Advanced Scikit-Learn

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

Overview

- Reminder: Basic sklearn concepts
- Model building and evaluation:
  - Pipelines and Feature Unions
  - Randomized Parameter Search
  - Scoring Interface
  - Bias Variance Tradeoff
- Out of Core learning
  - Feature Hashing
  - Kernel Approximation
- PyStruct – Structured Prediction in Python

Get the notebooks!

nbviewer FAQ IPython

pydata-nyc-advanced-sklearn / Chapter 0 - Reminder.ipynb /

Scikit-Learn is simple

Classification

In [4]:
```
from sklearn.datasets import load_iris
from sklearn.cross_validation import train_test_split


iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

In [5]:
```
from sklearn.svm import SVC
clf = SVC()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Transformations

In [6]:
```
from sklearn.decomposition import PCA
```

In [7]:
```
pca = PCA(n_components=2)
pca.fit(X)
X_pca = pca.transform(X)
```

Tools

Cross-validation scoring

In [30]:
```
from sklearn.cross_validation import cross_val_score, StratifiedKFold
scores = cross_val_score(SVC(), X_train, y_train, cv=5)
print(scores)
```

```
[ 0.95652174  1.          0.95652174  0.91304348  0.9        ]
```

https://github.com/amueller/pydata-nyc-advanced-sklearn
http://nbviewer.ipython.org/github/amueller/pydata-nyc-advanced-sklearn

Reminder: Estimators

Classification / Regression

```
from sklearn.svm import SVC
clf = SVC()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Reminder: Estimators

Classification / Regression

```
from sklearn.svm import SVC  # import
clf = SVC()                  # instantiate
clf.fit(X_train, y_train)    # fit
y_pred = clf.predict(X_test) # apply
```

Reminder: Estimators

Classification / Regression

```
from sklearn.svm import SVC  # import
clf = SVC()                  # instantiate
clf.fit(X_train, y_train)    # fit
y_pred = clf.predict(X_test) # apply
```

Transformations

```
from sklearn.decomposition import PCA  # import
trans = PCA(n_components=2)            # instantiate
trans.fit(X)                           # fit
X_pca = clf.transform(X)               # apply
```

Reminder: Tools

Cross -Validation

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(SVC(), X_train, y_train, cv=5)
print(scores)

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

Reminder: Tools

Cross -Validation

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(SVC(), X_train, y_train, cv=5)
print(scores)

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

cv_ss = ShuffleSplit(len(X_train))
scores_shuffle_split = cross_val_score(SVC(), X_train,
y_train, cv=cv_ss)
```

Reminder: Tools

Cross -Validation

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(SVC(), X_train, y_train, cv=5)
print(scores)

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

cv_ss = ShuffleSplit(len(X_train))
scores_shuffle_split = cross_val_score(SVC(), X_train,
y_train, cv=cv_ss)

cv_labels = LeaveOneLabelOut(labels)
scores_pout = cross_val_score(SVC(), X_train, y_train,
cv=cv_labels)
```

Reminder: Tools

Cross -Validated Grid Search

```
from sklearn.grid_search import GridSearchCV
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)
```

Reminder: Tools

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)
```

Do cross-validation over all steps.
Keep a separate test set till the very end.

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

```
char_and_word = make_union(TfidfVectorizer(analyzer="char"),
                           TfidfVectorizer(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__tfidfvectorizer-1__ngram_range':
               [(1, 3), (1, 5), (2, 5)],
               'featureunion__tfidfvectorizer-2__ngram_range':
               [(1, 1), (1, 2), (2, 2)],
               'linearsvc__C': 10. ** np.arange(-3, 3)}
```

Randomized Parameter Search

Randomized Parameter Search

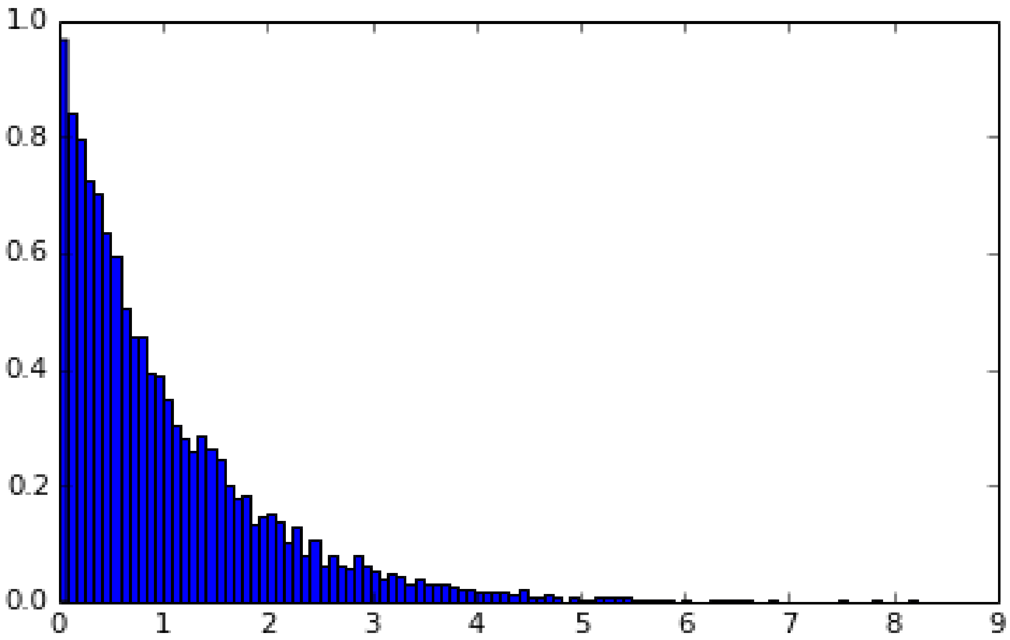
```
params = {'featureunion__tfidfvectorizer-1__ngram_range':
          [(1, 3), (1, 5), (2, 5)],
          'featureunion__tfidfvectorizer-2__ngram_range':
          [(1, 1), (1, 2), (2, 2)],
          'linearsvc__C': 10. ** np.arange(-3, 3)}
```

Randomized Parameter Search

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

Randomized Parameter Search

```
params = {'featureunion__tfidfvectorizer-1__ngram_range':
          [(1, 3), (1, 5), (2, 5)],
          'featureunion__tfidfvectorizer-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

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

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

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

Randomized Parameter Search

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

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

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

Randomized Parameter Search

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

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

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

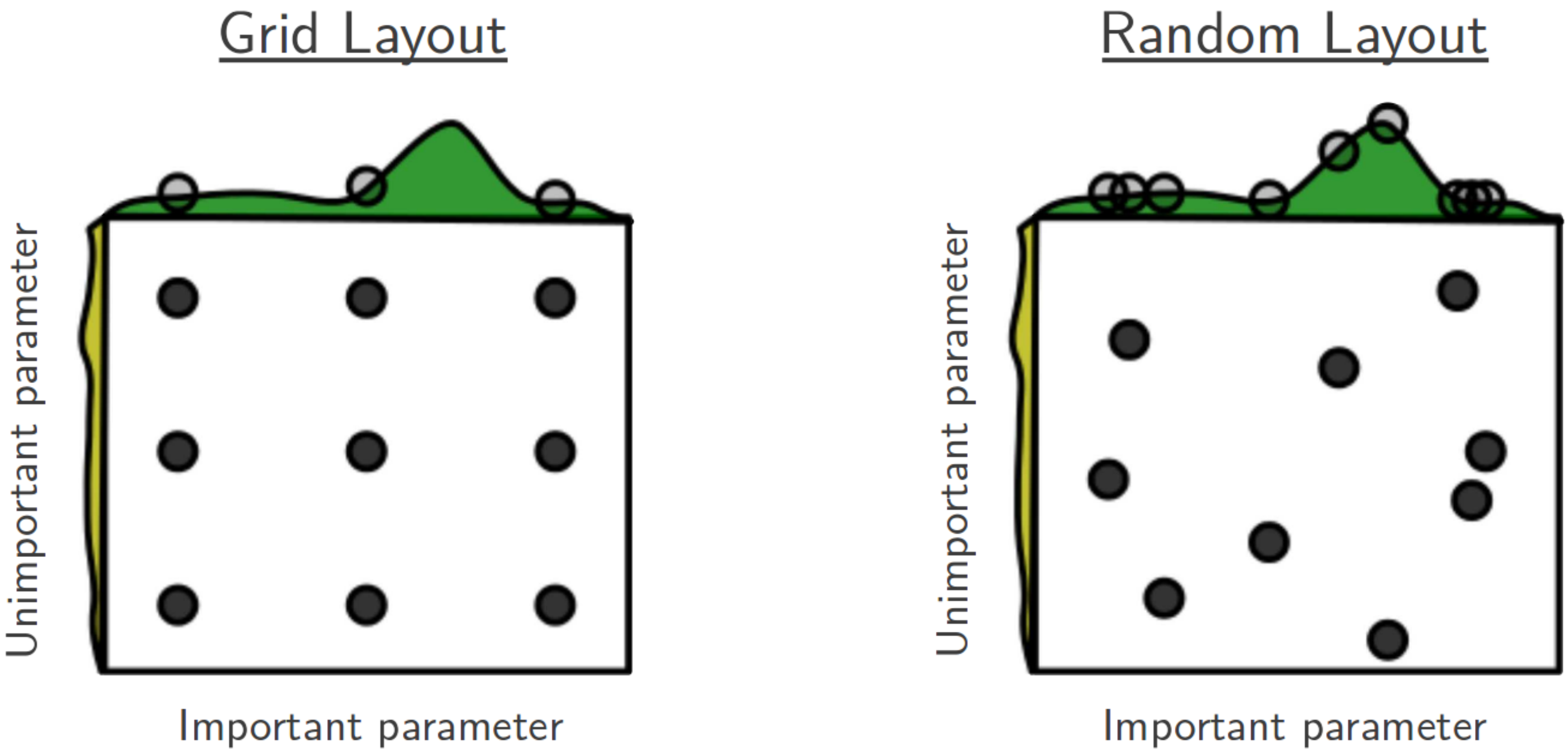
Randomized Parameter Search

