Machine Learning: Overview

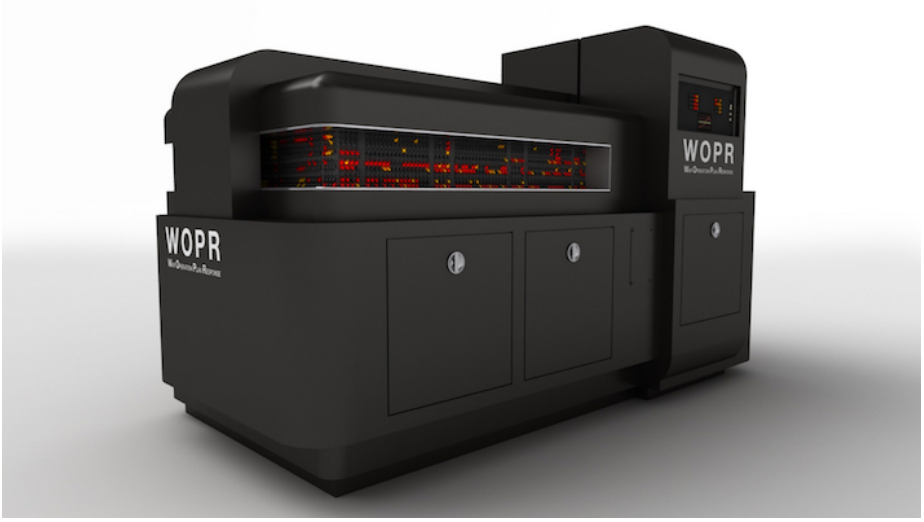
Speaker Qualifications

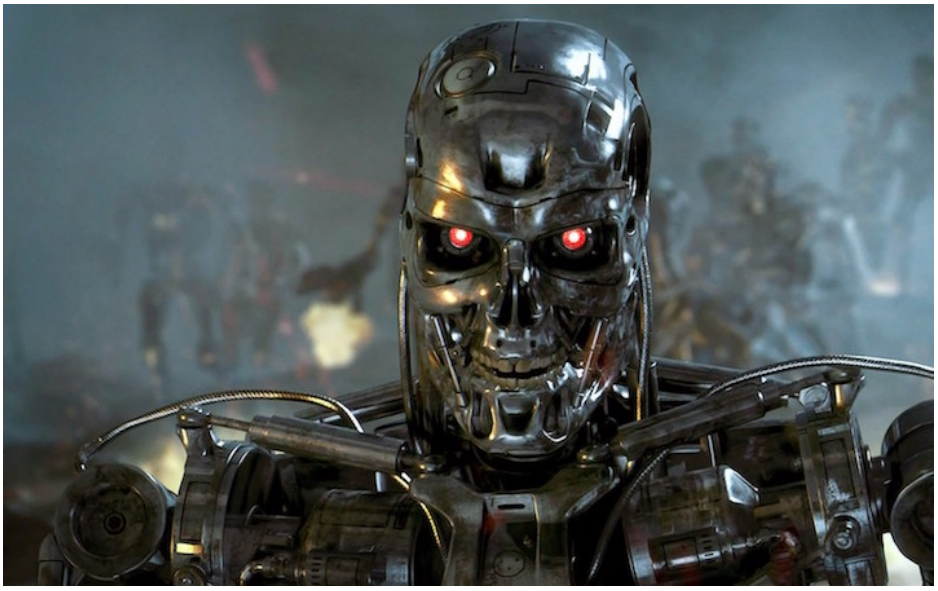
- Specialize in next-generation technologies
- Author of O'Reilly Videos on Hypermedia, Linking Data, Security and Encryption
- Author of 'Resource-Oriented Architecture Patterns for Webs of Data'
- Teaches and speaks internationally about REST, Semantic Web, Data Science, Machine Learning, GPU Computing, Security, Visualization, Architecture
- Worked in Defense, Finance, Retail, Hospitality, Video Game, Health Care, Telecommunications and Publishing Industries
- International Pop Recording Artist

