Linear predictions with scikit-learn: simple and efficient

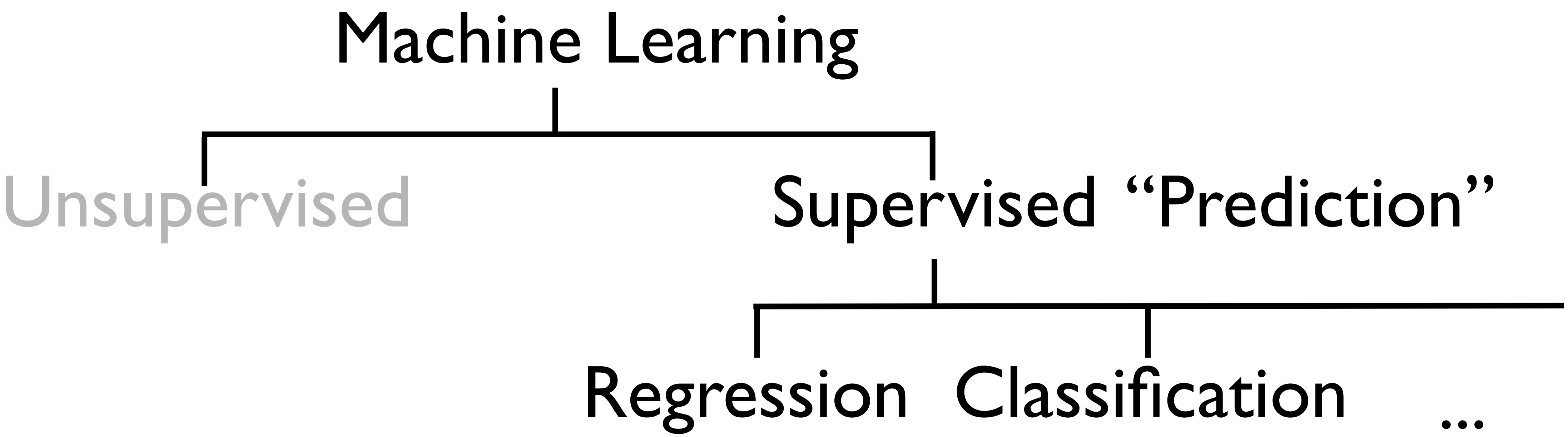
Alexandre Gramfort

Telecom ParisTech - CNRS LTCI
alexandre.gramfort@telecom-paristech.fr

GitHub : @agramfort

Twitter : @agramfort

ML Taxonomy



Machine Learning

Unsupervised

Supervised “Prediction”

Regression

Classification

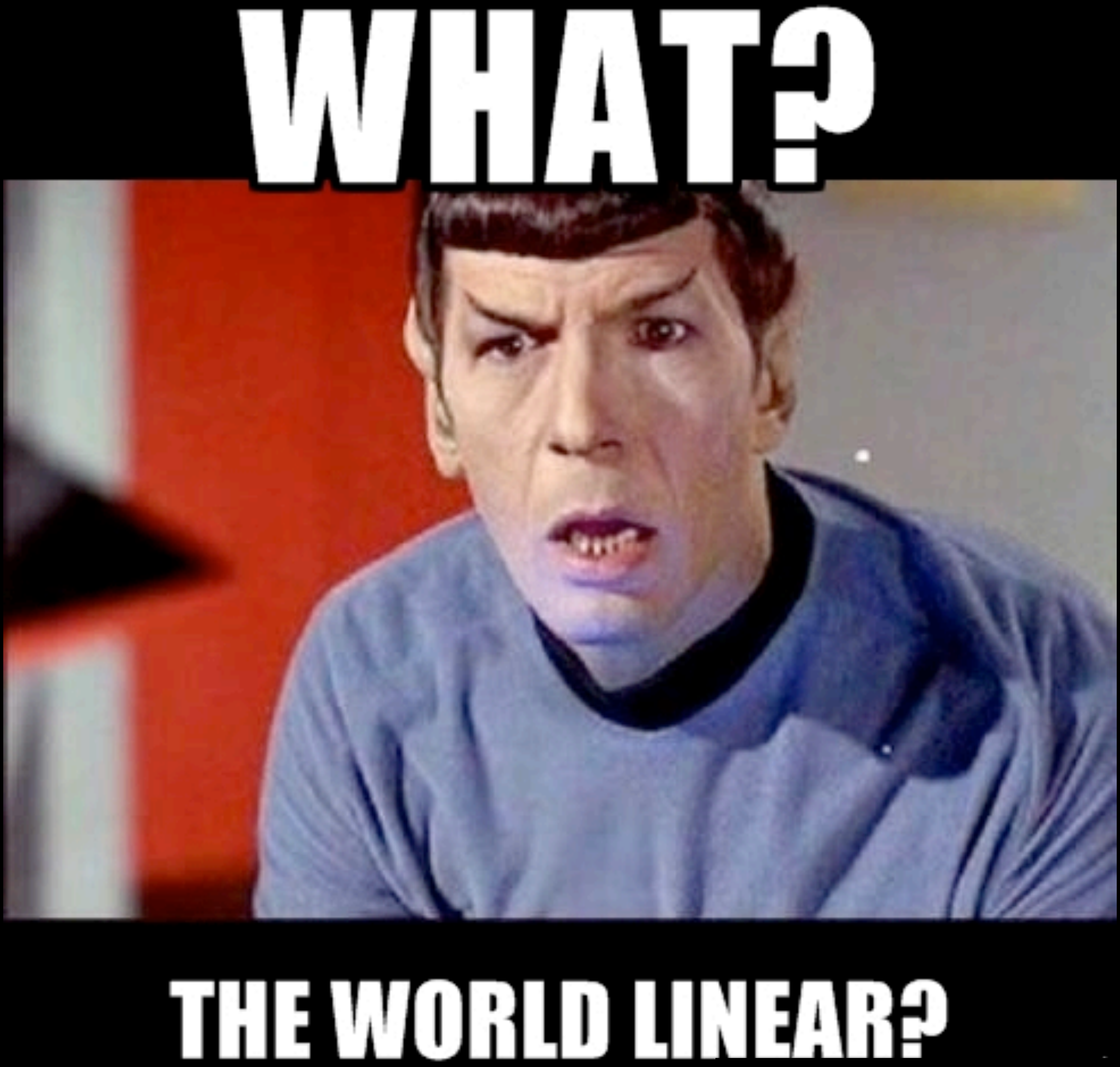
...

Examples of predictions:

customer churn, traffic, equipment failure, prices, optimal bid price for online ads, spam/ham, etc.

“Give me X and I will predict y”

Linearly or non-linearly….


WHAT?
THE WORLD LINEAR?