```
params = {'featureunion__tfidfvectorizer-1__ngram_range':
          [(1, 3), (1, 5), (2, 5)],
          'featureunion__tfidfvectorizer-2__ngram_range':
          [(1, 1), (1, 2), (2, 2)],
          'linearsvc__C': expon(), 'linearsvc__tol':
expon(scale=0.001)}

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

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

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

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

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

Bias and Variance

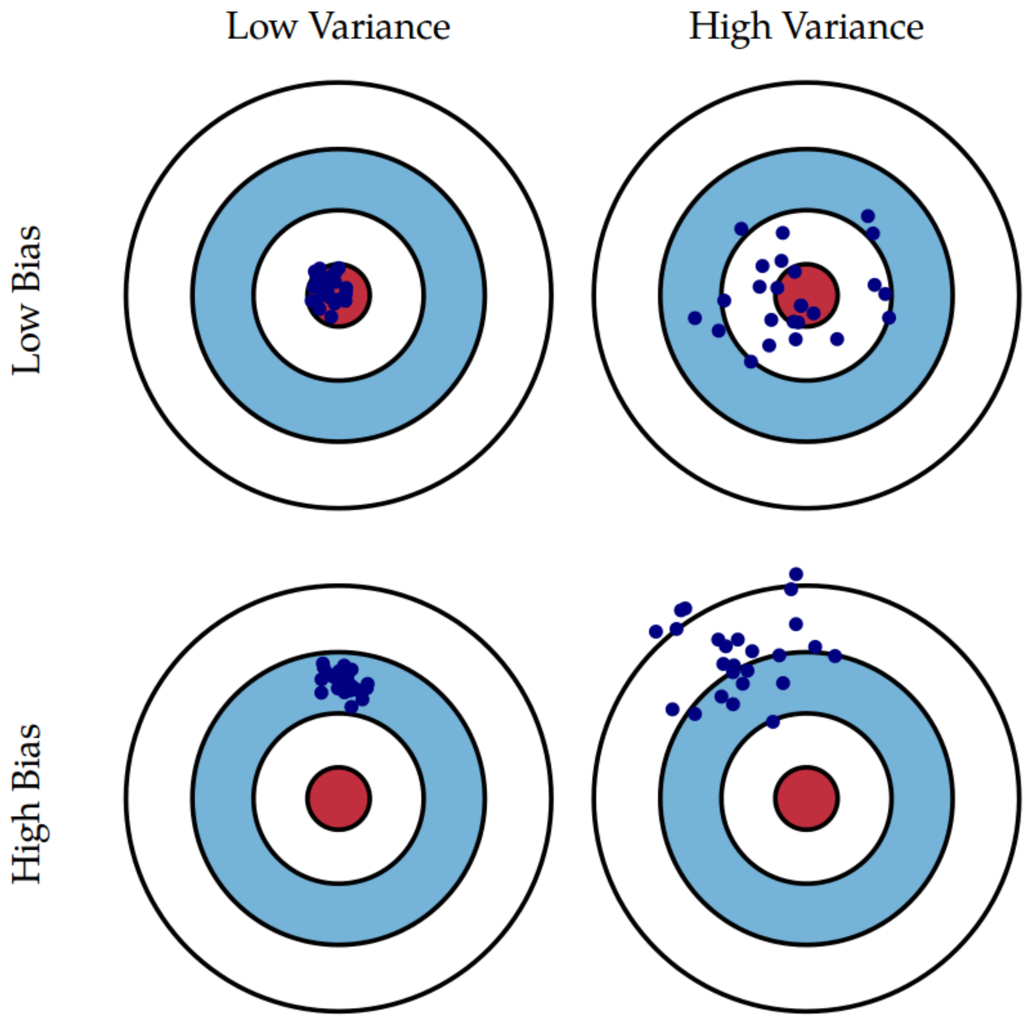


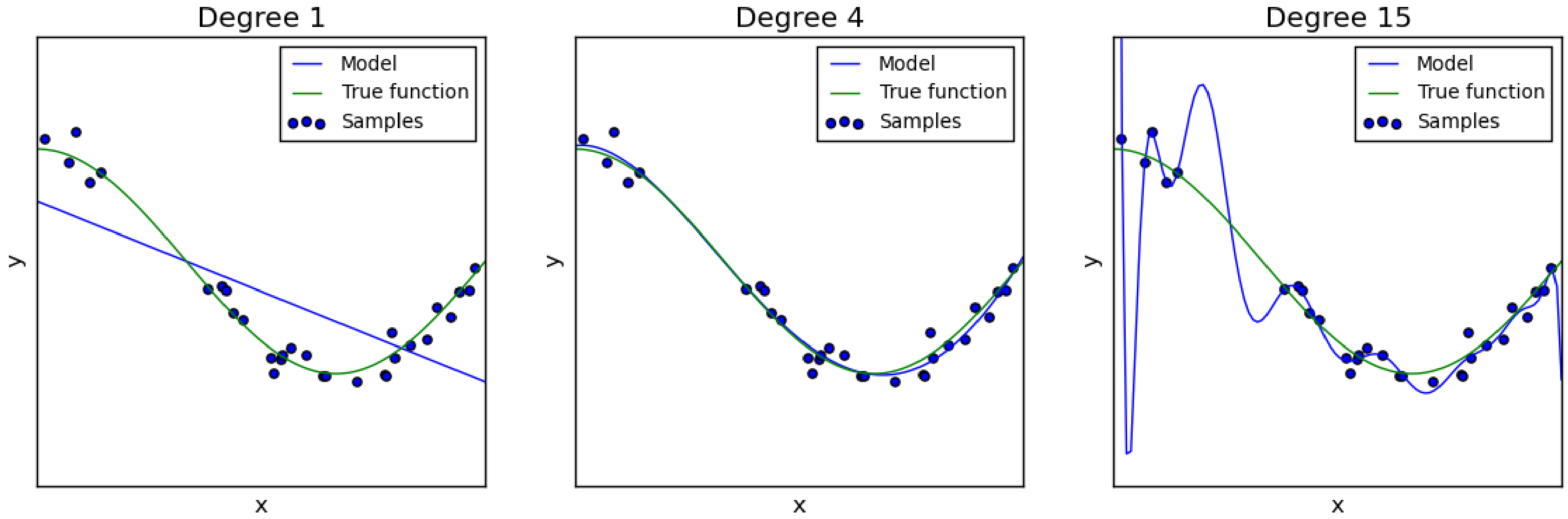
Illustration: Scott Fortmann-Roe

Bias and Variance

$$\begin{aligned}
\mathrm{E}[(y - \hat{f}(x))^2] &= \mathrm{E}[(f(x) - \mathrm{E}[\hat{f}(x)])^2] \\
&\quad + \mathrm{E}[(\hat{f}(x) - \mathrm{E}[\hat{f}(x)])^2] + \mathrm{E}[\epsilon^2] \\
&= \mathrm{Bias}(\hat{f}(x))^2 + \mathrm{Var}(\hat{f}(x)) + \sigma^2
\end{aligned}$$