Agenda

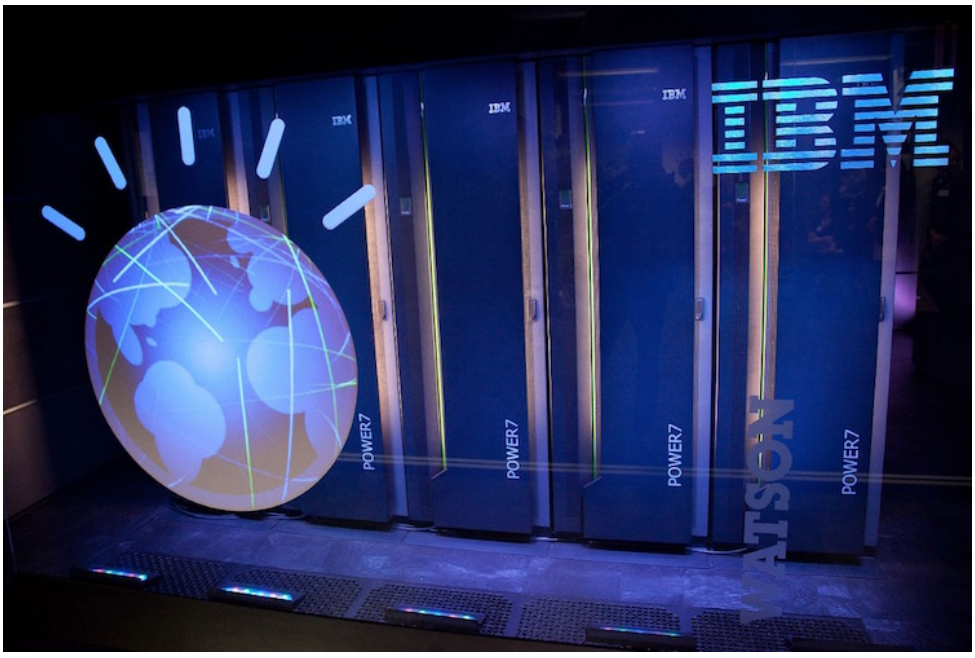
- Introduction
- Algorithms
- Neural Networks

Introduction


WOPR
WOPR



https://youtu.be/rVlhMGQgDkY?t=85



"The term machine learning refers to the automated detection of meaningful patterns in data."

Source: Shavel-Shwartz and Ben-David, "Understanding Machine Learning: From Theory to Algorithms"

Advancement of Machine Learning

Advancement of Machine Learning

- Stock market prediction in the 1980s

Advancement of Machine Learning

- Stock market prediction in the 1980s
- Mining corporate databases in the 1990s (direct marketing, CRM, credit scoring, fraud detection)

Advancement of Machine Learning

- Stock market prediction in the 1980s
- Mining corporate databases in the 1990s (direct marketing, CRM, credit scoring, fraud detection)
- E-commerce (personalization, click analysis)

Advancement of Machine Learning

- Stock market prediction in the 1980s
- Mining corporate databases in the 1990s (direct marketing, CRM, credit scoring, fraud detection)
- E-commerce (personalization, click analysis)
- 9/11 brought interest to applying ML to fighting terror

Advancement of Machine Learning

- Stock market prediction in the 1980s
- Mining corporate databases in the 1990s (direct marketing, CRM, credit scoring, fraud detection)
- E-commerce (personalization, click analysis)
- 9/11 brought interest to applying ML to fighting terror
- Web 2.0 (social networks, sentiment analysis, etc.)

Advancement of Machine Learning

- Stock market prediction in the 1980s
- Mining corporate databases in the 1990s (direct marketing, CRM, credit scoring, fraud detection)
- E-commerce (personalization, click analysis)
- 9/11 brought interest to applying ML to fighting terror
- Web 2.0 (social networks, sentiment analysis, etc.)
- Science (molecular biologists and astronomers were early adopters)

Advancement of Machine Learning

- Stock market prediction in the 1980s
- Mining corporate databases in the 1990s (direct marketing, CRM, credit scoring, fraud detection)
- E-commerce (personalization, click analysis)
- 9/11 brought interest to applying ML to fighting terror
- Web 2.0 (social networks, sentiment analysis, etc.)
- Science (molecular biologists and astronomers were early adopters)
- Housing bust freed up a lot of talent