Predicting House Prices

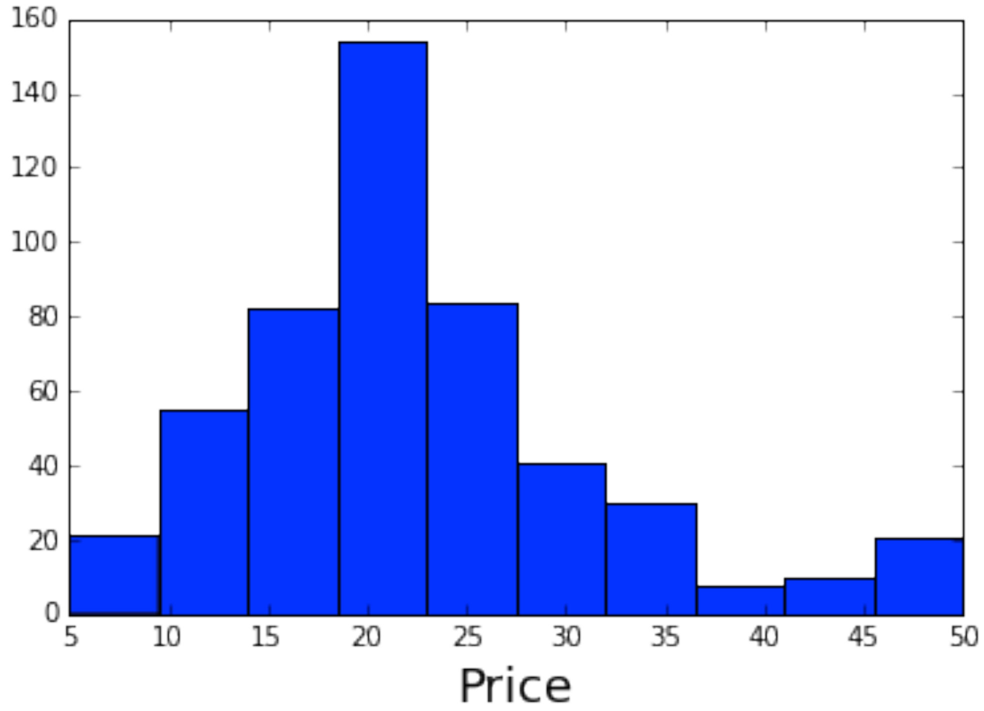
```
>>> from sklearn.datasets import load_boston
>>> boston = load_boston()
>>> print(boston.DESCR)
Boston House Prices dataset

Data Set Characteristics:

 :Number of Instances: 506
 :Number of Attributes: 13 numeric/categorical predictive
 :Median Value (attribute 14) is usually the target
 :Attribute Information (in order):
  - CRIM    per capita crime rate by town
  - ZN      proportion of residential land zoned for lots over 25,000 sq.ft.
  - INDUS   proportion of non-retail business acres per town
  - CHAS    Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
  - NOX     nitric oxides concentration (parts per 10 million)
  - RM      average number of rooms per dwelling
  - AGE     proportion of owner-occupied units built prior to 1940
...
```

Predicting House Prices

```
>>> from sklearn.datasets import load_boston
>>> boston = load_boston()
>>> X, y = boston.data, boston.target
>>> n_samples, n_features = X.shape
>>> print(n_samples, n_features)
(506, 13)
>>> print(boston.feature_names)
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX'
 'PTRATIO' 'B' 'LSTAT']
```

Let's look at the target:

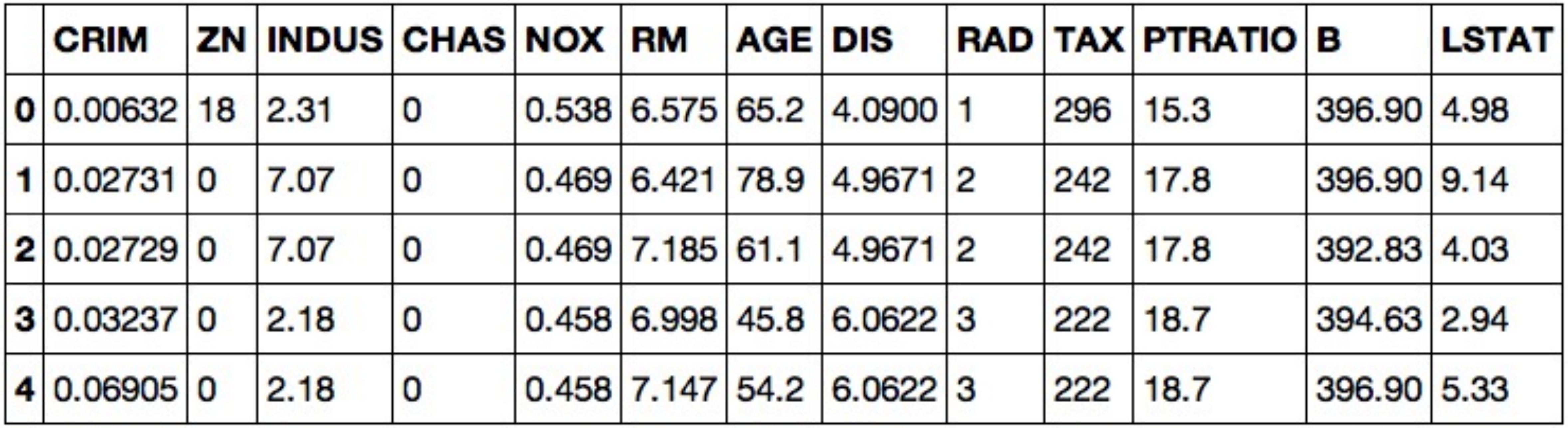
```
>>> plt.hist(y)
>>> plt.xlabel('Price', fontsize=18)
```



Predicting House Prices

Let's look at the features:

```
>>> import pandas as pd
>>> df = pd.DataFrame(X, columns=boston.feature_names)
>>> df.head()
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

Predicting with a linear model

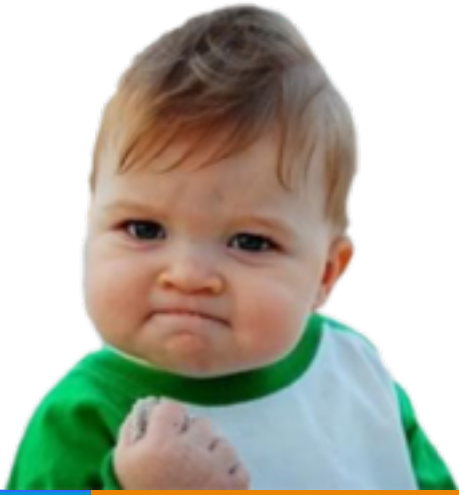
Linear regression: $y = \theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p$

Example with House Prices

$$\text{price} = \theta_0 + \theta_1 \text{CRIM} + \theta_2 \text{ZN} + \cdots + \theta_{13} \text{LSTAT}$$

```
>>> from sklearn.linear_model import LinearRegression
>>> model = LinearRegression()
>>> model.fit(X, y)
>>> print(model.intercept_)  # the intercept (theta0)
36.4911032804
>>> print(model.coef_.shape)  # the coefficients (theta1, …, theta13)
(13,)
```

```
>>> model.fit(X[::2], y[::2])
>>> print("R2 score: %s" % model.score(X[1::2], y[1::2]))
R2 score: 0.744395023361
```



Predicting with a linear model

