



Machine Learning with scikit-learn

Andreas Mueller

Overview

- Basic concepts of machine learning
- Introduction to scikit-learn
- Some useful algorithms
- Selecting a model
- Working with text data

scikit-learn

- Collection of machine learning algorithms and tools in Python.
- BSD Licensed, used in academia and industry (Spotify, bit.ly, Evernote).
- ~20 core developers.
- Take pride in good code and documentation.
- We want YOU to participate!

Two (three) kinds of learning

- Supervised
- Unsupervised
- Reinforcement

Supervised learning

Training: Examples X_{train} together with labels y_{train} .

Testing: Given X_{test} , predict y_{test} .

Examples

- Classification (spam, sentiment analysis, ...)
- Regression (stocks, sales, ...)
- Ranking (retrieval, search, ...)

Unsupervised Learning

Examples X. Learn something about X.

Examples

- Dimensionality reduction
- Clustering
- Manifold learning

scikit-learn algorithm cheat-sheet

START

classification

- get more data
 - >50 samples
 - predicting a category
 - <100K samples
 - Text Data
 - Naive Bayes (YES)
 - KNeighbors Classifier (NO)
 - SVC Ensemble Classifiers (NOT WORKING)
 - kernel approximation (NOT WORKING)
 - Linear SVC (NOT WORKING)
 - SGD Classifier (NO)
 - SGD Regressor (NOT WORKING)

regression

- few features should be important
 - YES
 - ElasticNet Lasso
 - NO
 - SGD Regressor
- <100K samples
 - YES
 - SVR(kernel='rbf') EnsembleRegressors (NOT WORKING)
 - NO
 - RidgeRegression SVR (kernel='linear')

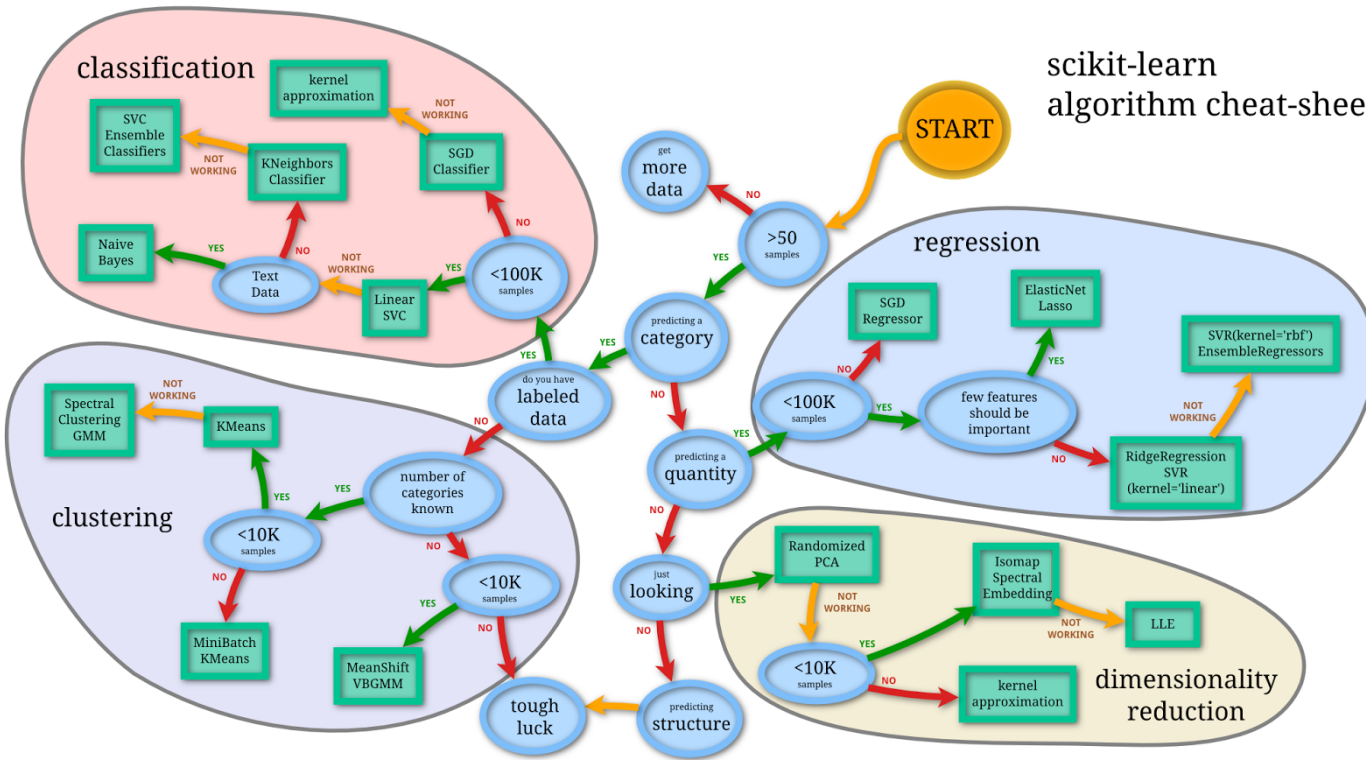
clustering

- number of categories known
 - YES
 - <10K samples
 - MiniBatch KMeans (NO)
 - KMeans
 - Spectral Clustering GMM (NOT WORKING)
 - NO
 - <10K samples
 - MeanShift VBGM

dimensionality reduction

 - just looking
 - YES
 - Randomized PCA
 - <10K samples
 - kernel approximation
 - NO
 - Isomap Spectral Embedding
 - LLE (NOT WORKING)

tough luck



Data representation

Everything is a **numpy array** (or a scipy sparse matrix)!

