



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

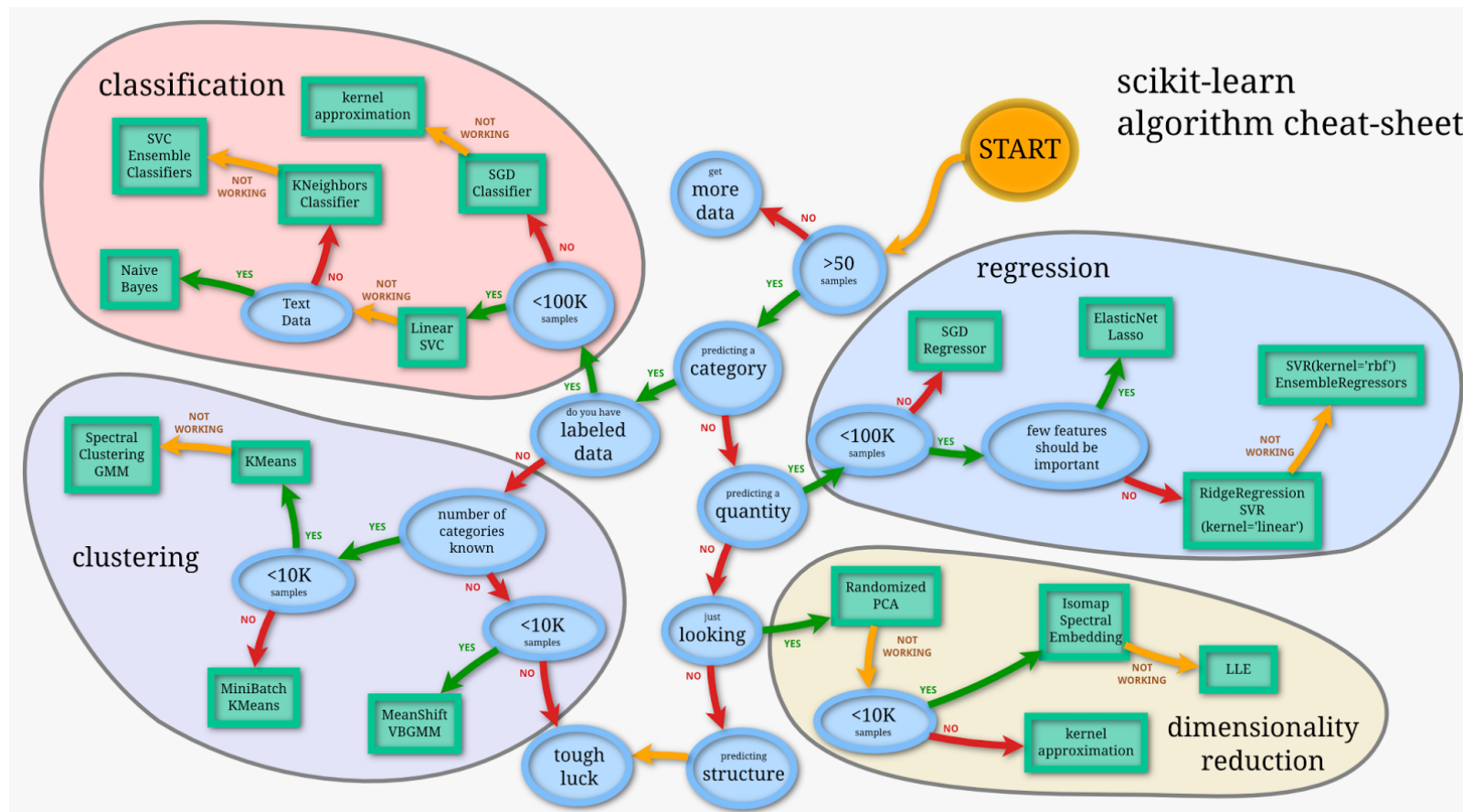
- Classification (spam, sentiment analysis, ...)
- Regression (stocks, sales, ...)
- Ranking (retrieval, search, ...)

Unsupervised Learning

Examples X. Learn something about X.

