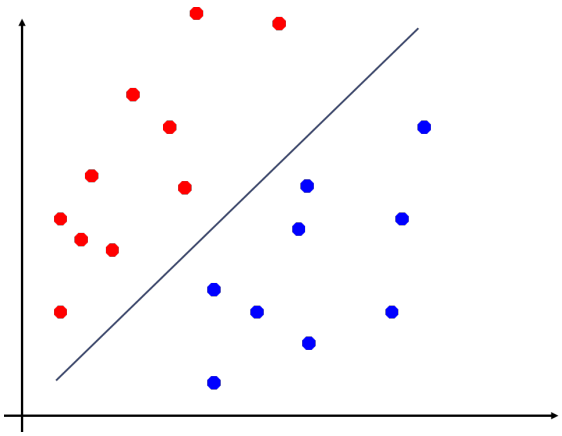
Beginner-friendly ML school:

Support Vector Machines and Hyperparameter fitting

Viviana Acquaviva
(CUNY)
vacquaviva@citytech.cuny.edu

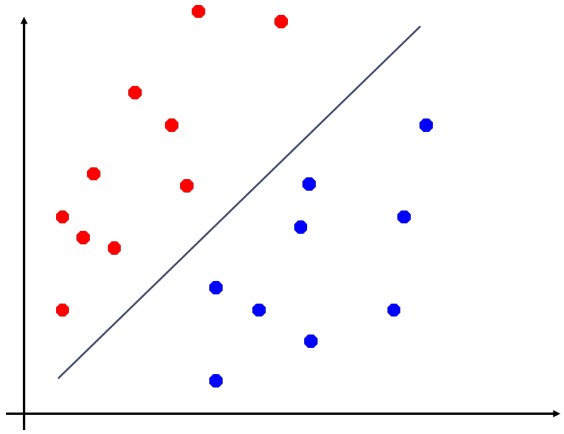
Support Vector Machines (SVM)



(was) one of the most popular classification algorithms.

Pros: Powerful, accurate Cons: Slow

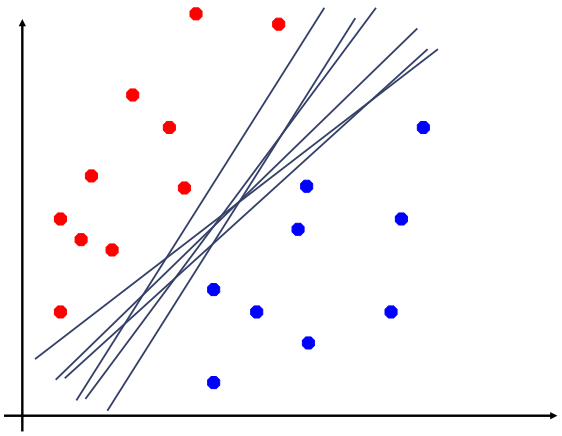
Support Vector Machines (SVM)



How can we split the two classes?

Linear SVMs

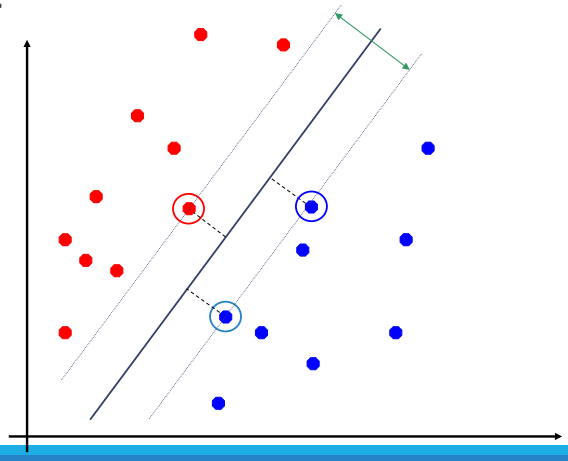
Which of the linear separators is optimal?



Idea 1. If the data are separable, find the decision boundary that maximizes separation between the two classes

Examples closest to the hyperplane are ***support vectors***.