Let's get some toy data.

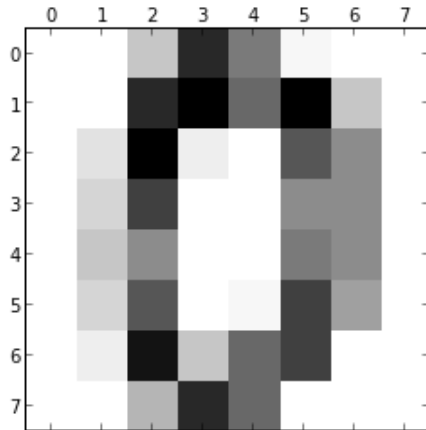
```
In [1]: from sklearn.datasets import load_digits  
digits = load_digits()
```

```
In [2]: print("images shape: %s" % str(digits.images.shape))  
print("targets shape: %s" % str(digits.target.shape))
```

images shape: (1797, 8, 8)

targets shape: (1797,)

```
In [3]: plt.matshow(digits.images[0], cmap=plt.cm.Greys);
```



```
In [4]: digits.target
```



```
Out[4]: array([0, 1, 2, ..., 8, 9, 8])
```

Prepare the data

```
In [6]: X = digits.data.reshape(-1, 64)
        print(X.shape)
```

```
(1797, 64)
```

```
In [7]: y = digits.target
        print(y.shape)
```

```
(1797,)
```

We have 1797 data points, each an 8x8 image -> 64 dimensional vector.

X.shape is always (n_samples, n_feature)

```
In [8]: print(X)
```

```
[[ 0.      0.      0.3125 ...,  0.      0.      0.      ]
 [ 0.      0.      0.      ...,  0.625    0.      0.      ]
 [ 0.      0.      0.      ...,  1.      0.5625  0.      ]
 ...,
 [ 0.      0.      0.0625 ...,  0.375    0.      0.      ]
 [ 0.      0.      0.125   ...,  0.75     0.      0.      ]
 [ 0.      0.      0.625   ...,  0.75     0.0625  0.      ]
 ...
]]
```

Taking a Peek

Dimensionality Reduction and Manifold Learning

- Always first have a look at your data!
- Projecting to two dimensions is the easiest way.

Principal Component Analysis (PCA)

In [9]: `from sklearn.decomposition import PCA`

Instantiate the model. Set parameters.

In [10]: `pca = PCA(n_components=2)`

Fit the model.

In [11]: `pca.fit(X);`

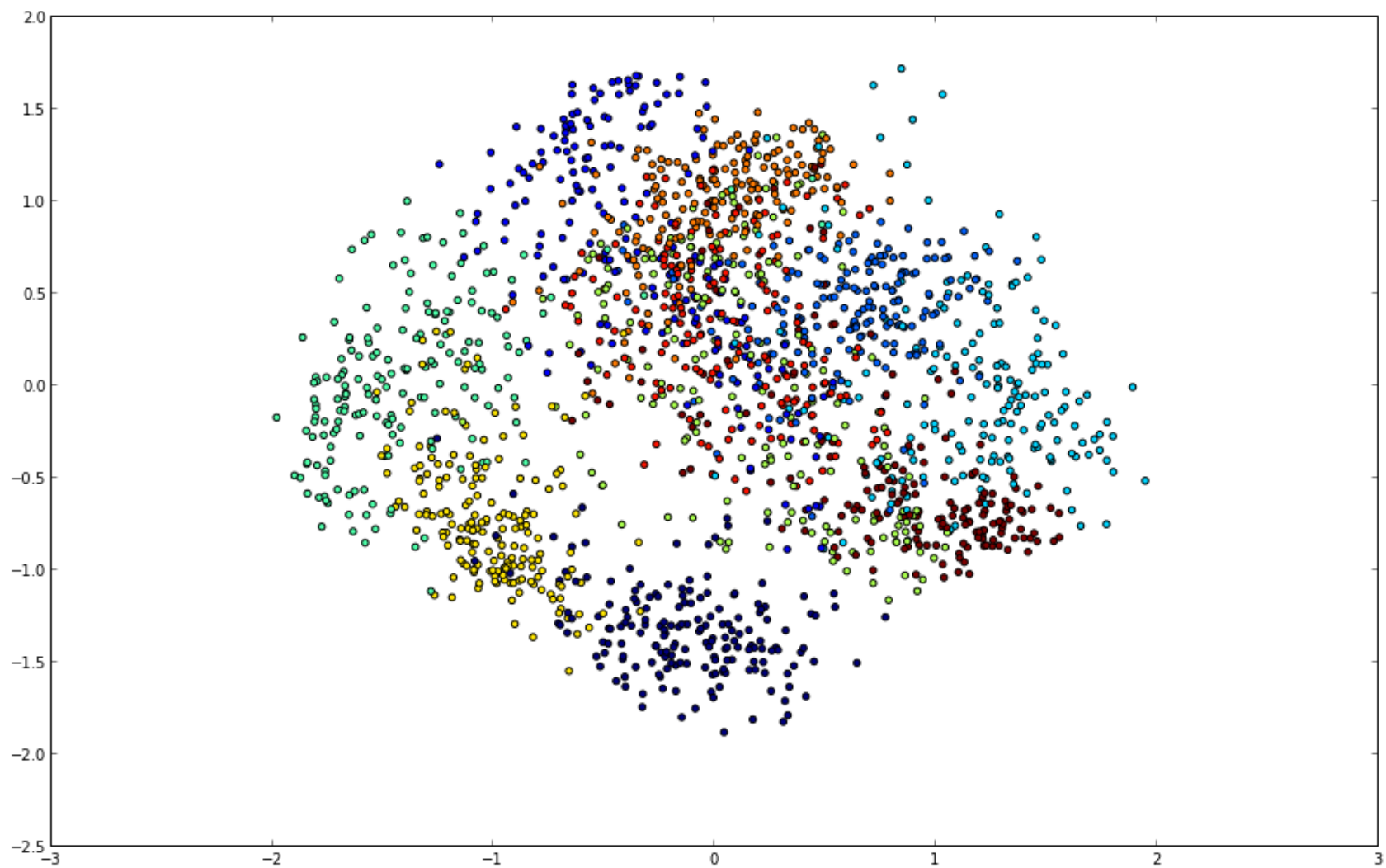
Apply the model. For embeddings / decompositions, this is `transform`.

