COMP 432 Machine Learning

Hyperparameter Search

Computer Science & Software Engineering
Concordia University, Fall 2021

Hyper comes from Greek word for “over” or “above”

Validation & Hyperparameters

- **Cross validation** is a method for *estimating* the test-time performance of a trainable model.
  - *i.e.*, for *fixed* model configuration*, CV* is a scheme to estimate its performance on held-out data
- **Model selection** is *choosing* the model (or model type) expected to perform the best on test data.
- **Hyperparameter search** is about finding the best *version* of a particular model type, and is a form of model selection.
  - Specifically, choices that are normally held fixed during training (*max_depth*, *etc.*) are tuned so as to maximize *estimated test-time* performance.
  - Parameters tuned on *training* data, but hyperparameters tuned on *validation* data --- never on test data!

3. Model selection and evaluation

Read

3.1. Cross-validation: evaluating estimator performance

- 3.1.1. Computing cross-validated metrics
- 3.1.2. Cross validation iterators
- 3.1.3. A note on shuffling
- 3.1.4. Cross validation and model selection

3.2. Tuning the hyper-parameters of an estimator

- 3.2.1. Exhaustive Grid Search
- 3.2.2. Randomized Parameter Optimization
- 3.2.3. Tips for parameter search
- 3.2.4. Alternatives to brute force parameter search

Lab covers this

3.3. Metrics and scoring: quantifying the quality of predictions

- 3.3.1. The `scoring` parameter: defining model evaluation rules
- 3.3.2. Classification metrics
- 3.3.3. Multilabel ranking metrics
- 3.3.4. Regression metrics
- 3.3.5. Clustering metrics
- 3.3.6. Dummy estimators

Important: Good for sanity-checking that your prediction task is not trivial! Avoids embarrassment!

K-Fold Cross Validation

- This is your go-to cross validation method when the data is balanced and independently sampled

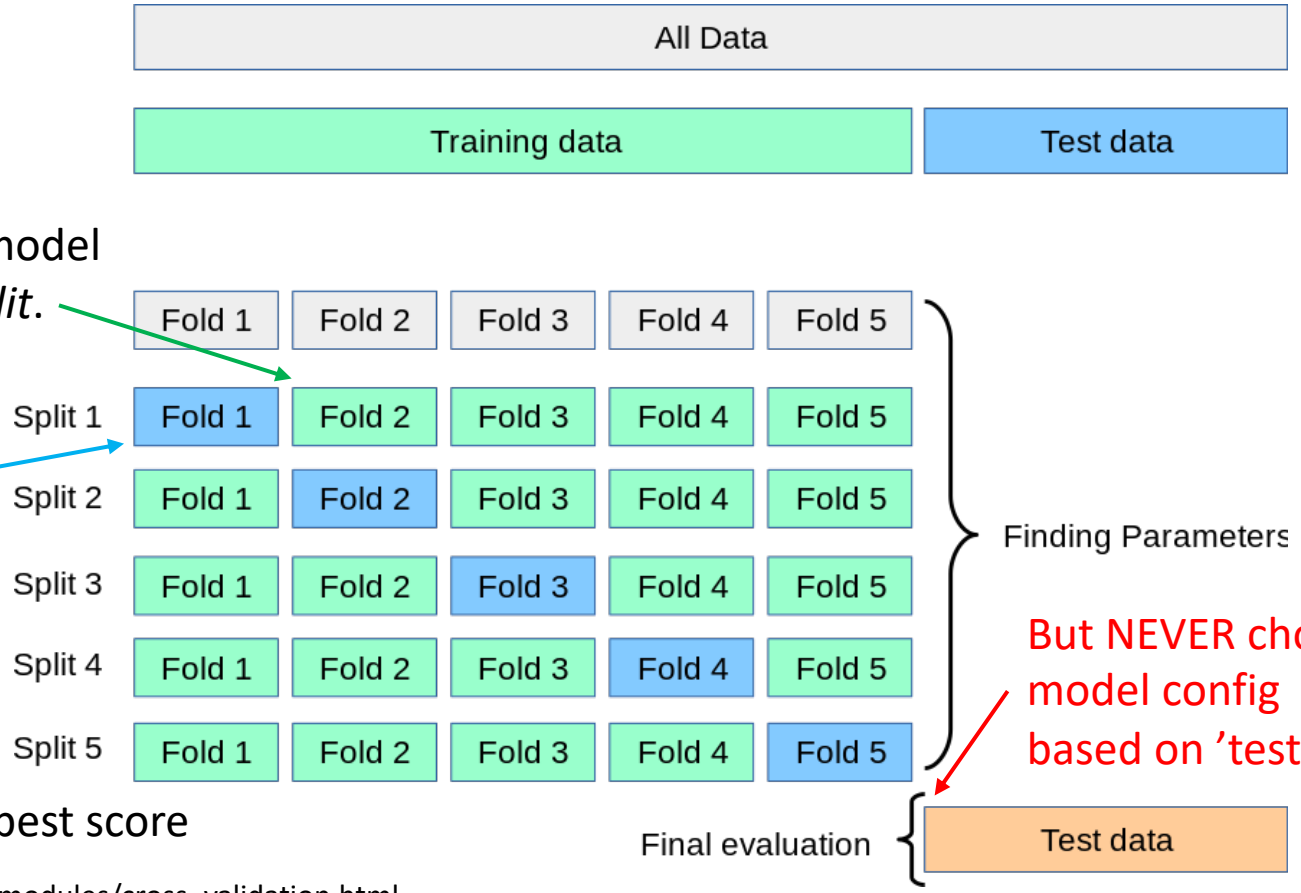
Example of 5-fold CV.