$$y_i = f(x_i) + \epsilon$$

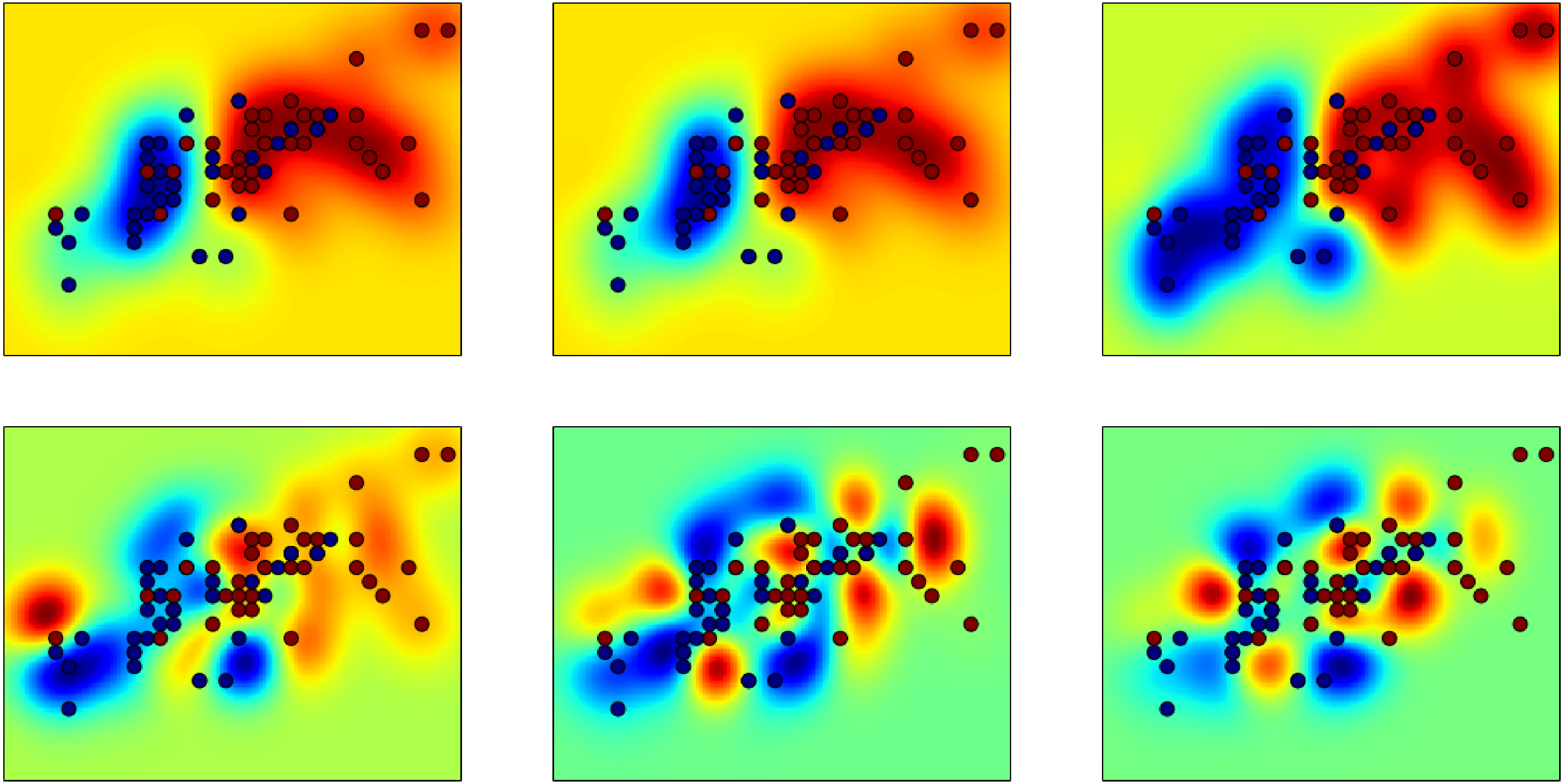
Curve (over and under) fitting



High bias
Low variance

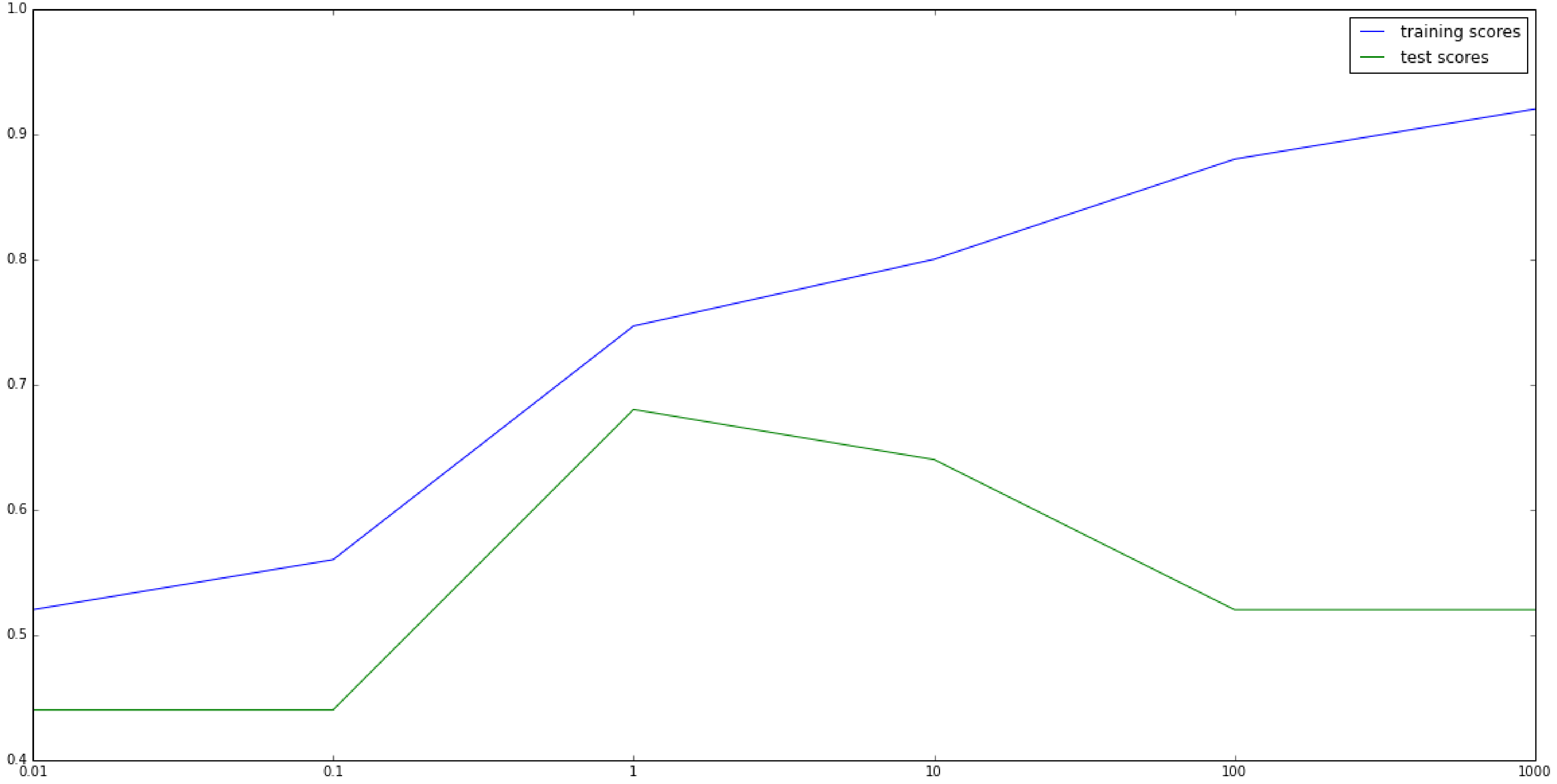
Low bias
High variance

RBF-SVM wigglyness



Bias Variance Tradeoff

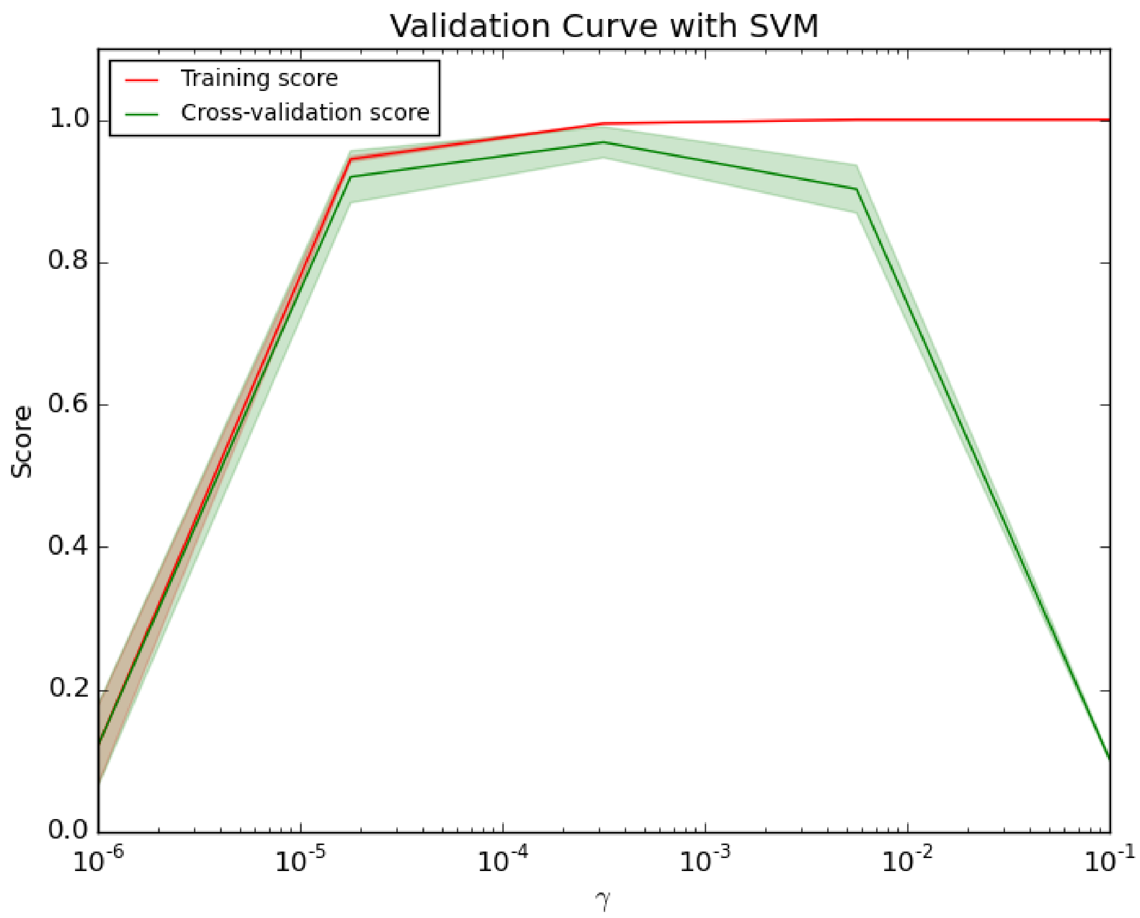
aka overfitting vs underfitting



Know where you are on the bias-variance tradeoff