In [12]: `X_pca = pca.transform(X)`
`X_pca.shape`

Out[12]: `(1797, 2)`

In [13]:

```
plt.figure(figsize=(16, 10))  
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y);
```



In [14]:

```
print(pca.mean_.shape)
print(pca.components_.shape)
```

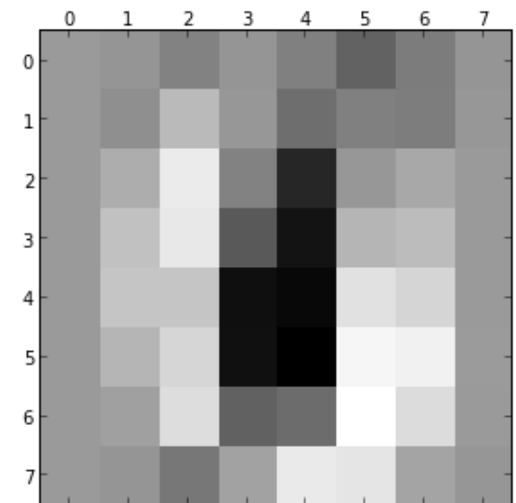
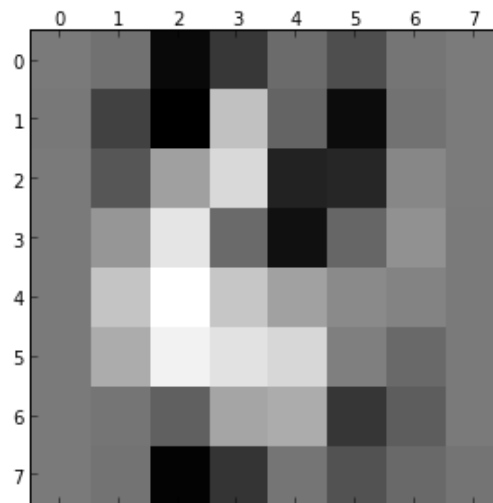
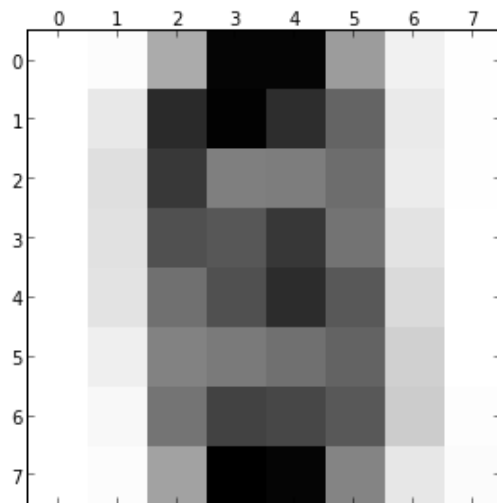
(64,)

(2,

64)

In [15]:

```
fig, ax = plt.subplots(1, 3)
ax[0].matshow(pca.mean_.reshape(8, 8), cmap=plt.cm.Greys)
ax[1].matshow(pca.components_[0, :].reshape(8, 8), cmap=plt.cm.Greys)
ax[2].matshow(pca.components_[1, :].reshape(8, 8), cmap=plt.cm.Greys);
```



Isomap

In [16]: `from sklearn.manifold import Isomap`

Instantiate the model. Set parameters.

In [17]: `isomap = Isomap(n_components=2, n_neighbors=20)`

Fit the model.