```
>>> from sklearn import linear_model
>>> dir(linear_model)
['ARDRegression',
 'BayesianRidge',
 'ElasticNet',
 'Lars',
 'Lasso',
 'LassoLars'
 'LinearRegression',
 'LogisticRegression',
 'LogisticRegressionCV',
 'OrthogonalMatchingPursuit',
 'Perceptron',
 'Ridge',
 'RidgeCV',
 'RidgeClassifier',
 'RidgeClassifierCV',
 'SGDClassifier',
 'SGDRegressor',
…]
```

Predicting with a linear model

Want to try another model?

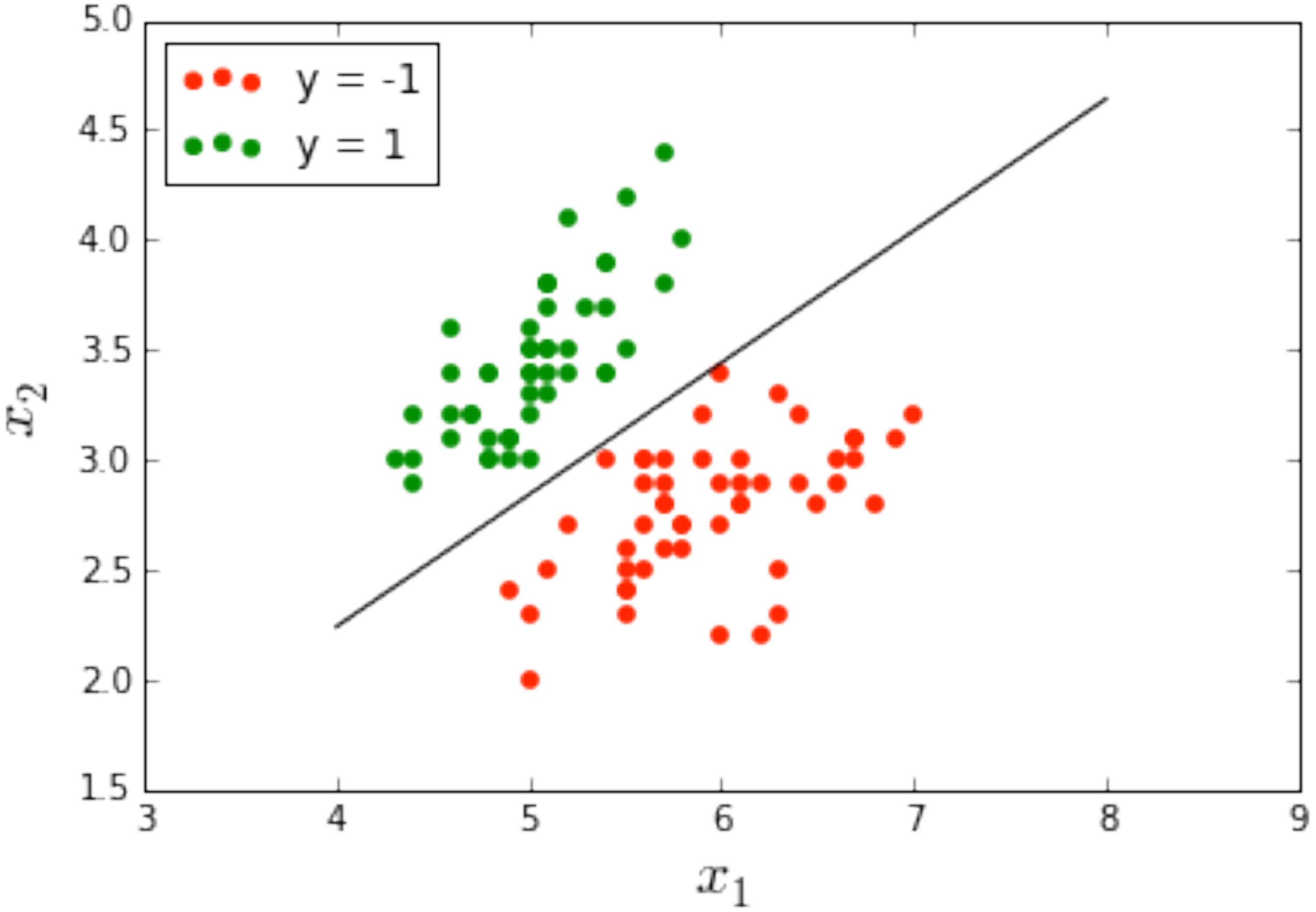
```
>>> from sklearn.linear_model import Ridge
>>> model = Ridge(alpha=0.1)
>>> model.fit(X, y)
>>> print(model.intercept_)  # the intercept (theta0)
35.7235452294
>>> print(model.coef_.shape)  # the coefficients (theta1, …, theta13)
(13,)
```

Predicting with a linear model

Linear classification (binary):

$$y = \text{sign}(\theta_0 + \theta_1 x_1 + \cdots + \theta_p x_p) \qquad y = 1 \text{ or } -1$$

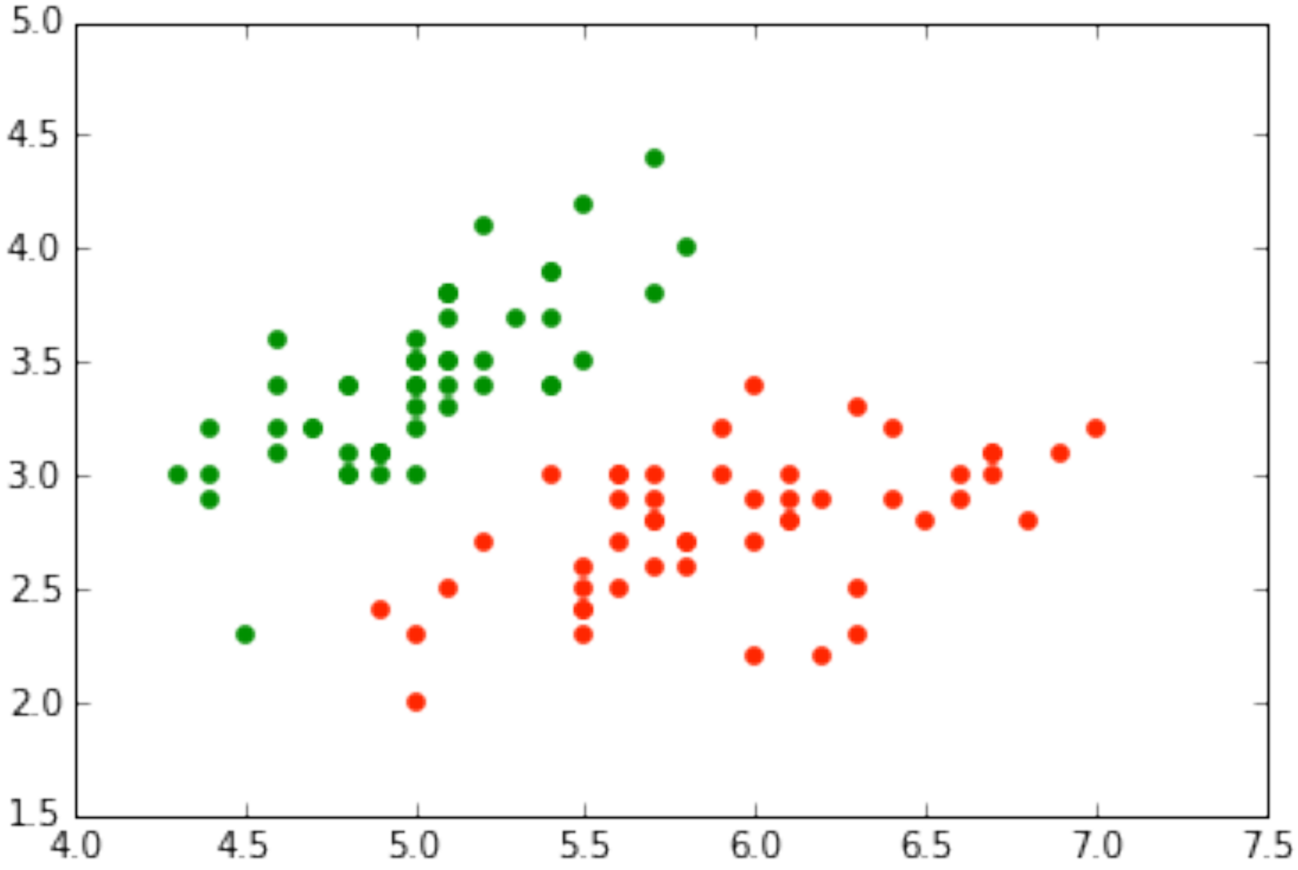
