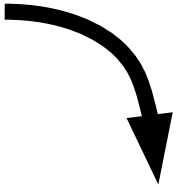
Advanced Scikit-Learn

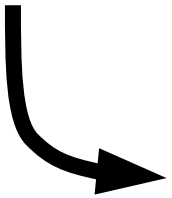
Andreas Mueller (NYU Center for Data Science, scikit-learn)

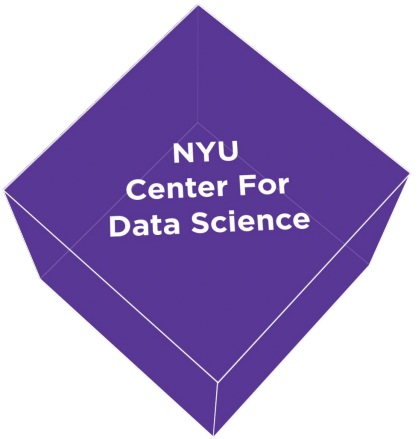
Me


universitätbonn
ais




Amazon
Machine
Learning




NYU
Center For
Data Science

Classification
Regression
Clustering
Semi-Supervised Learning
Feature Selection
Feature Extraction
Manifold Learning
Dimensionality Reduction
Kernel Approximation
Hyperparameter Optimization
Evaluation Metrics
Out-of-core learning
……

agramfort
Alexandre Gramfort

AlexanderFabisch
Alexander Fabisch

alextp
Alexandre Passos

amueller
Andreas Mueller

arjoly
Arnaud Joly

bdholt1
Brian Holt

GaelVaroquaux
Gael Varoquaux

glouppe
Gilles Louppe

jakevdp
Jake Vanderplas

jaquesgrobler
Jaques Grobler

jnothman

kastnerkyle
Kyle Kastner

kuantkid
Wei LI

larsmans
Lars

lucidfrontier45
Shiqiao Du

mblondel
Mathieu Blondel

MechCoder
Manoj Kumar

ndawe
Noel Dawe

NelleV
Varoquaux

ogrisel
Olivier Grisel

paolo-losi
Paolo Losi

pprett
Peter Prettenhofer

robertlayton
Robert Layton

ronw
Ron Weiss

satra
Satrajit Ghosh

sklearn-ci

vene
Vlad Niculae

VirgileFritsch
Virgile Fritsch

vmichel
Vincent Michel

yarikoptic
Yaroslav Halchenko

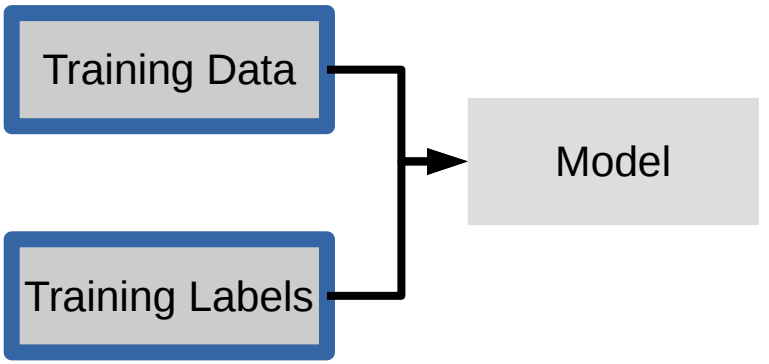
Overview

- Reminder: Basic sklearn concepts
- Model building and evaluation:
  - Pipelines and Feature Unions
  - Randomized Parameter Search
  - Scoring Interface
- Out of Core learning
  - Feature Hashing
  - Kernel Approximation
- New stuff in 0.16.0
  - Overview
  - Calibration

Supervised Machine Learning

```
clf = RandomForestClassifier()
```

```
clf.fit(X_train, y_train)
```

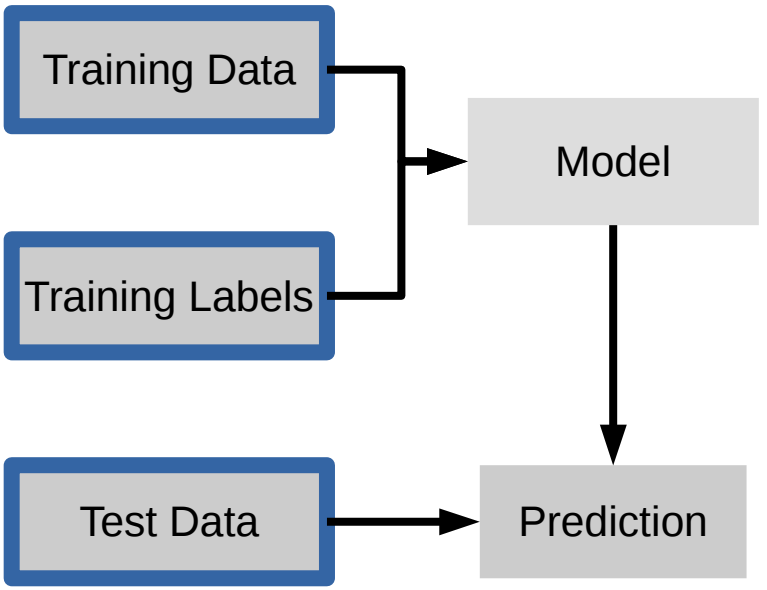


Supervised Machine Learning

```
clf = RandomForestClassifier()
```

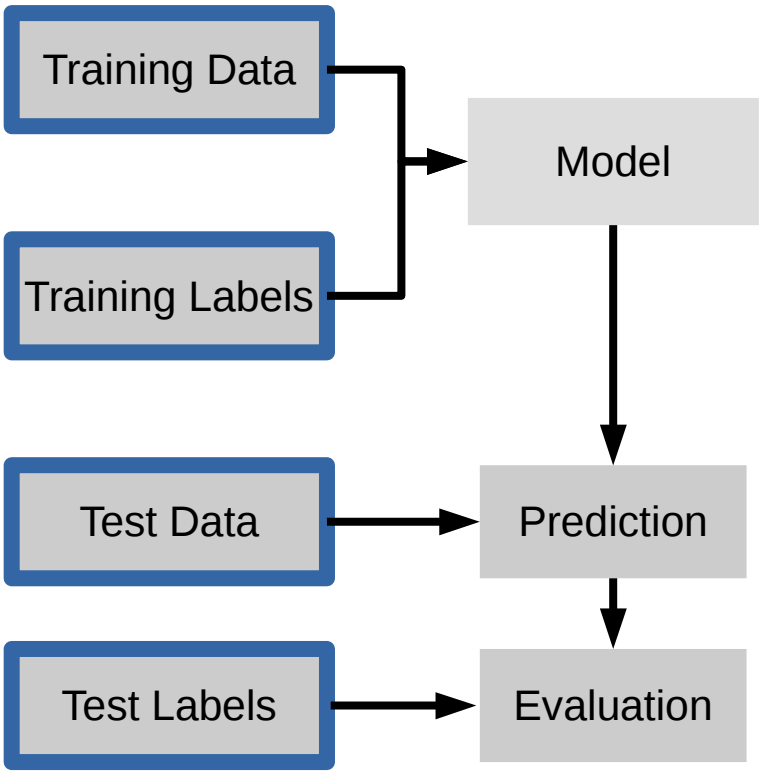
```
clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)
```



Supervised Machine Learning

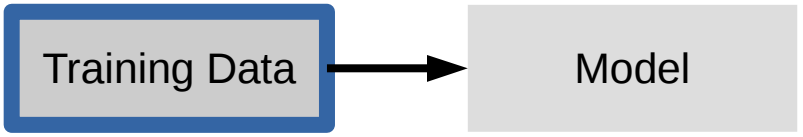
```
clf = RandomForestClassifier()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```



Unsupervised Transformations

```
pca = PCA(n_components=3)
```

```
pca.fit(X_train)
```

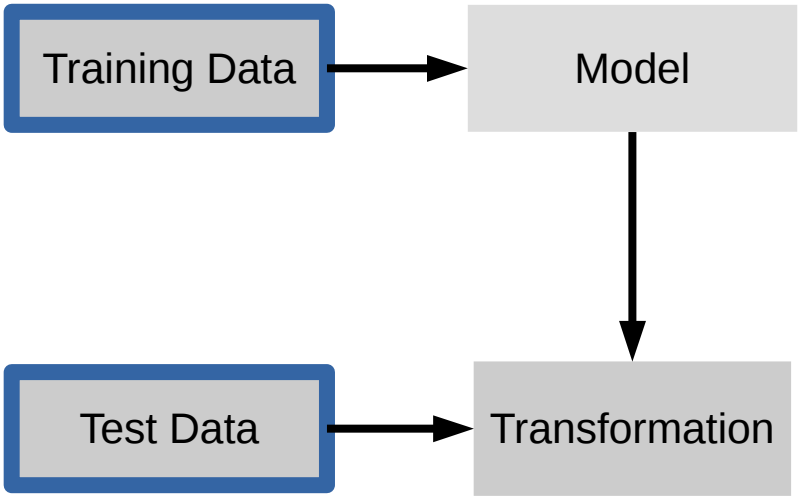


Unsupervised Transformations

```
pca = PCA(n_components=3)
```

```
pca.fit(X_train)
```

```
X_new = pca.transform(X_test)
```



Basic API

```
estimator.fit(X, [y])
```

`estimator.predict`	`estimator.transform`
Classification	Preprocessing
Regression	Dimensionality reduction
Clustering	Feature selection
	Feature extraction

Cross-Validation

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(SVC(), X, y, cv=5)
print(scores)

>> [ 0.92  1.    1.    1.    1.  ]
```

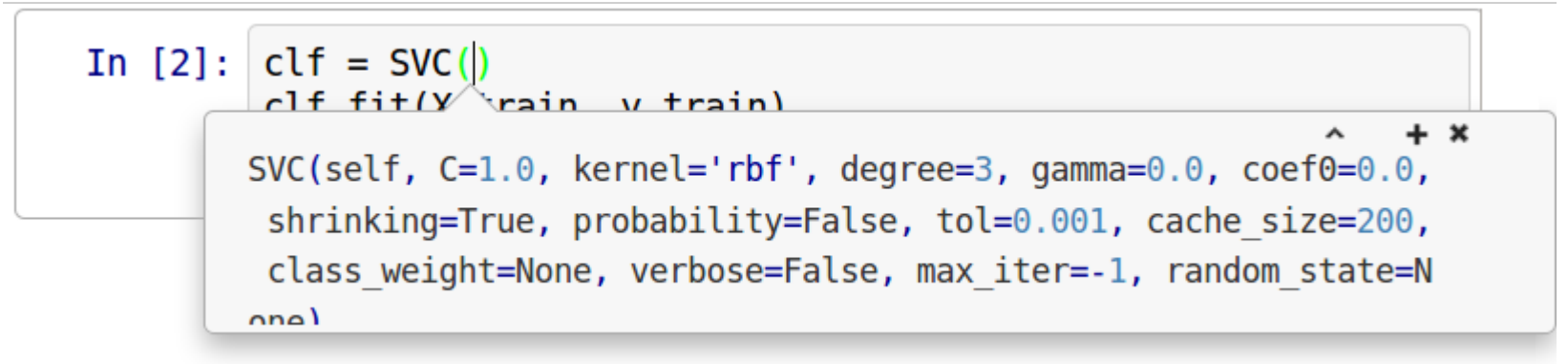
Cross-Validation

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(SVC(), X, y, cv=5)
print(scores)

>> [ 0.92  1.    1.    1.    1.  ]

cv_ss = ShuffleSplit(len(X_train), test_size=.3,
                     n_iter=10)
scores_shuffle_split = cross_val_score(SVC(), X, y,
                                       cv=cv_ss)
```

Cross-Validation

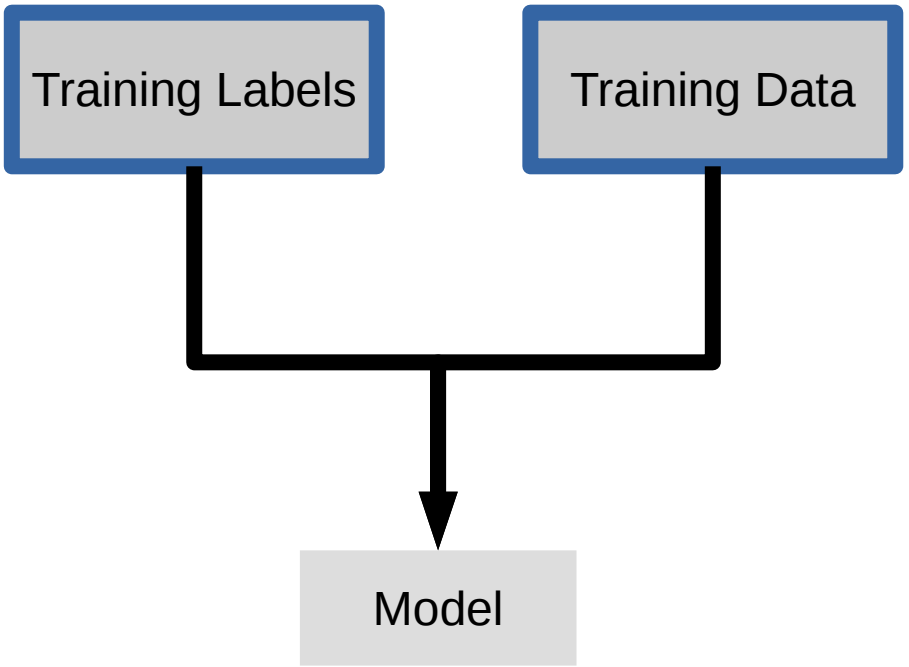
```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(SVC(), X, y, cv=5)
print(scores)

>> [ 0.92  1.    1.    1.    1.  ]

cv_ss = ShuffleSplit(len(X_train), test_size=.3,
                     n_iter=10)
scores_shuffle_split = cross_val_score(SVC(), X, y,
                                       cv=cv_ss)

