```
LinearRegression(copy_X, fit_intercept, n_jobs, normalize)
```

$$y = w^T x + c$$

$$\min_w \|w^T x - y\|_2^2$$

```
Ridge(alpha, copy_X, fit_intercept, max_iter,
normalize, random_state, solver, tol)
```

$$y = w^T x + c$$

$$\min_w \|w^T x - y\|_2^2 + \alpha\|w\|_2^2$$

regularization

prevents overfitting

```
Lasso(alpha, copy_X, fit_intercept, max_iter,
normalize, positive, precompute, random_state,
selection, tol, warm_start)
```

$$y = w^T x + c$$

$$\min_{w} \frac{1}{2n_{samples}} \|w^T x - y\|_2^2 + \alpha \|w\|_1$$

sparse model with few non-zero weights

```
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True,
l1_ratio=0.5,  max_iter=1000, normalize=False,
positive=False, precompute=False, random_state=None,
selection='cyclic', tol=0.0001, warm_start=False)
```

$$y = w_0 + w_1 x_1 + \ldots + w_p x_p$$

$$\min_w \frac{1}{2n_{samples}} \|Xw - y\|_2^2 + \alpha\rho\|w\|_1 + \frac{\alpha(1-\rho)}{2}\|w\|_2^2$$

Ridge + Lasso

```
LogisticRegression(C=1.0, class_weight=None,
dual=False, fit_intercept=True, intercept_scaling=1,
max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear',
tol=0.0001, verbose=0, warm_start=False)
```

$$\text{logit}(p) = \log\left(\frac{P(y=1)}{P(y=0)}\right) = w^T x + c$$

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \log(\exp(-y_i(x_i^T w + c)) + 1)$$

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{N} \log(\exp(-y_i(x_i^T w + c)) + 1)$$

Classification, NOT regression

Логистическая регрессия

$$\log \frac{P(y=1)}{P(y=0)} = w^T x + c$$

$$\log \frac{P(y=1)}{1-P(y=1)} = w^T x + c$$

$$h_w(x) := P(y=1) = \frac{1}{1+e^{-(w^T x + c)}}$$

$$[1-h_w(x)] = P(y=0) = \frac{1}{1+e^{+(w^T x + c)}}$$

Логистическая регрессия

- Обучающее множество (x_i, y_i), $x_i \in R^d$, $y_i \in \{0, 1\}$.

- $h_w(x_i)$ показывает вероятность принадлежности к классу 1 при данных x_i, w и c.

- $[1 - h_w(x_i)]$ показывает вероятность принадлежности к классу 0 при данных x_i, w и c.

- $h_w(x_i)_i^y[1 - h_w(x_i)]^{1-y_i}$ показывает... что?

Логистическая регрессия. Максимальное правдоподобие

$$\prod_{i=1}^{n} h_w(x_i)^{y_i}[1 - h_w(x_i)]^{1-y_i} \to \max_{w,c}$$

$$L(w,c) = \sum_{i=1}^{n} \log h_w(x_i)^{y_i}[1 - h_w(x_i)]^{1-y_i} \to \max_{w,c}$$

$$L(w,c) = \sum_{i=1}^{n} \log \frac{e^{y_i(w^T x + c)}}{1 + e^{(w^T x + c)}} \to \max_{w,c}$$

SGD? BFGS?

Логистическая регрессия

- cross-entropy loss

$$L(w,c) = \sum_{i=1}^{n} \log \frac{e^{y_i(w^T x + c)}}{1 + e^{(w^T x + c)}} \to \max_{w,c}$$

- logistic loss

$$L(w,c) = \sum_{i=1}^{n} \log(\exp(-y_i(w^T x_i + c)) + 1)$$

- wiki: Loss functions for classification

Функции потерь для классификации ($y \in \{-1, 1\}$)

- square loss: $(1 - yf(x))^2$
- hinge loss: $\max(0, 1 - yf(x))$
- logistic loss: $\log(1 + e^{-yf(x)})$
- cross-entropy loss: $-t \log f(x) - (1 - t) \log(1 - f(x))$, $t \in \{0, 1\}$

Activation functions

- logistic: $\dfrac{1}{1+e^{-x}}$
- TanH: $\tanh(x)$
- ReLu: $\begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
- Softmax: $f_i(\vec{x}) = \dfrac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$

Алгоритм обратного распространения ошибки

```
initialize network weights (often small random values)
  do
    forEach training example named ex
      prediction = neural-net-output(network, ex)
      actual = teacher-output(ex)
      compute error (prediction - actual)
          at the output units
      compute Delta w_h for all weights
          from hidden layer to output layer
      compute Delta w_i for all weights
          from input layer to hidden layer
      update network weights
  until all examples classified correctly
      or another stopping criterion satisfied
  return the network
```

The APIs for Neural Networks in TensorFlow

```
import numpy as np
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('/tmp/data', one_hot=True)

Xtrain = mnist.train.images
ytrain = mnist.train.labels
Xtest = mnist.test.images
ytest = mnist.test.labels

N_PIXELS = 28 * 28
N_CLASSES = 10
```

```
HIDDEN_SIZE = 64
EPOCHS = 20
BATCH_SIZE = 64

sess = tf.Session()

x = tf.placeholder(tf.float32, [None, N_PIXELS], name="pixels")
y_label = tf.placeholder(tf.float32,
    [None, N_CLASSES], name="labels")

W1 = tf.Variable(tf.truncated_normal([N_PIXELS, HIDDEN_SIZE],
                                     stddev=N_PIXELS**-0.5))
b1 = tf.Variable(tf.zeros([HIDDEN_SIZE]))

hidden = tf.nn.sigmoid(tf.matmul(x, W1) + b1)
```

```
W2 = tf.Variable(tf.truncated_normal([HIDDEN_SIZE, N_CLASSES],
                                     stddev=HIDDEN_SIZE**-0.5))
b2 = tf.Variable(tf.zeros([N_CLASSES]))

y = tf.matmul(hidden, W2) + b2

loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(logits=y,
                                            labels=y_label))

accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(y, 1),
                      tf.argmax(y_label, 1)), tf.float32))

sgd = tf.train.GradientDescentOptimizer(0.5).minimize(loss)
```

```
sess.run(tf.global_variables_initializer())
inds = range(Xtrain.shape[0])

for i in xrange(EPOCHS):
  np.random.shuffle(inds)
  for j in xrange(0, len(inds), BATCH_SIZE):
    sess.run(sgd, feed_dict={x: Xtrain[inds[j:j+BATCH_SIZE]],
                          label: ytrain[inds[j:j+BATCH_SIZE]]})

  print sess.run([loss, accuracy],
      feed_dict={x: Xtest, y_label: ytest})
```