Examples

- Dimensionality reduction
- Clustering
- Manifold learning



Data representation

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

Let's get some toy data.

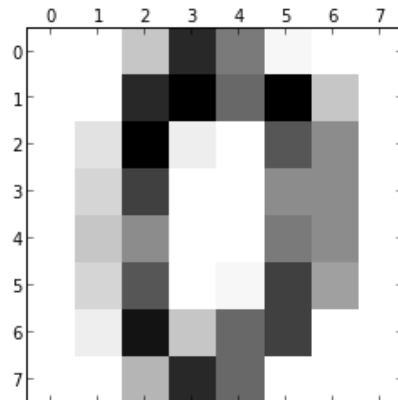
```
In [1]: from sklearn.datasets import load_digits  
digits = load_digits()
```

```
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.

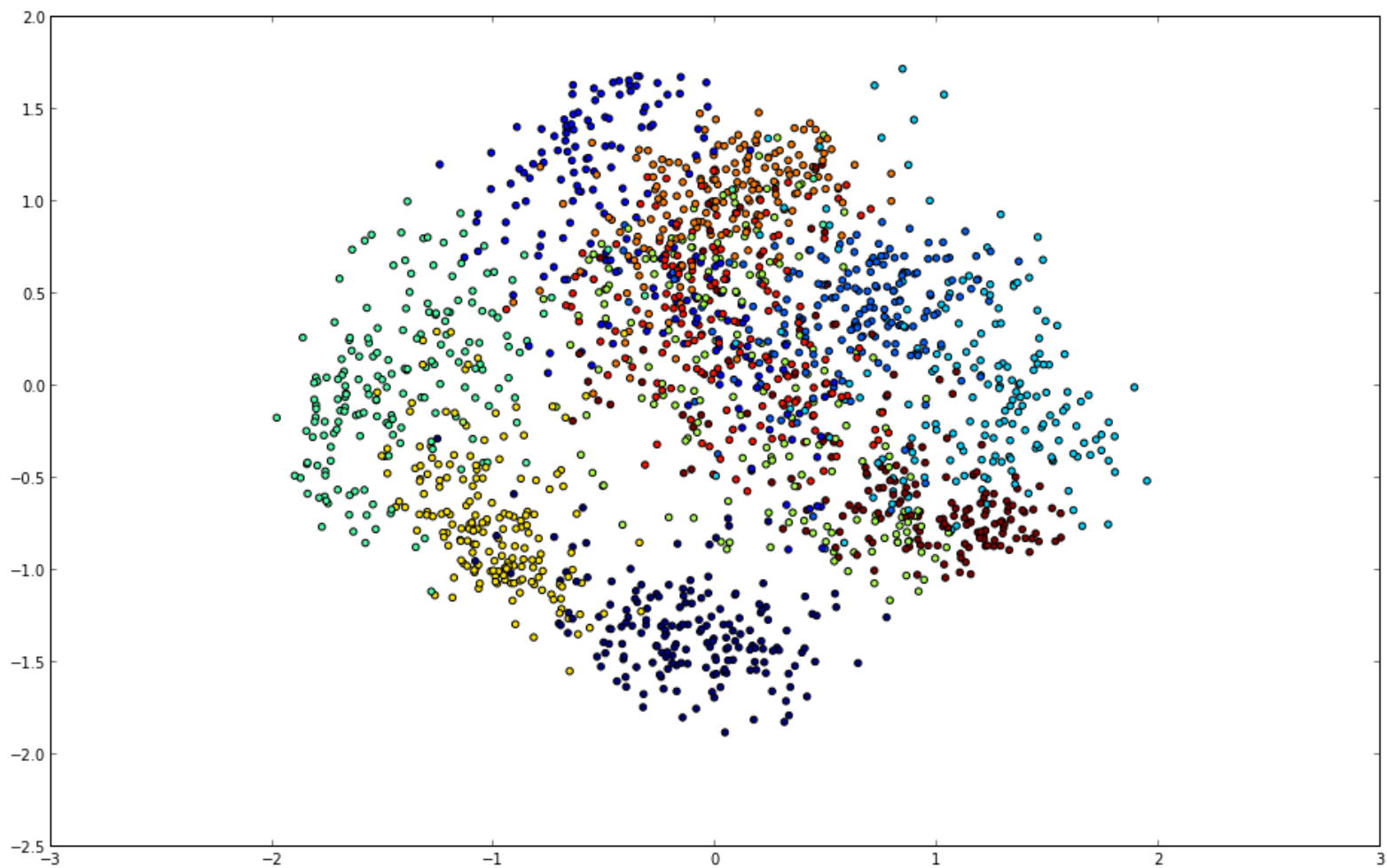
```