cv_labels = LeaveOneLabelOut(labels)
scores_pout = cross_val_score(SVC(), X, y, cv=cv_labels)
```

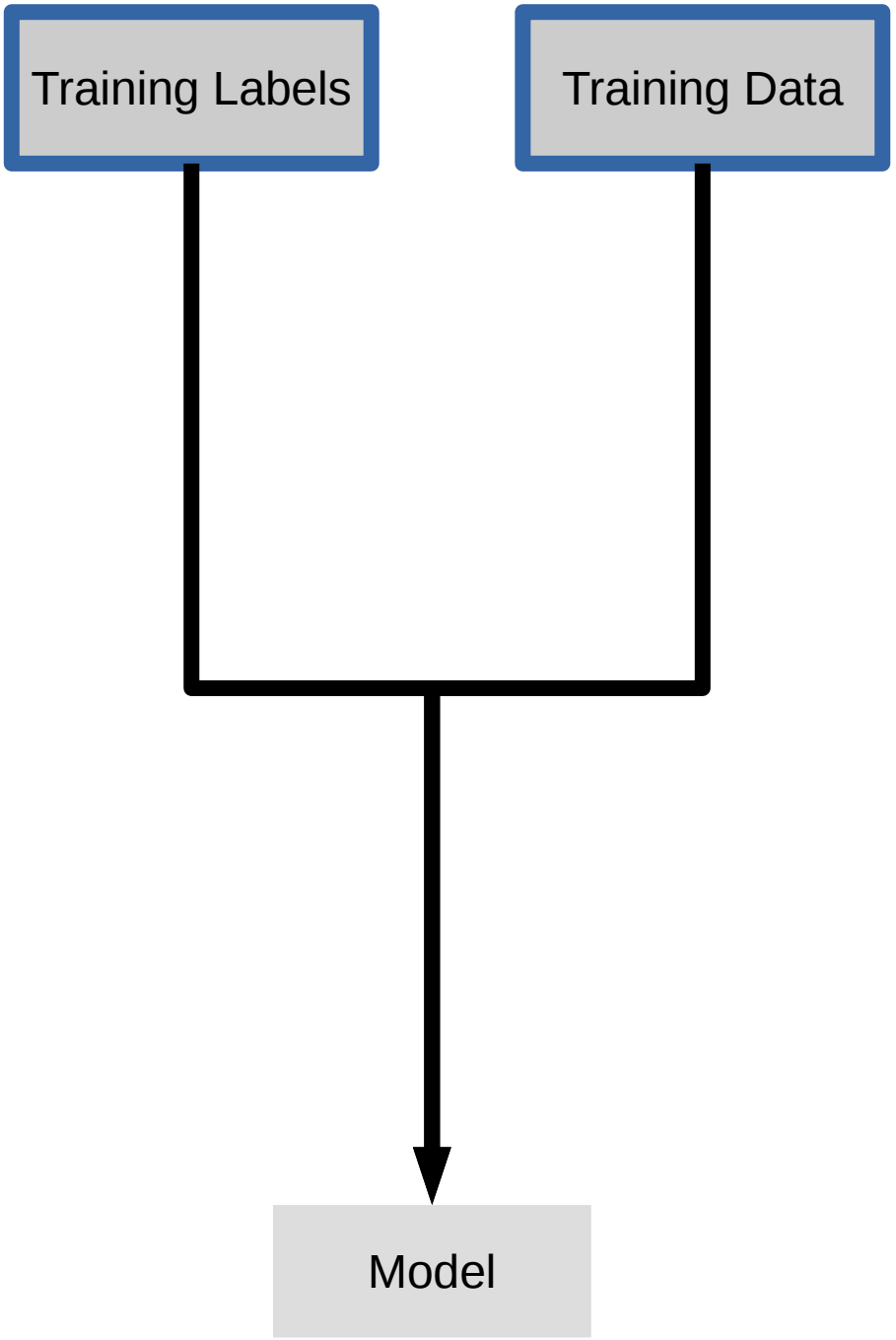
Cross -Validated Grid Search

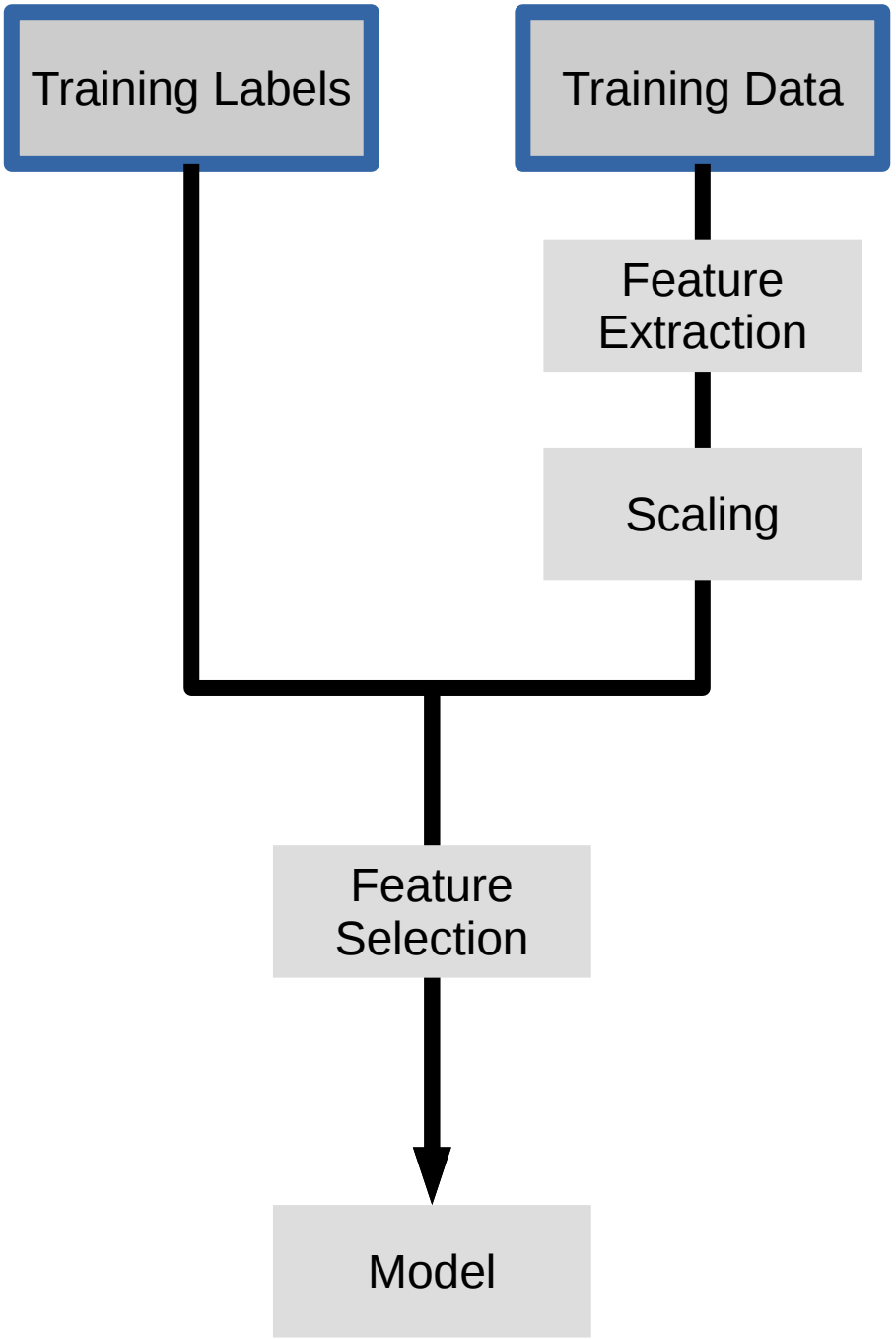


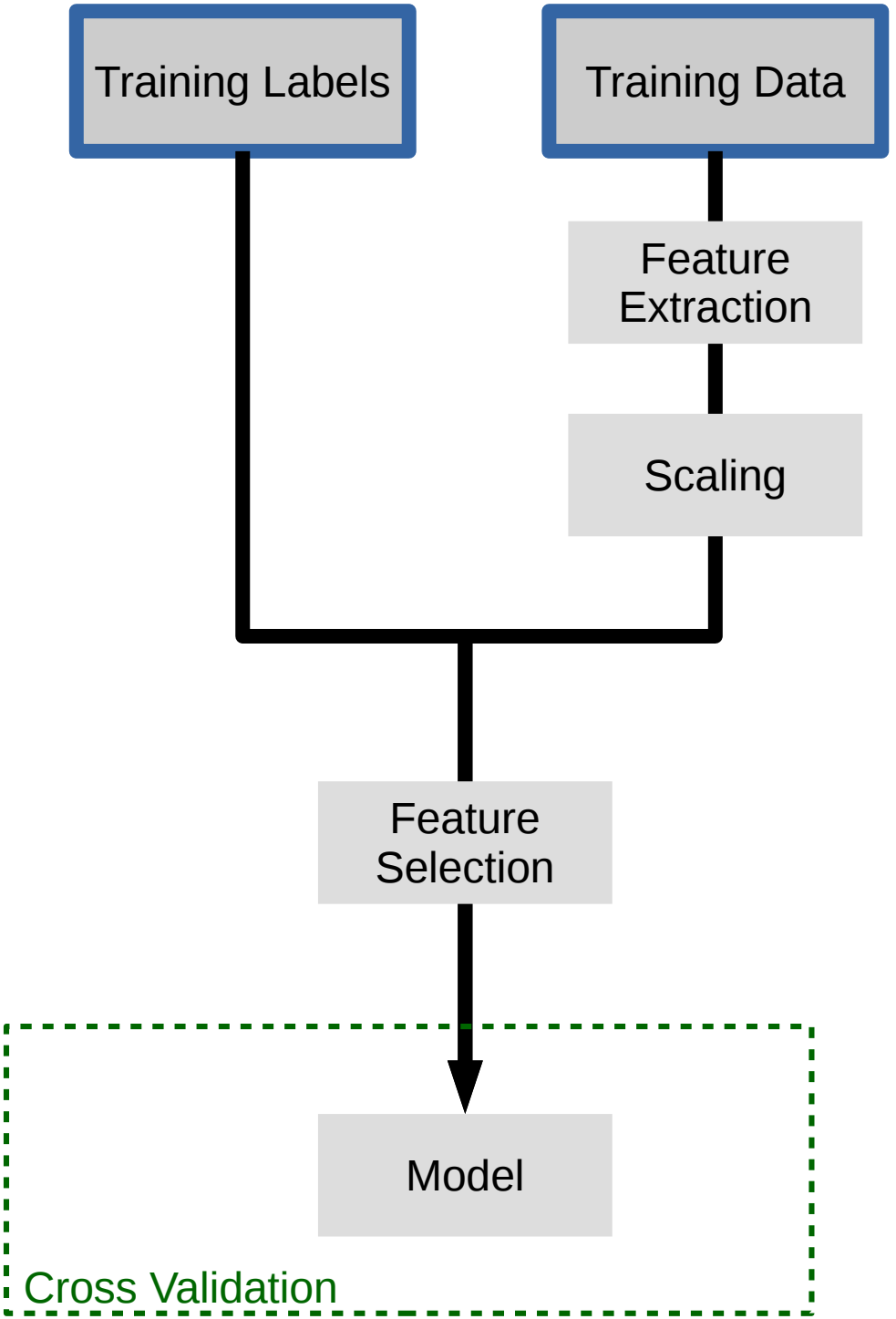
Cross -Validated Grid Search

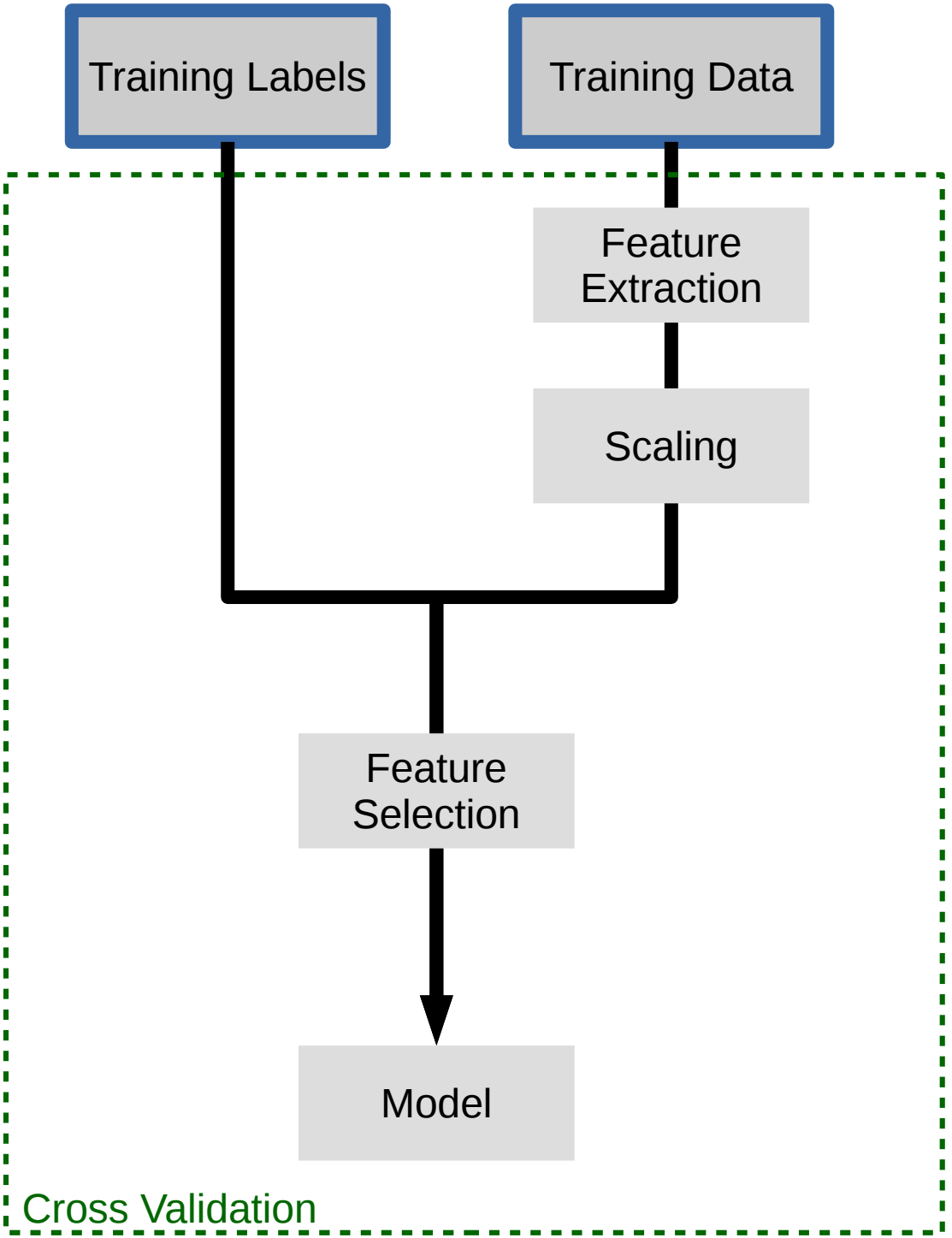
```
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)

param_grid = {'C': 10. ** np.arange(-3, 3),
              'gamma': 10. ** np.arange(-3, 3)}
grid = GridSearchCV(SVC(), param_grid=param_grid)
grid.fit(X_train, y_train)
grid.predict(X_test)
grid.score(X_test, y_test)
```


Training Labels
Training Data
Model


Training Labels
Training Data
Model


Training Labels
Training Data
Feature Extraction
Scaling
Feature Selection
Model


Training Labels
Training Data
Feature Extraction
Scaling
Feature Selection
Model
Cross Validation


Training Labels
Training Data
Feature Extraction
Scaling
Feature Selection
Model
Cross Validation

Pipelines

```
from sklearn.pipeline import make_pipeline

pipe = make_pipeline(StandardScaler(), SVC())
pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

Combining Pipelines and Grid Search

Proper cross-validation

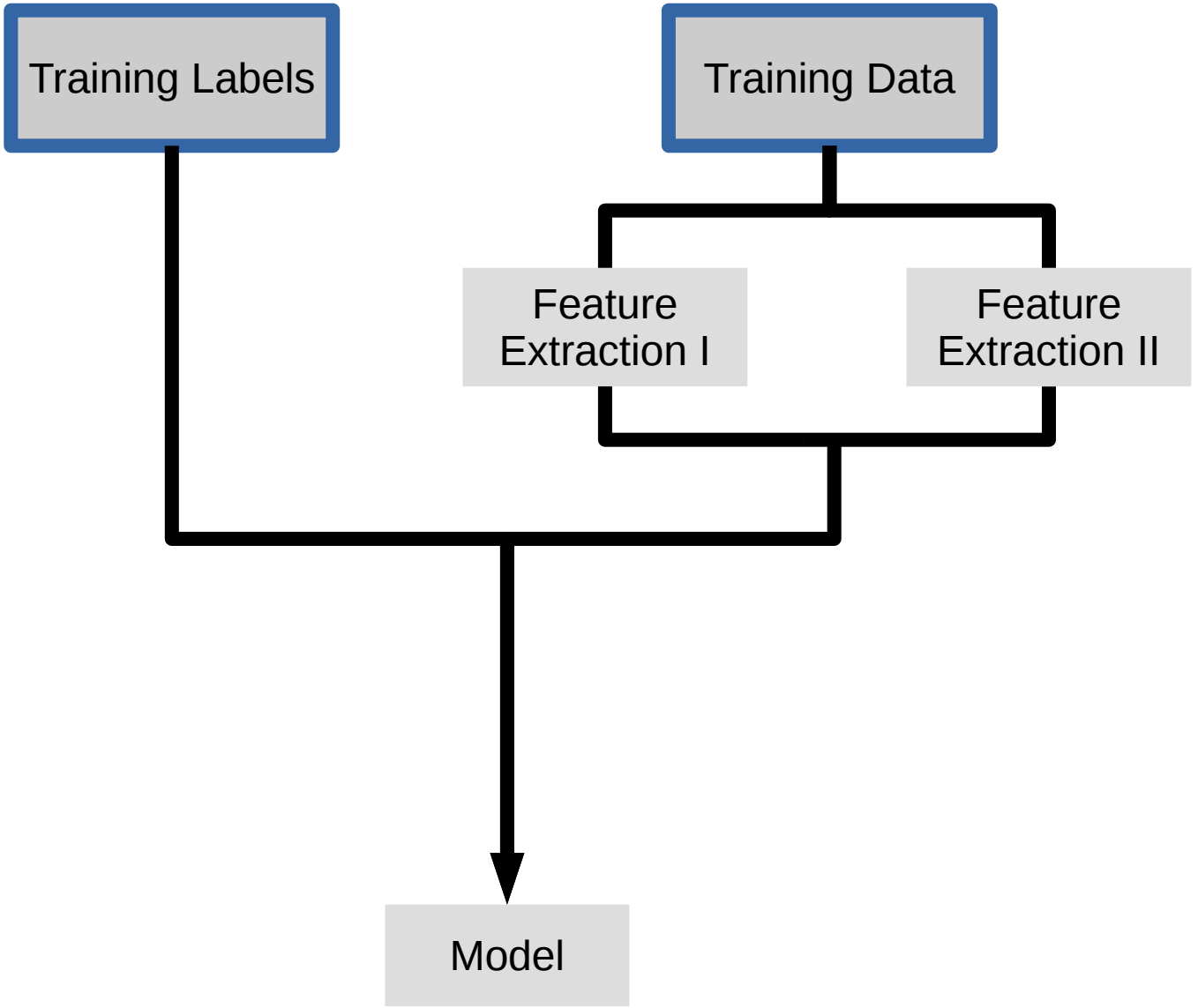
```
param_grid = {'svc__C': 10. ** np.arange(-3, 3),
              'svc__gamma': 10. ** np.arange(-3, 3)}

scaler_pipe = make_pipeline(StandardScaler(), SVC())
grid = GridSearchCV(scaler_pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

Combining Pipelines and Grid Search II

Searching over parameters of the preprocessing step

```
param_grid = {'selectkbest__k': [1, 2, 3, 4],
              'svc__C': 10. ** np.arange(-3, 3),
              'svc__gamma': 10. ** np.arange(-3, 3)}

scaler_pipe = make_pipeline(SelectKBest(), SVC())
grid = GridSearchCV(scaler_pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