Train 5 instances of a model config, one on each *split*.

Score each instance on its held-out *fold*.

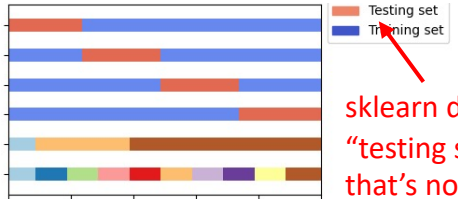
Score the config by average of 5 scores.

Repeat for many configs, pick one with best score



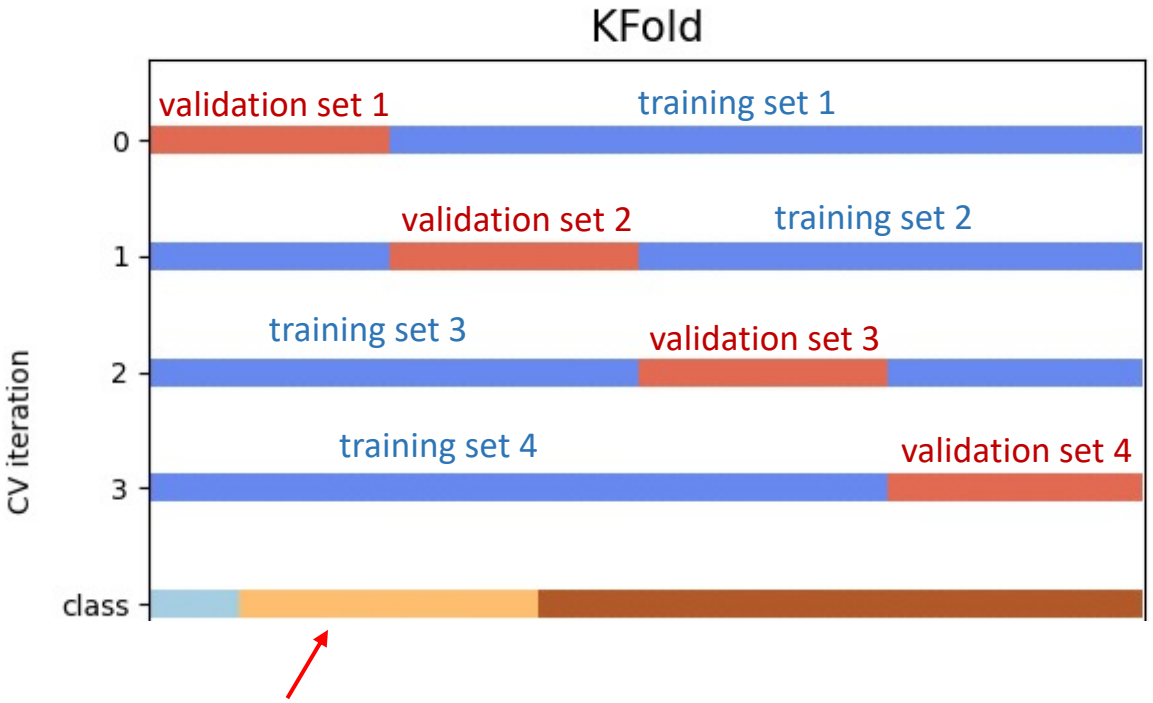
From https://scikit-learn.org/stable/modules/cross_validation.html

K-Fold Cross Validation



sklearn docs say "testing set", but that's not right: it is "validation set"

- This is your go-to cross validation method when the data is expected to be independently sampled



Example of 4-fold CV.