In [11]: pca.fit(X);
```

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

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

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

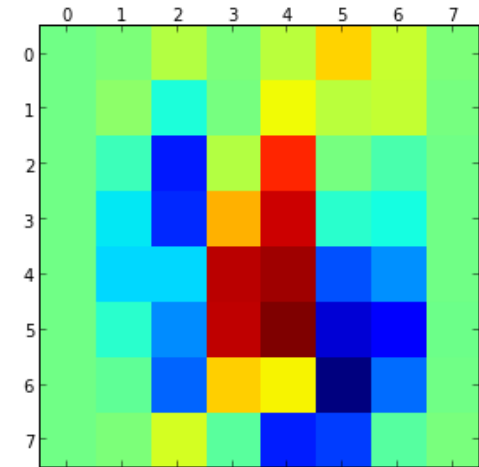
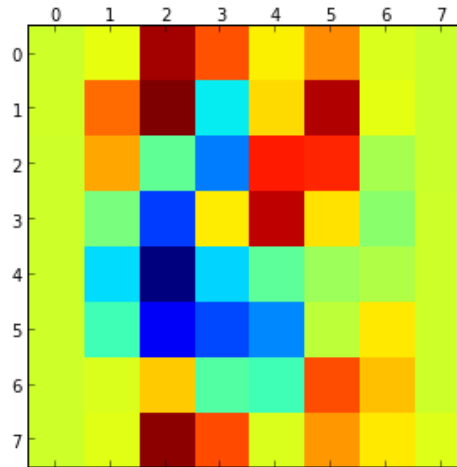
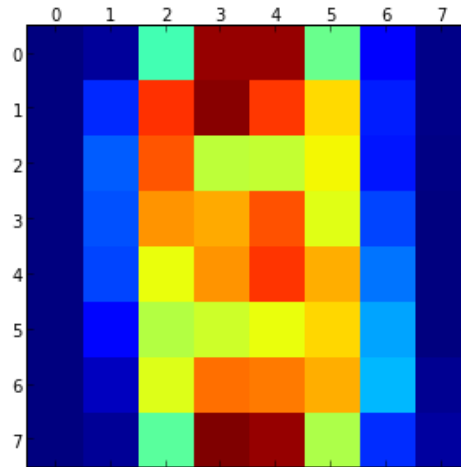
```
In [13]: plt.figure(16, 10)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y);
```