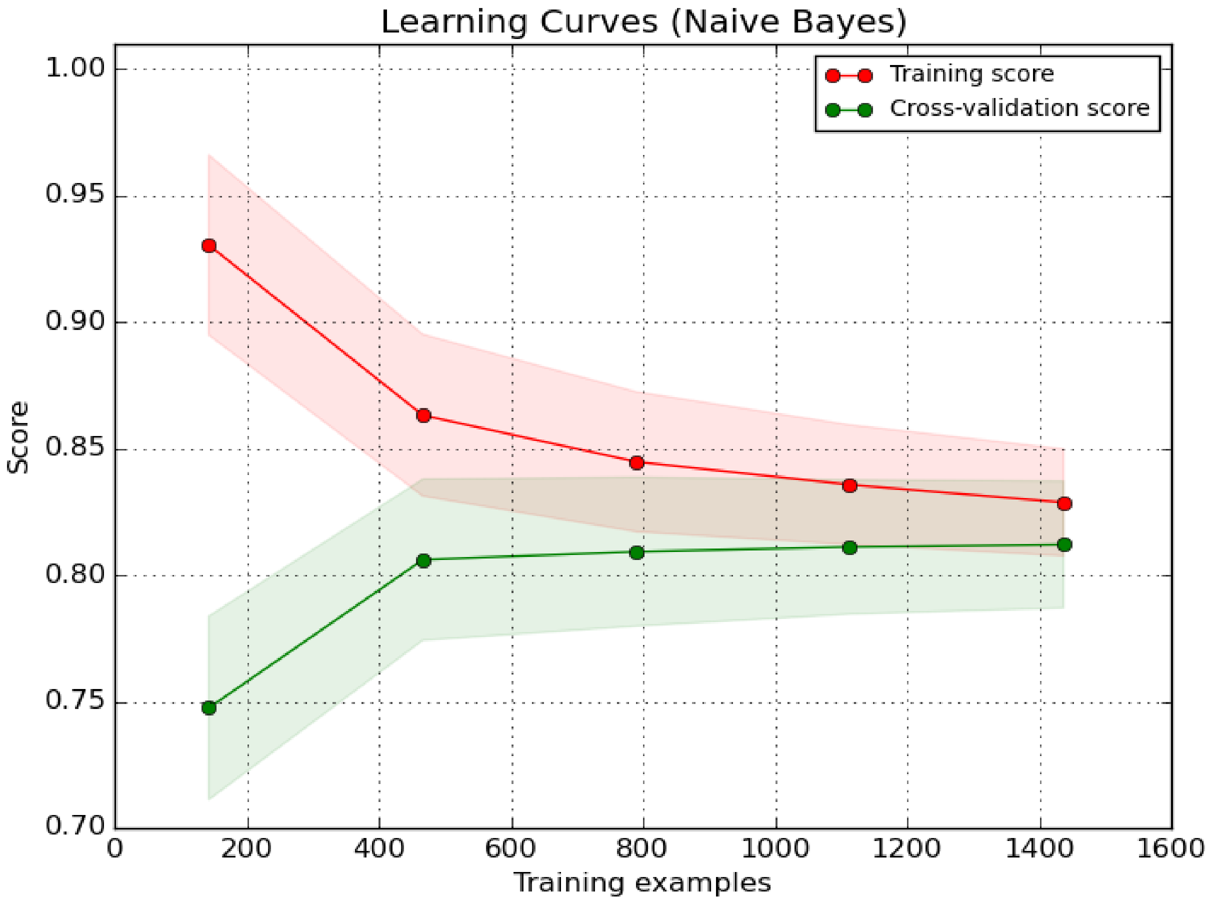
Validation Curves

```
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)
```



Scoring Functions

GridSeachCV
cross_val_score
validation_curve

Default:
Accuracy (classification)
R2 (regression)

Scoring

Motivation: Imbalanced data 1:9

```
cross_val_score(SVC(), X_train, y_train)

>>> array([ 0.9,  0.9,  0.9])
```