Feature Union



Feature Union

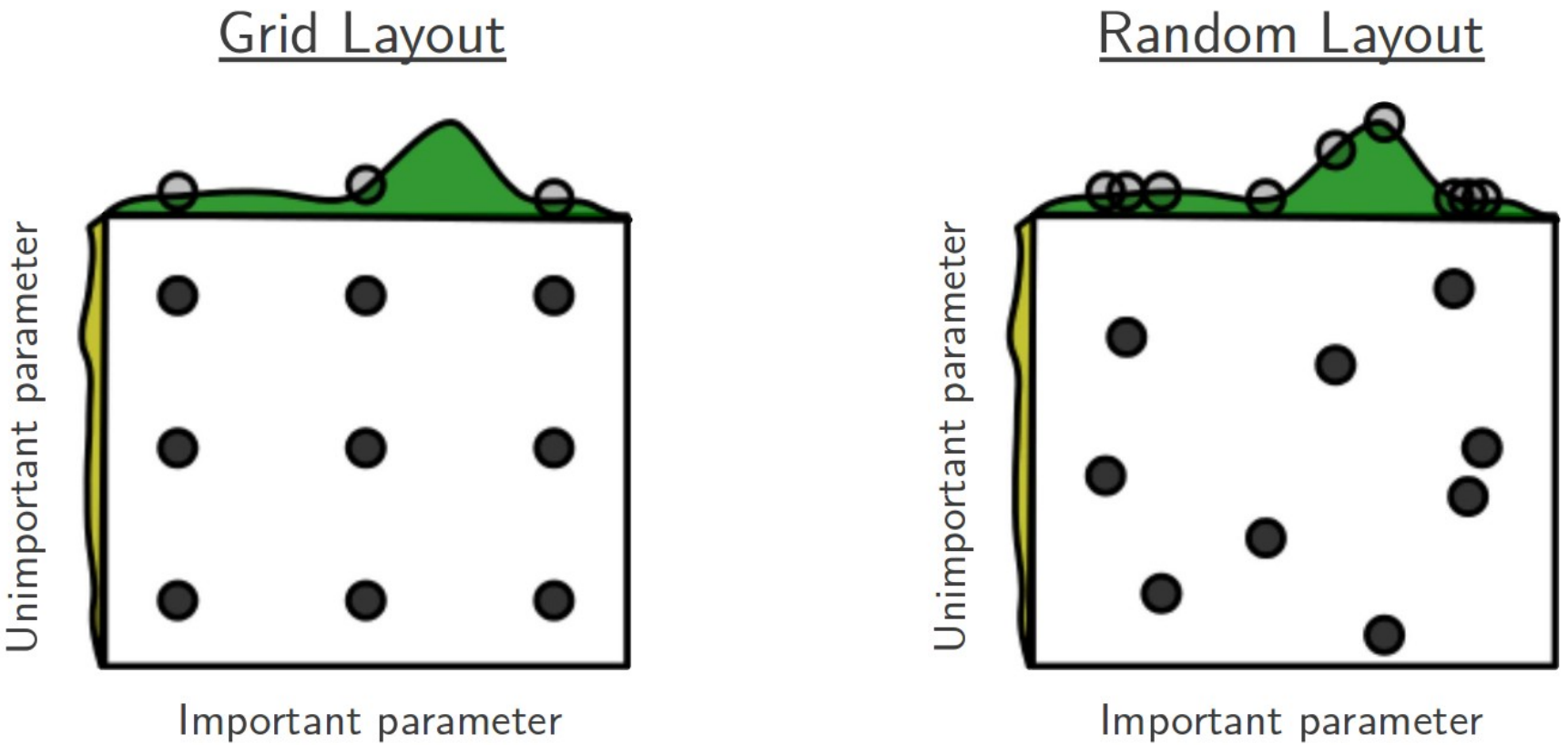
```
char_and_word = make_union(CountVectorizer(analyzer="char"),
                           CountVectorizer(analyzer="word"))

text_pipe = make_pipeline(char_and_word, LinearSVC(dual=False))

param_grid = {'linearsvc__C': 10. ** np.arange(-3, 3)}
grid = GridSearchCV(text_pipe, param_grid=param_grid, cv=5)
```

Feature Union

```
char_and_word = make_union(CountVectorizer(analyzer="char"),
                           CountVectorizer(analyzer="word"))

text_pipe = make_pipeline(char_and_word, LinearSVC(dual=False))

param_grid = {'linearsvc__C': 10. ** np.arange(-3, 3)}
grid = GridSearchCV(text_pipe, param_grid=param_grid, cv=5)


param_grid2 = {'featureunion__countvectorizer-1__ngram_range': [(1, 3), (1, 5), (2, 5)],
               'featureunion__countvectorizer-2__ngram_range': [(1, 1), (1, 2), (2, 2)],
               'linearsvc__C': 10. ** np.arange(-3, 3)}
```

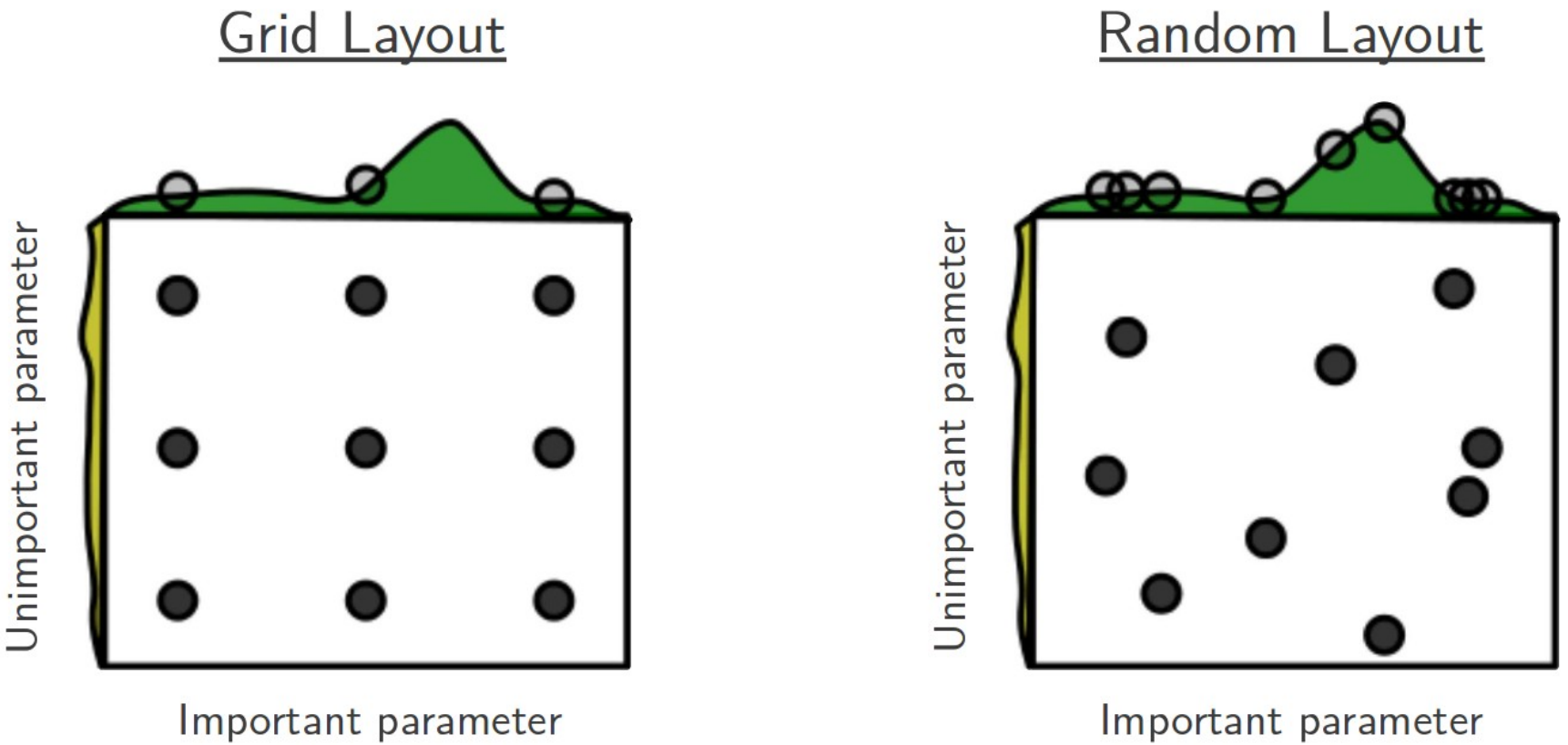
Randomized Parameter Search

Randomized Parameter Search



Source: Bergstra and Bengio

Randomized Parameter Search



Step-size free for continuous parameters
Decouples runtime from search-space size
Robust against irrelevant parameters

Source: Bergstra and Bengio

Randomized Parameter Search

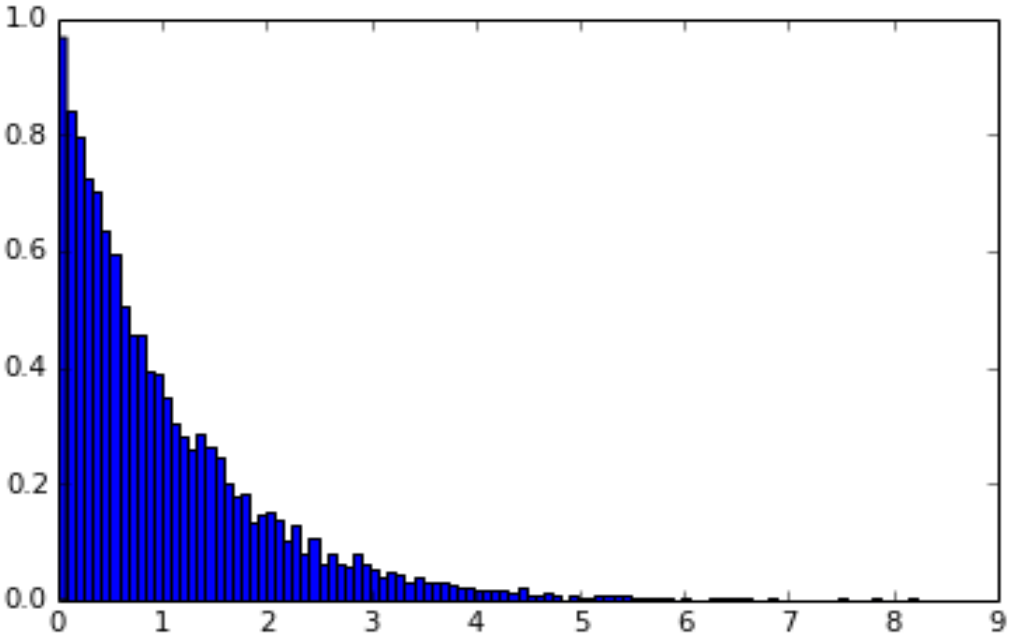
```
params = {'featureunion__countvectorizer-1__ngram_range':
          [(1, 3), (1, 5), (2, 5)],
          'featureunion__countvectorizer-2__ngram_range':
          [(1, 1), (1, 2), (2, 2)],
          'linearsvc__C': 10. ** np.arange(-3, 3)}
```

Randomized Parameter Search

```
params = {'featureunion__countvectorizer-1__ngram_range':
          [(1, 3), (1, 5), (2, 5)],
          'featureunion__countvectorizer-2__ngram_range':
          [(1, 1), (1, 2), (2, 2)],
          'linearsvc__C': expon()}
```

Randomized Parameter Search

```
params = {'featureunion__countvectorizer-1__ngram_range':
          [(1, 3), (1, 5), (2, 5)],
          'featureunion__countvectorizer-2__ngram_range':
          [(1, 1), (1, 2), (2, 2)],
          'linearsvc__C': expon()}
```




```
rs = RandomizedSearchCV(text_pipe,
    param_distributions=param_distributins, n_iter=50)
```

Randomized Parameter Search

- Always use distributions for continuous variables.
- Don't use for low dimensional spaces.
- Future: Bayesian optimization based search.

Generalized Cross-Validation and Path Algorithms

```
rfe = RFE(LogisticRegression())
```

```
rfe = RFE(LogisticRegression())
param_grid = {'n_features_to_select': range(1, n_features)}
gridsearch = GridSearchCV(rfe, param_grid)
grid.fit(X, y)
```

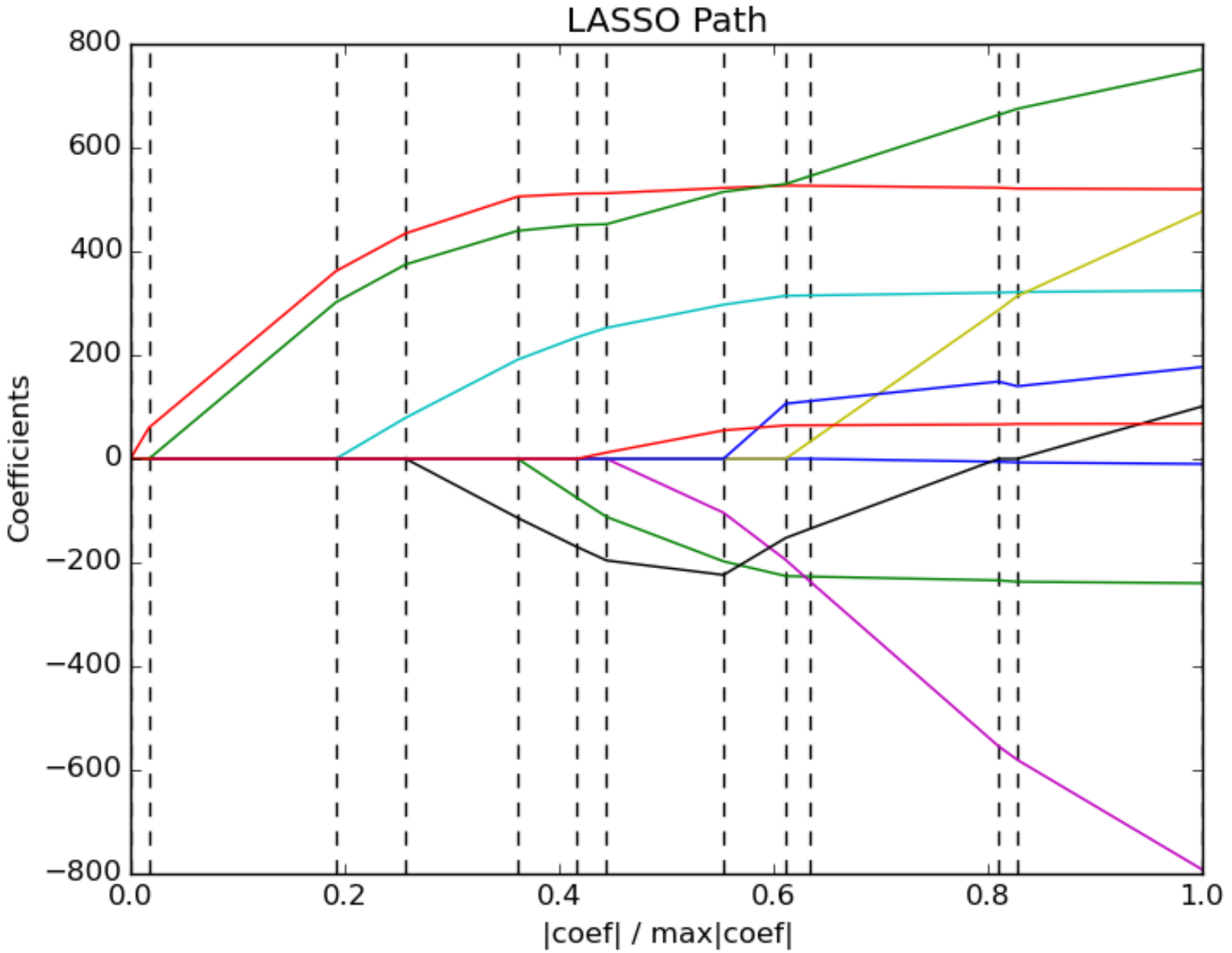

rfe = RFE(L
param_grid
gridsearch
grid.fit(X,

```
rfe = RFE(L  is  Regressi  ())
param_grid  {'n_  tures_  _select': range(1, n_features)}
gridsearch  GridSe  hCV(  e, param_grid)
grid.fit(X,  )
```

```
rfecv = RFECV(LogisticRegression())
```

```
rfe = RFE(L  is  Regressi  ())
param_grid  {'n_  tures_  _select': range(1, n_features)}
gridsearch  GridSe  hCV(  e, param_grid)
grid.fit(X,  )
```

```
rfecv = RFECV(LogisticRegression())
rfecv.fit(X, y)
```


LASSO Path
Coefficients
|coef| / max|coef|
800
600
400
200
0
−200
−400
−600
−800
0.0
0.2
0.4
0.6
0.8
1.0

Linear Models

```
LogisticRegressionCV [new]

RidgeCV

RidgeClassifierCV

LarsCV

ElasticNetCV
...
```

Feature Selection

```
RFECV
```

Tree-Based models [possible]

```
[DecisionTreeCV]

[RandomForestClassifierCV]

[GradientBoostingClassifierCV]
```

Scoring Functions

GridSeachCV
RandomizedSearchCV
cross_val_score
...CV

Default:
Accuracy (classification)
R2 (regression)

Scoring with imbalanced data

```
cross_val_score(SVC(), X_train, y_train)
>>> array([ 0.9,  0.9,  0.9])
```