Advancement of Machine Learning

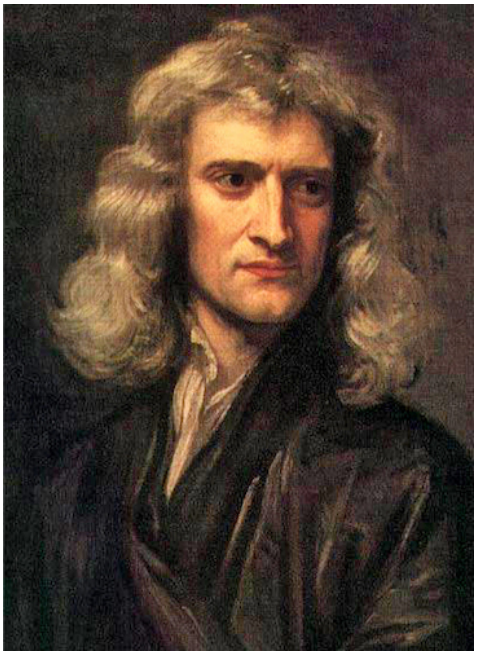
- Stock market prediction in the 1980s
- Mining corporate databases in the 1990s (direct marketing, CRM, credit scoring, fraud detection)
- E-commerce (personalization, click analysis)
- 9/11 brought interest to applying ML to fighting terror
- Web 2.0 (social networks, sentiment analysis, etc.)
- Science (molecular biologists and astronomers were early adopters)
- Housing bust freed up a lot of talent
- Big Data

