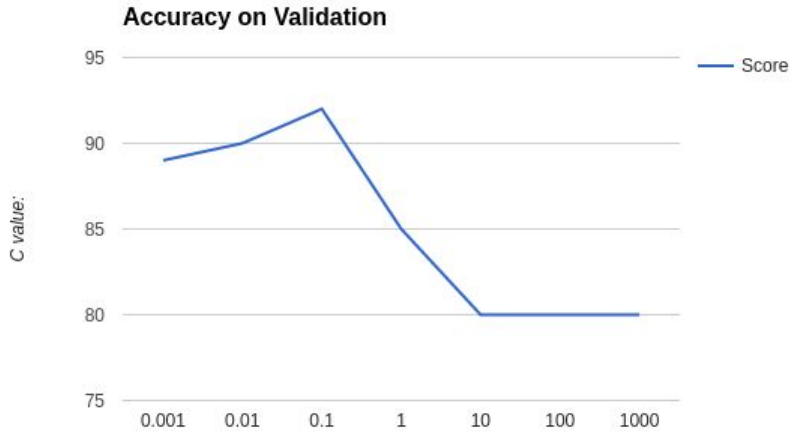
Changing the model based on performance on the test set is cheating!

Example - choosing C

Linear SVM -> we must optimize **C**

We select an appropriate range

C = [10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}]



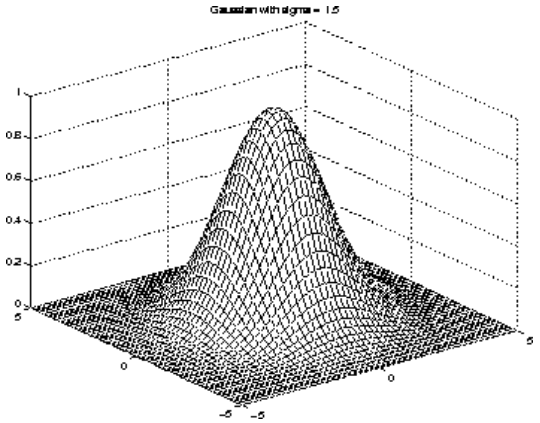
C value:	10^{-3}	10^{-2}	**10^{-1}**	10^{0}	10^{1}	10^{2}	10^{3}
Score on Validation	89%	90%	**92%**	85%	80%	80%	80%

Example - grid search

What if there are two (or more) parameters to optimize?

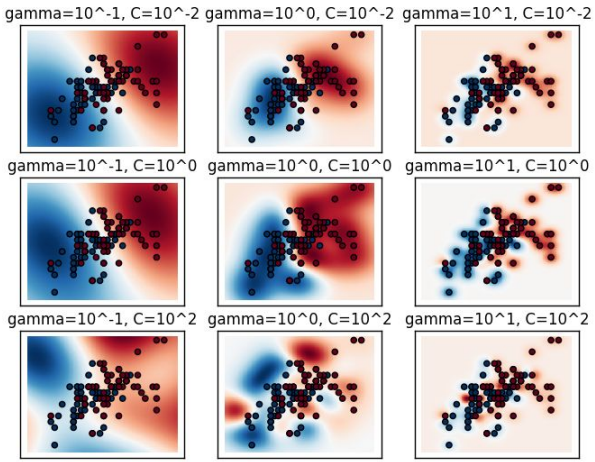
Grid search: try all possible combinations of these parameters.

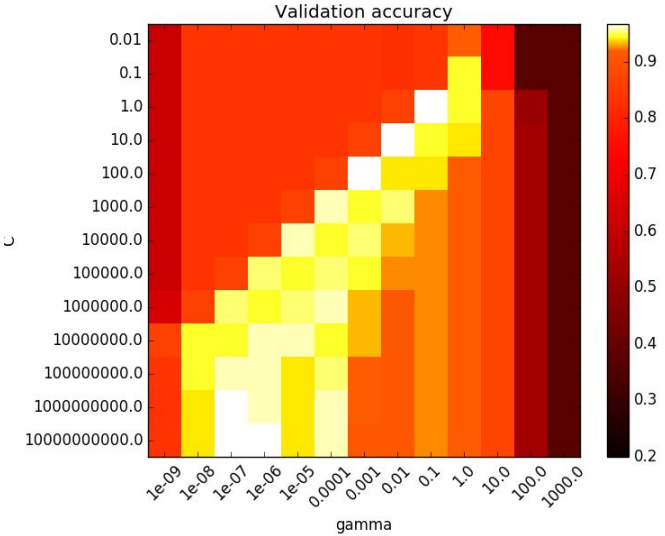
In the **RBF** case we have:
C and **gamma**



Gamma \ C	10^{-2}	10^{-1}	**10^{0}**	10^{1}
10^{-9}	66%	75%	60%	60%
10^{-7}	48%	67%	78%	79%
10^{-5}	80%	82%	**90%**	85%
10^{-3}	45%	66%	78%	74%

Grid search


gamma=10^-1, C=10^-2
gamma=10^0, C=10^-2
gamma=10^1, C=10^-2
gamma=10^-1, C=10^0
gamma=10^0, C=10^0
gamma=10^1, C=10^0
gamma=10^-1, C=10^2
gamma=10^0, C=10^2
gamma=10^1, C=10^2


Validation accuracy
C
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
100000.0
1000000.0
10000000.0
100000000.0
1000000000.0
10000000000.0
gamma
1e-09
1e-08
1e-07
1e-06
1e-05
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