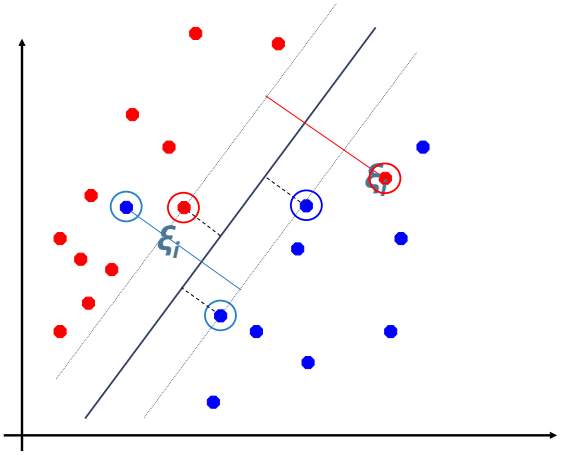
Margin of the separator is the distance between support vectors of opposite classes.



Idea 2. Add slack variables that allow for misclassifications

What if the training set is not linearly separable?

Slack variables can be added to allow misclassification of difficult or noisy examples close to the boundary, while keeping separation large **(resulting in a "soft" margin).**

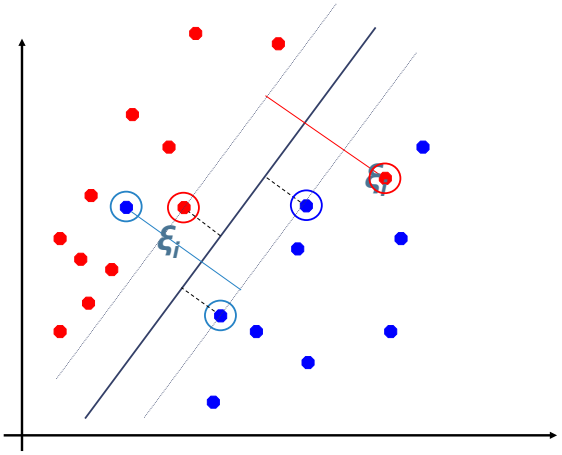


Idea 2. Add slack variables ξ_i that allow for misclassifications

What if the training set is not linearly separable?

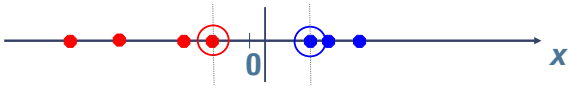
The problem becomes a minimization of

1/(width of margin) + C $\sum_i \xi_i$



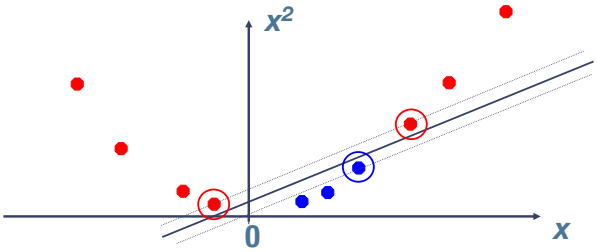
Non-linear SVMs

Datasets that are linearly separable, even with some noise, work out great:



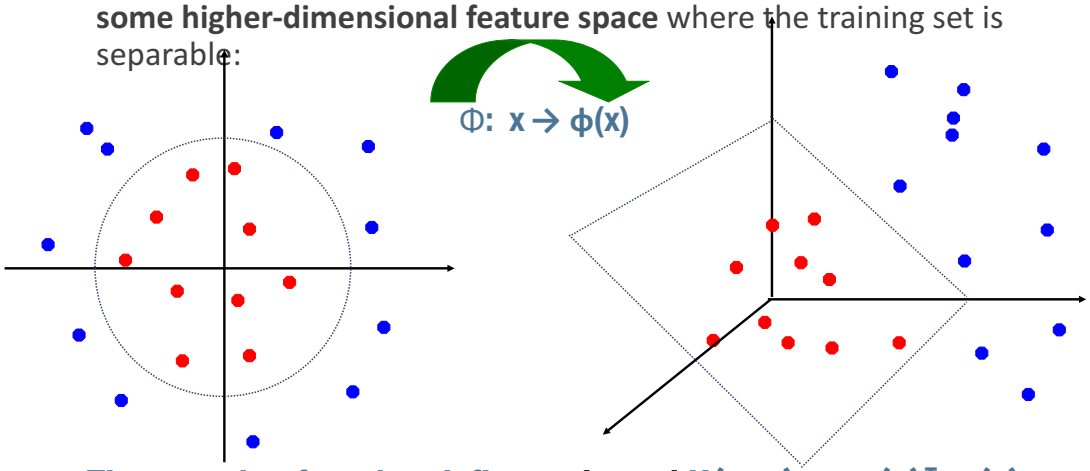
But what are we going to do if the dataset cannot be separated with a line?