In [18]: `isomap.fit(X);`

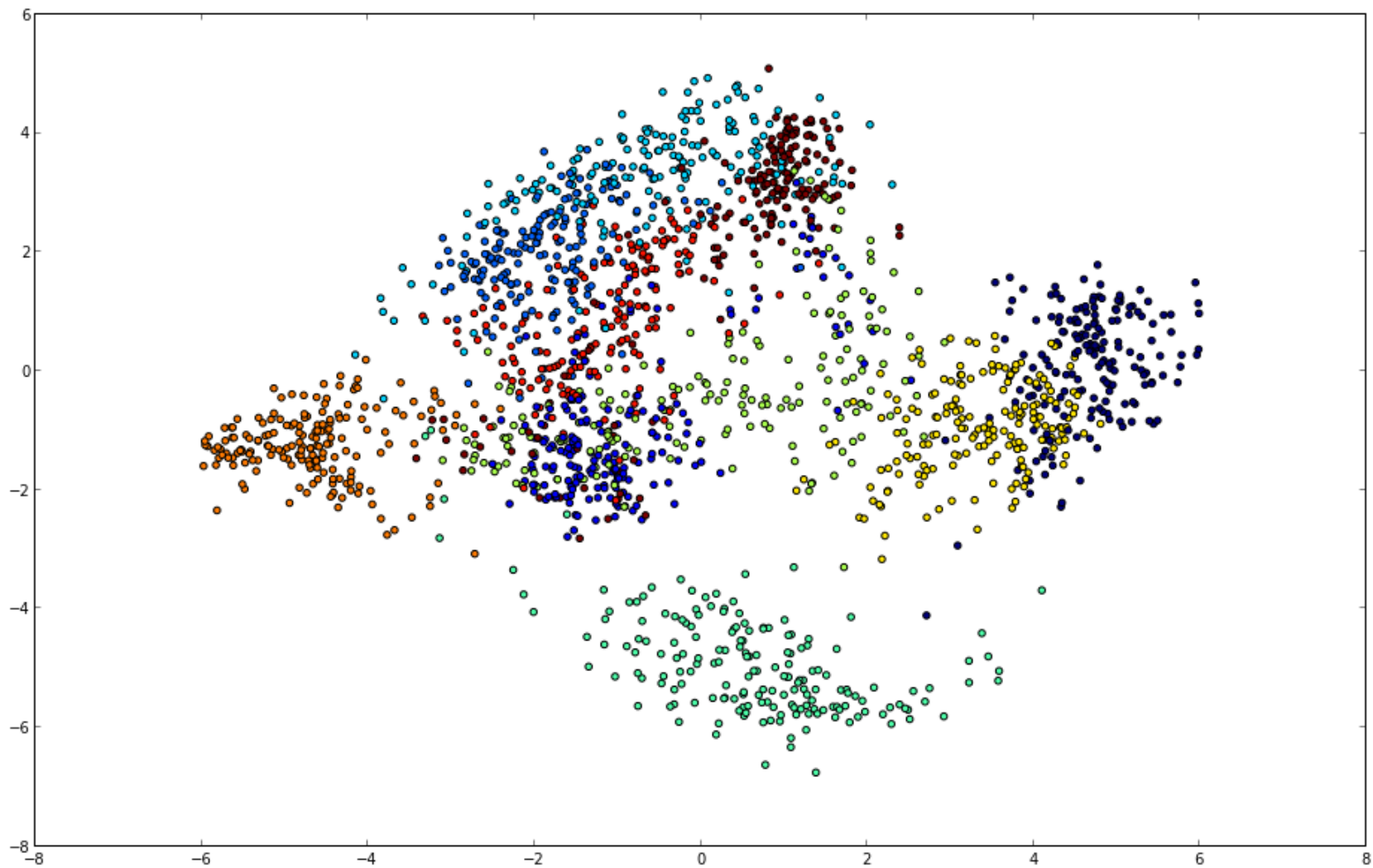
Apply the model.

In [19]: `X_isomap = isomap.transform(X)`
`X_isomap.shape`

Out[19]: `(1797, 2)`

In [20]:

```
plt.scatter(X_isomap[:, 0], X_isomap[:, 1], c=y);
```



Classification

To evaluate the algorithm, split data into training and testing part.

```
In [21]: from sklearn.cross_validation import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
In [22]: print("X_train shape: %s" % repr(X_train.shape))  
print("y_train shape: %s" % repr(y_train.shape))  
print("X_test shape: %s" % repr(X_test.shape))  
print("y_test shape: %s" % repr(y_test.shape))
```

X_train shape: (1347, 64)

y_train shape: (1347,)

X_test shape: (450, 64)

y_test shape: (450,)

Start Simple: Linear SVMs

In [23]: `from sklearn.svm import LinearSVC`

Finds a linear separation between the classes.

Instantiate the model.

In [24]: `svm = LinearSVC()`

Fit the model using the known labels.

In [25]: `svm.fit(X_train, y_train);`

Apply the model. For supervised algorithms, this is `predict`.

In [26]: `svm.predict(X_train)`

Out[26]: `array([2, 8, 9, ..., 7, 7, 8])`

Evaluate the model.