David Hume

Inductivist Turkey

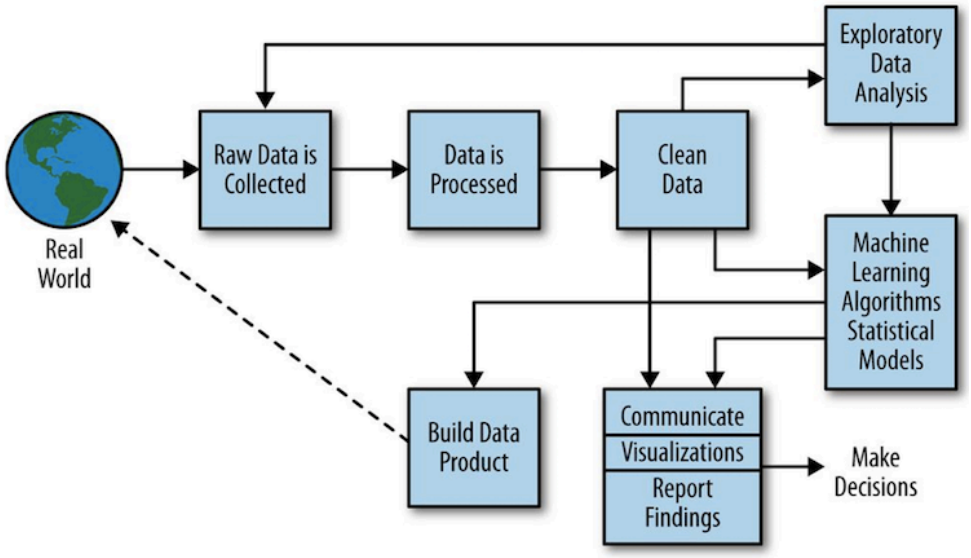
Tycho Brahe

Johannes Kepler

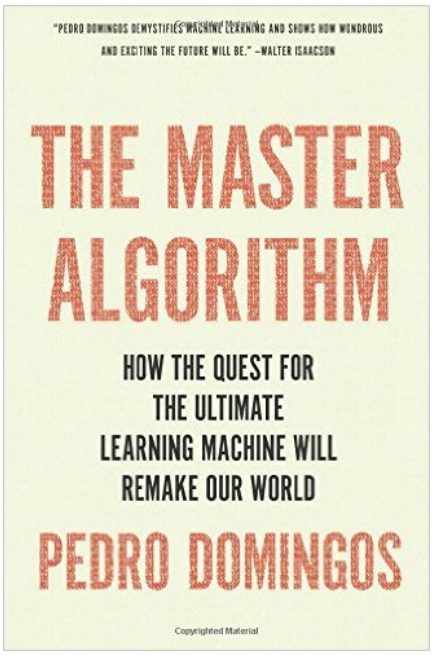


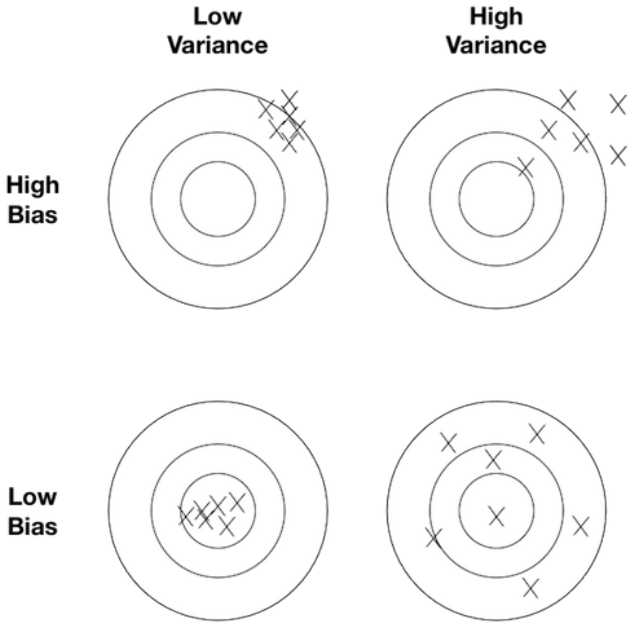
Isaac Newton

Algorithms



O'Neil and Schutt, "Doing Data Science"





Domingos, "Master Algortihm : : How the Quest for the Ultimate Learning Machine Will Remake Our World"

Tribe	Approach	Master Algorithm
Evolutionaries	Natural Selection/Deriving Learning Structure	Genetic Programming
Connectionists	Reverse engineer the brain	Backpropagation
Symbolists	Symbol manipulation from initial knowledge	Inverse Deduction
Bayesians	Managing uncertainty, noisy, incomplete and contradictory information	Probabilistic Inference (Bayes' Theorem)
Analogizers	Recognizing similarities	Support Vector Machines

Sample Algorithms

Sample Algorithms

- Linear Regression

Sample Algorithms

- Linear Regression
- k-Nearest Neighbors

Sample Algorithms

- Linear Regression
- k-Nearest Neighbors
- Naive Bayes

Sample Algorithms

- Linear Regression
- k-Nearest Neighbors
- Naive Bayes
- Decision Trees

Sample Algorithms

- Linear Regression
- k-Nearest Neighbors
- Naive Bayes
- Decision Trees
- Support Vector Machines

Linear Regression

"Linear Regression: In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables denoted X."

Source: https://en.wikipedia.org/wiki/Linear_regression

Ordinary Least Squares

Ordinary Least Squares

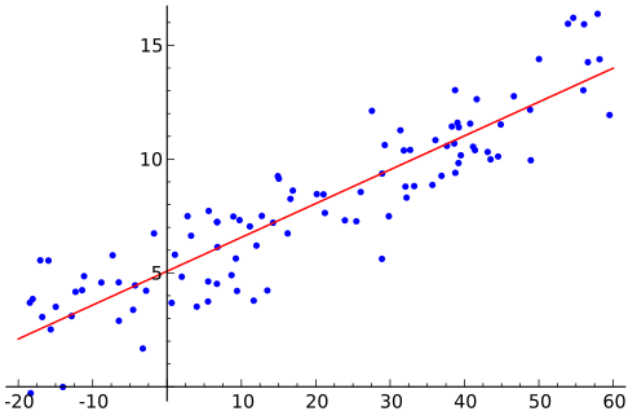
- Simple Linear Regression

Ordinary Least Squares

- Simple Linear Regression
- Fit a straight line through the observed points

Ordinary Least Squares

- Simple Linear Regression
- Fit a straight line through the observed points
- Minimizes the sum of square residuals of the model