Scoring

Motivation: Imbalanced data 1:9

```
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

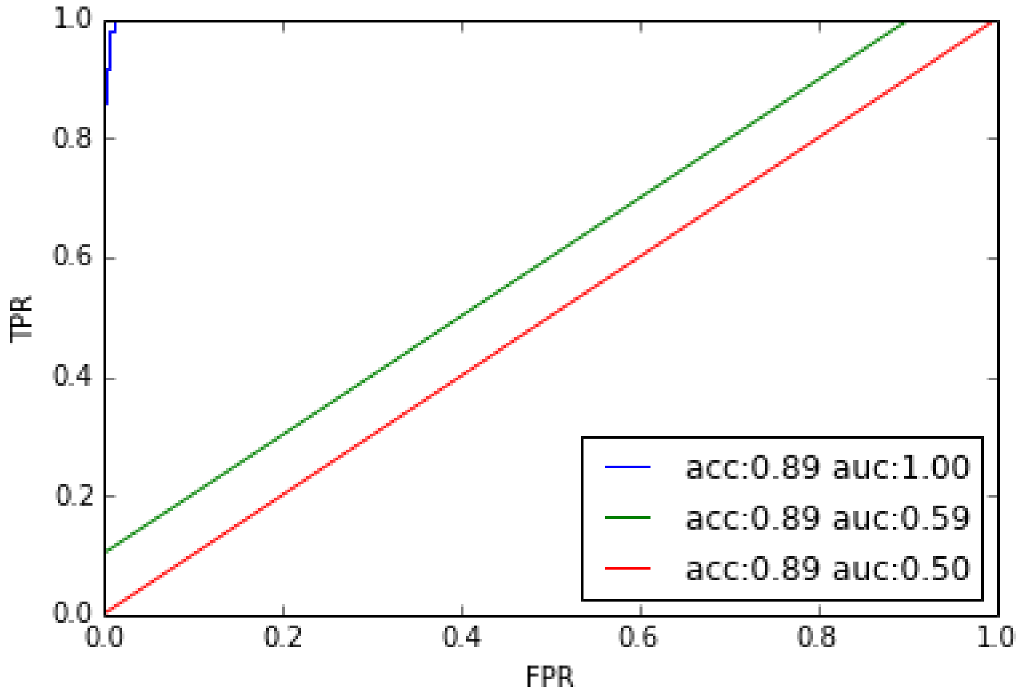
Motivation: Imbalanced data 1:9

```
cross_val_score(SVC(), 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

Motivation: Imbalanced data 1:9

```
cross_val_score(SVC(), 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

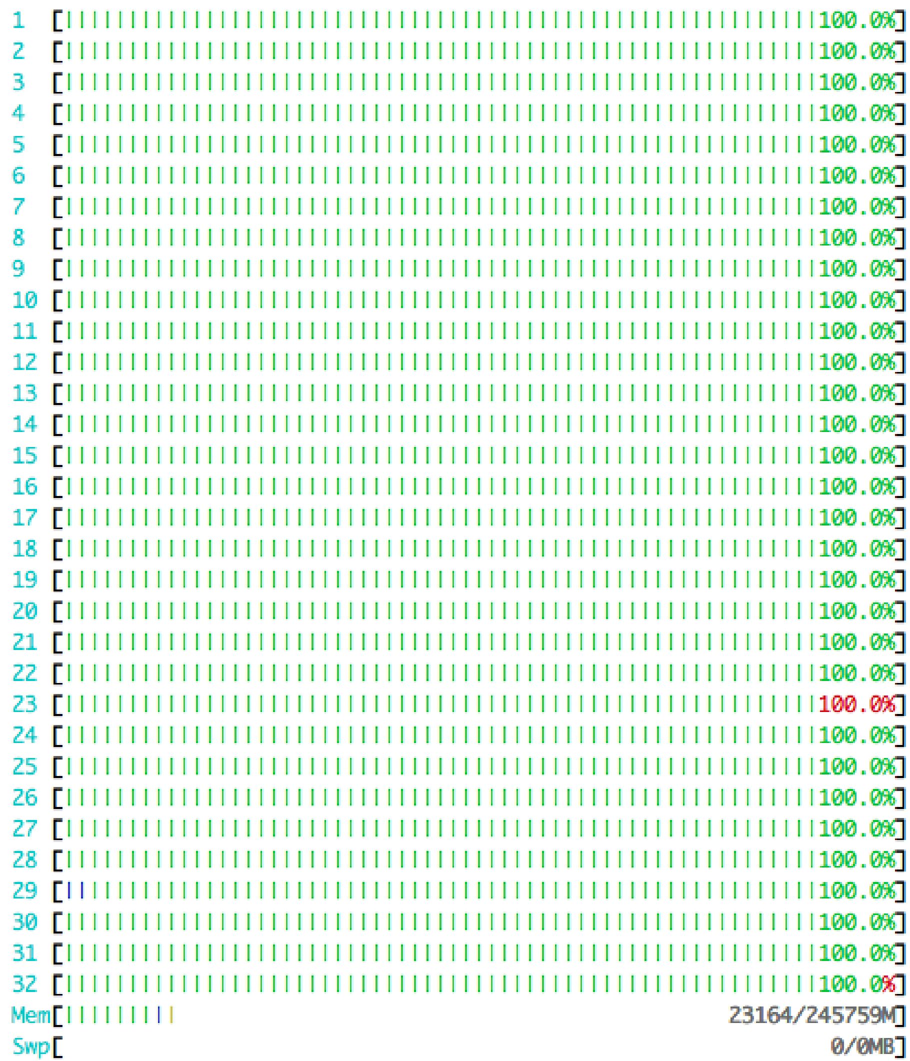
```python
def my_super_scoring(est, X, y):
    return accuracy_scorer(est, X, y) - np.sum(est.coef_ != 0)
```

Defining your own scoring

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


def scoring_function(y_true, y_pred):
    return (np.abs(y_true – y_pred) < 2).mean()

tolerant_scoring = make_scorer(scoring_function)
```

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`

`“You better call Kenny Loggins”`

tokenizer

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

Sparse matrix encoding

aardvak better call you zyxst
[0, …, 0, 1, 0, … , 0, 1 , 0, …, 0, 1, 0, …., 0]

Hacking Trick

`HashingVectorizer`

`“You better call Kenny Loggins”`

↓ tokenizer

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

↓ hashing

[hash('you'), hash('better'), hash('call'), hash('kenny'), hash('loggins')]
= [832412, 223788, 366226, 81185, 835749]

↓ Sparse matrix encoding

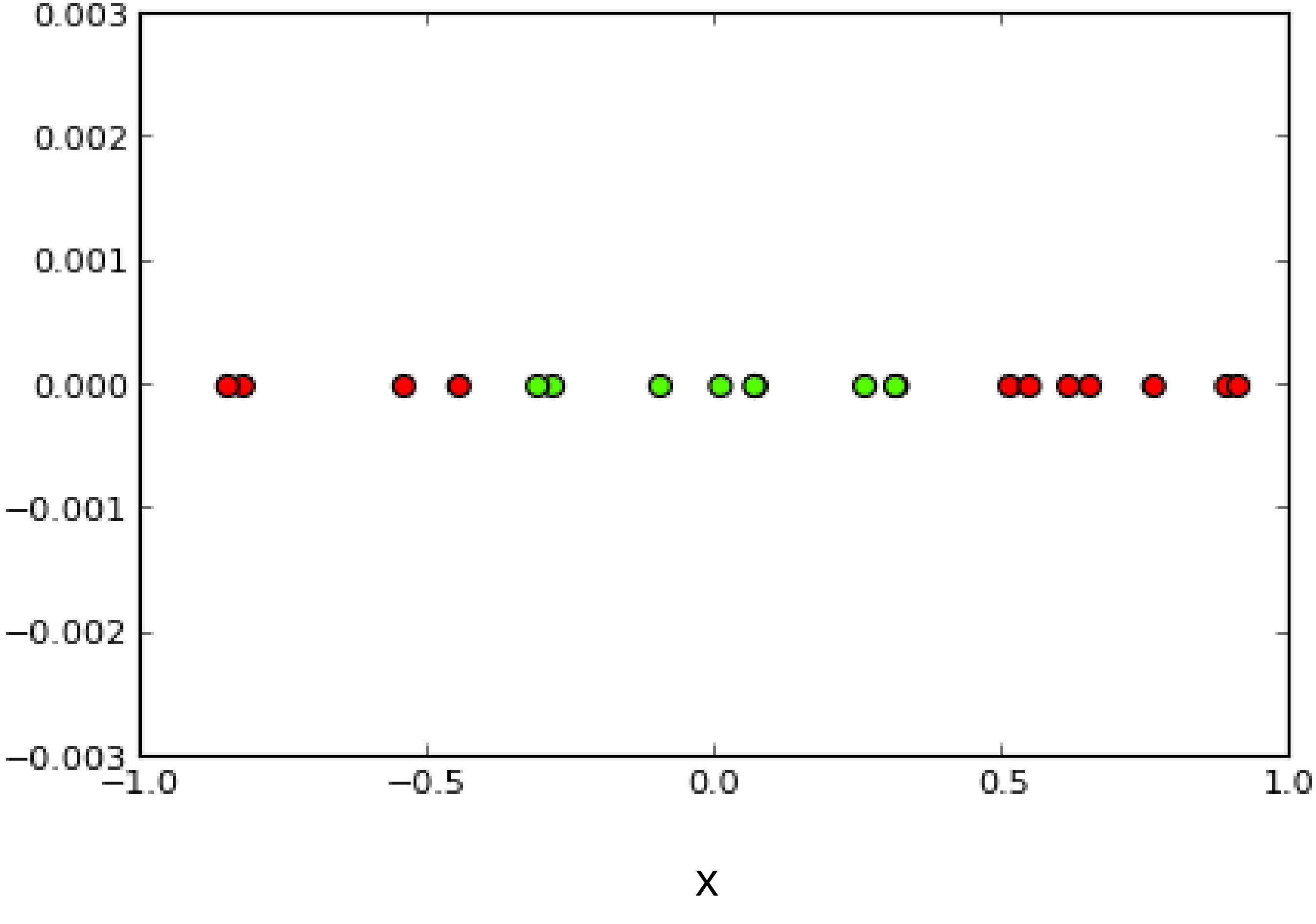
[0, …, 0, 1, 0, … , 0, 1 , 0, …, 0, 1, 0, ... 0]