Advantage over doing *K* independent random splits: every data point gets chance to be in one validation set.

class label *not* used

Keep in mind that data set gets shuffled by default, so not really ordered by class like this.

https://scikit-learn.org/stable/modules/cross_validation.html

sklearn.model_selection.KFold

class sklearn.model_selection.**KFold**(*n_splits=5*, *, *shuffle=False*, *random_state=None*) ¶ [source]

K-Folds cross-validator

Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default).

Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Read more in the User Guide.

Parameters:

n_splits : *int, default=5*

Number of folds. Must be at least 2.

Changed in version 0.22: `n_splits` default value changed from 3 to 5.

shuffle : *bool, default=False*

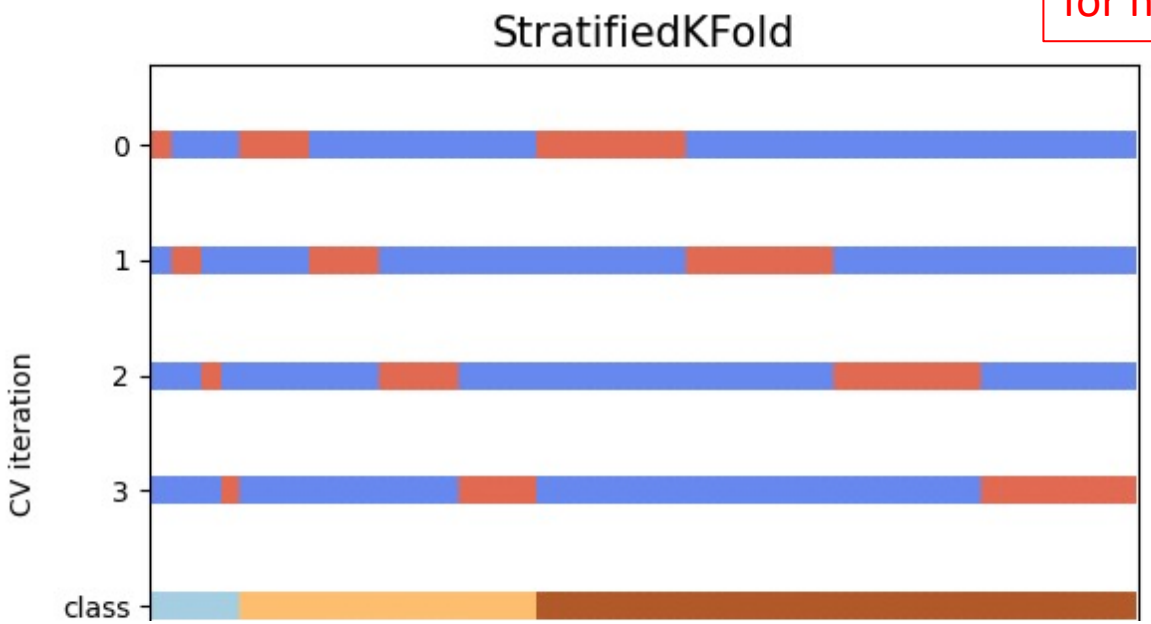
Whether to shuffle the data before splitting into batches. Note that the samples within each split will not be shuffled.

https://scikit-learn.org/stable/modules/cross_validation.html

Stratified *K*-Fold

- Stratified *K*-fold tries to ensure validation set have same class proportions as full data set.
  - Doesn't leave it up to "chance"!

This is sklearn's default splitter for hyperparameter search.