Example: spam $y = 1$ *or ham* $y = -1$



Predicting with a linear model

Example: classification of iris dataset

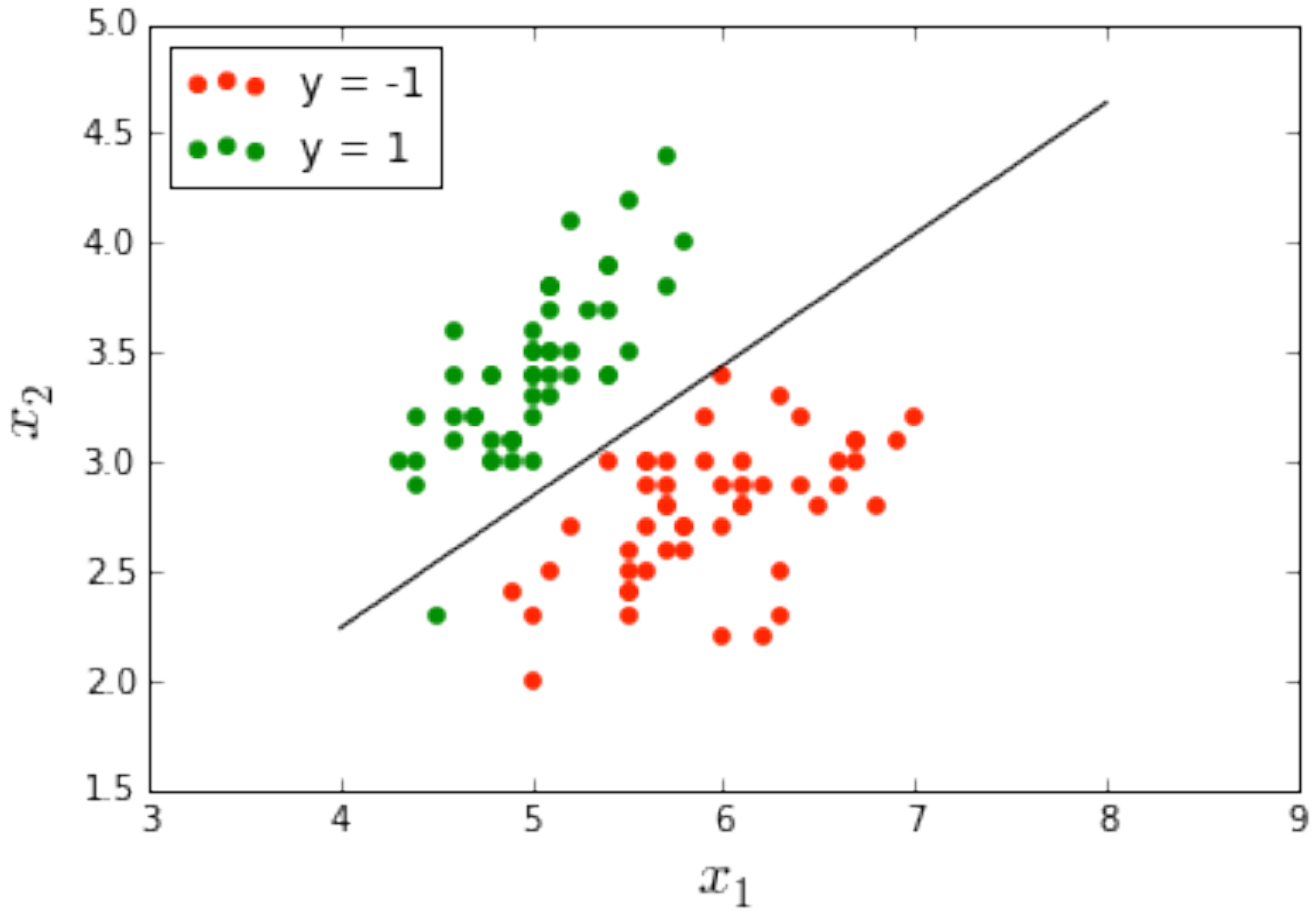
```
>>> from sklearn import datasets
>>> from sklearn.linear_model import LogisticRegression
>>> iris = datasets.load_iris()
>>> X = iris.data[:, :2]  # Make it 2d
>>> y = iris.target
>>> X, y = X[y < 2], y[y < 2]  # Make it binary
>>> y[y == 0] = -1
>>> print(X.shape)
(100, 2)
>>> print(np.unique(y))
[-1  1]
```



Predicting with a linear model

Classification with Logistic Regression

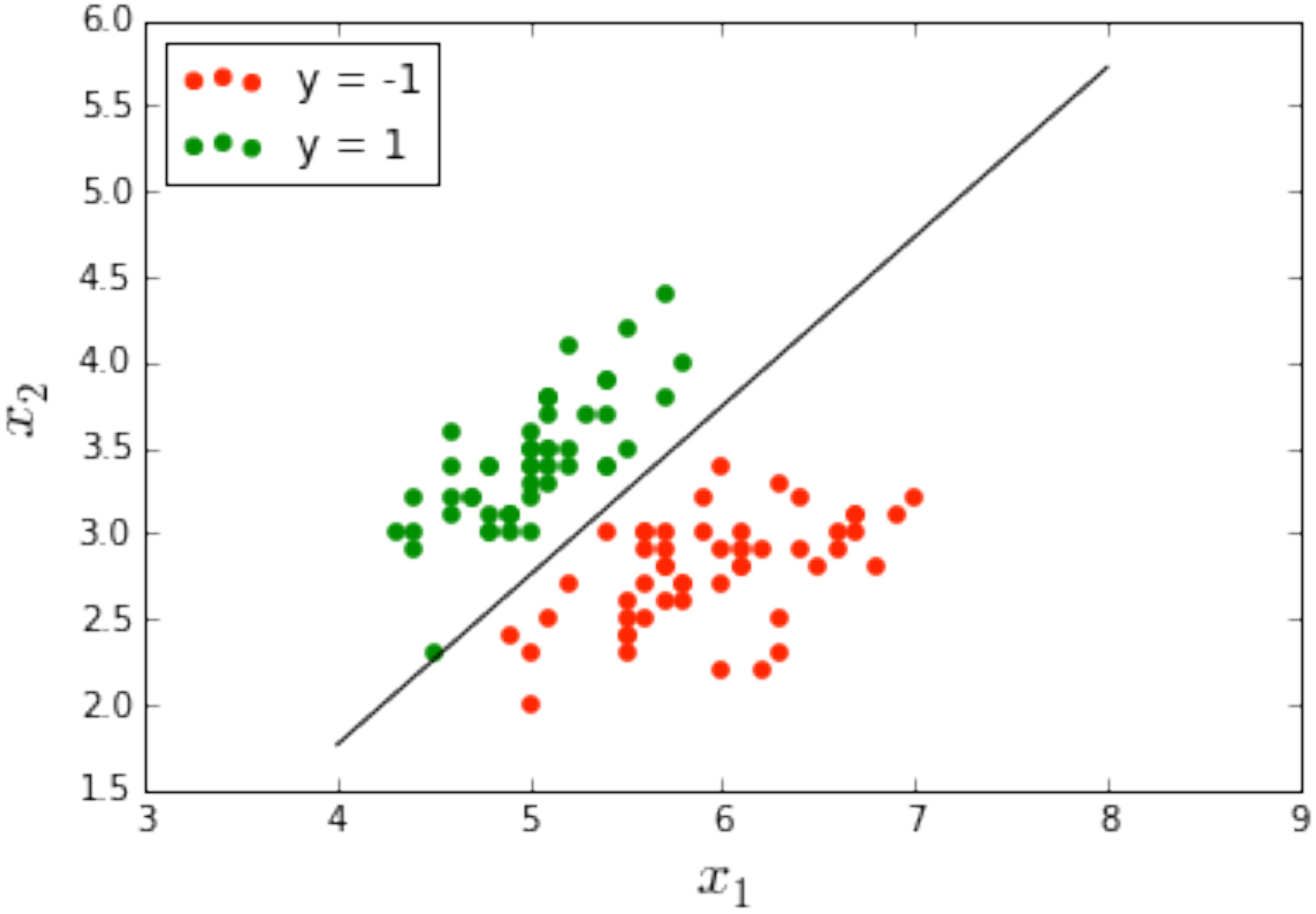
```
>>> from sklearn.linear_model import LogisticRegression
>>> model = LogisticRegression(C=1.)
>>> model.fit(X, y)
>>> theta0 = model.intercept_  # the intercept (theta0)
>>> theta = model.coef_[0]  # the coefficients (theta1, …, theta13)
```



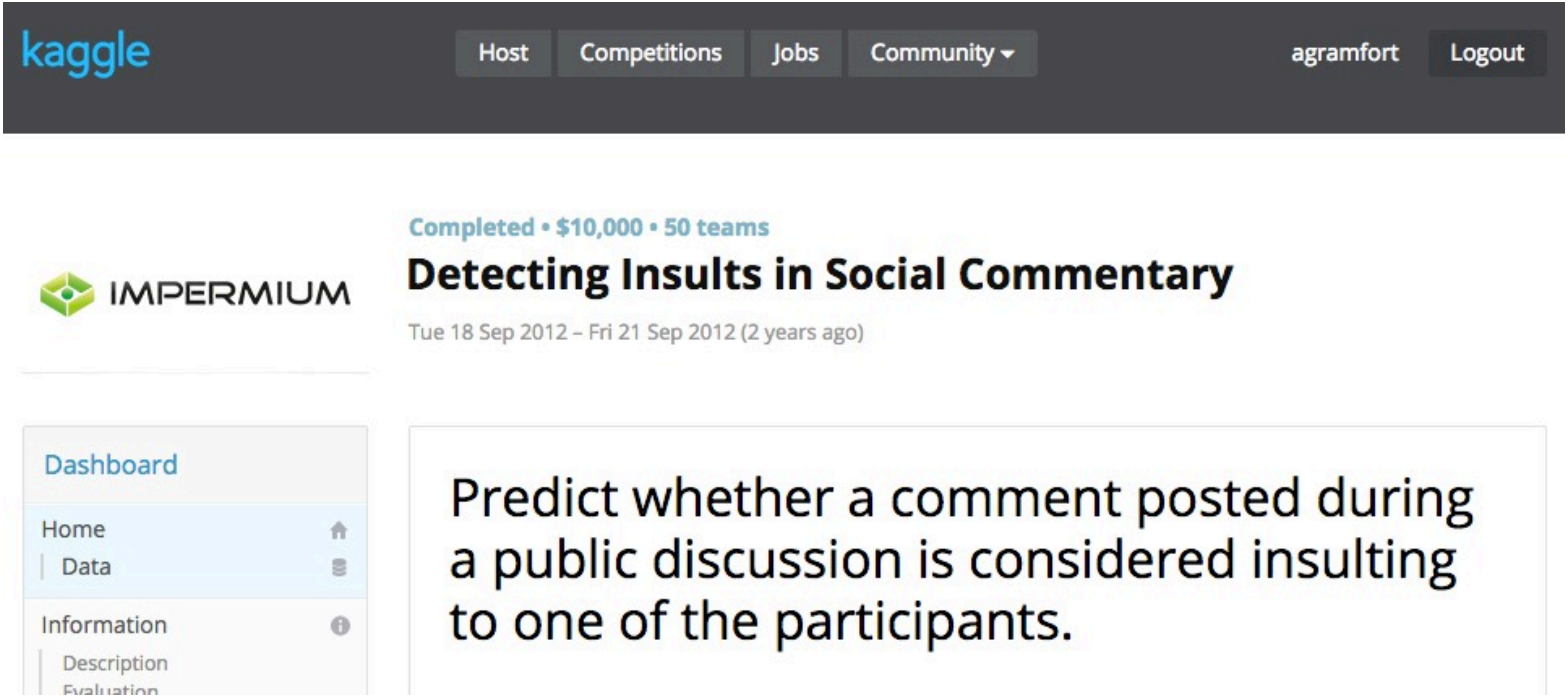
Predicting with a linear model

Classification with Support Vector Machine (SVM)