Out of Core Text Classification

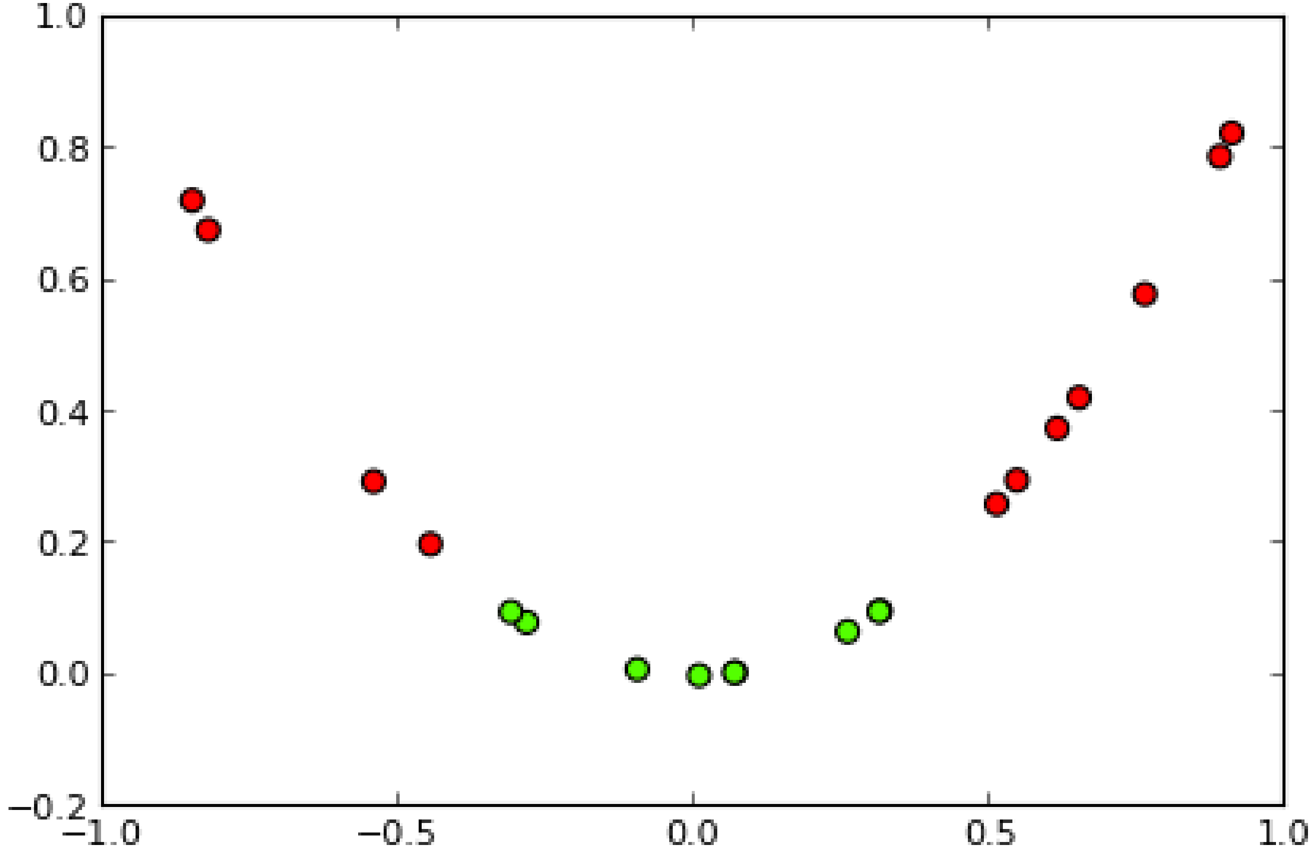
```
sgd = SGDClassifier()
hashing_vectorizer = HashingVectorizer()

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))
```

Questions so far?

PyStruct – Structured Prediction in Python

Structured Prediction

$$y = (y_1, y_2, \dots y_{n_k})$$

Applications: Multi-Label Classification

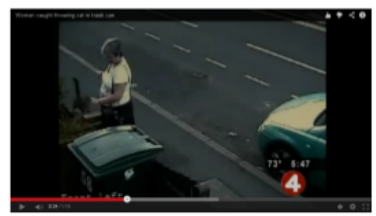
	Politics	Sports	Finance	Domestic	Religion
News Story1	1	0	0	1	1
News Story2	0	1	0	1	0
News Story3	0	0	1	0	0

	Owns Car	Smokes	Married	Self-Employed	Has Kids
Customer1	1	0	1	0	1
Customer2	1	1	0	1	0
Customer3	0	1	1	0	0

Applications: Sequence Tagging


Stroke cat.


Stroke cat.


Stroke cat.


Open trash can.


Put cat in trash can.

Applications: Image Segmentation

Pairwise Structured Models

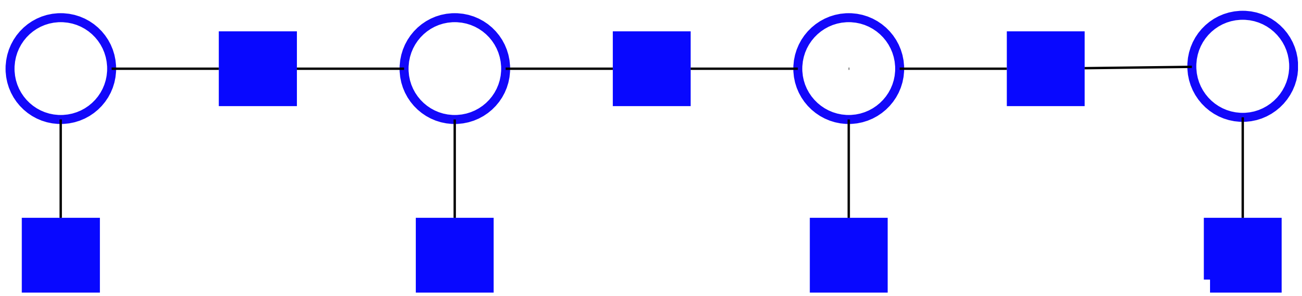
$$\arg\max_{y_1, y_2, \ldots, y_n} w^T \psi(x, y)$$

$$= \arg\max_{y_1, y_2, \ldots, y_n} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j) \in E} w_{i,j}^T \psi(x, y_i, y_j)$$

Pairwise Structured Models

$$\underset{y_1, y_2, \ldots, y_n}{\arg\max}\ w^T \psi(x, y)$$

$$= \underset{y_1, y_2, \ldots, y_n}{\arg\max} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j) \in E} w_{i,j}^T \psi(x, y_i, y_j)$$