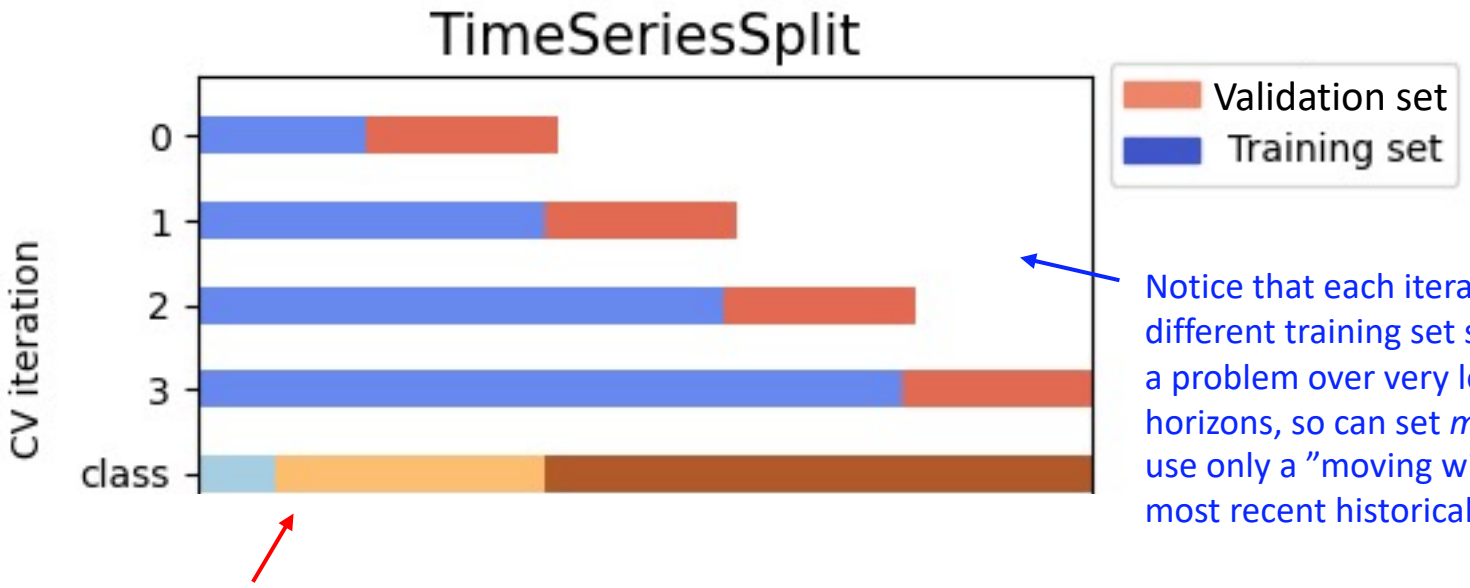


class label *used* to distribute classes *evenly* into each fold

https://scikit-learn.org/stable/modules/cross_validation.html

Time Series Split

- Each training set comprises only historical data, never data from the “future”
  - Estimates ability to extrapolate “forward in time” only
  - Never use “k-fold CV” if what you care about is ability to extrapolate into the future.



Notice that each iteration has different training set size. This can be a problem over very long time horizons, so can set *max_train_size* to use only a ”moving window” of the most recent historical examples.

TimeSeriesSplit relies on data being ordered oldest-to-newest so does NOT shuffle.
(Real time series data would have class labels mixed up through time, so not realistic.)

https://scikit-learn.org/stable/modules/cross_validation.html

sklearn.model_selection.LeaveOneOut

class sklearn.model_selection.LeaveOneOut [source]

Leave-One-Out cross-validator

Provides train/test indices to split data in train/test sets. Each sample is used once as a test set (singleton) while the remaining samples form the training set.

- Suppose you have training set size *N*.
- **Leave-One-Out CV** (LOOCV) means performs *K*-fold CV with *K*=*N*, i.e., every single training example gets a chance to "play the test data" separately, on its own.
- Pro: Good estimate of test-time performance, training set size are all *N*-1, as close as possible to full training set that final model will be trained on.
- Con: Requires training *N* models, very slow in general.