Scoring with imbalanced data

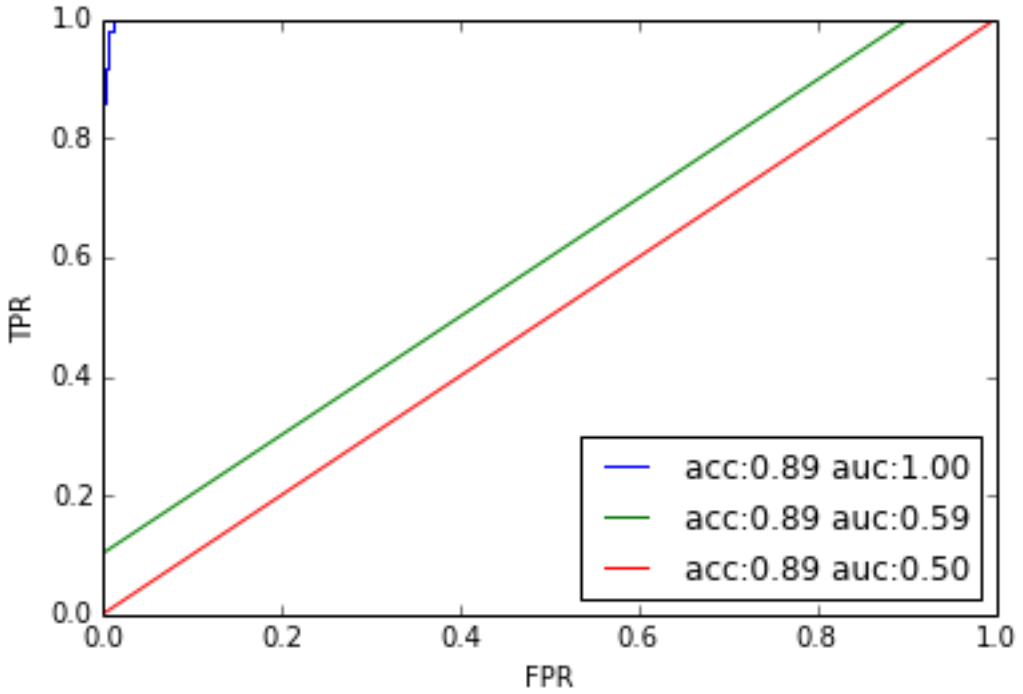
```
cross_val_score(SVC(), X_train, y_train)
>>> array([ 0.9,  0.9,  0.9])

cross_val_score(DummyClassifier("most_frequent"), X_train, y_train)
>>> array([ 0.9,  0.9,  0.9])
```

Scoring with imbalanced data

```
cross_val_score(SVC(), X_train, y_train)
>>> array([ 0.9,  0.9,  0.9])

cross_val_score(DummyClassifier("most_frequent"), X_train, y_train)
>>> array([ 0.9,  0.9,  0.9])

cross_val_score(SVC(), X_train, y_train, scoring="roc_auc")
array([ 0.99961591,  0.99983498,  0.99966247])
```

Scoring with imbalanced data

```
cross_val_score(SVC(), X_train, y_train)
>>> array([ 0.9,  0.9,  0.9])

cross_val_score(DummyClassifier("most_frequent"), X_train, y_train)
>>> array([ 0.9,  0.9,  0.9])

cross_val_score(SVC(), X_train, y_train, scoring="roc_auc")
array([ 0.99961591,  0.99983498,  0.99966247])
```



Available metrics

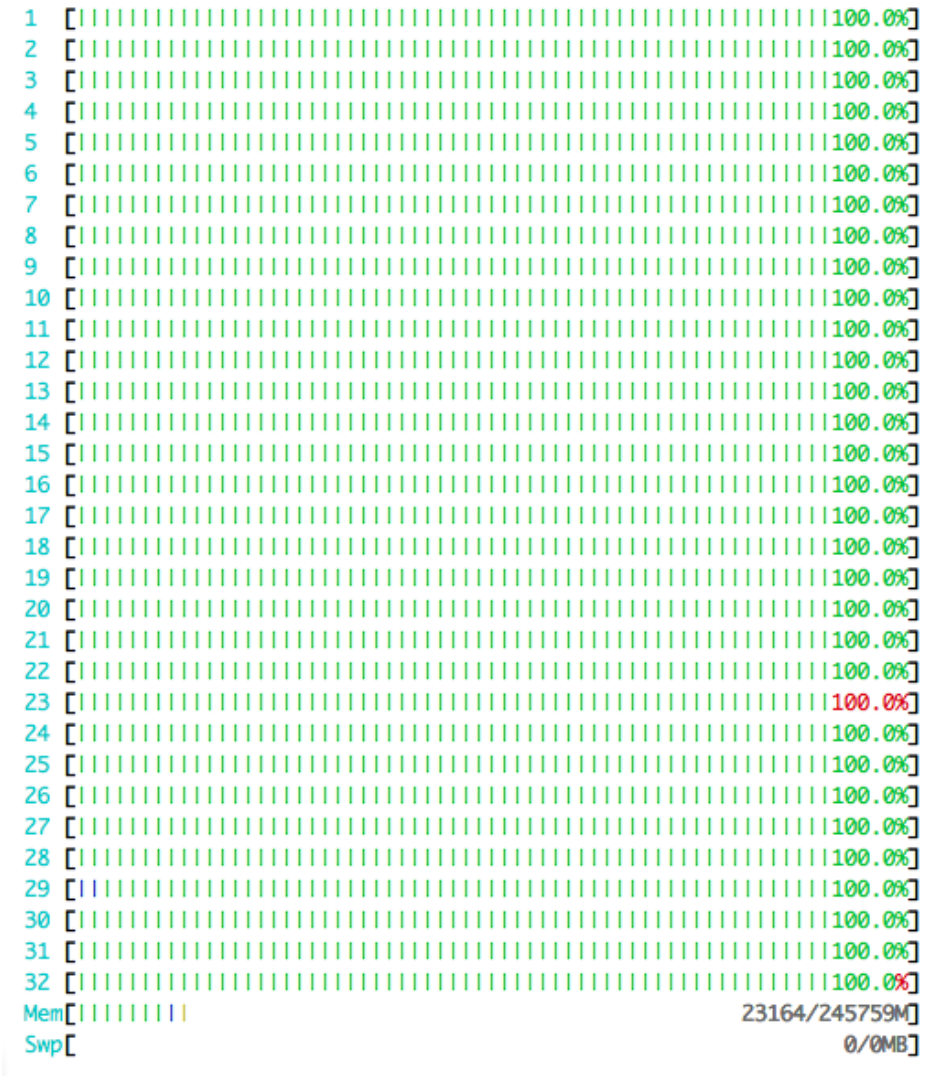
```
print(SCORERS.keys())

>> ['adjusted_rand_score',
 'f1',
 'mean_absolute_error',
 'r2',
 'recall',
 'median_absolute_error',
 'precision',
 'log_loss',
 'mean_squared_error',
 'roc_auc',
 'average_precision',
 'accuracy']
```

Defining your own scoring

```
def my_super_scoring(est, X, y):
    return accuracy_scorer(est, X, y) - np.sum(est.coef_ != 0)
```

Out of Core Learning

Or: save ourself the effort



Think twice!

- Old laptop: 4GB Ram
- 1073741824 float32
- Or 1mio data points with 1000 features
- EC2 : 256 GB Ram
- 68719476736 float32
- Or 68mio data points with 1000 features

Supported Algorithms

- All `SGDClassifier` derivatives
- Naive Bayes
- `MinibatchKMeans`
- `IncrementalPCA`
- `MiniBatchDictionaryLearning`

Out of Core Learning

```
sgd = SGDClassifier()

for i in range(9):
    X_batch, y_batch = cPickle.load(open("batch_%02d" % i))
    sgd.partial_fit(X_batch, y_batch, classes=range(10))
```

Possibly go over the data multiple times.

Stateless Transformers

- `Normalizer`
- `HashingVectorizer`
- `RBFSampler` (and other kernel approx)