```
>>> from sklearn.svm import SVC
>>> model = SVC(kernel='linear', C=1.)
>>> model.fit(X, y)
>>> theta0 = model.intercept_   # the intercept (theta0)
>>> theta = model.coef_[0]  # the coefficients (theta1, …, theta13)
```



“Real” life example



Completed • $10,000 • 50 teams

Detecting Insults in Social Commentary

Tue 18 Sep 2012 – Fri 21 Sep 2012 (2 years ago)

Predict whether a comment posted during a public discussion is considered insulting to one of the participants.

https://www.kaggle.com/c/detecting-insults-in-social-commentary

“Real” life example

kaggle

Detecting Insults in Social Commentary

```
>>> !head -2 train.csv
0,"""Imagine being able say, you know what, no sanctions, no forever
hearings on IEAA regulations, no more hiding\xa0under\xa0the pretense of
friendly nuclear energy. \xa0You have 2 days to; \xa0i.e. \xa0let in the
inspectors, quit killing the civilians, respect the border and rights of
your neighboring country, \xa0or we ( whoever we are) will shut off your
nuclear plant, your monitoring system and whatever else we fancy, like
your water\xa0treatment\xa0plants and early warning sandstorm system and
the traffic lights of all major cities...\xa0\nand yes..( pinky finger to
lip edge) so your teenagers revolt and topple your regime...
\xa0disconnect ... FACEBOOK.... buwhahjahahaha."""
0,"""""But Jack from Raleigh wasn't done. He came back with this bit of
furious grammatical genius:""\n""Holy hell, Jack. Calm down.""\n\nGOD
D@MN HILARIOUS!\n\nWho writes your material GraziD?  \n\nMM never even
acknowledged we were here (well accept when Uber ticked him off)  GraziD
not only interacts with us, he calls you dumb when you're being dumb...
right beeaner?"""
```

“Real” life example

kaggle **Detecting Insults in Social Commentary**