Grid Search & Random Search

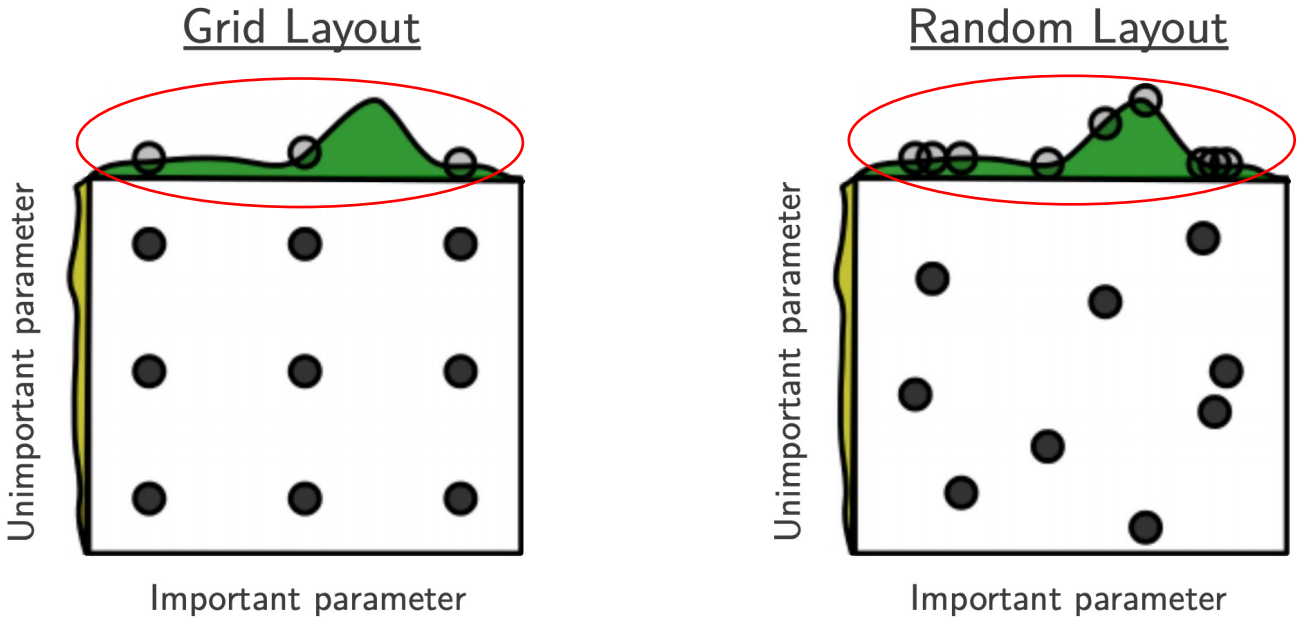


Figure taken from the paper below, which argues that random search is better because it manages to evaluate 'important parameters' more densely.

Random Search for Hyper-Parameter Optimization

James Bergstra JAMES.BERGSTRA@UMONTREAL.CA
Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Département d'Informatique et de recherche opérationnelle
Université de Montréal

sklearn.model_selection.GridSearchCV

class `sklearn.model_selection.` **`GridSearchCV`**(*estimator*, *param_grid*, ***, *scoring=None*, *n_jobs=None*, *iid='deprecated'*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score=nan*, *return_train_score=False*) [source]

Exhaustive search over specified parameter values for an estimator.

Parameters:

estimator : ***estimator object.***

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

param_grid : ***dict or list of dictionaries***

Dictionary with parameters names (`str`) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

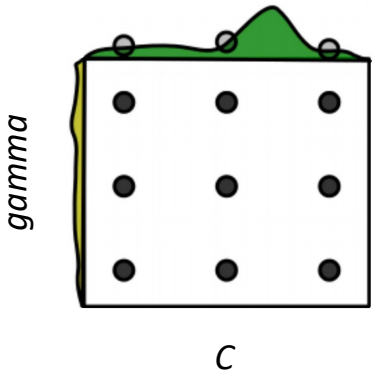
scoring : ***str, callable, list/tuple or dict, default=None***

A single str (see The scoring parameter: defining model evaluation rules) or a callable (see Defining your scoring strategy from metric functions) to evaluate the predictions on the test set.

cv : ***int, cross-validation generator or an iterable, defau***

Determines the cross-validation splitting strategy. Possib
cv are:

- None, to use the default 5-fold cross validation,
- integer, to specify the number of folds in a `(Stratified)KFold`,
- CV splitter,
- An iterable yielding (train, test) splits as arrays of indices.




```
param_grid = {
    'C' : [0.01, 0.1, 1.0],
    'gamma' : [.1, .5, 2.0],
}
```

THIS should be your APPLICATION-DRIVEN success metric, regardless of whether you were able to use as training loss!

Scoring	Function
Classification	
'accuracy'	metrics.accuracy_score
'balanced_accuracy'	metrics.balanced_accuracy_score
'top_k_accuracy'	metrics.top_k_accuracy_score
'average_precision'	metrics.average_precision_score

... many other choices, or custom score

sklearn.model_selection.RandomizedSearchCV

class `sklearn.model_selection.` **RandomizedSearchCV**(*estimator*, *param_distributions*, ***, *n_iter=10*, *scoring=None*, *n_jobs=None*, *iid='deprecated'*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *random_state=None*, *error_score=nan*, *return_train_score=False*) [source]

Randomized search on hyper parameters.