```
In [14]: print(pca.mean_.shape)
print(pca.components_.shape)
```

```
(64,)
(2, 64)
```

```
In [15]: fix, ax = plt.subplots(1, 3)
ax[0].matshow(pca.mean_.reshape(8, 8))
ax[1].matshow(pca.components_[0, :].reshape(8, 8))
ax[2].matshow(pca.components_[1, :].reshape(8, 8));
```



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.

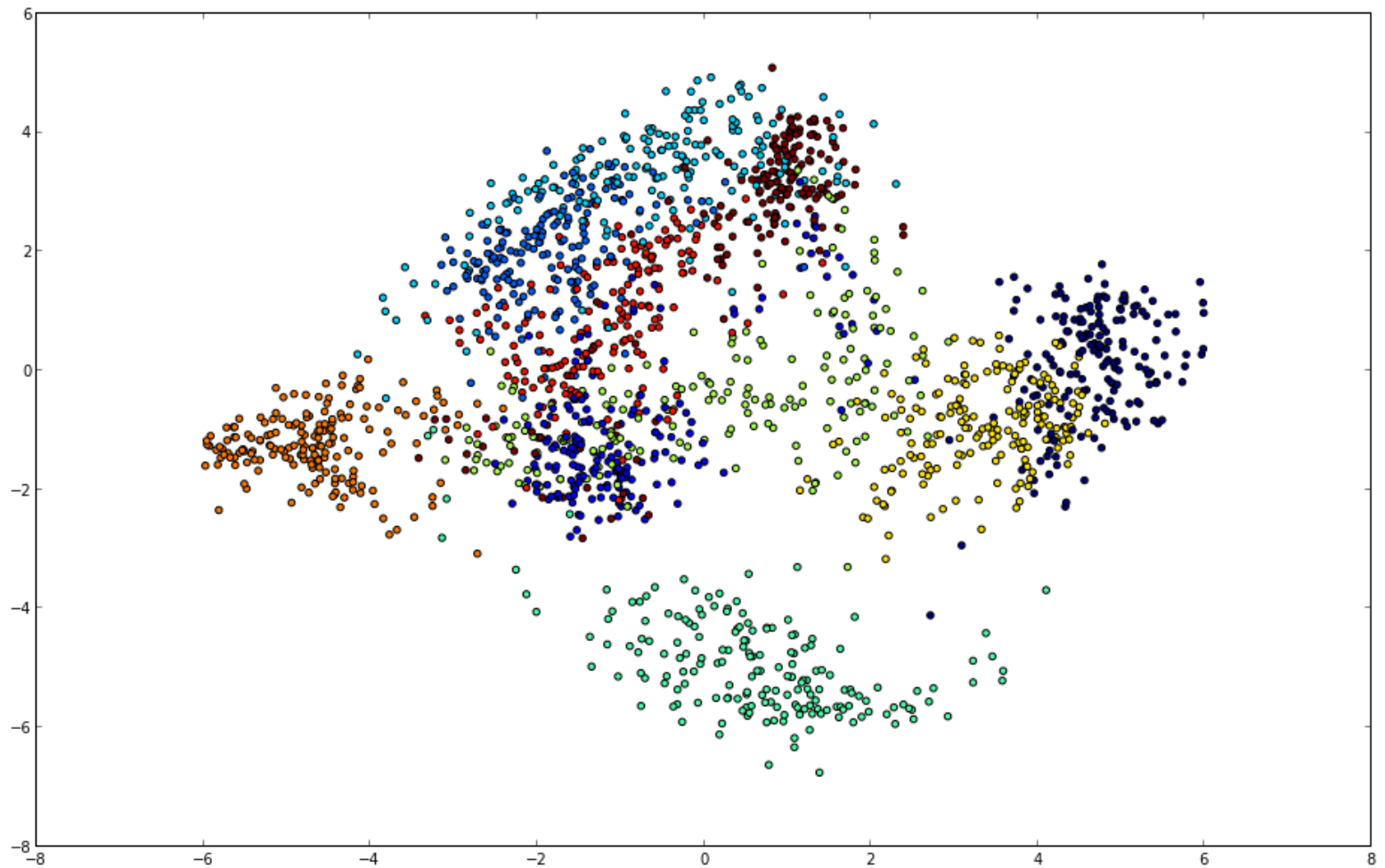
```
In [18]: isomap.fit(X);
```

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]: 1.0
```

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

```
Out[33]: 0.9488888888888889
```

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: [ 0.95555556  0.95555556  0.93680297  0.9330855  0.92193309] mean: 0.940587 std:
0.013166
```

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: [ 0.96296296  0.96666667  0.97769517  0.9739777  0.97026022] mean: 0.970313 std:
0.005201
```