K-Fold Cross Validation

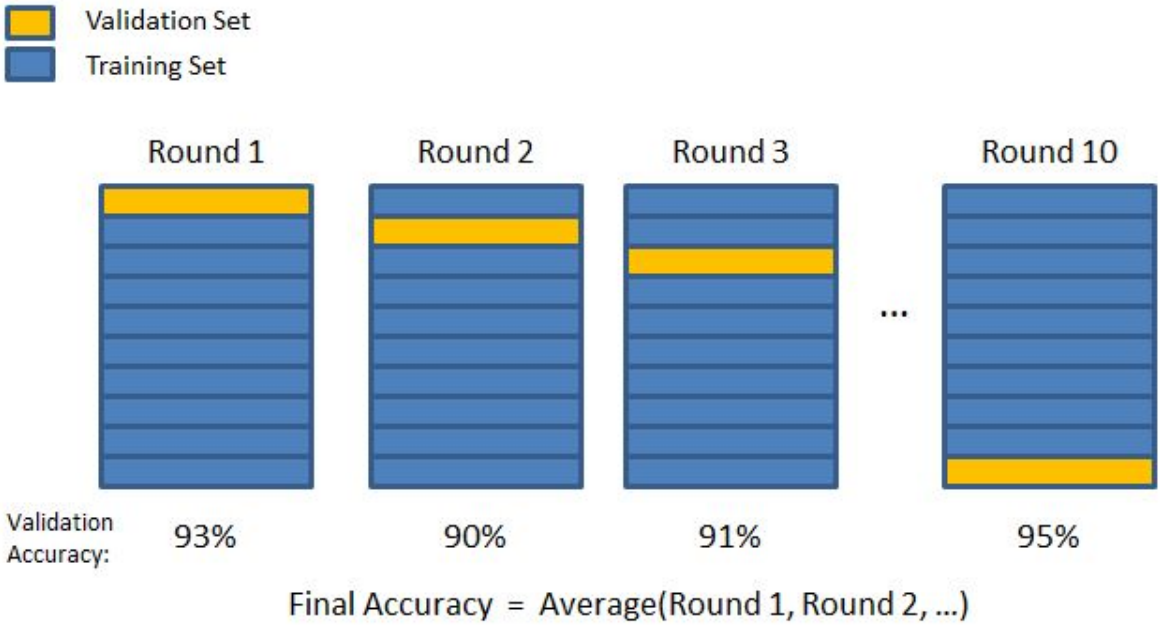
The validation set is a subset of the training data and is data we can no longer use to train our model.

Too small of a validation set is not very informative, too big and you lose too much training data.

Where the training set is small it’s best to use K-fold cross validation.

K-Fold Cross Validation

Divide training data into **k** equal parts and do **k** rounds of validation. Then average the accuracy



K-Fold in practice

```
>>> import numpy as np
>>> X_folds = np.array_split(X_digits, 3)
>>> y_folds = np.array_split(y_digits, 3)
>>> scores = list()
>>> for k in range(3):
...     # We use 'list' to copy, in order to 'pop' later on
...     X_train = list(X_folds)
...     X_test  = X_train.pop(k)
...     X_train = np.concatenate(X_train)
...     y_train = list(y_folds)
...     y_test  = y_train.pop(k)
...     y_train = np.concatenate(y_train)
...     scores.append(svc.fit(X_train, y_train).score(X_test, y_test))
>>> print(scores)
[0.93489148580968284, 0.95659432387312182, 0.93989983305509184]
```

For more information, check

http://scikit-learn.org/stable/tutorial/statistical_inference/model_selection.html

What to do ⅓ - Linear SVM

1. Load ***Iris*** dataset
2. Simply select the first two dimensions (let's skip PCA this time)
3. Randomly split data into train, validation and test sets in proportion 5:2:3
4. For **C** from **10^{-3} to 10^{3}**: *(multiplying at each step by 10)*
    a. Train a **linear** SVM on the training set.
    b. Plot the data and the decision boundaries
    c. Evaluate the method on the validation set
5. Plot a graph showing how the accuracy on the validation set varies when changing **C**
6. How do the boundaries change? Why?
7. Use the best value of **C** and evaluate the model on the **test set**. How well does it go?

What to do ⅔ - RBF Kernel

8. Repeat point 4. (train, plot, etc..), but this time use an RBF kernel
9. Evaluate the best **C** on the **test set**.
10. Are there any differences compared to the linear kernel? How are the boundaries different?
11. Perform a grid search of the best parameters for an RBF kernel: we will now tune both ***gamma*** and **C** at the same time. Select an appropriate range for both parameters. Train the model and score it on the validation set.
12. Show the table showing how these parameters score on the validation set.
13. Evaluate the best parameters on the test set. Plot the decision boundaries.

What to do 3/3 - K-Fold

14. Merge the training and validation split. You should now have 70% training and 30% test data.
15. Repeat the grid search for ***gamma*** and ***C*** but this time perform 5-fold validation.
16. Evaluate the parameters on the test set. Is the final score different? Why?

A little help in pseudo code:

```
scores = np.empty((len(gamma_vals), len(C_vals), len(k_folds)))
for i, gamma in enumerate(gamma_vals):
    for j, C in enumerate(C_vals):
        for k, fold in enumerate(k_folds):
            model = SVM(gamma, C)
            model.fit(fold.train_data, fold.train_labels)
            scores[i,j,k] = model.score(fold.validation_data, fold.validation_labels)
```

Your turn now! Questions?