```
>>> X = []
y = []
with open('train.csv') as f:
    for line in f:
        y.append(int(line[0]))
        X.append(line[5:-6])
>>> len(X) # number of samples
4415
>>> X[:1]
['Imagine being able say, you know what, no sanctions, no forever
hearings on IEAA regulations, no more hiding\\xa0under\\xa0the pretense
of friendly nuclear energy. \\xa0You have 2 days to; \\xa0i.e. \\xa0let
in the inspectors, quit killing the civilians, respect the border and
rights of your neighboring country, \\xa0or we ( whoever we are) will
shut off your nuclear plant, your monitoring system and whatever else we
fancy, like your water\\xa0treatment\\xa0plants and early warning
sandstorm system and the traffic lights of all major cities...\\xa0\\nand
yes..( pinky finger to lip edge) so your teenagers revolt and topple your
regime... \\xa0disconnect ... FACEBOOK.... buwhahjahahaha']
```

“Real” life example

kaggle

Detecting Insults in Social Commentary

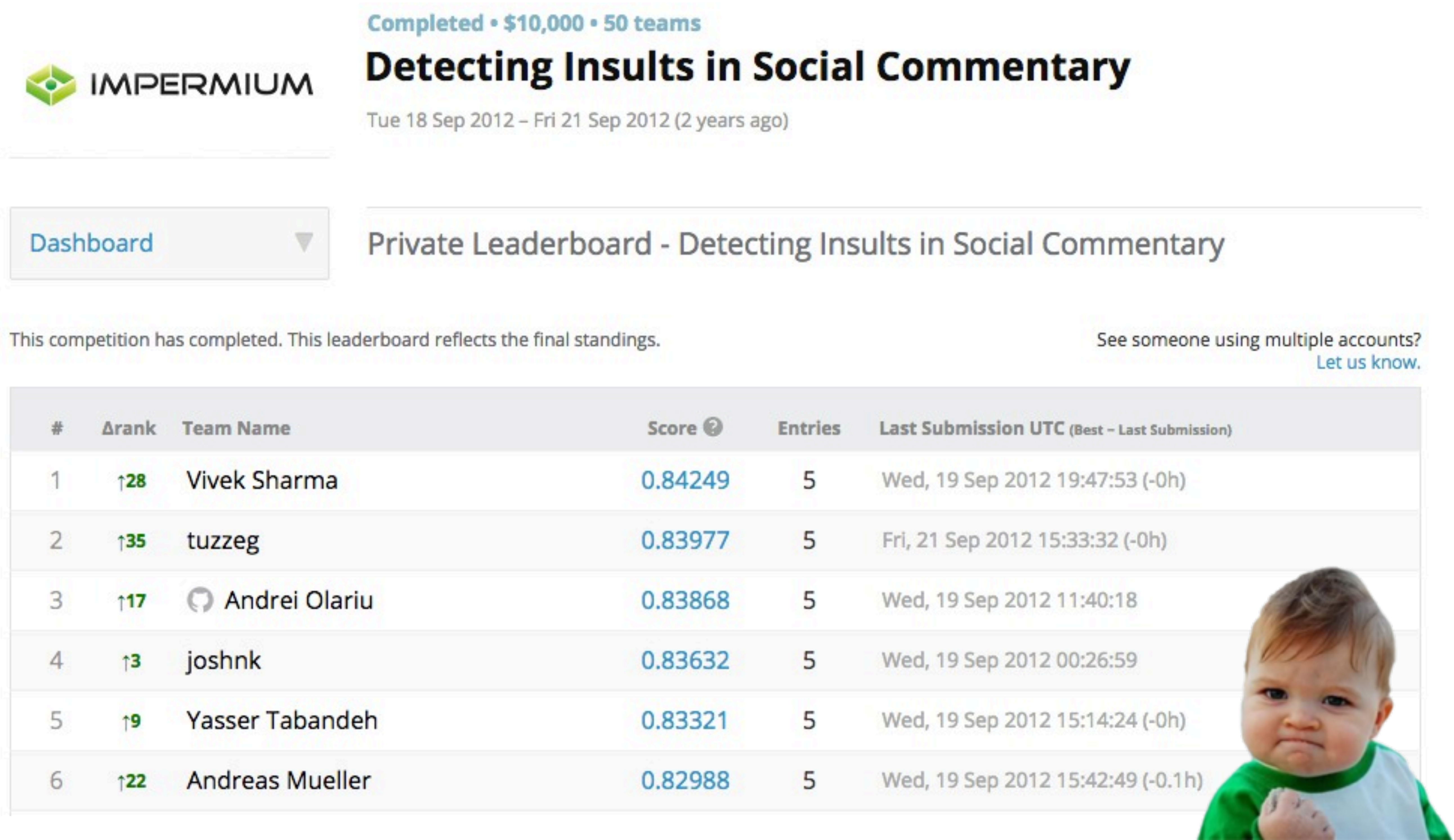
```
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.pipeline import make_pipeline, FeatureUnion
>>> from sklearn.feature_selection import SelectPercentile, chi2
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> from sklearn.cross_validation import cross_val_score
>>> # Define pipeline (text vectorizer, selection, logistic)
>>> select = SelectPercentile(score_func=chi2, percentile=16)
>>> lr = LogisticRegression(tol=1e-8, penalty='l2', C=10.,
                            intercept_scaling=1e3)
>>> char_vect = TfidfVectorizer(ngram_range=(1, 5), analyzer="char")
>>> word_vect = TfidfVectorizer(ngram_range=(1, 3), analyzer="word",
                                min_df=3)
>>> ft = FeatureUnion([("chars", char_vect), ("words", word_vect)])
>>> clf = make_pipeline(ft, select, lr)
```

11 lines of code…

kaggle

Detecting Insults in Social Commentary

```
>>> # run classification
>>> scores = cross_val_score(clf, X, y, cv=2)
>>> print(np.mean(scores))
0.819479193344
```



Completed • $10,000 • 50 teams

IMPERMIUM

Detecting Insults in Social Commentary

Tue 18 Sep 2012 – Fri 21 Sep 2012 (2 years ago)

Private Leaderboard - Detecting Insults in Social Commentary

This competition has completed. This leaderboard reflects the final standings.

See someone using multiple accounts? Let us know.

#	Δrank	Team Name	Score	Entries	Last Submission UTC (Best – Last Submission)
1	↑28	Vivek Sharma	0.84249	5	Wed, 19 Sep 2012 19:47:53 (-0h)
2	↑35	tuzzeg	0.83977	5	Fri, 21 Sep 2012 15:33:32 (-0h)
3	↑17	Andrei Olariu	0.83868	5	Wed, 19 Sep 2012 11:40:18
4	↑3	joshnk	0.83632	5	Wed, 19 Sep 2012 00:26:59
5	↑9	Yasser Tabandeh	0.83321	5	Wed, 19 Sep 2012 15:14:24 (-0h)
6	↑22	Andreas Mueller	0.82988	5	Wed, 19 Sep 2012 15:42:49 (-0.1h)

kaggle

Detecting Insults in Social Commentary

```
>>> XX = ft.fit_transform(X)
>>> print('n_samples: %s, n_features: %s' % XX.shape)
n_samples: 4415, n_features: 226779
```

```
>>> lr = LogisticRegression(tol=1e-8, penalty='l2', C=10.,
                            intercept_scaling=1e3)
```

```
>>> %timeit lr.fit(XX, y)
1 loops, best of 3: 2.36 s per loop
```

Scaling up !

You cannot store everything in memory?

Go online / out of core !

```
>>> from sklearn.linear_model import SGDClassifier

>>> clf = SGDClassifier(alpha=0.1, learning_rate='optimal')