Text data and the hashing trick

Bag Of Word Representations

`CountVectorizer / TfidfVectorizer`

`“You better call Kenny Loggins”`

Bag Of Word Representations

`CountVectorizer / TfidfVectorizer`

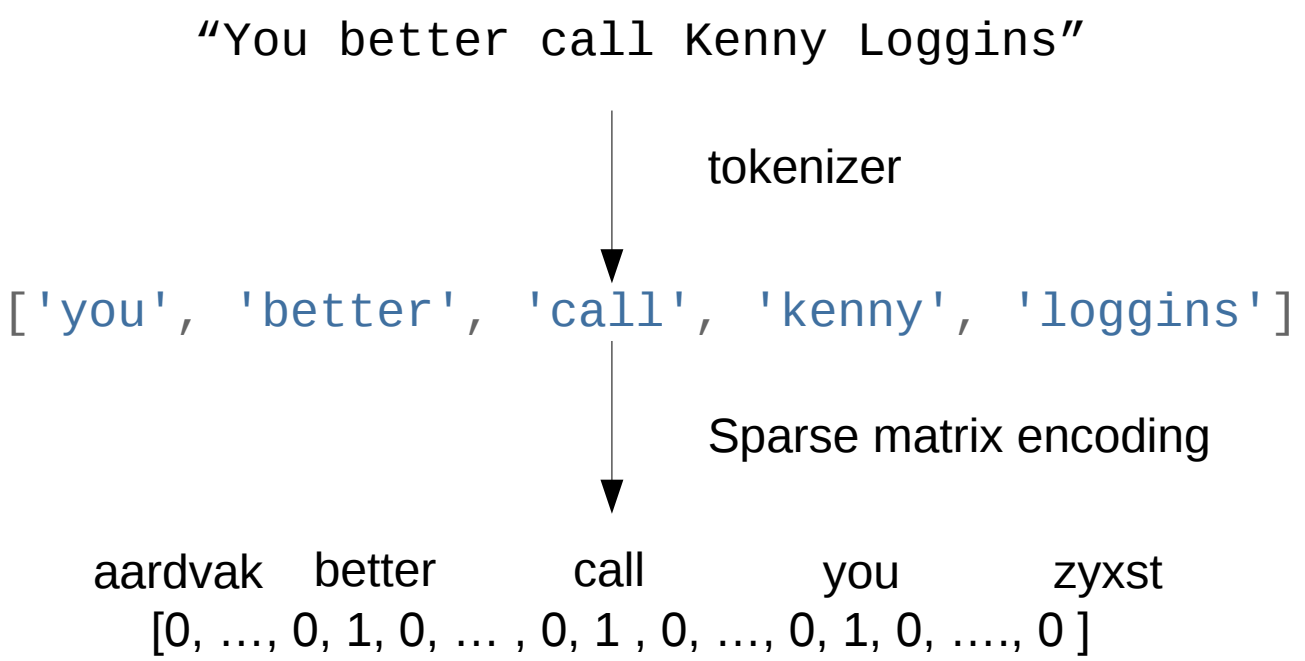
`“You better call Kenny Loggins”`

tokenizer

```
['you', 'better', 'call', 'kenny', 'loggins']
```

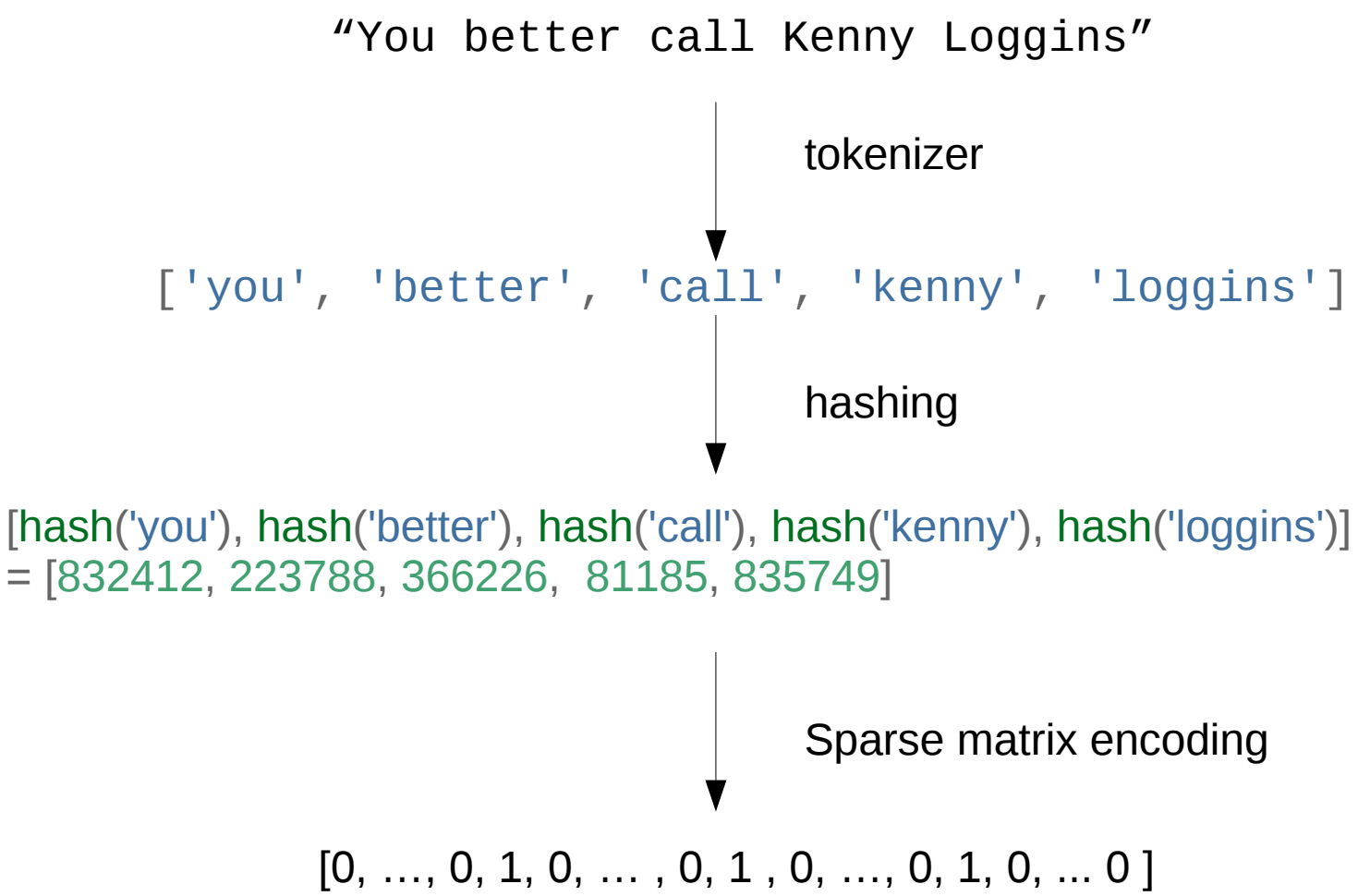
Bag Of Word Representations

`CountVectorizer / TfidfVectorizer`



Hashing Trick

`HashingVectorizer`

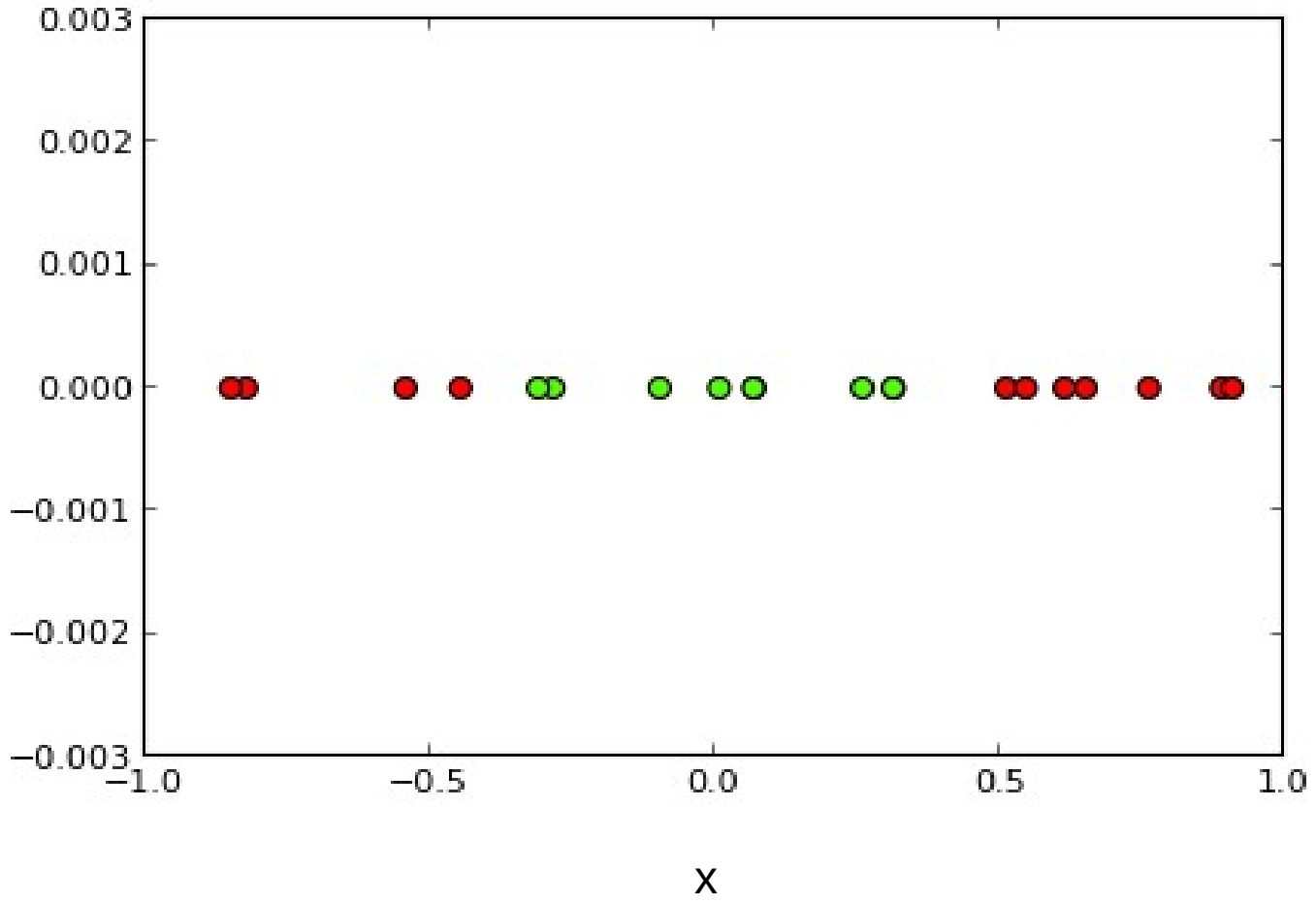


Out of Core Text Classification

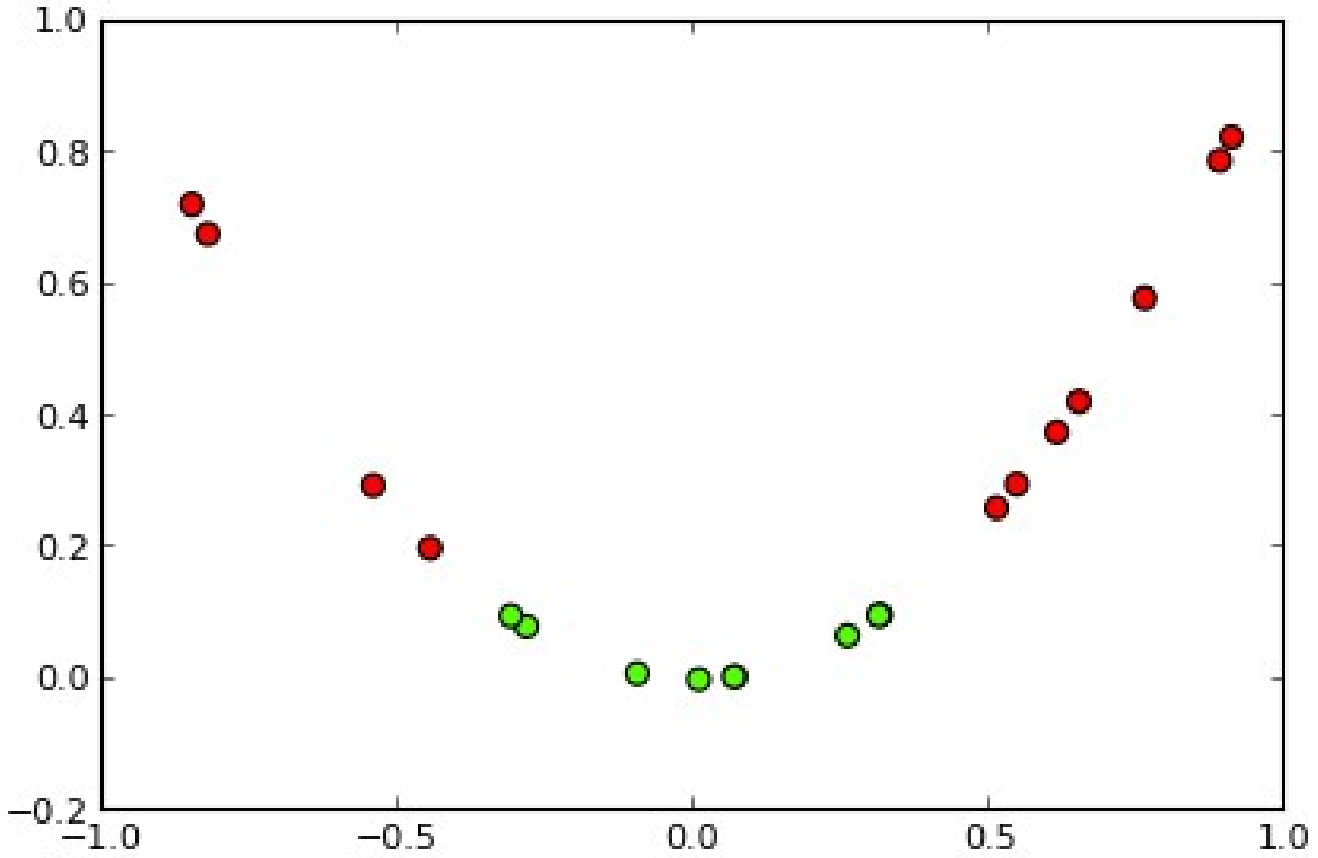
```
sgd = SGDClassifier()
hashing_vectorizer = HashingVectorizer()

for i in range(9):
    text_batch, y_batch = cPickle.load(open("text_%02d" % I))
    X_batch = hashing_vectorizer.transform(text_batch)
    sgd.partial_fit(X_batch, y_batch, classes=range(10))
```

Kernel Approximations

Reminder: Kernel Trick



Reminder: Kernel Trick



$\phi(x)$

Reminder: Kernel Trick

Classifier linear → need only

$$\langle \phi(x_i), \phi(x_j) \rangle = k(x_i, x_j)$$

Reminder: Kernel Trick

Classifier linear → need only

$$\langle \phi(x_i), \phi(x_j) \rangle = k(x_i, x_j)$$

Linear: $\langle x, x' \rangle$

Polynomial: $(\gamma \langle x, x' \rangle + r)^d$

RBF: $\exp(-\gamma |x - x'|^2)$

Sigmoid: $\tanh(\gamma \langle x, x' \rangle + r)$

Complexity

- Solving kernelized SVM:
  ~O(n_samples ** 3)
- Solving linear (primal) SVM:
  ~O(n_samples * n_features)

n_samples large? Go primal!

Undoing the Kernel Trick

- Kernel approximation:

$$\langle \hat{\phi}(x_i), \hat{\phi}(x_j) \rangle \approx k(x_i, x_j)$$

- k = $\exp(-\gamma|x - x'|^2)$

  $\hat{\phi}$ = RBFSampler

Usage

```
sgd = SGDClassifier()
kernel_approximation = RBFSampler(gamma=.001, n_components=400)

for i in range(9):
    X_batch, y_batch = cPickle.load(open("batch_%02d" % i))
    if i == 0:
        kernel_approximation.fit(X_batch)
    X_transformed = kernel_approximation.transform(X_batch)
    sgd.partial_fit(X_transformed, y_batch, classes=range(10))
```

Highlights from 0.16.0

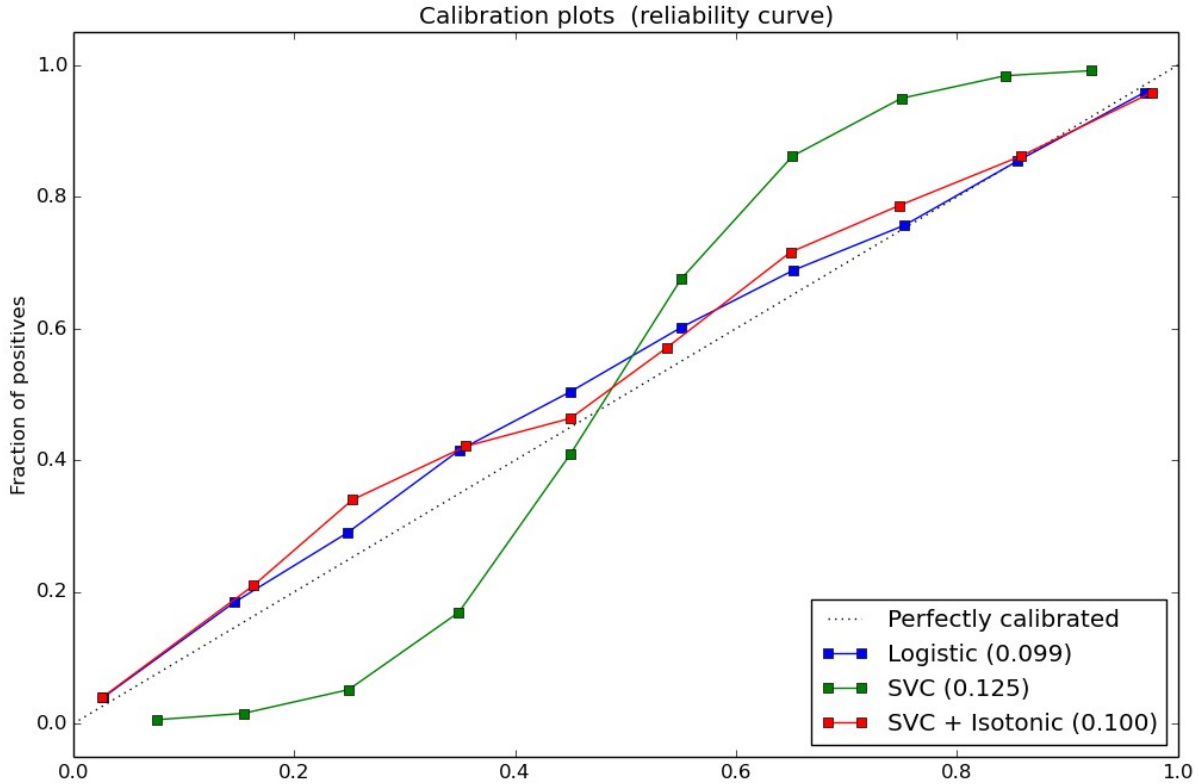
Highlights from 0.16.0

- Multinomial Logistic Regression, LogisticRegressionCV.
- IncrementalPCA.
- Probability callibration of classifiers.
- Birch clustering.
- LSHForest.
- More robust integration with pandas.

Probability Calibration

```
SVC().decision_function()
→  CalibratedClassifierCV(SVC()).predict_proba()
```

```
RandomForestClassifier().predict_proba()
→ CalibratedClassifierCV(RandomForestClassifier()).predict_proba()
```



scikit learn

Home Installation Documentation Examples

Google™ Custom Search Search

Documentation of scikit-learn 0.16-git

Quick Start

A very short introduction into machine learning problems and how to solve them using scikit-learn. Introduced basic concepts and conventions.

User Guide

The main documentation. This contains an in-depth description of all algorithms and how to apply them.

Other Versions

- scikit-learn 0.15 (stable)
- scikit-learn 0.16 (development)
- scikit-learn 0.14
- scikit-learn 0.13
- scikit-learn 0.12
- Older versions

Tutorials

Useful tutorials for developing a feel for some of scikit-learn's applications in the machine learning field.

API

The exact API of all functions and classes, as given by the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

Additional Resources

Talks given, slide-sets and other information relevant to scikit-learn.

Contributing

Information on how to contribute. This also contains useful information for advanced users, for example how to build their own estimators.

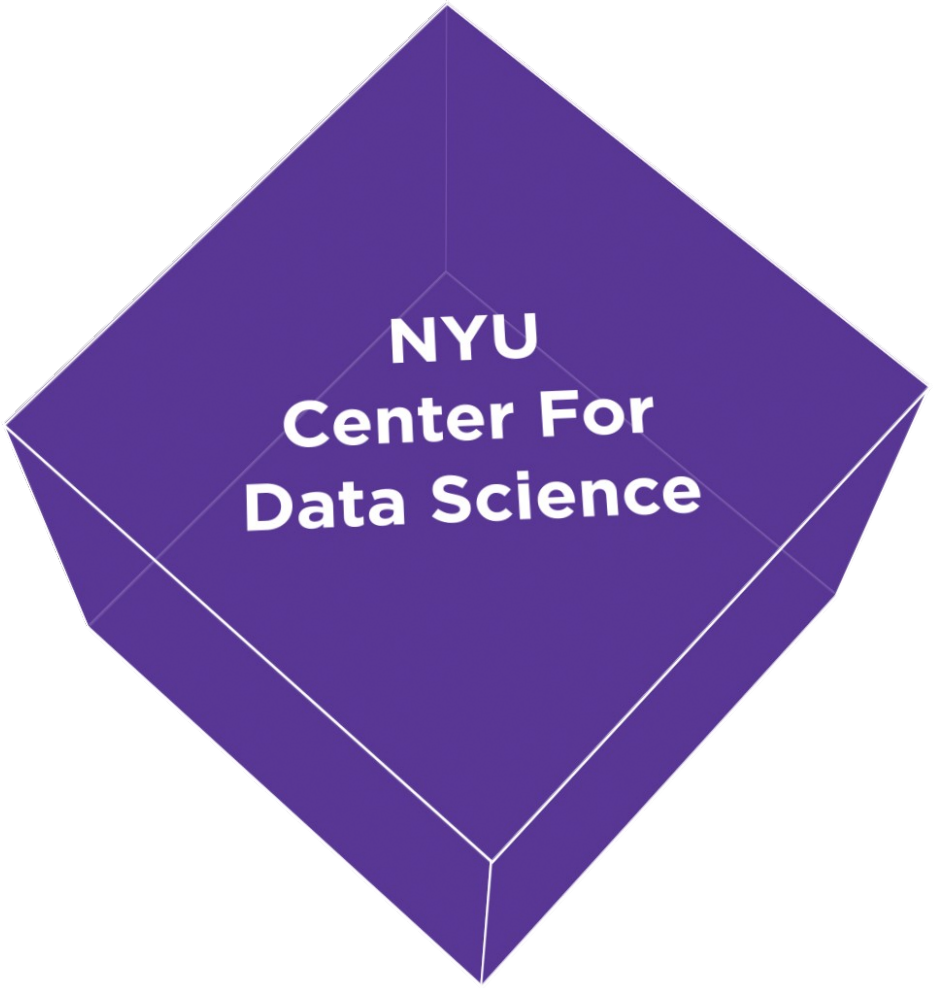
Flow Chart

A graphical overview of basic areas of machine learning, and guidance which kind of algorithms to use in a given situation.

FAQ

Frequently asked questions about the project and contributing.

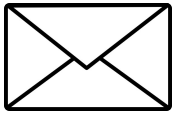
CDS is hiring Research Engineers



Thank you for your attention.