How about… mapping data to a higher-dimensional space:



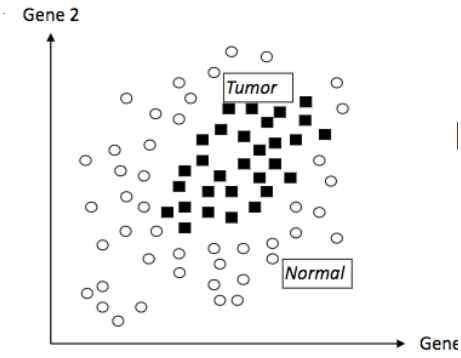
Non-linear SVMs: Feature spaces (kernel trick)

General idea: the original feature space **can always be mapped to some higher-dimensional feature space** where the training set is separable:



The mapping function defines a kernel $K(x, z) = \phi(x)^T \phi(z)$.
Often linear, polynomial, or Gaussian.

Example: Gaussian (RBF) Kernel



Data is not linearly separable in the input space

Magic: You don't need to specify the mapping function, only the kernel, because it is possible to re-write the decision function as a function of $K(x, x_j)$ where x_j are the support vectors and that's O(n) to boot

In fact: the math of SVM is beautiful
(this is a tiny preview of how I understand it)

Given boundary equation w x + b = 0:

- rescale w and b so that the edges of margin have equations w x + b = -1; w x + b = 1

- for all correctly classified examples, y * (wx + b) > 0 and the decision function is g(w x + b) > 1 or g(w x + b) < -1

- maximize 1/|| w2 || with constraint y * (wx + b) > 0 an all training data (Quadratic Programming)

- To solve efficiently: turn this constrained maximization problem into a Lagrange multiplier formulation (primal/dual), allows one to write decision function as a function of inner product of example x with training examples xj

- Non separable case: add slack variables (for which g(w x + b) < -1 does not hold) and maximize 1/|| w2 || + C ∑i ξi

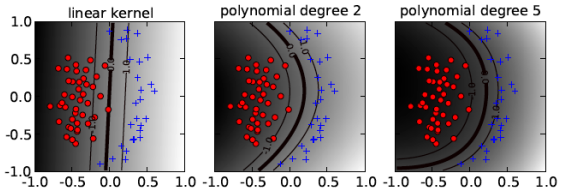
- To solve: find new mapping x -> phi(x) to higher dimensional space where instances are more separated

- Kernel trick (Mercer theorem): it can be shown that if K is semipositive definite, inner product of phi(xi) phi (xj) is prop to K(xi, xj) and its complexity is O(n). Gives decision function in mapping space without explicit mapping.