In [27]: `svm.score(X_train, y_train)`

Out[27]: `0.99257609502598365`

In [28]: `svm.score(X_test, y_test)`

Out[28]: 0.9644444444444444

More complex: Random Forests

In [29]: `from sklearn.ensemble import RandomForestClassifier`

Builds many randomized decision trees and averages their results.

Instantiate the model.

In [30]: `rf = RandomForestClassifier()`

Fit the model.

In [31]: `rf.fit(X_train, y_train);`

Evaluate.

In [32]: `rf.score(X_train, y_train)`

Out[32]: 0.99925760950259834

In [33]: `rf.score(X_test, y_test)`

Out[33]: 0.9511111111111113

Model Selection and Evaluation

Always keep a separate test set to the end.

- Measure performance using cross-validation

In [34]:

```
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(rf, X_train, y_train, cv=5)
print("scores: %s mean: %f std: %f" % (str(scores), np.mean(scores),
np.std(scores)))
```

```
scores: [ 0.95185185  0.94074074  0.93680297  0.95910781  0.92936803] mean: 0.943574 std: 0.010635
```

Maybe more trees will help?

In [35]:

```
rf2 = RandomForestClassifier(n_estimators=50)
scores = cross_val_score(rf2, X_train, y_train, cv=5)
print("scores: %s mean: %f std: %f" % (str(scores), np.mean(scores),
np.std(scores)))
```

```
scores: [ 0.95555556  0.97407407  0.97026022  0.97769517  0.96282528] mean: 0.968082 std: 0.007970
```

Adjust important parameters using grid search

In [36]:

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

- Let's look at LinearSVC again.
- Only important parameter: C

In [37]:

```
param_grid = {'C': 10. ** np.arange(-3, 4)}  
grid_search = GridSearchCV(svm, param_grid=param_grid, cv=3, verbose=3,  
compute_training_score=True)
```

In [38]:

```
grid_search.fit(X_train, y_train);
```

```
[GridSearchCV] C=0.001 .....
[GridSearchCV] ..... C=0.001, score=0.902004 - 0.1s
[GridSearchCV] C=0.001 .....
[GridSearchCV] ..... C=0.001, score=0.895323 - 0.1s
[GridSearchCV] C=0.001 .....
[GridSearchCV] ..... C=0.001, score=0.879733 - 0.1s
[GridSearchCV] C=0.01 .....
[GridSearchCV] ..... C=0.01, score=0.953229 - 0.1s
[GridSearchCV] C=0.01 .....
[GridSearchCV] ..... C=0.01, score=0.937639 - 0.1s
[GridSearchCV] C=0.01 .....
[GridSearchCV] ..... C=0.01, score=0.919822 - 0.1s
[GridSearchCV] C=0.1 .....
[GridSearchCV] ..... C=0.1, score=0.973274 - 0.1s
[GridSearchCV] C=0.1 .....
[GridSearchCV] ..... C=0.1, score=0.951002 - 0.1s
[GridSearchCV] C=0.1 .....
[GridSearchCV] ..... C=0.1, score=0.951002 - 0.1s
[GridSearchCV] C=1.0 .....
[GridSearchCV] ..... C=1.0, score=0.977728 - 0.2s
[GridSearchCV] C=1.0 .....
[GridSearchCV] ..... C=1.0, score=0.957684 - 0.2s
[GridSearchCV] C=1.0 .....
[GridSearchCV] ..... C=1.0, score=0.964365 - 0.2s
[GridSearchCV] C=10.0 .....
[GridSearchCV] ..... C=10.0, score=0.975501 - 0.2s
[GridSearchCV] C=10.0 .....
[GridSearchCV] ..... C=10.0, score=0.944321 - 0.2s
[GridSearchCV] C=10.0 .....
[GridSearchCV] ..... C=10.0, score=0.962138 - 0.2s
[GridSearchCV] C=100.0 .....
```