@t3kcit

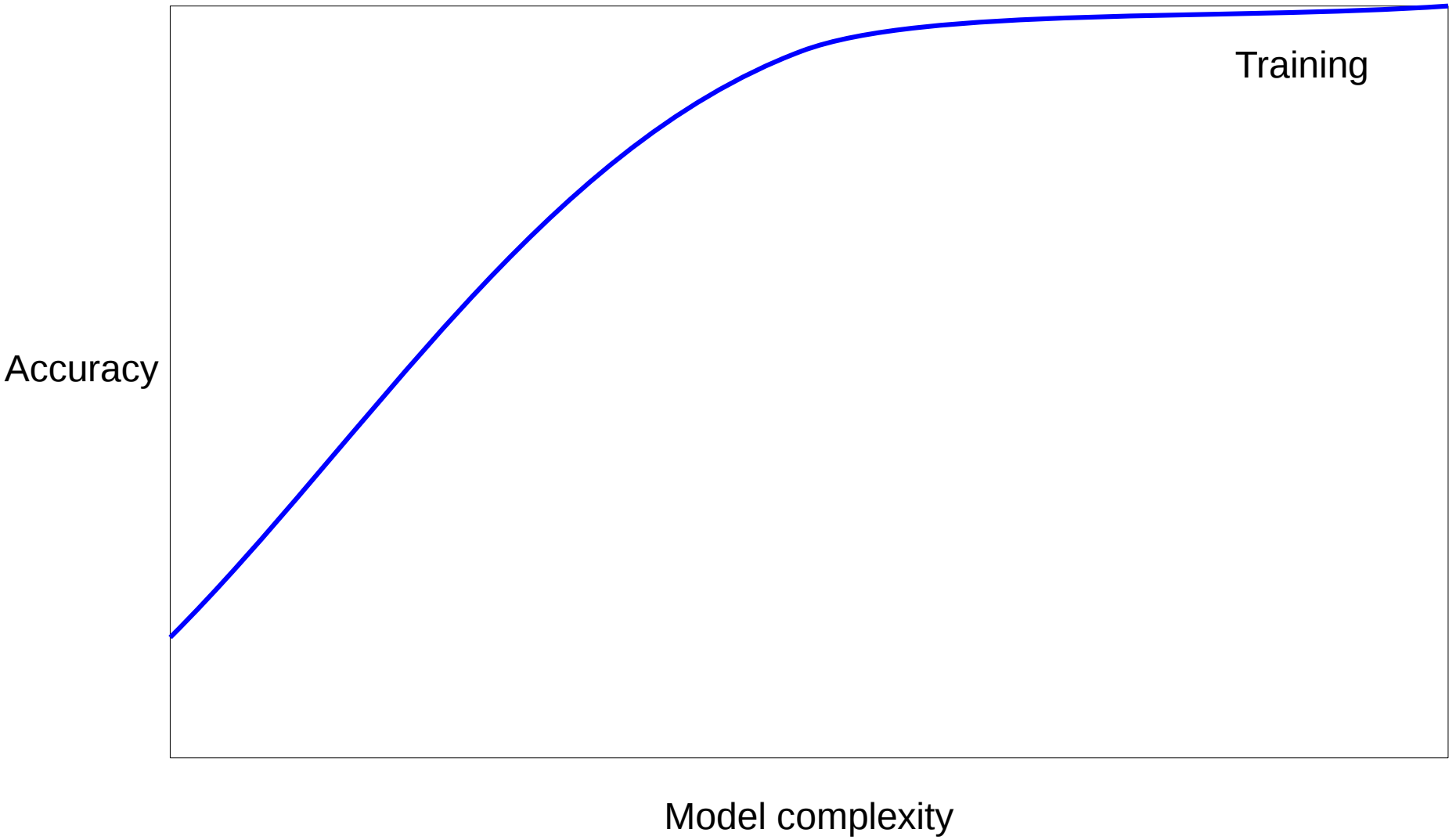
@amueller



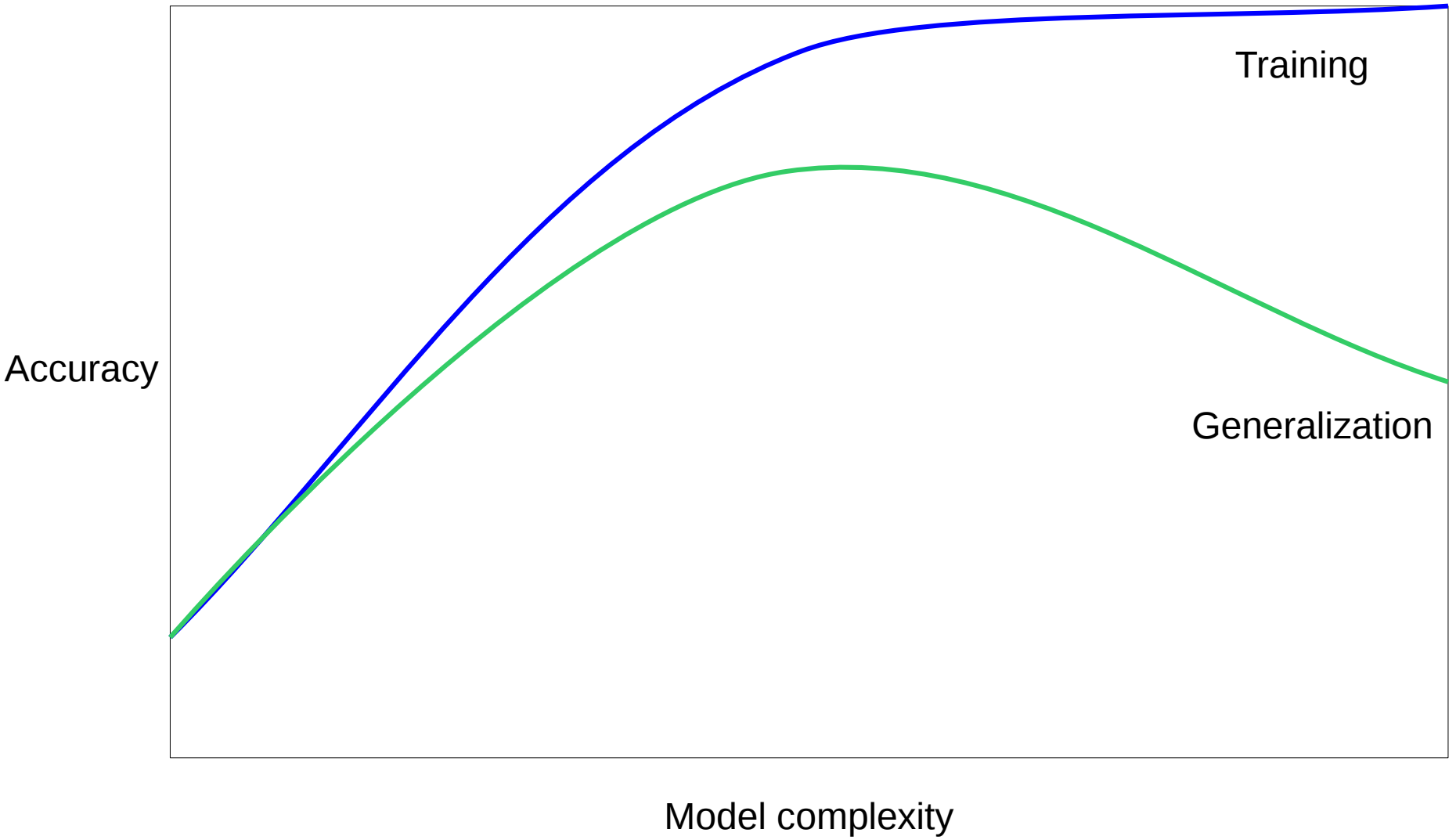
t3kcit@gmail.comx

Bias Variance Tradeoff
(why we do cross validation and grid searches)

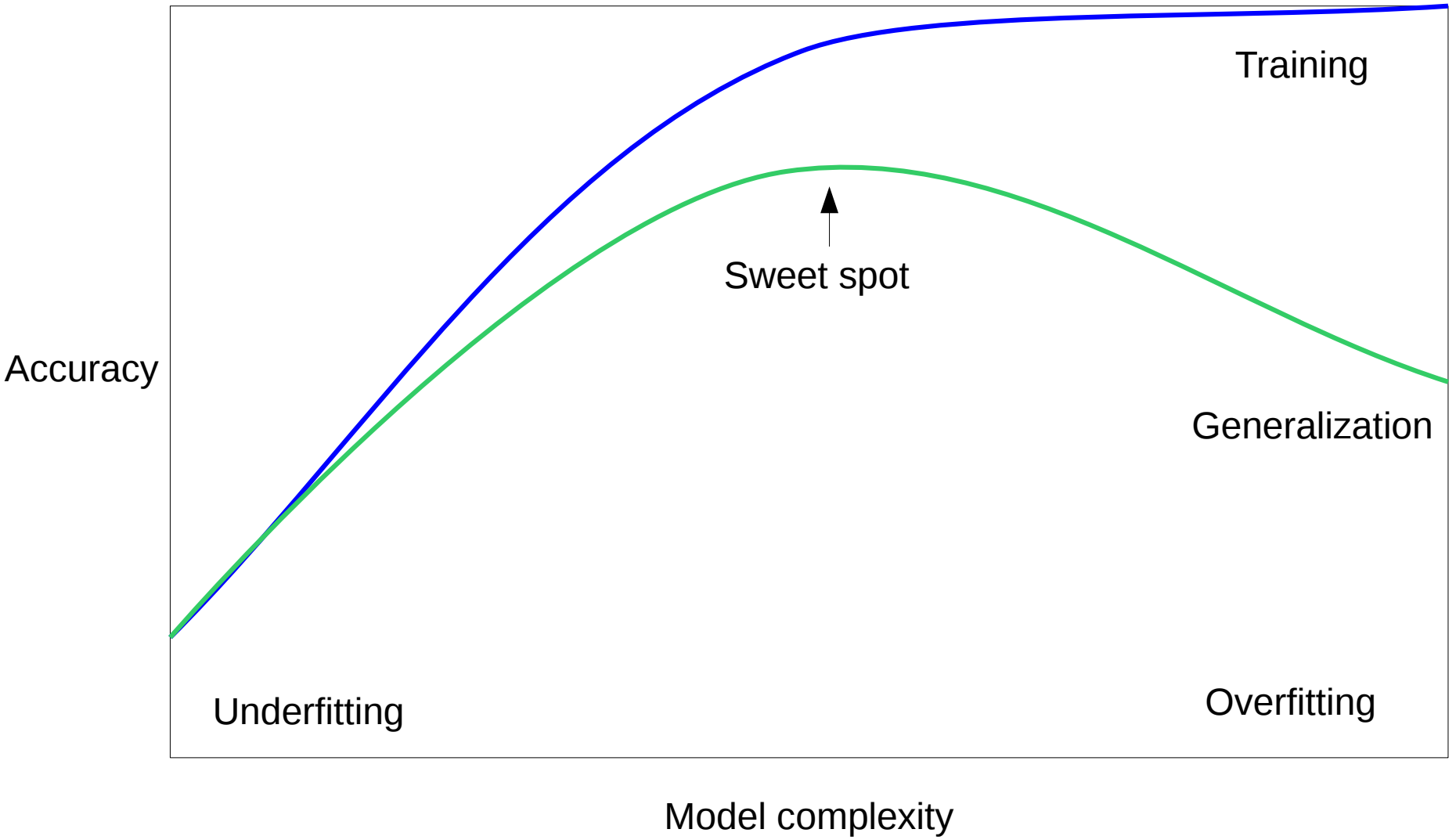
Overfitting and Underfitting



Overfitting and Underfitting



Overfitting and Underfitting

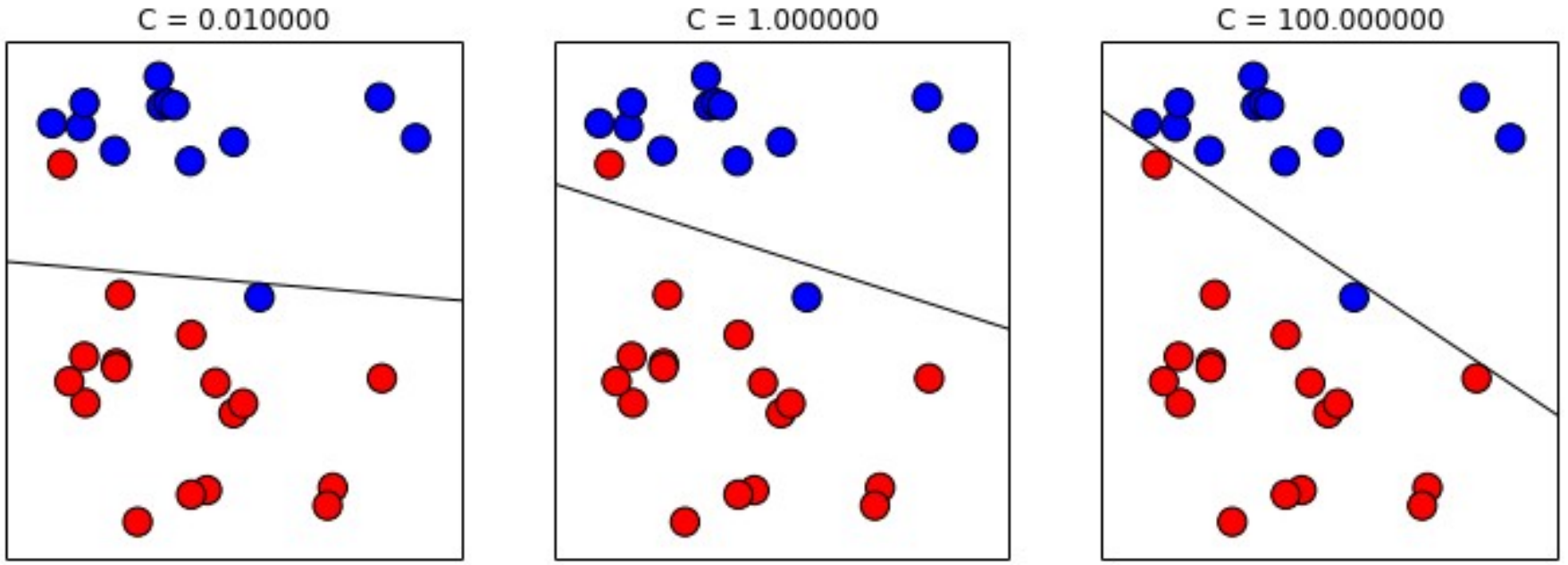


Linear SVM

$$\hat{y} = \text{sign}(w_0 + \sum_i w_i x_i)$$

Linear SVM

$$\hat{y} = \text{sign}(w_0 + \sum_i w_i x_i)$$



(RBF) Kernel SVM

$$\hat{y} = \text{sign}(\alpha_0 + \sum_j \alpha_j y_j k(\mathbf{x}^{(\mathbf{j})}, \mathbf{x}))$$

(RBF) Kernel SVM

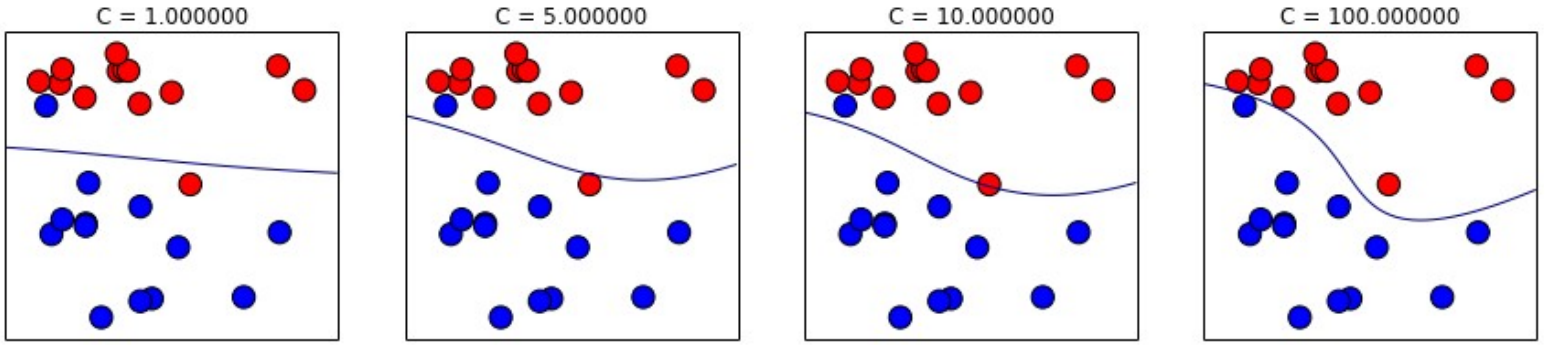
$$\hat{y} = \text{sign}(\alpha_0 + \sum_j \alpha_j y_j k(\mathbf{x^{(j)}}, \mathbf{x}))$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2)$$

(RBF) Kernel SVM

$$\hat{y} = \text{sign}(\alpha_0 + \sum_j \alpha_j y_j k(\mathbf{x}^{(\mathbf{j})}, \mathbf{x}))$$

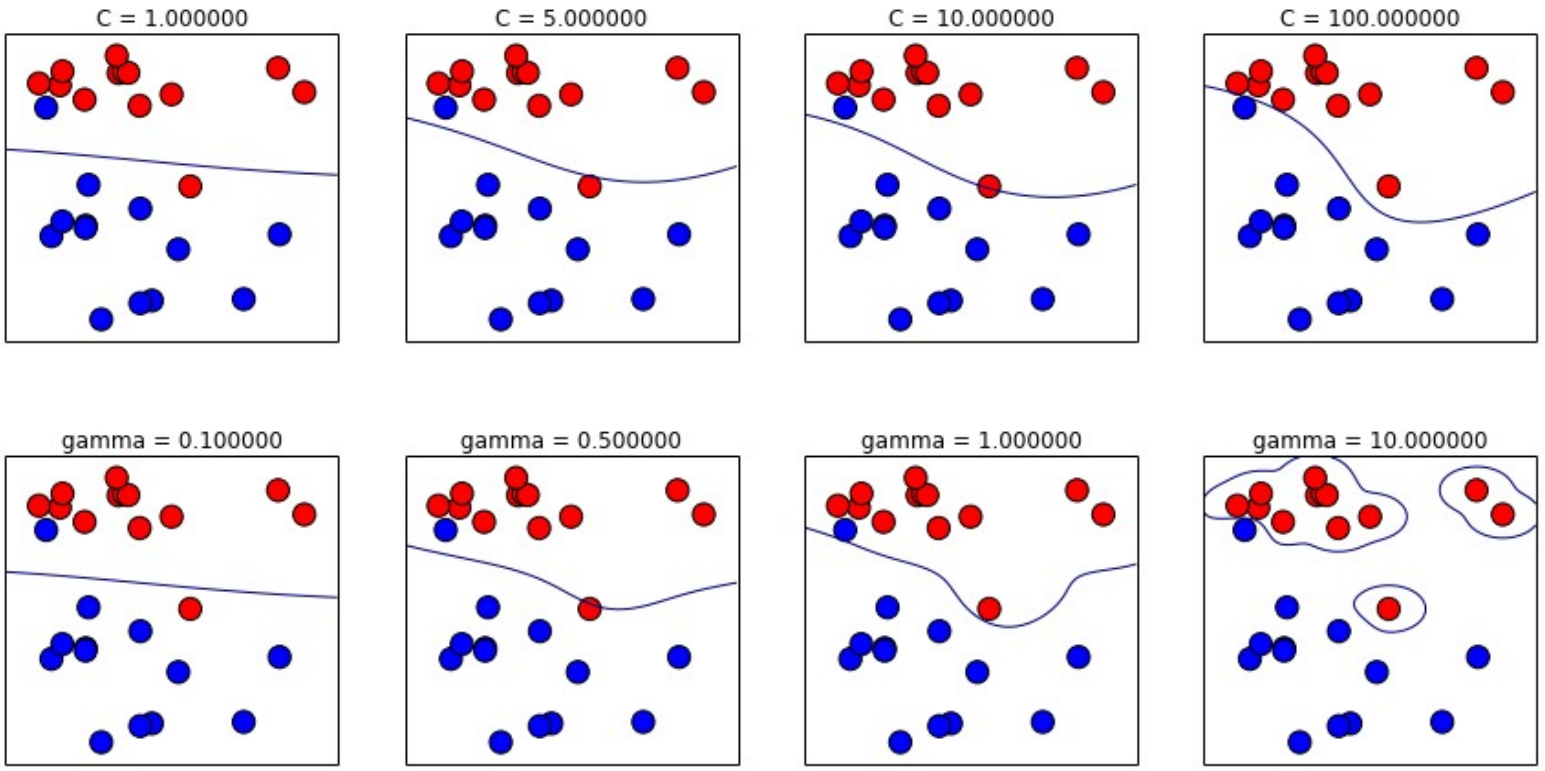
$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2)$$