But if you really want to see it, see Andrew Ng's notes

http://cs229.stanford.edu/notes/cs229-notes3.pdf

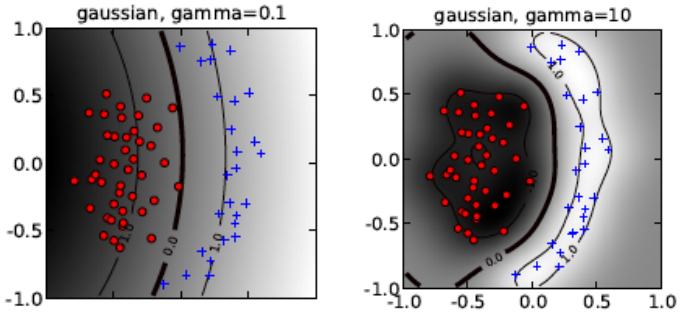
Hyperparameters of SVMs
(hyperparameters = parameters of the model)



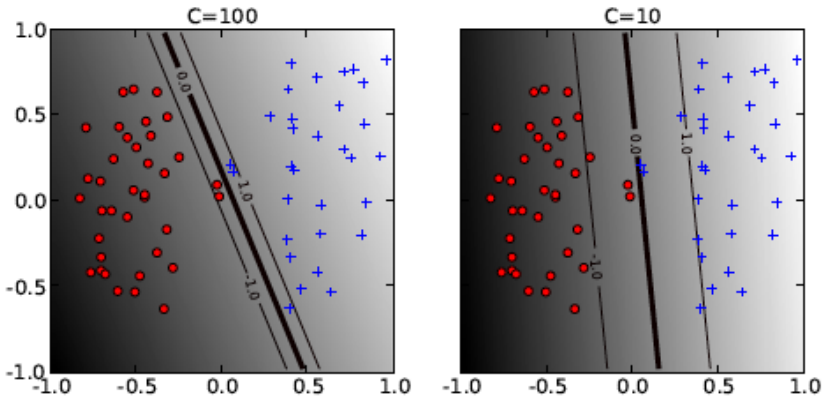
1) Type of kernel (functional form of function describing the boundary).

A linear kernel is a line or plane. A polynomial kernel corresponds to a more curvy boundary.

A common choice is a Gaussian kernel, called rbf (radial basis function).



2) Gamma (shape factor of Gaussian kernel boundary). Small gamma is more linear, high gamma is more wiggly.

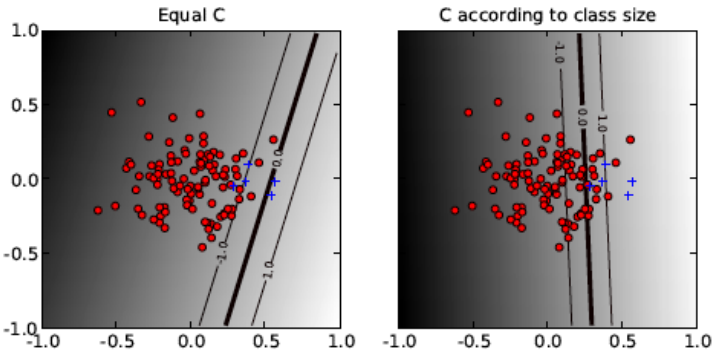


3) C = Penalty function for misclassifications in soft margin SVM.

Large C means that misclassifications near the boundary pay a large penalty, and drives the separation between classes to be small.

Small C means that misclassifications near the boundary pay a small penalty, and creates a larger separation (margin) between classes.