Parameters:

estimator : ***estimator object.***

A object of that type is instantiated for each grid point. This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

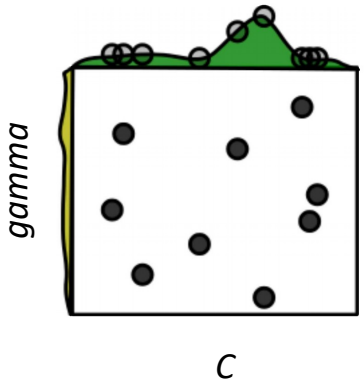
param_distributions : ***dict or list of dicts***

Dictionary with parameters names (`str`) as keys and distributions or lists of parameters to try. Distributions must provide a `rvs` method for sampling (such as those from scipy.stats.distributions). If a list is given, it is sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a parameter is sampled using that dict as above.

n_iter : ***int, default=10***

Number of parameter settings that are sampled. of the solution.

scoring : ***str, callable, list/tuple or dict, default=None***



Example that generates log-spaced samples:

```
param_distributions = {
    'C'     : scipy.stats.reciprocal(0.01, 1.0),
    'gamma' : scipy.stats.reciprocal(0.1, 2.0),
}
```

But you can still pass in lists of values and it'll just randomly sample from them – nice!

```
param_distributions = {
    'C' : [0.01, 0.1, 1.0],
    'gamma' : [.1, .5, 2.0],
}
```

```
X = np.random.rand(100, 2)
y = np.random.randint(0, 2, 100)
```

```
param_distributions = {
    'C': [0.01, 0.1, 1.0],
    'gamma': [0.1, 0.5, 2.],
}
svm = SVC(kernel='rbf', random_state=0)
search = RandomizedSearchCV(svm, param_distributions, cv=3, n_iter=7,
                            verbose=2, random_state=0)
search.fit(X, y);
search.score(X, y)
```

```
Fitting 3 folds for each of 7 candidates, totalling 21 fits
[CV] END .....................................C=1.0, gamma=0.5; total time=   0.0s
[CV] END .....................................C=1.0, gamma=0.5; total time=   0.0s
[CV] END .....................................C=1.0, gamma=0.5; total time=   0.0s
[CV] END ....................................C=0.01, gamma=2.0; total time=   0.0s
[CV] END ....................................C=0.01, gamma=2.0; total time=   0.0s
[CV] END ....................................C=0.01, gamma=2.0; total time=   0.0s
[CV] END ....................................C=0.01, gamma=0.5; total time=   0.0s
[CV] END ....................................C=0.01, gamma=0.5; total t
[CV] END ....................................C=0.01, gamma=0.5; total t
[CV] END .....................................C=0.1, gamma=0.5; total t
[CV] END .....................................C=0.1, gamma=0.5; total t
[CV] END .....................................C=0.1, gamma=0.5; total t
[CV] END .....................................C=1.0, gamma=2.0; total t
[CV] END .....................................C=1.0, gamma=2.0; total time=
[CV] END .....................................C=1.0, gamma=2.0; total time=   0.0s
[CV] END .....................................C=1.0, gamma=0.1; total time=   0.0s
[CV] END .....................................C=1.0, gamma=0.1; total time=   0.0s
[CV] END .....................................C=1.0, gamma=0.1; total time=   0.0s
[CV] END .....................................C=0.1, gamma=0.1; total time=   0.0s
[CV] END .....................................C=0.1, gamma=0.1; total time=   0.0s
[CV] END .....................................C=0.1, gamma=0.1; total time=   0.0s
```

```
0.52
```

On large training jobs, set *n_jobs*=-1 on the search object and it will run these in parallel using as many CPUs as you have available.

Tried 7 of the 9 possible hyperparameter settings.
For each of the 7 settings, 3 models were trained using that setting (1 per split).

```
search.best_params_
```

```
{'gamma': 2.0, 'C': 0.01}
```

This was the best hyperparameter setting with best cross-validation score.

```
search.best_estimator_
```

```
SVC(C=0.01, gamma=2.0, random_state=0)
```

This model was trained on ALL data, using the best hyperparameter settings

PRML Readings

None.

Scikit-learn documentation does a better job of explaining classification metrics and cross-validation schemes.