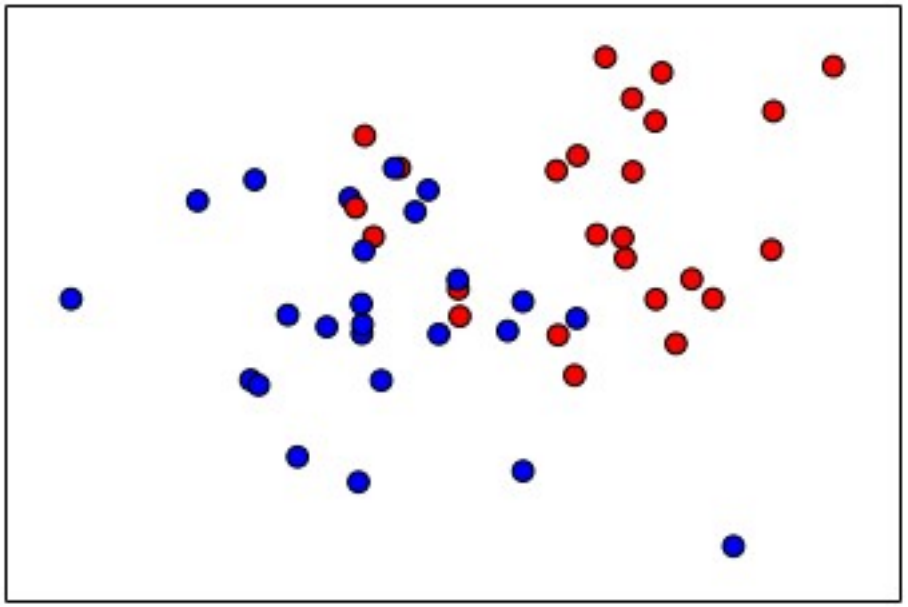
(RBF) Kernel SVM

$$\hat{y} = \text{sign}(\alpha_0 + \sum_j \alpha_j y_j k(\mathbf{x}^{(\mathbf{j})}, \mathbf{x}))$$

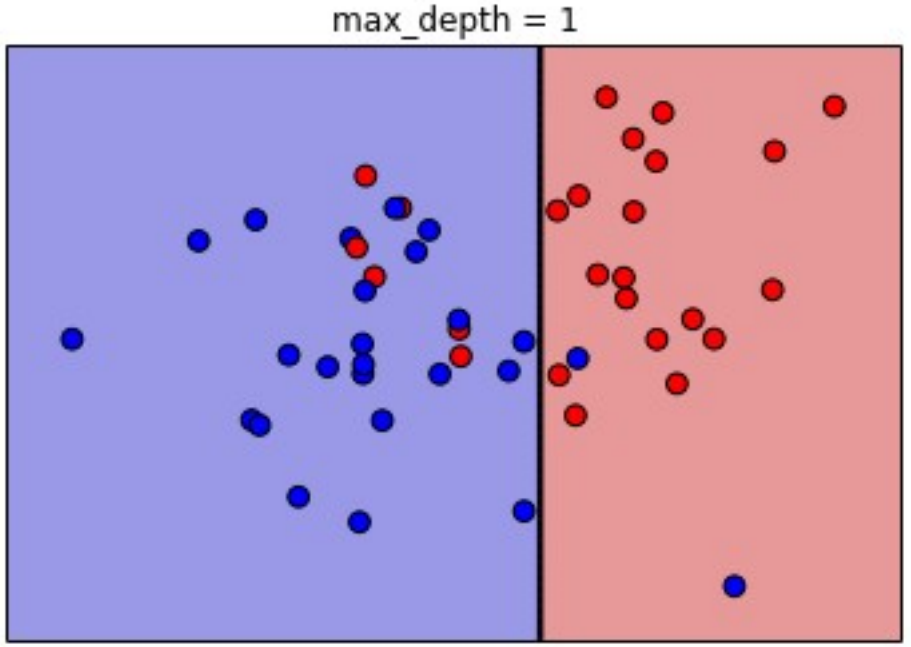
$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2)$$