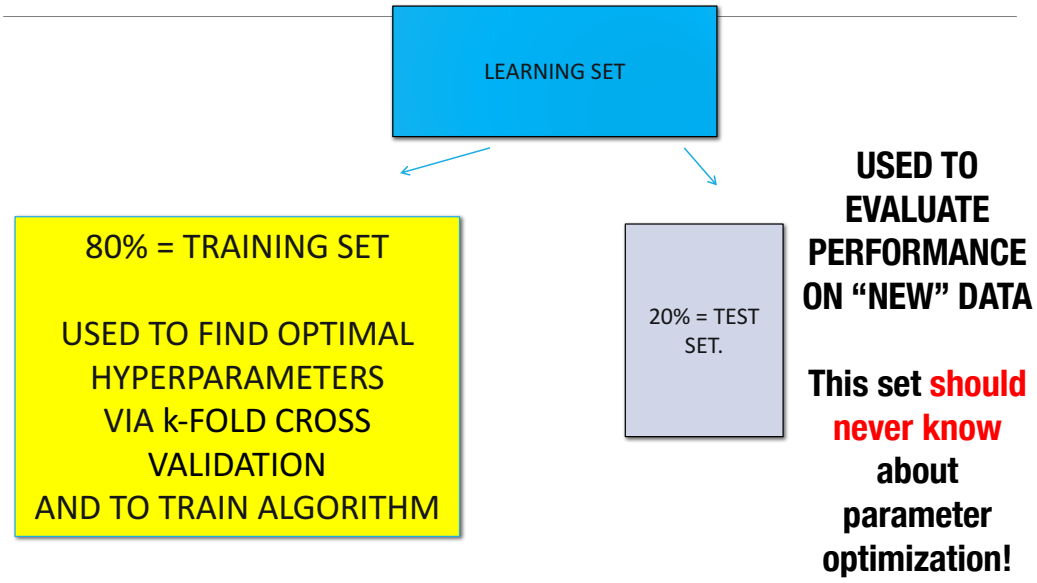
A cool hyperparameter: class weight



4) Class weight

It gives a different penalty for each class, useful for unbalanced data or when we care about one type of error (e.g. false positive) more than the other

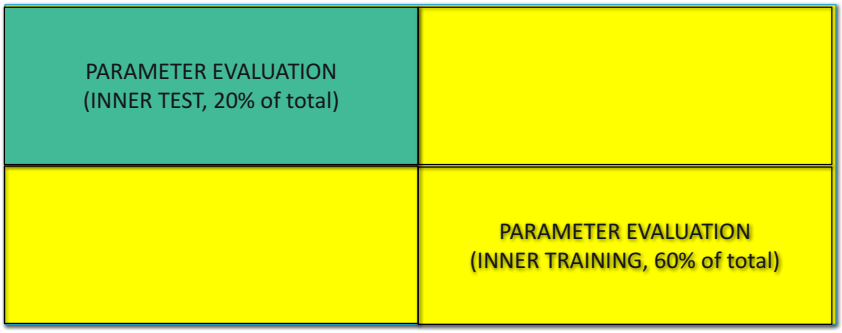
How can we optimize the hyperparameters?



Nested cross validation (nested = two levels, outer/inner)

Outer 5 fold cross validation:
takes 80% for training, 20% for testing
and averages performance 5 times (nothing new so far)

Inner cross validation **(inside yellow outer training set)**
will be used to pick hyper parameters;
we will do this 5 times
(or as many outer CV folds we have).



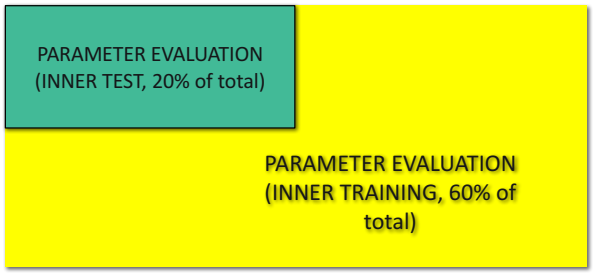
Here we show 4 fold inner cross validation on the outer training set.

Let's see it with one parameter:
C = {1, 10, 100}

1) Set C = 1

2) Do k fold inner cross validation on the outer training test (yellow, 80% of total). Here we divide in 4 folds where 3 are used for inner training, 1 for inner testing.

3) Report average of performance.

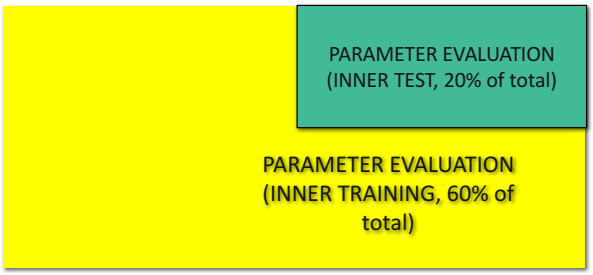


Let's see it with one parameter: C = {1, 10, 100}

1) Set C = 1

2) Do k fold inner cross validation on the outer training test (yellow, 80% of total). Here we divide in 4 folds where 3 are used for inner training, 1 for inner testing.