https://commons.wikimedia.org/wiki/File:Linear_regression.svg

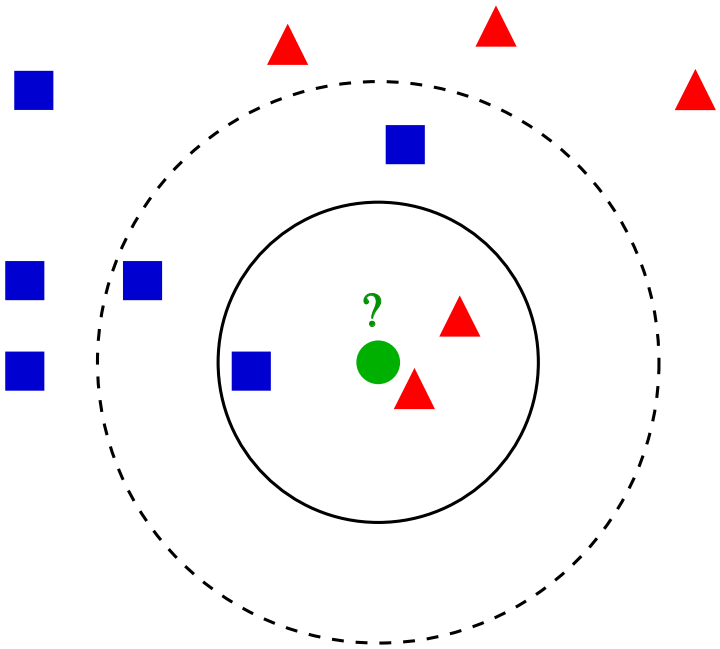
Linear Regression

Pros	Cons
Common approach for numeric data	Strong assumptions about the data
Can handle most modeling tasks	Model form must be specified in advance
Estimates the strength and size of the relationships among features and outcomes	Does not handle missing data
	Only numeric features
	Requires statistical knowledge to understand the model

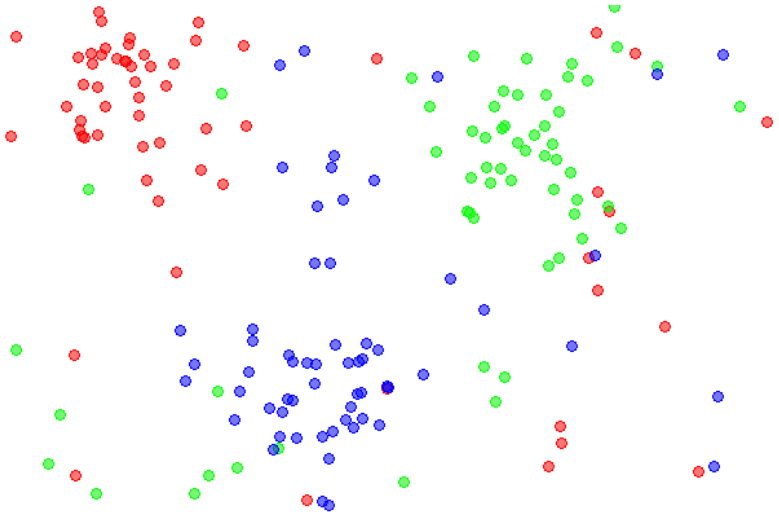
k-Nearest Neighbors

"k-Nearest Neighbor: a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression."