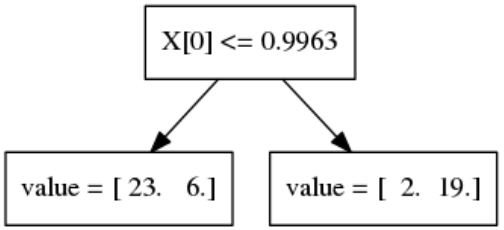


Decision Trees

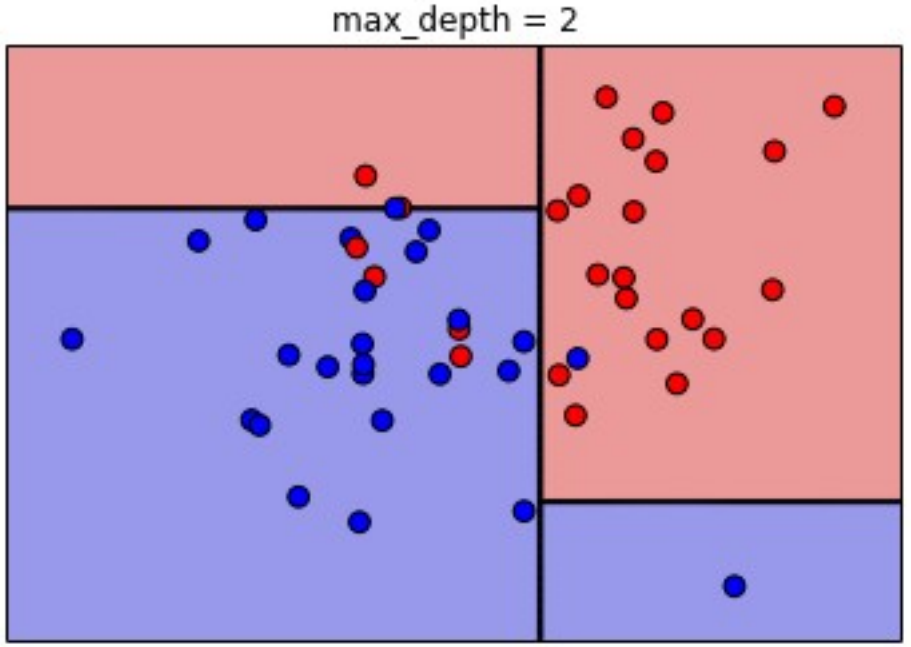


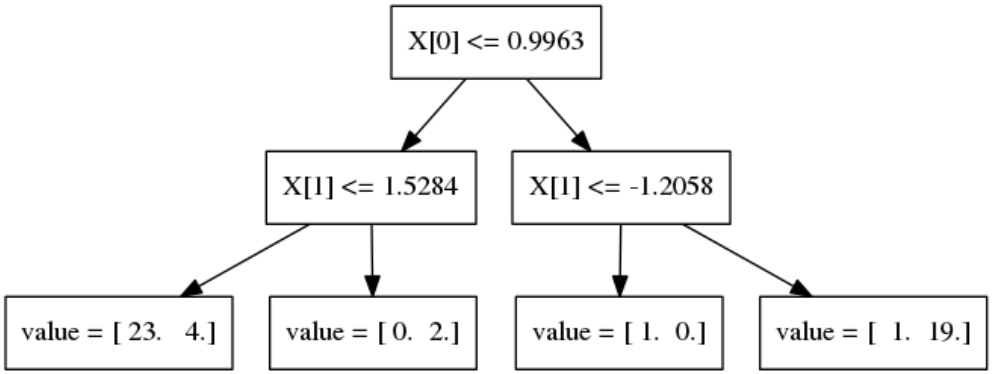
Decision Trees



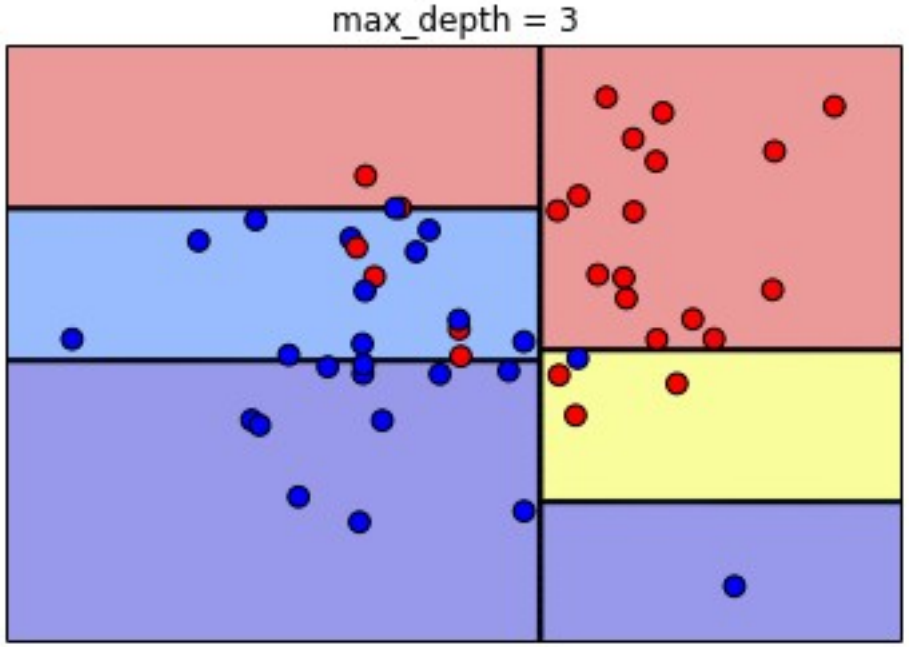


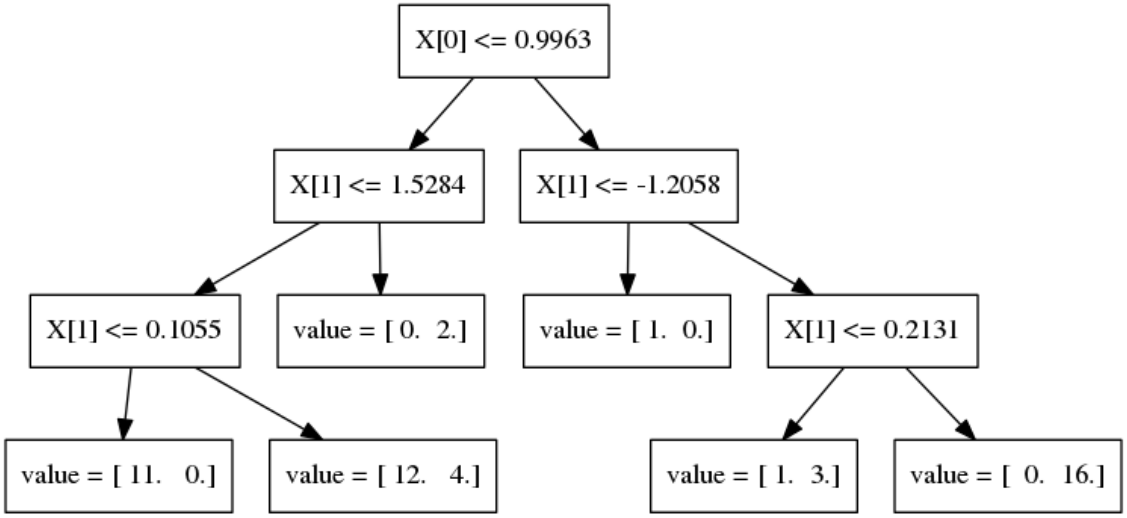
Decision Trees



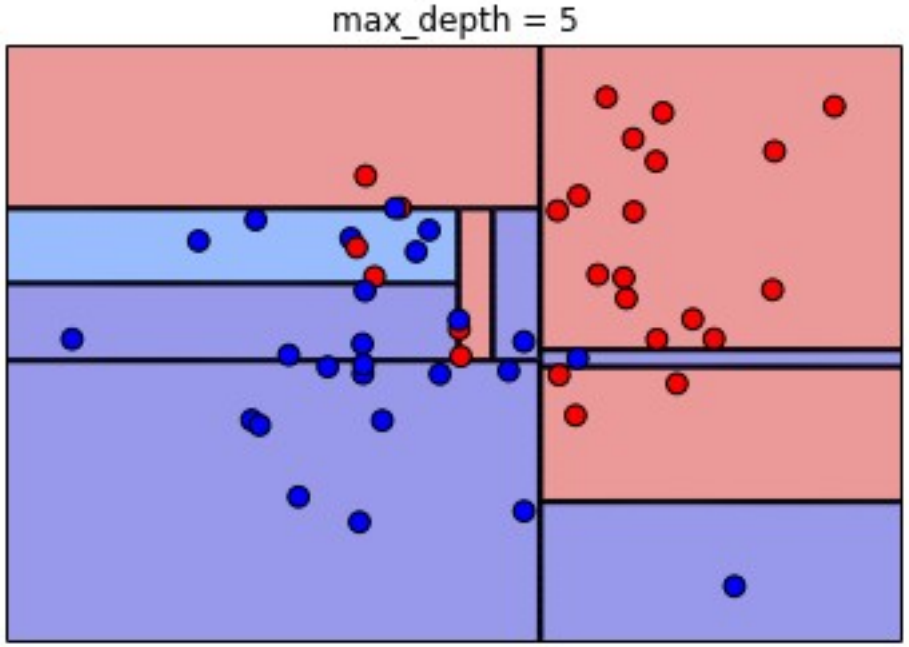


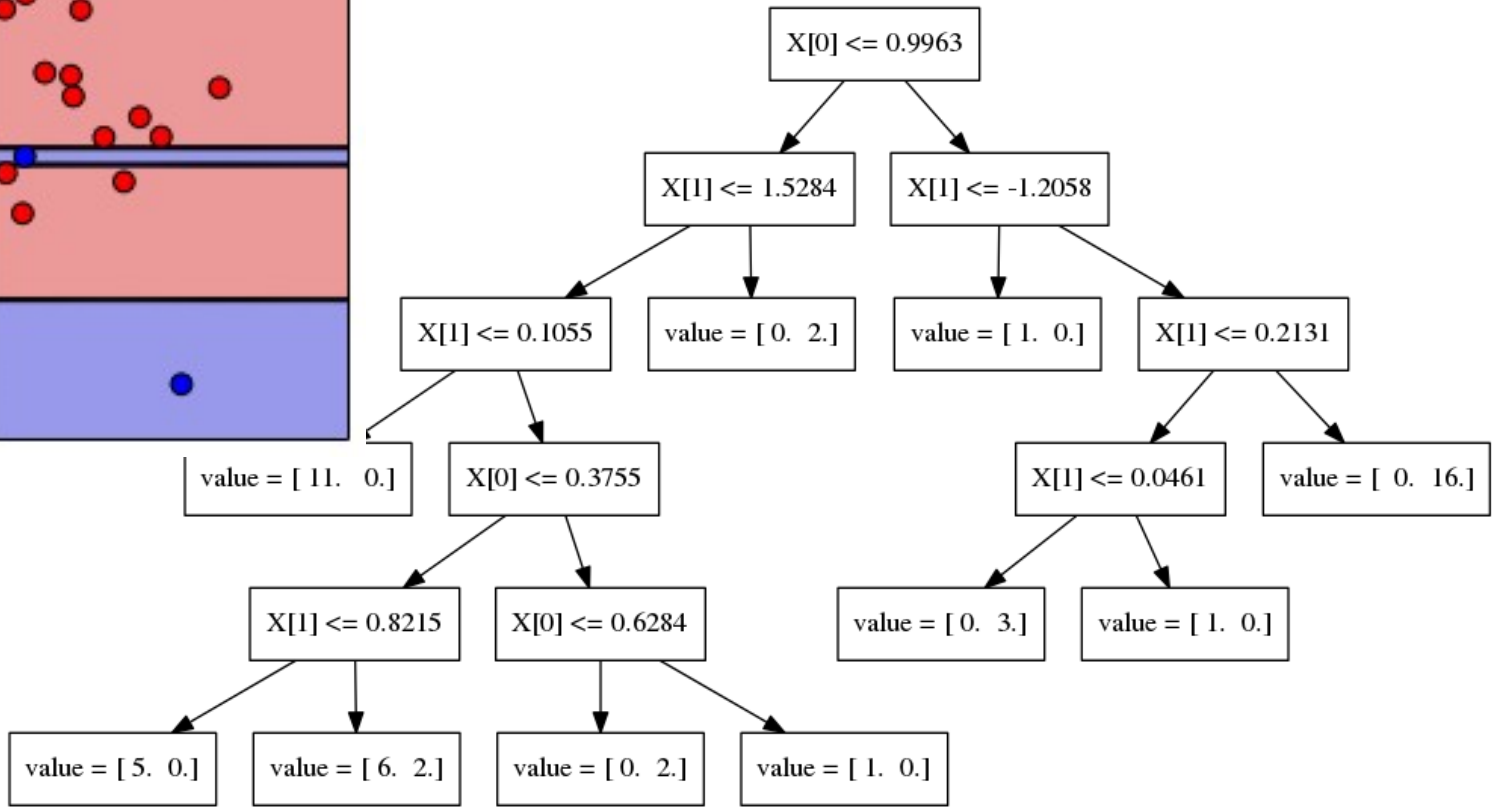
Decision Trees



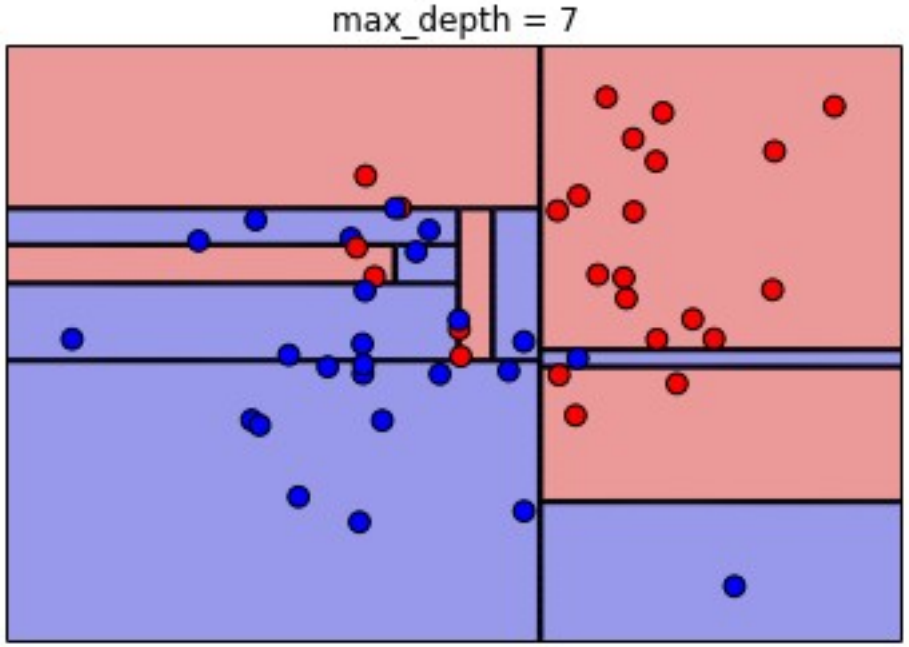


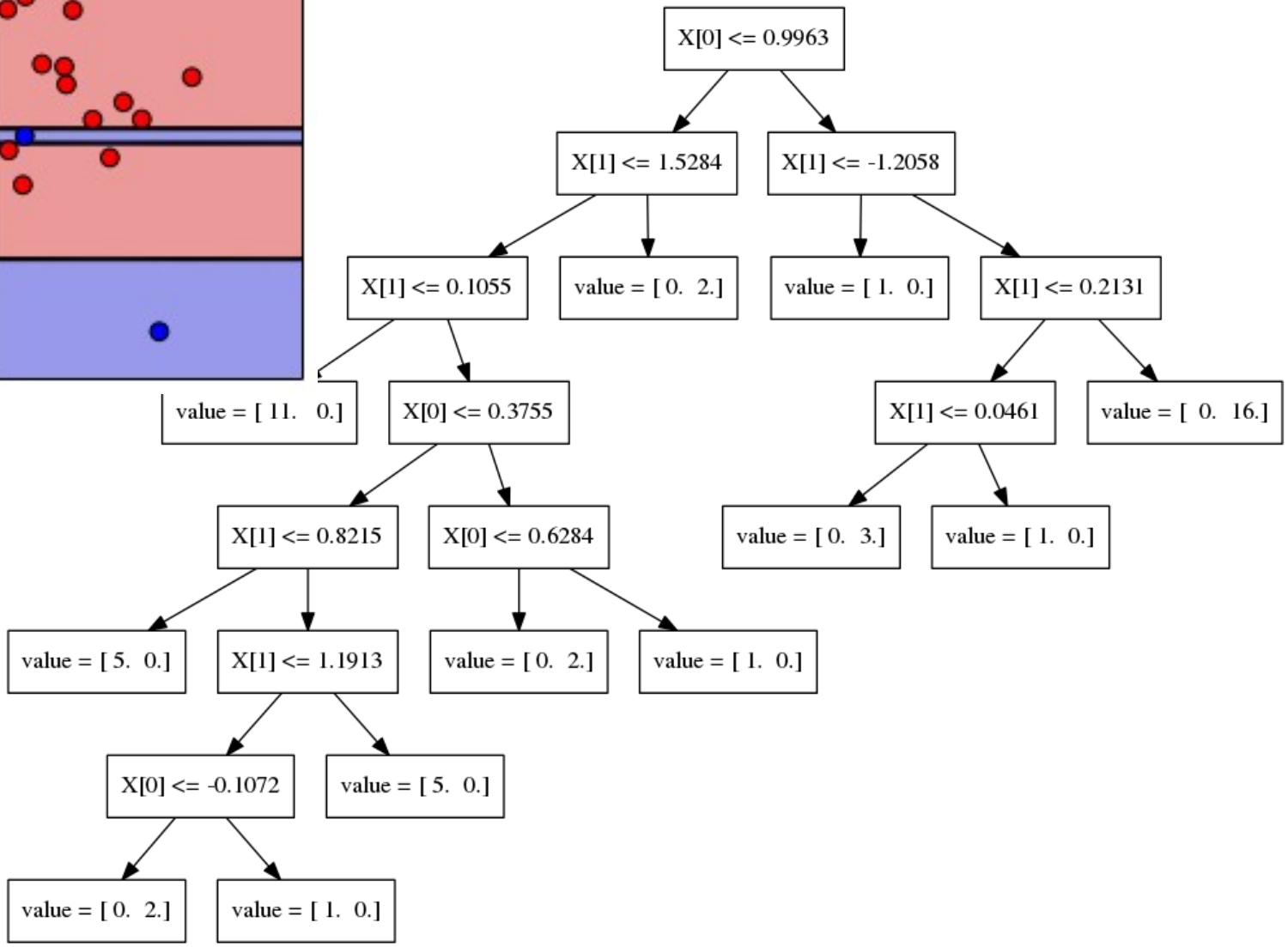
Decision Trees



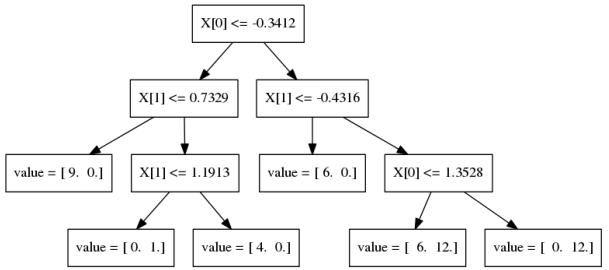


Decision Trees

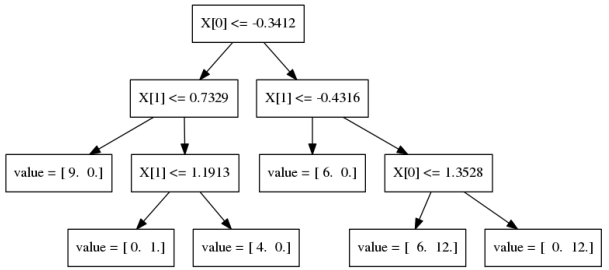


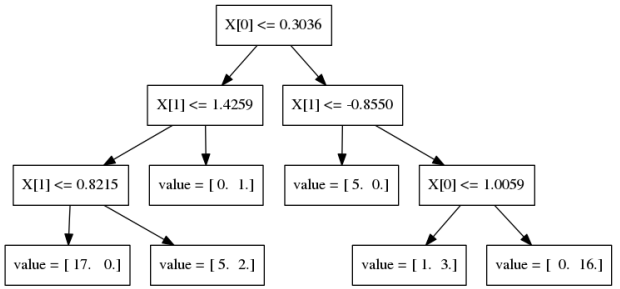


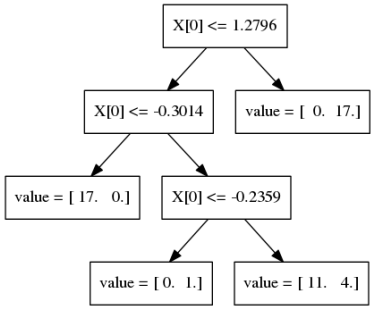
Random Forests



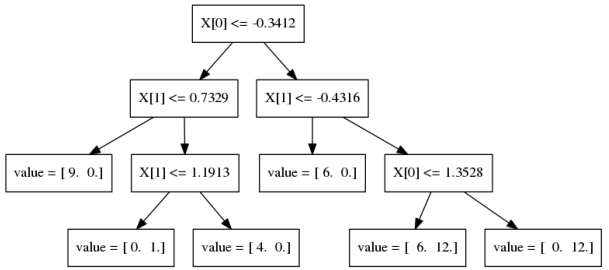
Random Forests

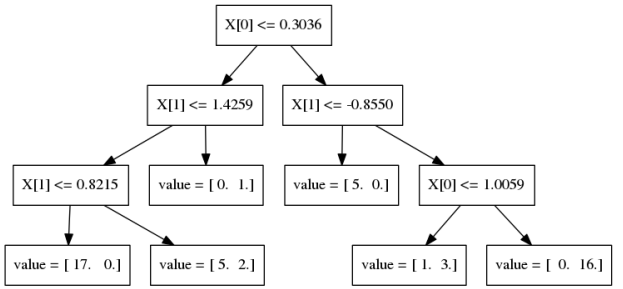


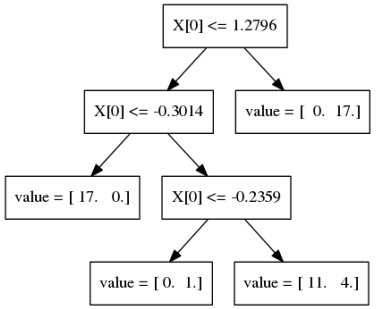


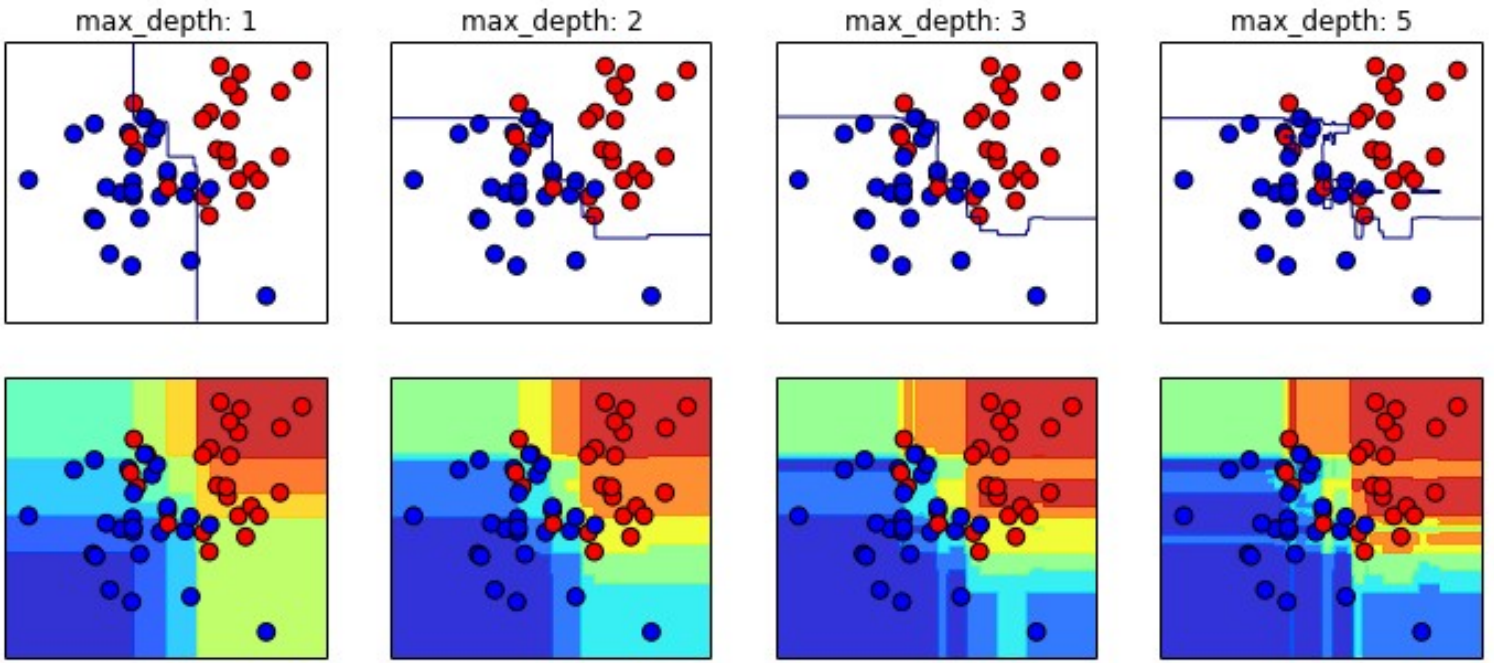


Random Forests





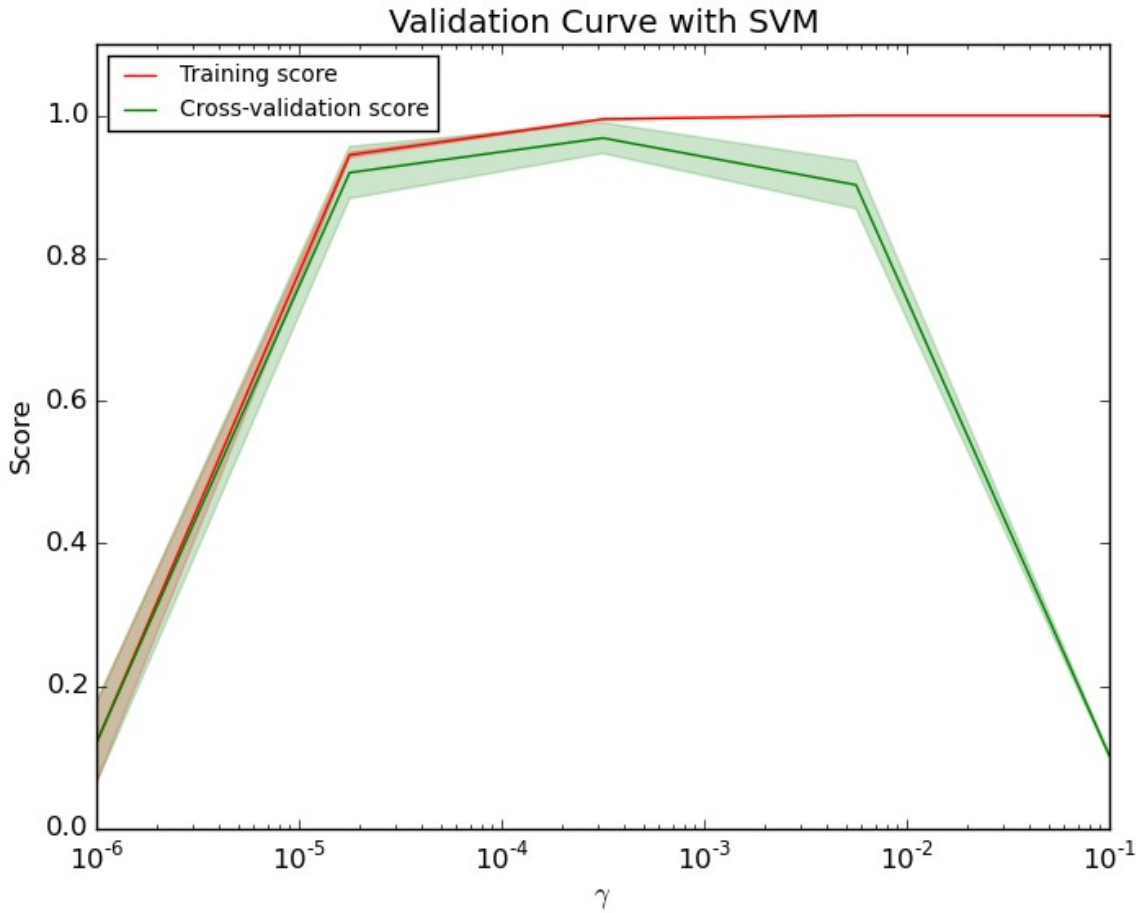




Know where you are on the bias-variance tradeoff

Validation Curves

```
train_scores, test_scores = validation_curve(SVC(), X, y,
param_name="gamma", param_range=param_range)
```



Learning Curves

```
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y,train_sizes=train_sizes)
```