3) Report average of performance

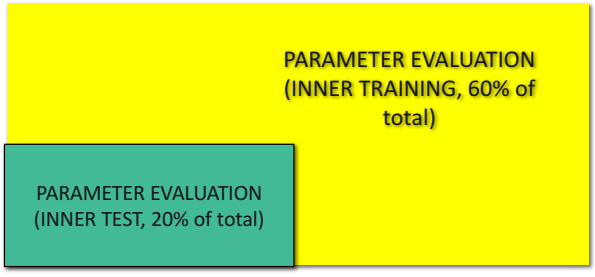


Let's see it with one parameter: C = {1, 10, 100}

1) Set C = 1

2) Do k fold inner cross validation on the outer training test (yellow, 80% of total). Here we divide in 4 folds where 3 are used for inner training, 1 for inner testing.

3) Report average of performance

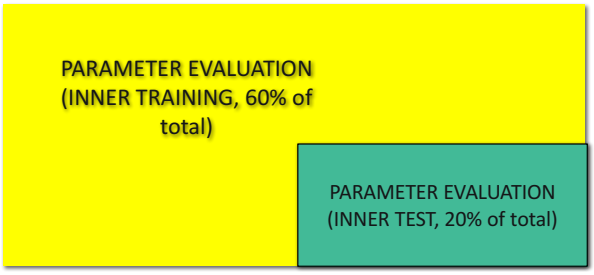


Let's see it with one parameter: C = {1, 10, 100}

1) Set C = 1

2) Do k fold inner cross validation on the outer training test (yellow, 80% of total). Here we divide in 4 folds where 3 are used for inner training, 1 for inner testing.

3) Report average of performance (inner test scores) on the four folds.

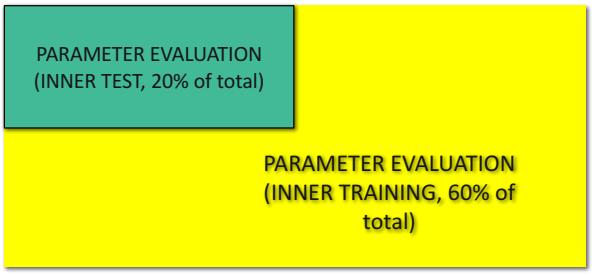


Let's see it with one parameter: C = {1, 10, 100}

1) Set C = 10 and repeat:

2) Do k fold cross validation on the outer training test (yellow, 80% of total). Here we divide in 4 folds where 3 are used for inner training, 1 for inner testing.

3) Report average of performance

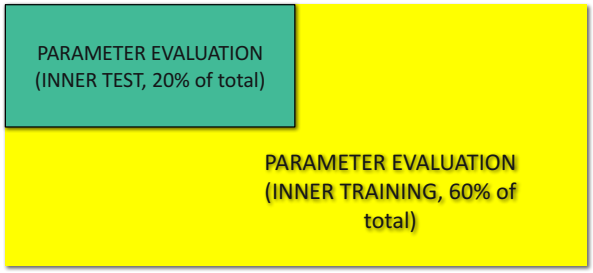


Let's see it with one parameter: C = {1, 10, 100}

1) Set C = 100 and repeat:

2) Do k fold cross validation on the outer training test (yellow, 80% of total). Here we divide in 4 folds where 3 are used for inner training, 1 for inner testing.