PyStruct Architecture

Estimator = Learner + Model + Inference

$$\underset{y_1, y_2, \ldots, y_n}{\arg\max}\; w^T \psi(x, y)$$

PyStruct Architecture

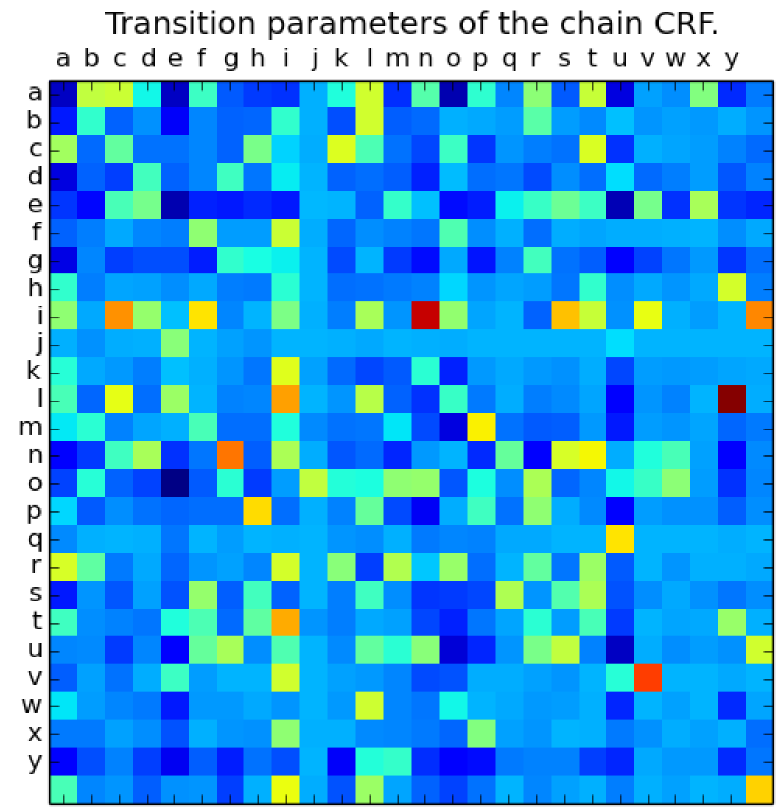
Estimator = Learner + Model + Inference

$$\underset{y_1, y_2, \ldots, y_n}{\arg\max}\; w^T \psi(x, y)$$

```
model = ChainCRF(inference="max_product")
ssvm = OneSlackSSVM(model=model, C=.1, inference_cache=50,,
                    tol=0.1, verbose=3)
ssvm.fit(X_train, y_train)
```

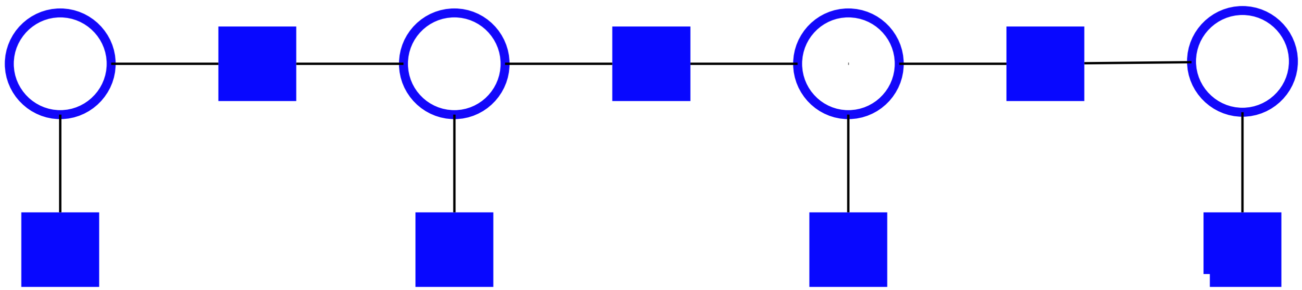
Sequence Tagging example

Sequence Tagging example



The Devil is in the Inference

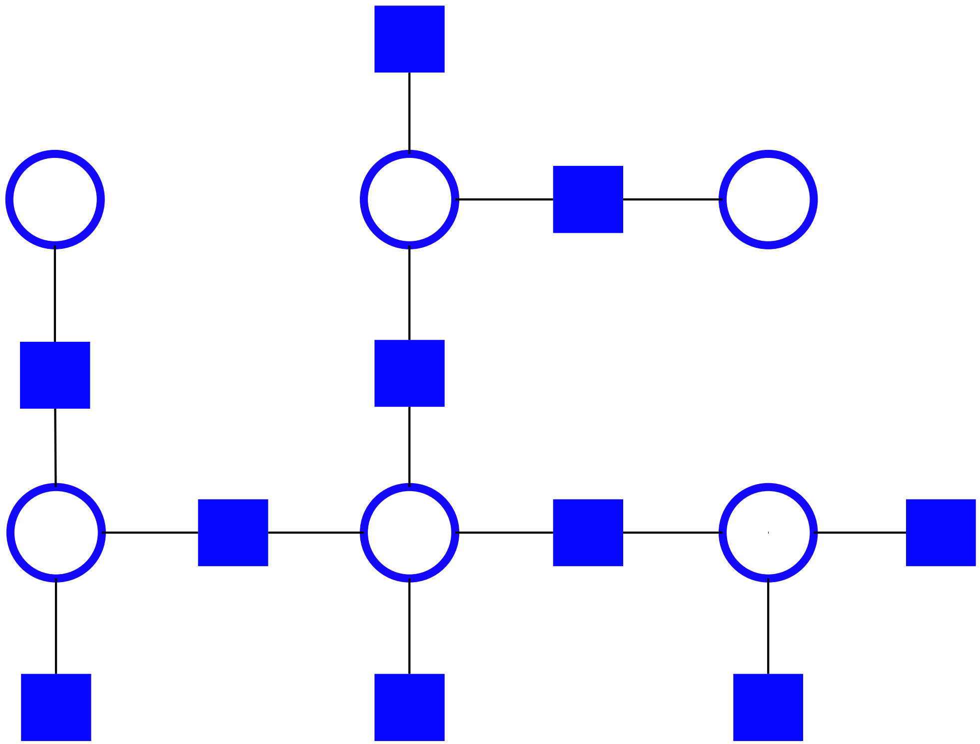
$$\underset{y_1, y_2, \ldots, y_n}{\arg\max} \; w^T \psi(x, y)$$



Easy: Dynamic Programming

The Devil is in the Inference

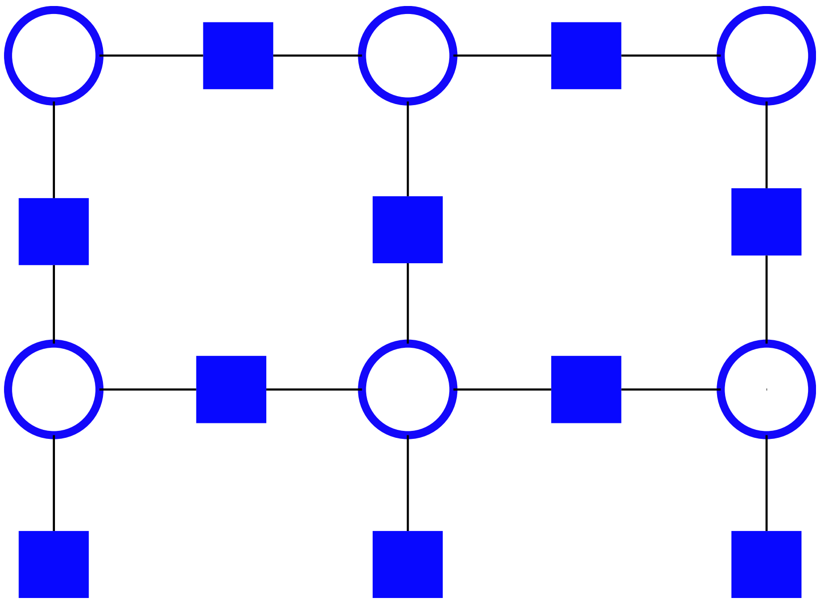
$$\underset{y_1, y_2, \ldots, y_n}{\arg\max}\; w^T \psi(x, y)$$



Easy: Dynamic Programming

The Devil is in the Inference

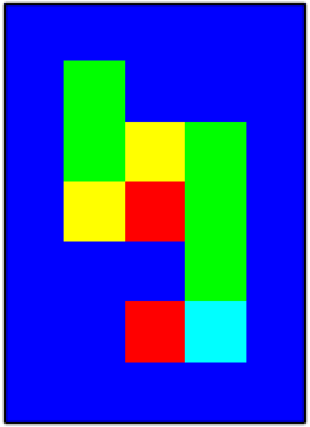
$$\underset{y_1, y_2, \ldots, y_n}{\arg\max}\ w^T \psi(x, y)$$



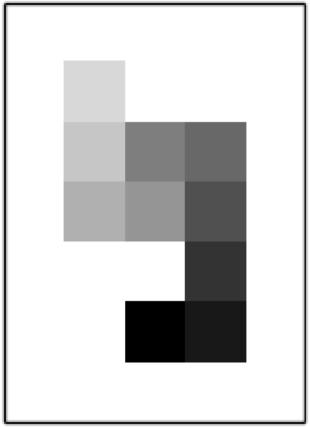
HARD!
AD3, QPBO, LP, Loopy BP, ….