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);
```

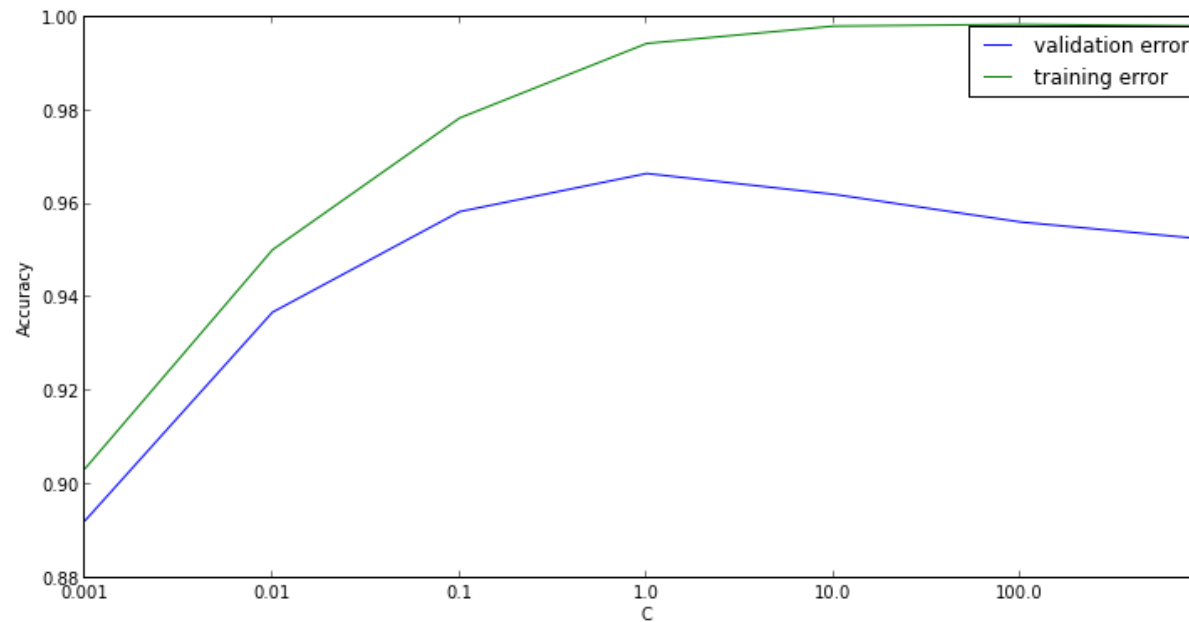
```
[GridSearchCV] C=0.001 .....  
[GridSearchCV] ..... C=0.001, score=0.902004 - 0.1s  
[GridSearchCV] C=0.001 .....  
[GridSearchCV] ..... C=0.001, score=0.895323 - 0.1s  
[GridSearchCV] C=0.001 .....  
[GridSearchCV] ..... C=0.001, score=0.879733 - 0.1s  
[GridSearchCV] C=0.01 .....  
[GridSearchCV] ..... 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=0.946548 - 0.2s  
[GridSearchCV] C=10.0 .....  
[GridSearchCV] ..... C=10.0, score=0.964365 - 0.2s  
[GridSearchCV] C=100.0 .....  
[GridSearchCV] ..... C=100.0, score=0.968820 - 0.2s  
[GridSearchCV] C=100.0 .....  
[GridSearchCV] ..... C=100.0, score=0.939866 - 0.2s  
[GridSearchCV] C=100.0 .....  
[GridSearchCV] ..... C=100.0, score=0.959911 - 0.2s  
[GridSearchCV] C=1000.0 .....  
[GridSearchCV] ..... C=1000.0, score=0.957684 - 0.2s
```

```
[GridSearchCV] ..... C=1000.0, score=0.957684 - 0.2s
[GridSearchCV] C=1000.0 .....
[GridSearchCV] ..... C=1000.0, score=0.944321 - 0.2s
[GridSearchCV] C=1000.0 .....
[GridSearchCV] ..... C=1000.0, score=0.955457 - 0.2s
[Parallel(n_jobs=1)]: Done 1 jobs | elapsed: 0.1s
[Parallel(n_jobs=1)]: Done 21 out of 21 | elapsed: 3.4s finished
```

```
In [39]: print(grid_search.best_params_)  
print(grid_search.best_score_)
```