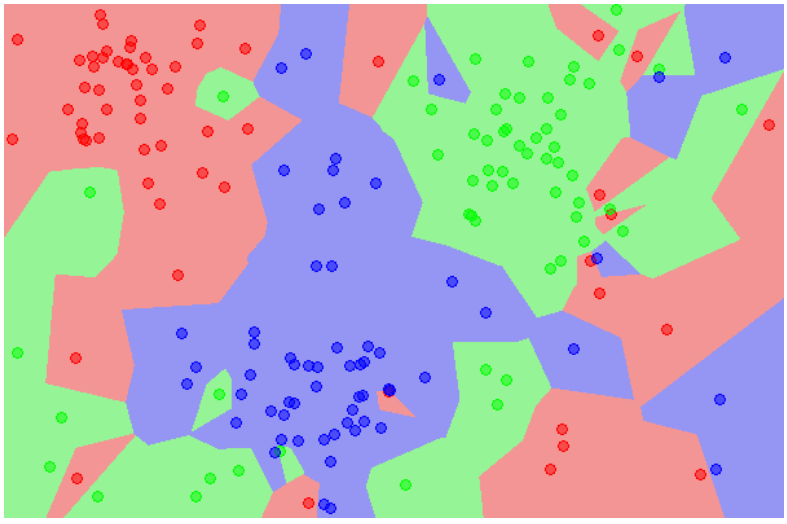
Source: http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm



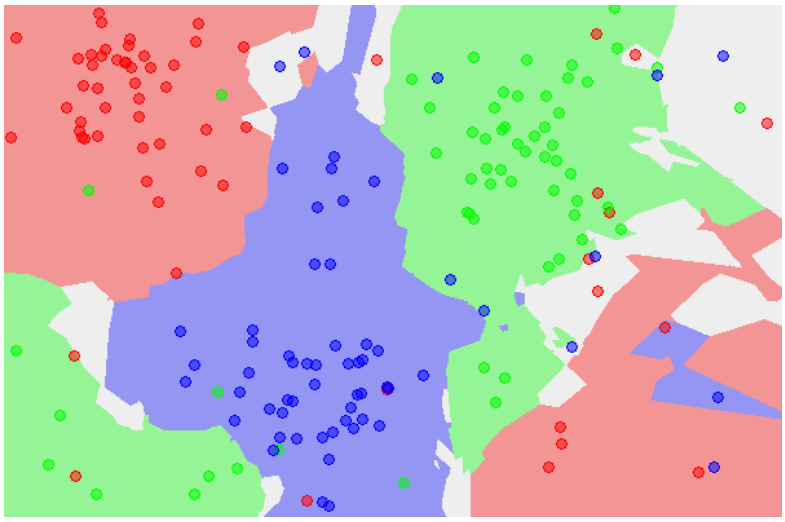
http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg