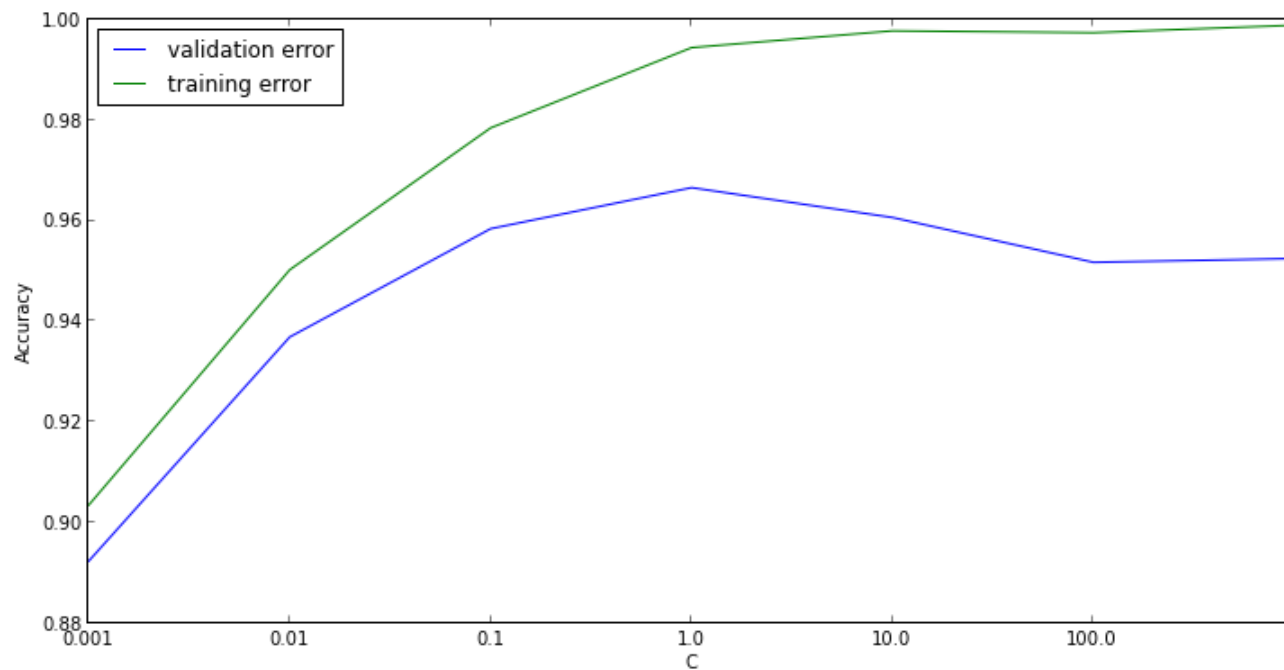

In [39]:

```
print(grid_search.best_params_)  
print(grid_search.best_score_)
```

```
{'C': 1.0}  
0.96659242761  
7
```

In [41]:

```
plt.figure(12, 6)  
plt.plot([c.mean_validation_score for c in grid_search.cv_scores_],  
label="validation error")  
plt.plot([c.mean_training_score for c in grid_search.cv_scores_],  
label="training error")  
plt.xticks(np.arange(6), param_grid['C']); plt.xlabel("C");  
plt.ylabel("Accuracy"); plt.legend(loc='best');
```

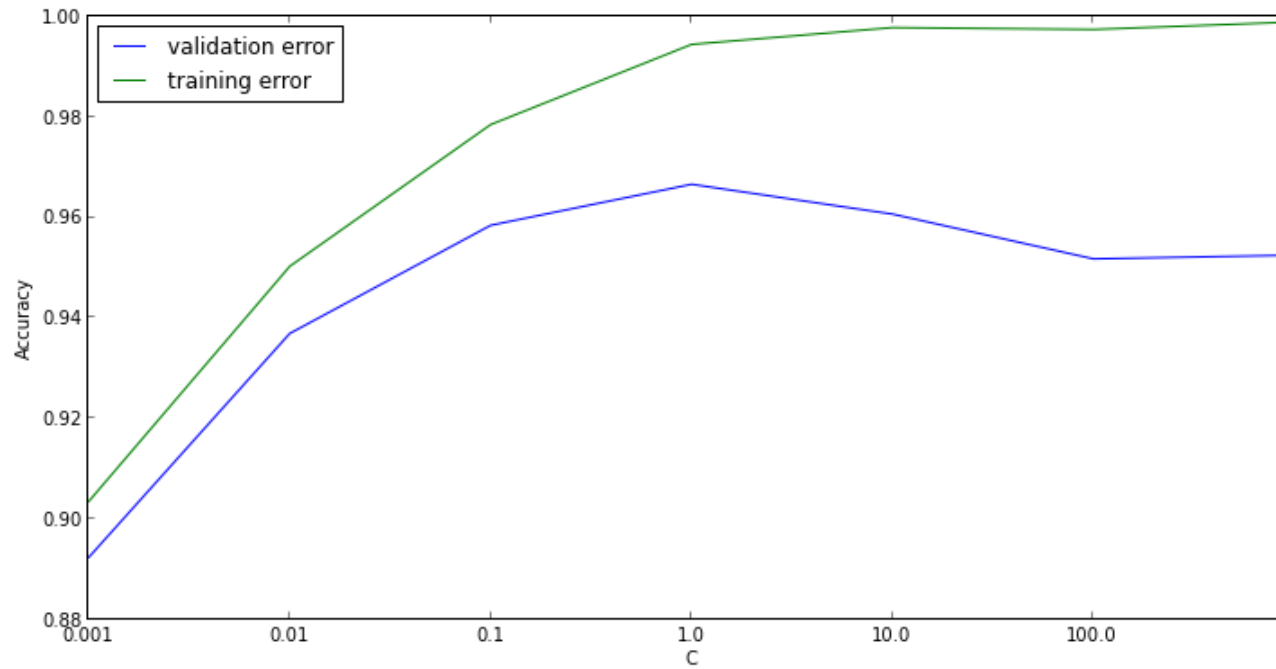


Overfitting and Complexity Control

- to the right: overfitting aka high variance.
 - Means no generalization.
- to the left: underfitting aka high bias.
 - Means bad even on training set.

In [42]:

```
plt.plot([c.mean_validation_score for c in grid_search.cv_scores_],  
label="validation error")  
plt.plot([c.mean_training_score for c in grid_search.cv_scores_],  
label="training error")  
plt.xticks(np.arange(6), param_grid['C']); plt.xlabel("C");  
plt.ylabel("Accuracy"); plt.legend(loc='best');
```



Detecting Insults in Social Commentary

- My first (and only) kaggle entry.
- Classify short forum posts as insulting or not.
- A simple bag of word model carries quite far.
- Linear classifiers are usually the best for text data.