>>> for df in pd.read_csv('data.csv', chunksize=20):
        y = df['target'].values
        X = df.drop('target', axis=1).values
        clf.partial_fit(X, y, classes=[-1, 1])
```

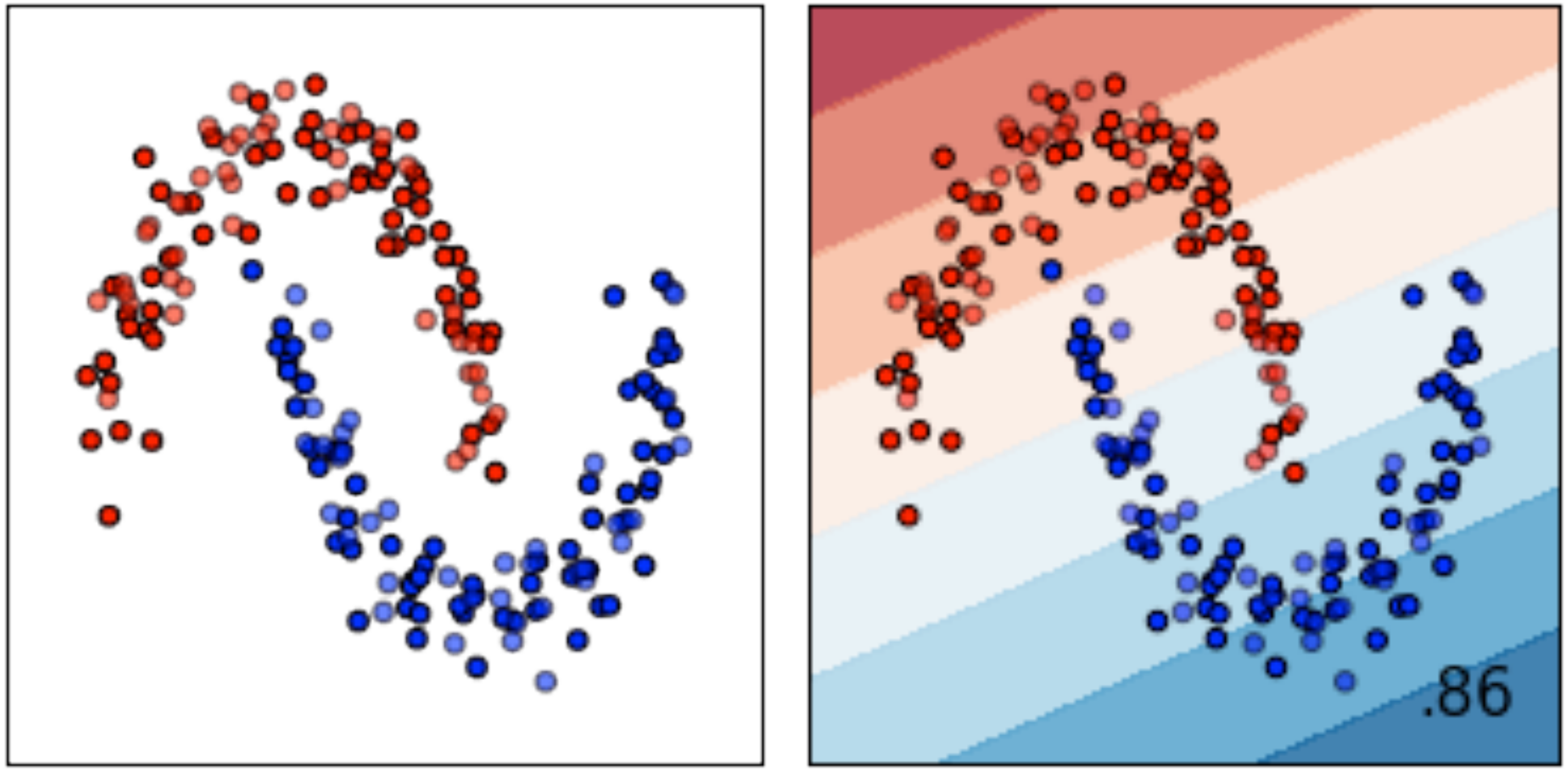
More online algorithms: SGDRegressor, Perceptron, ...

Full out of core example:

http://scikit-learn.org/stable/auto_examples/applications/plot_out_of_core_classification.html

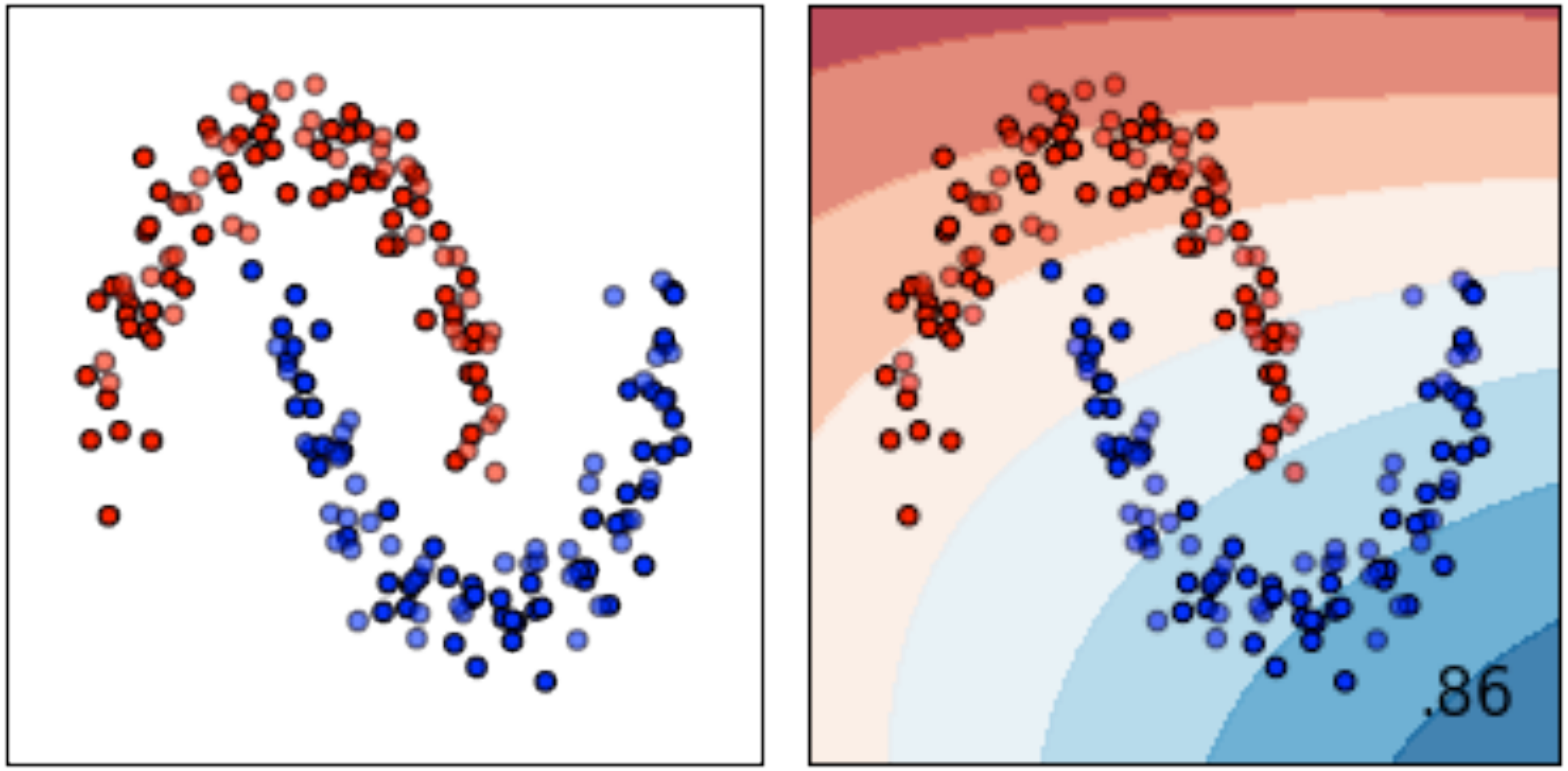
Need to be non-linear?

```
>>> from sklearn.datasets import make_moons
>>> from sklearn.linear_model import LogisticRegression

>>> model = LogisticRegression()
>>> X, y = make_moons(n_samples=200, noise=0.1, random_state=0)
>>> plot_model(model, X, y)
```



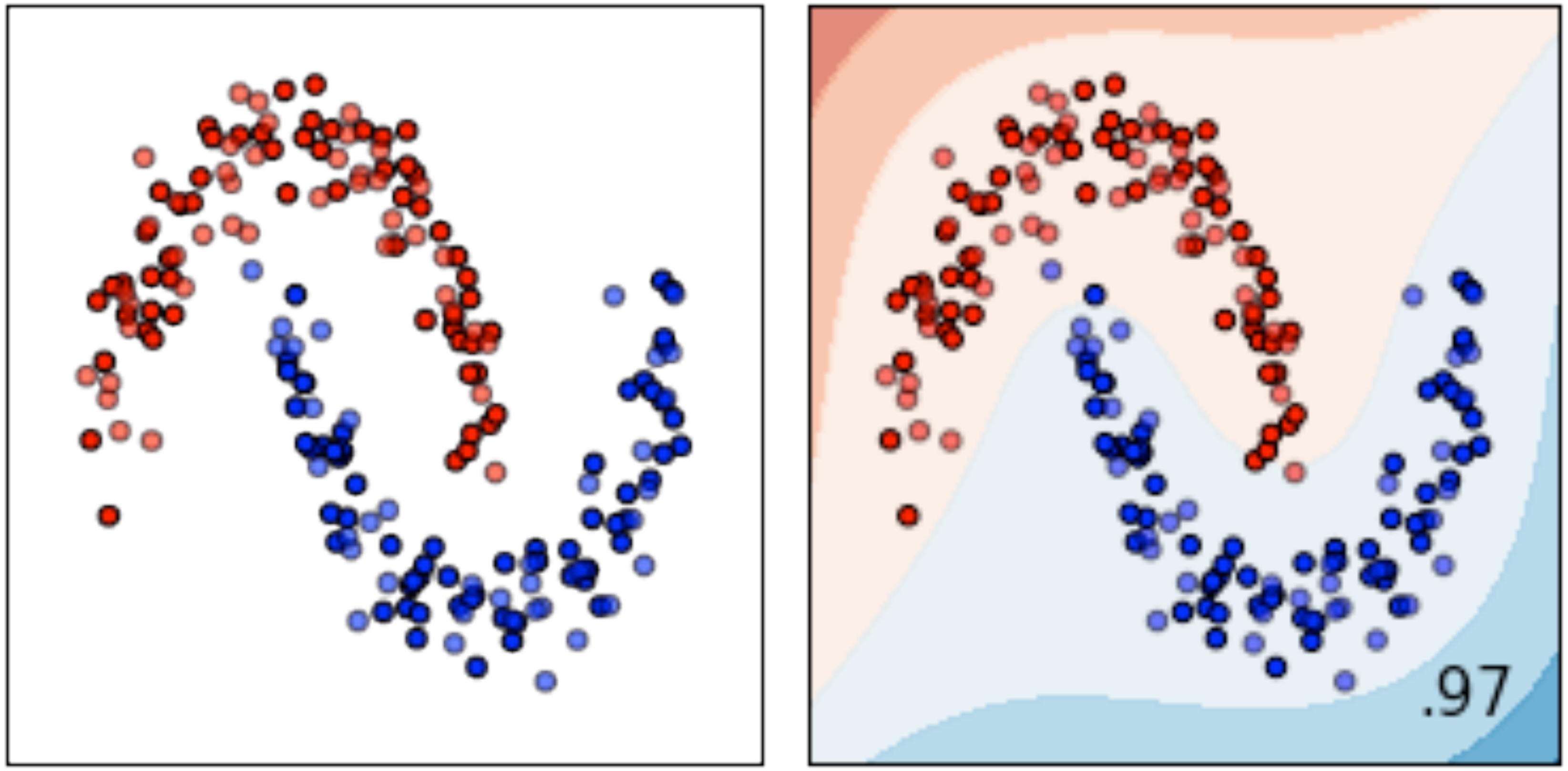
Need to be non-linear?