3) Report average of performance



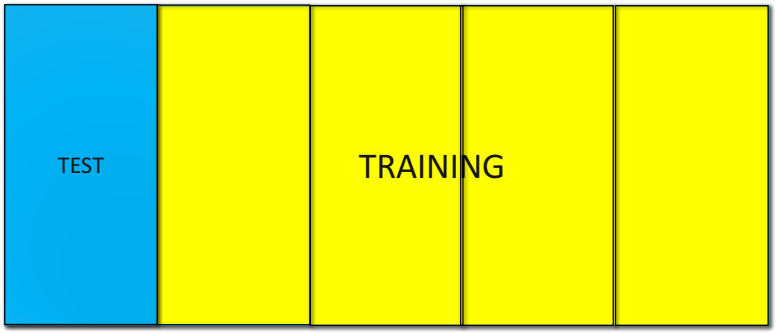
Summary of inner CV results: picking parameters Metric: accuracy

	Fold 1 = test Folds 2, 3, 4 = training	Fold 2 = test Folds 1, 3, 4 = training	Fold 3 = test Folds 1, 2, 4 = training	Fold 4 = test Folds 1, 2, 3 = training	Average accuracy
C = 1	80%	85%	79%	84%	82%
C = 10	83%	87%	78%	77%	81%
C = 100	81%	83%	80%	76%	80%

Summary of inner CV results: picking parameters Metric: accuracy

	Fold 1 = test Folds 2, 3, 4 = training	Fold 2 = test Folds 1, 3, 4 = training	Fold 3 = test Folds 1, 2, 4 = training	Fold 4 = test Folds 1, 2, 3 = training	Average accuracy
C = 1	**80%**	**85%**	**79%**	**84%**	**82%**
C = 10	83%	87%	78%	77%	81%
C = 100	81%	83%	80%	76%	80%

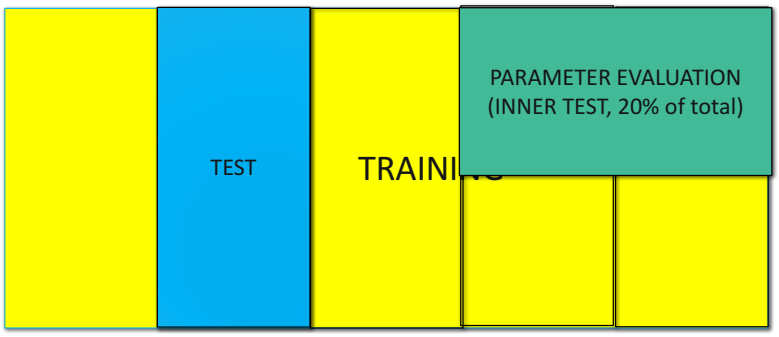
Now I apply the winning model (with C = 1) to the test set of the first outer fold.



This way the "test" set has never seen the training data and has never participated in the hyper-parameter optimization.

I'll save the accuracy score: e.g., 82%.

Now we go back to outer cross validation, pick the second fold as test and repeat the process.



Summary of inner CV results on this second outer fold: picking parameters
Metric: accuracy

	Fold 1 = test Folds 2, 3, 4 = training	Fold 2 = test Folds 1, 3, 4 = training	Fold 3 = test Folds 1, 2, 4 = training	Fold 4 = test Folds 1, 2, 3 = training	Average
C = 1	80%	83%	78%	83%	81%
C = 10	84%	87%	81%	80%	83%
C = 100	81%	83%	80%	76%	80%

Summary of inner CV results on this second outer fold: picking parameters
Metric: accuracy

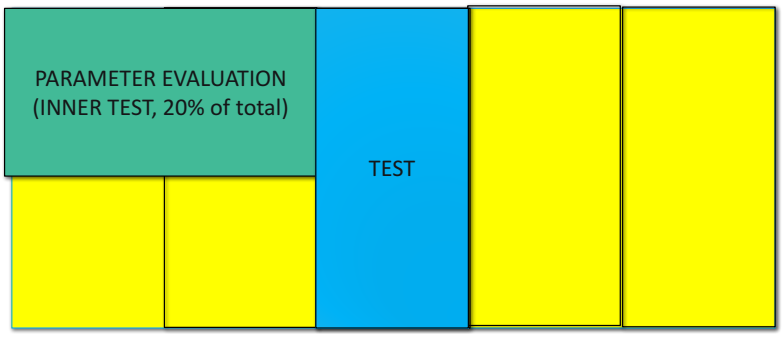
	Fold 1 = test Folds 2, 3, 4 = training	Fold 2 = test Folds 1, 3, 4 = training	Fold 3 = test Folds 1, 2, 4 = training	Fold 4 = test Folds 1, 2, 3 = training	Average
C = 1	80%	83%	78%	83%	81%
C = 10	**84%**	**87%**	**81%**	**80%**	**83%**
C = 100	81%	83%	80%	76%	80%

Now I apply the winning model (with C = 10) to the test set of the second outer fold.