Read the CSV using Pandas (a bit overkill).

In [43]:

```
import pandas as pd  
train_data = pd.read_csv("kaggle_insult/train.csv")  
test_data = pd.read_csv("kaggle_insult/test_with_solutions.csv")
```

- The column "Insult" contains the target.
- The column "Comment" contains the text.

In [44]:

```
y_train = np.array(train_data.Insult)
comments_train = np.array(train_data.Comment)
print(comments_train.shape)
print(y_train.shape)
```

(3947,)

(3947,)

In [45]:

```
print(comments_train[0])
print("Insult: %d" % y_train[0])
```

"You fuck your dad."

Insult: 1

In [46]:

```
print(comments_train[5])
print("Insult: %d" % y_train[5])
```

"@SDL OK, but I would hope they'd sign him to a one-year contract to start with. Give him the chance

Insult: 0

Vectorizing the Data

```
In [47]: from sklearn.feature_extraction.text import CountVectorizer
```

- Use bag of words model as implemented in CountVectorizer.
- Extracts a dictionary, then counts word occurrences.

```
In [48]: cv = CountVectorizer()
cv.fit(comments_train)
print(cv.get_feature_names()[:15])

[u'00', u'000', u'01', u'014', u'01k4wu4w', u'02', u'034', u'05', u'06', u'0612', u'07', u'075', u'08', u'081', u'082', u'083', u'084', u'085', u'086', u'087', u'088', u'089', u'09', u'091', u'092', u'093', u'094', u'095', u'096', u'097', u'098', u'099', u'1', u'10', u'100', u'101', u'102', u'103', u'104', u'105', u'106', u'107', u'108', u'109', u'11', u'110', u'111', u'112', u'113', u'114', u'115', u'116', u'117', u'118', u'119', u'12', u'120', u'121', u'122', u'123', u'124', u'125', u'126', u'127', u'128', u'129', u'13', u'130', u'131', u'132', u'133', u'134', u'135', u'136', u'137', u'138', u'139', u'14', u'140', u'141', u'142', u'143', u'144', u'145', u'146', u'147', u'148', u'149', u'15', u'150', u'151', u'152', u'153', u'154', u'155', u'156', u'157', u'158', u'159', u'16', u'160', u'161', u'162', u'163', u'164', u'165', u'166', u'167', u'168', u'169', u'17', u'170', u'171', u'172', u'173', u'174', u'175', u'176', u'177', u'178', u'179', u'18', u'180', u'181', u'182', u'183', u'184', u'185', u'186', u'187', u'188', u'189', u'19', u'190', u'191', u'192', u'193', u'194', u'195', u'196', u'197', u'198', u'199', u'2', u'20', u'200', u'201', u'202', u'203', u'204', u'205', u'206', u'207', u'208', u'209', u'21', u'210', u'211', u'212', u'213', u'214', u'215', u'216', u'217', u'218', u'219', u'22', u'220', u'221', u'222', u'223', u'224', u'225', u'226', u'227', u'228', u'229', u'23', u'230', u'231', u'232', u'233', u'234', u'235', u'236', u'237', u'238', u'239', u'24', u'240', u'241', u'242', u'243', u'244', u'245', u'246', u'247', u'248', u'249', u'25', u'250', u'251', u'252', u'253', u'254', u'255', u'256', u'257', u'258', u'259', u'26', u'260', u'261', u'262', u'263', u'264', u'265', u'266', u'267', u'268', u'269', u'27', u'270', u'271', u'272', u'273', u'274', u'275', u'276', u'277', u'278', u'279', u'28', u'280', u'281', u'282', u'283', u'284', u'285', u'286', u'287', u'288', u'289', u'29', u'290', u'291', u'292', u'293', u'294', u'295', u'296', u'297', u'298', u'299', u'3', u'30', u'300', u'301', u'302', u'303', u'304', u'305', u'306', u'307', u'308', u'309', u'31', u'310', u'311', u'312', u'313', u'314', u'315', u'316', u'317', u'318', u'319', u'32', u'320', u'321', u'322', u'323', u'324', u'325', u'326', u'327', u'328', u'329', u'33', u'330', u'331', u'332', u'333', u'334', u'335', u'336', u'337', u'338', u'339', u'34', u'340', u'341', u'342', u'343', u'344', u'345', u'346', u'347', u'348', u'349', u'35', u'350', u'351', u'352', u'353', u'354', u'355', u'356', u'357', u'358', u'359', u'36', u'360', u'361', u'362', u'363', u'364', u'365', u'366', u'367', u'368', u'369', u'37', u'370', u'371', u'372', u'373', u'374', u'375', u'376', u'377', u'378', u'379', u'38', u'380', u'381', u'382', u'383', u'384', u'385', u'386', u'387', u'388', u'389', u'39', u'390', u'391', u'392', u'393', u'394', u'395', u'396', u'397', u'398', u'399', u'4', u'40', u'400', u'401', u'402', u'403', u'404', u'405', u'406', u'407', u'408', u'409', u'41', u'410', u'411', u'412', u'413', u'414', u'415', u'416', u'417', u'418', u'419', u'42', u'420', u'421', u'422', u'423', u'424', u'425', u'426', u'427', u'428', u'429', u'43', u'430', u'431', u'432', u'433', u'434', u'435', u'436', u'437', u'438', u'439', u'44', u'440', u'441', u'442', u'443', u'444', u'445', u'446', u'447', u'448', u'449', u'45', u'450', u'451', u'452', u'453', u'454', u'455', u'456', u'457', u'458', u'459', u'46', u'460', u'461', u'462', u'463', u'464', u'465', u'466', u'467', u'468', u'469', u'47', u'470', u'471', u'472', u'473', u'474', u'475', u'476', u'477', u'478', u'479', u'48', u'480', u'481', u'482', u'483', u'484', u'485', u'486', u'487', u'488', u'489', u'49', u'490', u'491', u'492', u'493', u'494', u'495', u'496', u'497', u'498', u'499', u'5', u'50', u'500', u'501', u'502', u'503', u'504', u'505', u'506', u'507', u'508', u'509', u'51', u'510', u'511', u'512', u'513', u'514', u'515', u'516', u'517', u'518', u'519', u'52', u'520', u'521', u'522', u'523', u'524', u'525', u'526', u'527', u'528', u'529', u'53', u'530', u'531', u'532', u'533', u'534', u'535', u'536', u'537', u'538', u'539', u'54', u'540', u'541', u'542', u'543', u'544', u'545', u'546', u'547', u'548', u'549', u'55', u'550', u'551', u'552', u'553', u'554', u'555', u'556', u'557', u'558', u'559', u'56', u'560', u'561', u'562', u'563', u'564', u'565', u'566', u'567', u'568', u'569', u'57', u'570', u'571', u'572', u'573', u'574', u'575', u'576', u'577', u'578', u'579', u'58', u'580', u'581', u'582', u'583', u'584', u'585', u'586', u'587', u'588', u'589', u'59', u'590', u'591', u'592', u'593', u'594', u'595', u'596', u'597', u'598', u'599', u'6', u'60', u'600', u'601', u'602', u'603', u'604', u'605', u'606', u'607', u'608', u'609', u'61', u'610', u'611', u'612', u'613', u'614', u'615', u'616', u'617', u'618', u'619', u'62', u'620', u'621', u'622', u'623', u'624', u'625', u'626', u'627', u'628', u'629', u'63', u'630', u'631', u'632', u'633', u'634', u'635', u'636', u'637', u'638', u'639', u'64', u'640', u'641', u'642', u'643', u'644', u'645', u'646', u'647', u'648', u'649', u'65', u'650', u'651', u'652', u'653', u'654', u'655', u'656', u'657', u'658', u'659', u'66', u'660', u'661', u'662', u'663', u'664', u'665', u'666', u'667', u'668', u'669', u'67', u'670', u'671', u'672', u'673', u'674', u'675', u'676', u'677', u'678', u'679', u'68', u'680', u'681', u'682', u'683', u'684', u'685', u'686', u'687', u'688', u'689', u'69', u'690', u'
```

```
In [49]: print(cv.get_feature_names()[1000:1015])
```

```
[u'argue', u'argued', u'argument', u'arguments', u'arguing', u'argument', u'arguments', u'aries', ]
```

```
In [50]: X_train = cv.transform(comments_train).tocsr()
          print("X_train.shape: %s" % str(X_train.shape))
          print(X_train[0, :])
```

```
X_train.shape: (3947, 16469)
```