```
>>> from sklearn.datasets import make_moons
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.preprocessing import PolynomialFeatures

>>> model = make_pipeline(PolynomialFeatures(degree=2),
                          LogisticRegression())
>>> X, y = make_moons(n_samples=200, noise=0.1, random_state=0)
>>> plot_model(model, X, y)
```



Need to be non-linear?

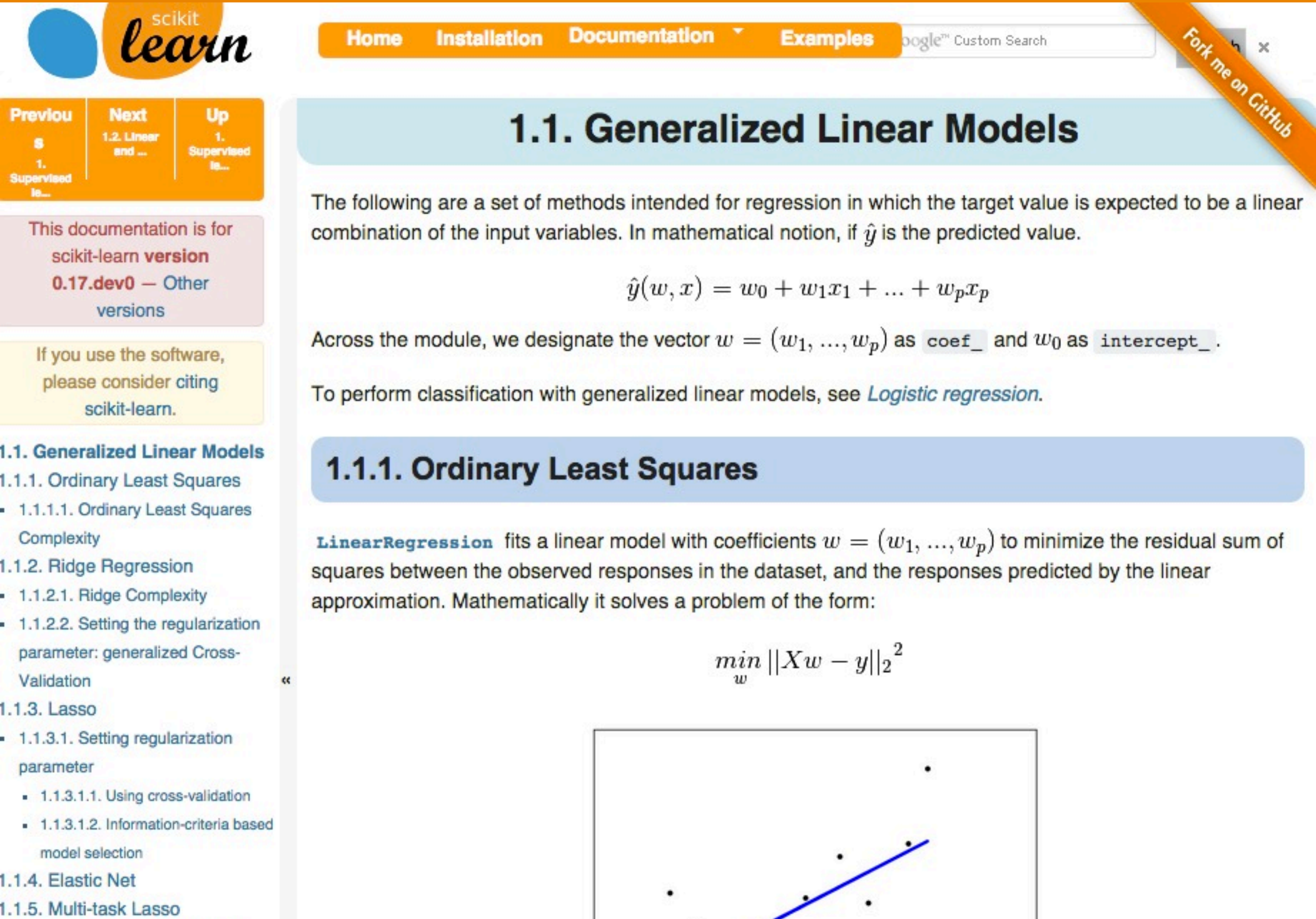
```
>>> from sklearn.datasets import make_moons
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.preprocessing import PolynomialFeatures

>>> model = make_pipeline(PolynomialFeatures(degree=3),
                          LogisticRegression())
>>> X, y = make_moons(n_samples=200, noise=0.1, random_state=0)
>>> plot_model(model, X, y)
```



When to use a linear model?

- When it is the true model
- When your data are linearly separable
- When non-linear models overfit
- When you the number of samples is low compared to number of features
- Because they are simple and efficient !

http://scikit-learn.org/dev/modules/linear_model.html



This documentation is for scikit-learn **version 0.17.dev0** — Other versions

If you use the software, please consider citing scikit-learn.

1.1. Generalized Linear Models

1.1.1. Ordinary Least Squares

- 1.1.1.1. Ordinary Least Squares Complexity

1.1.2. Ridge Regression

- 1.1.2.1. Ridge Complexity
- 1.1.2.2. Setting the regularization parameter: generalized Cross-Validation

1.1.3. Lasso

- 1.1.3.1. Setting regularization parameter
  - 1.1.3.1.1. Using cross-validation
  - 1.1.3.1.2. Information-criteria based model selection

1.1.4. Elastic Net

1.1.5. Multi-task Lasso

1.1. Generalized Linear Models

The following are a set of methods intended for regression in which the target value is expected to be a linear combination of the input variables. In mathematical notion, if $\hat{y}$ is the predicted value.

$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p$$

Across the module, we designate the vector $w = (w_1, ..., w_p)$ as `coef_` and w_0 as `intercept_`.

To perform classification with generalized linear models, see *Logistic regression*.

1.1.1. Ordinary Least Squares

`LinearRegression` fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\underset{w}{min\,} {|| X w - y||_2}^2$$

Contact:

Alexandre Gramfort
alexandre.gramfort@telecom-paristech.fr

GitHub : @agramfort

Twitter : @agramfort

Questions?

2 positions to work on Scikit-Learn and Scipy stack available !

Paris-Saclay
Center for Data Science

TELECOM
ParisTech