```
{'C': 1.0}  
0.966592427617
```

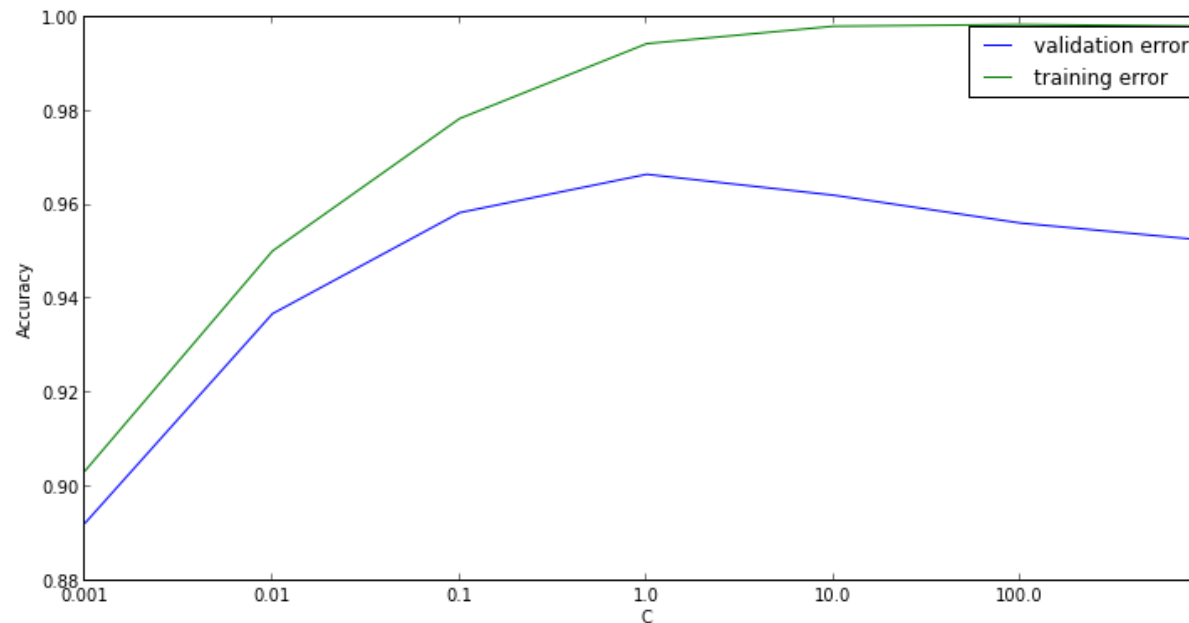
```
In [40]: plt.figure(12, 6)  
plt.plot([c.mean_test_score for c in grid_search.cv_scores_], label="validation  
error")  
plt.plot([c.mean_training_score for c in grid_search.cv_scores_], label="training  
error")  
plt.xticks(np.arange(6), param_grid['C']); plt.xlabel("C");  
plt.ylabel("Accuracy"); 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 [41]: plt.plot([c.mean_test_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 [42]: 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 [43]: 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 [44]: print(comments_train[0])
         print("Insult: %d" % y_train[0])
```

```
"You fuck your dad."
```

```
Insult: 1
```

```
In [45]: 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 to be reliable and productive, but give themselves the out if all his time off has
hurt his playing skills or if he falls back into old habits."
```

```
Insult: 0
```

Vectorizing the Data

```
In [46]: from sklearn.feature_extraction.text import CountVectorizer
```

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

```
In [47]: 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'09', u'0bama']
```

```
In [48]: print(cv.get_feature_names()[1000:1015])
```

```
[u'argue', u'argued', u'argument', u'arguments', u'arguing', u'argument', u'arguments',  
u'aries', u'aristocracy', u'aritlett', u'arizona', u'arkan', u'arlington', u'arm',  
u'armando']
```

```
In [49]: X_train = cv.transform(comments_train).tocsr()  
print(uquot;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 [50]: from sklearn.svm import LinearSVC
svm = LinearSVC()
svm.fit(X_train, y_train)
```

```
Out[50]: 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 [51]: 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[51]: 0.83037400831129582
```

```
In [52]: 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. It looks like you're bent on disregarding the efforts of the government."
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