http://commons.wikimedia.org/wiki/File:Data3classes.png

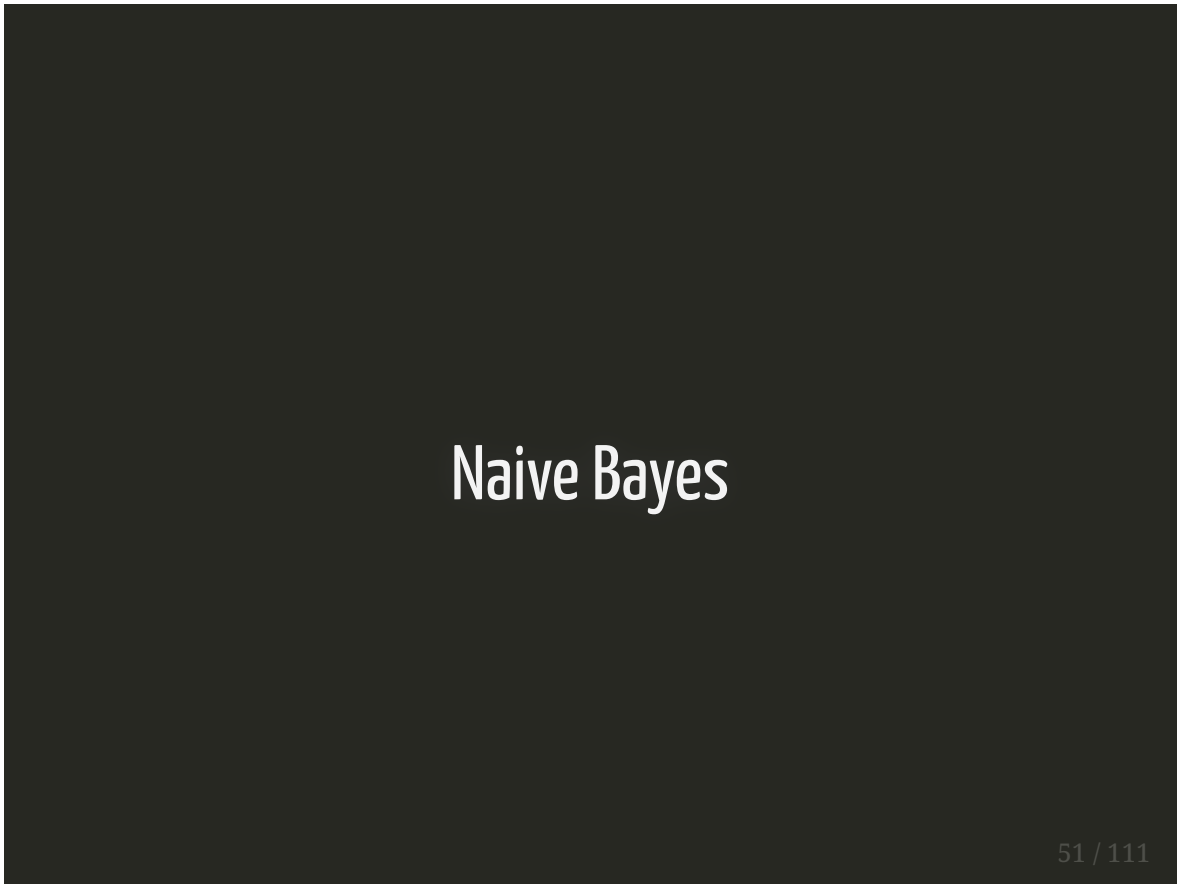


http://commons.wikimedia.org/wiki/File:Map1NN.png



http://commons.wikimedia.org/wiki/File:Map5NN.png

Pros	Cons
Simple and effective	Does not produce a model
Makes no assumptions about the data distribution	Efficacy affected by choice of k
Fast training phase	Slow classification phase



Naive Bayes

Naive Bayes

- Family of algorithms to produce probabilistic classifiers based on Bayes Theorem

Naive Bayes

- Family of algorithms to produce probabilistic classifiers based on Bayes Theorem
- Requires relatively little training data

Naive Bayes

- Family of algorithms to produce probabilistic classifiers based on Bayes Theorem
- Requires relatively little training data
- Often used for text/document classification

Naive Bayes

- Family of algorithms to produce probabilistic classifiers based on Bayes Theorem
- Requires relatively little training data
- Often used for text/document classification
- Assumes independence of the features

$$P(A), P(B)$$

$$P(A), P(B)$$

$$P(A \cap B) = P(A) \cdot P(B)$$

$$P(A), P(B)$$

$$P(A \cap B) = P(A) \cdot P(B)$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A), P(B)$$

$$P(A \cap B) = P(A) \cdot P(B)$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

$$P(A), P(B)$$

$$P(A \cap B) = P(A) \cdot P(B)$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

$$P(spam|Viagra) = \frac{P(Viagra|spam) \cdot P(spam)}{P(Viagra)}$$

Pros	Cons
Simple and effective	Assumption of the independence of features is usually wrong
Does well with noisy and missing data	Doesn't work well with lots of numeric features
Works well with arbitrary sizes of training data	Estimated probabilities aren't as reliable as the classifications
Easy to produce the estimated probability for predictions	

Decision Trees

Decision Trees

Decision Trees

- Tree structure-based classifier

Decision Trees

- Tree structure-based classifier
- Models relationships between features and outputs

Decision Trees

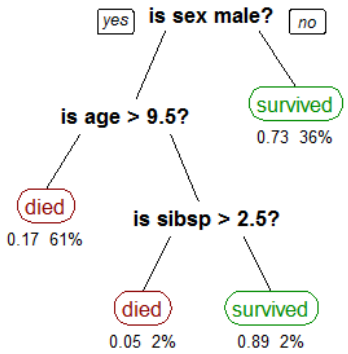
- Tree structure-based classifier
- Models relationships between features and outputs
- Easy to explain to users

Decision Trees

- Tree structure-based classifier
- Models relationships between features and outputs
- Easy to explain to users
- Can be turned into external representation beyond the classifier