(0, 3409)	1
(0, 5434)	1
(0, 16397)	1
(0, 16405)	1

Training a Classifier

- LinearSVC : linear SVM that is efficient for sparse data.

In [51]:

```
from sklearn.svm import LinearSVC
svm = LinearSVC()
svm.fit(X_train, y_train)
```

Out[51]:

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='l2', multi_class='ovr',
          penalty='l2',
          random_state=None, tol=0.0001, verbose=0)
```

In [52]:

```
comments_test = np.array(test_data.Comment)
y_test = np.array(test_data.Insult)
X_test = cv.transform(comments_test)
svm.score(X_test, y_test)
```

Out[52]:

```
0.83037400831129582
```

In [53]:

```
print(comments_test[8])
print("Target: %d, prediction: %d" % (y_test[8], svm.predict(X_test.tocsr()
[8])[0]))
```

```
"To engage in an intelligent debate with you is like debating to a retarded person. It's useless.
Target: 1, prediction: 1"
```


Next Steps

- Grid search C parameter of LinearSVC.
- Build a pipeline, adjust parameters of feature extraction.
- Combine different feature extraction methods.

Take Away

- Get your data into an array (`n_samples, n_features`).
- `model.fit(X)`, `model.predict(X)` / `model.transform(X)`
- Always do cross-validation. Leave the test set until the end.
- Internalize the complexity / generalization tradeoff.

Fin



amueller@ais.uni-bonn.de



[@t3kcit](https://twitter.com/t3kcit)



[@amueller](https://github.com/amueller)



peekaboo-vision.blogspot.com