Grid Graphs: Snakes

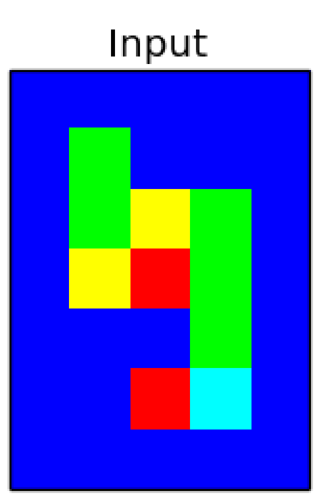
Input

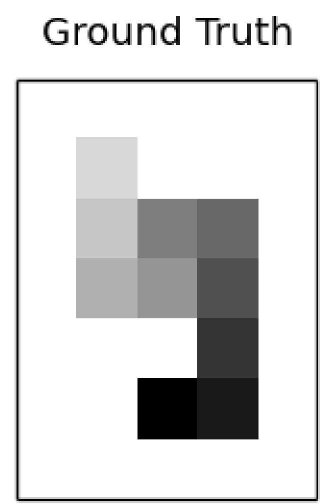


Ground Truth



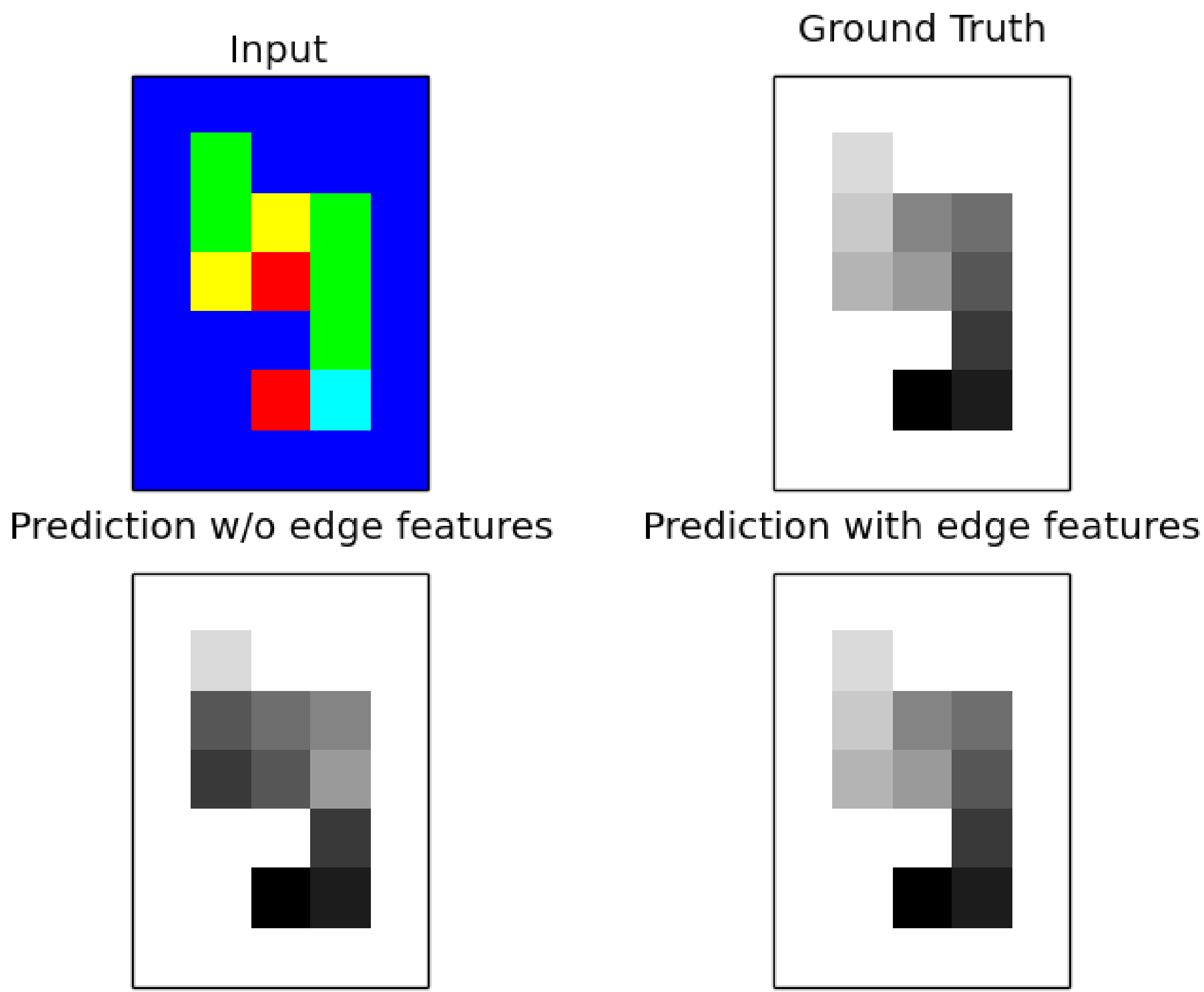
Grid Graphs: Snakes






```
crf = EdgeFeatureGraphCRF(inference_method='qpbo')
ssvm = OneSlackSSVM(crf, inference_cache=50, C=.1, tol=.1,
                    switch_to='ad3', n_jobs=1)
ssvm.fit(X_train_edge_features, Y_train_flat)
```

Grid Graphs: Snakes



Classes of Inference Algorithms

Exact Algorithms
Max-Product (Chains, Trees) `'max-product'`

Exhaustive (usually too expensive)

Relaxed algorithms + branch & bound
`('ad3', {'branch_and_bound': True})`

Relaxed
Linear Programming (slooow) `'lp'`
Dual Decomposition `'ad3'`

Approximate / heuristics
Loopy message passing `'max-product'`
QPBO `'qpbo'`

Classes of Inference Algorithms

Exact Algorithms
Max-Product (Chains, Trees) `'max-product'`

Exhaustive (usually too expensive)

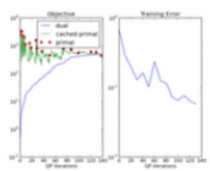
Relaxed algorithms + branch & bound
`('ad3', {'branch_and_bound': True})`

Relaxed
Linear Programming (slooow) `'lp'`
Dual Decomposition `'ad3'`

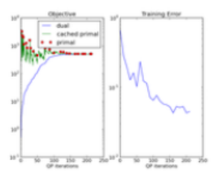
Approximate / heuristics
Loopy message passing `'max-product'`
QPBO `'qpbo'`

Install OpenGM for many more!

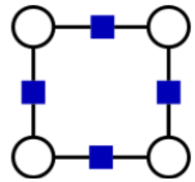
Examples



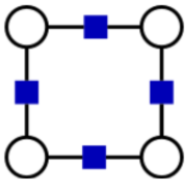
Plotting the objective and constraint caching in 1-slack SSVM



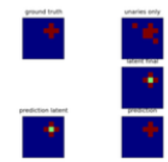
Efficient exact learning of 1-slack SSVMs



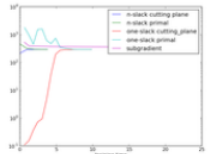
SVM as CRF



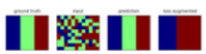
Semantic Image Segmentation on Pascal VOC



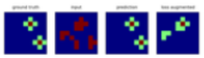
Latent Dynamics CRF



SVM objective values

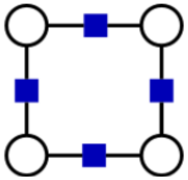


Learning directed interactions on a 2d grid

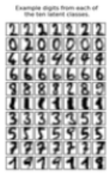


Learning interactions on a 2d grid

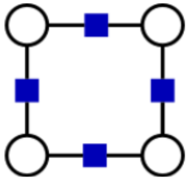
OCR Letter sequence recognition



Crammer-Singer Multi-Class SVM

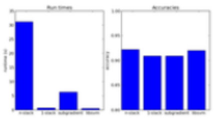


Latent SVM for odd vs. even digit classification

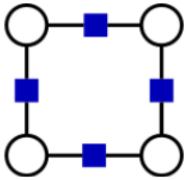


Mult-label classification

Latent Variable Hierarchical CRF



Binary SVM as SSVM



Comparing PyStruct and SVM-Struct

Back to top

Thank you for your attention.