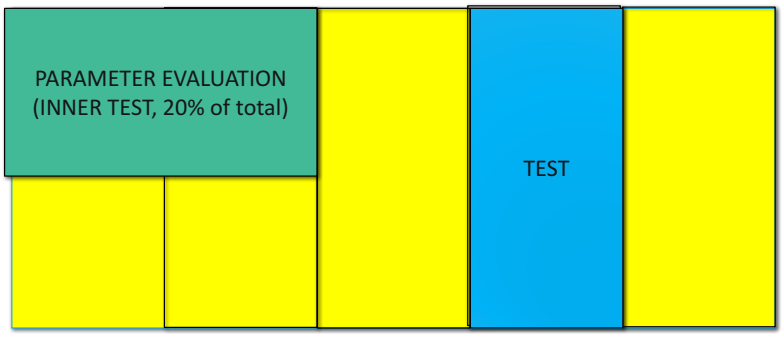
This way the "test" set has never seen the training data and has never participated in the hyper-parameter optimization.

I might get a different accuracy, e.g. 80%. I save this.

I repeat this process of inner CV for the other three folds, so I will have 5 winning models and 5 scores.



I repeat this process of inner CV for the other three folds, so I will have 5 winning models and 5 scores.



I repeat this process of inner CV for the other three folds, so I will have 5 winning models and 5 scores.

At the end I will have the five performances (outer test scores) of the five winning models.

The average of these gives me the average performance of my optimized SVM.

For example, 82, 80, 83, 84, 81 %. This gives me a "quotable generalization performance" (average) of 82% with a standard deviation of 1.6%.

Note that the five winning models might have different hyperparameters. That's ok as long as they are not CRAZY different. They are only used to evaluate the generalization error.

Final bit: If I am building an algorithm for a customer, I now will use all the training data to build a model and only perform one level of cross validation to find ideal hyperparameters. They might be different from what I found in the previous process as well. I will quote as accuracy (or any other metric) the performance found above.

Updated summary of how to build a ML model in view of today's slides

1. **Choose a class of model (aka a machine learning algorithm) by importing the appropriate estimator class from Scikit-Learn. Today: SVM.**
2. **Choose model hyperparameters by instantiating this class with desired values. Alternatively: optimize hyperparameters.**
3. **Arrange data into a features matrix and target vector, if necessary.**
4. **Split the learning set into training/test using k fold cross validation (outer CV), let's say 5.**
5. **Split the training set into inner training/inner validation set using k fold cross validation (inner CV), let's say 4.**
6. **Fit the model in the inner loop by calling the ``fit()`` method for every combination of parameters you want to try out.**
7. **Find combination of parameters with the best average "inner test" scores (averaged over the inner 4 folds).**
8. **Repeat for the outer 5 folds. Report mean scores and standard deviation of winning models. That's the generalization error.**
9. **If performance still not satisfactory, diagnose bias/variance and proceed.**
10. **Build final model using all training data, using k-fold CV to find ideal parameters, and reporting generalization error found above.**

Further reading

Simple to follow, good material about SVMs although doesn't talk about nested cross validation:

https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

Amazing and a little technical:

http://cs229.stanford.edu/notes/cs229-notes3.pdf

Long and has a lot of math but also awesome and great figures:

https://med.nyu.edu/chibi/sites/default/files/chibi/Final.pdf

Other References:

https://www.cs.utexas.edu/~mooney/cs391L/slides/svm.ppt

https://www.ncbi.nlm.nih.gov/pubmed/20221922