Decision Trees

- Tree structure-based classifier
- Models relationships between features and outputs
- Easy to explain to users
- Can be turned into external representation beyond the classifier
- Supports both classification and regression



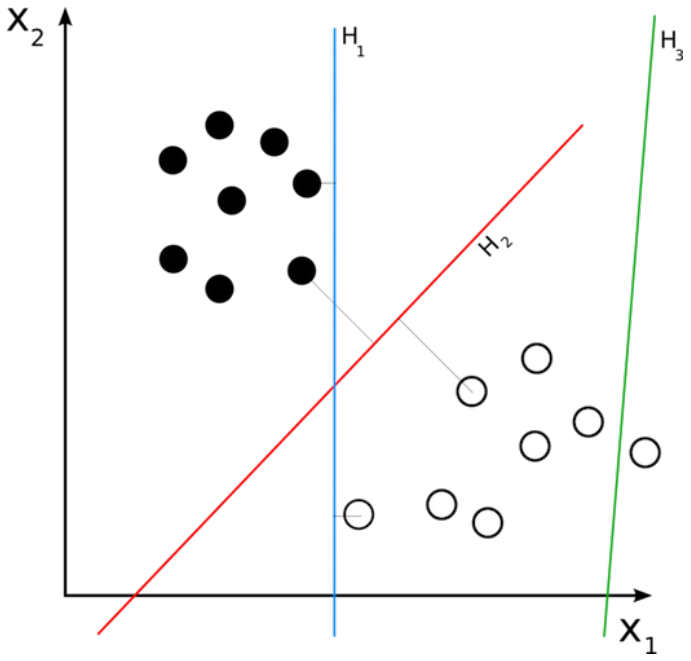
https://en.wikipedia.org/wiki/Decision_tree_learning

Pros	Cons
Useful classifier for most problems	Can be biased toward feature splits with several levels
Automated learning process	Easy to misfit the model
Supports numeric, nominal and missing data	Small changes in the training data can have been impact on decision logic
Works with large and small data sets	Large trees may be hard to interpret
Easily interpreted	
Efficient	

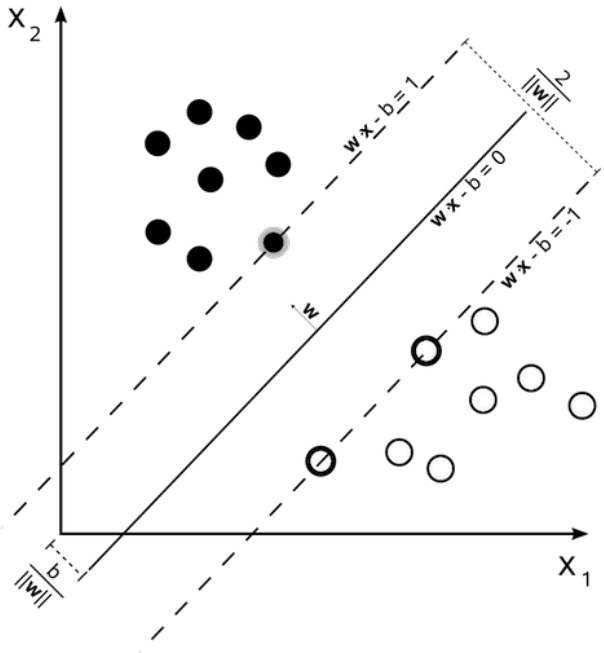
Support Vector Machines

"[a] support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks"

Source: https://en.wikipedia.org/wiki/Support_vector_machine



https://en.wikipedia.org/wiki/Support_vector_machine



https://en.wikipedia.org/wiki/Support_vector_machine

Neural Networks

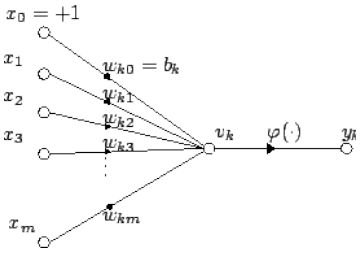
Modeling Neurons

Modeling Neurons

- Walter Pitts and Warren McCulloch in 1943

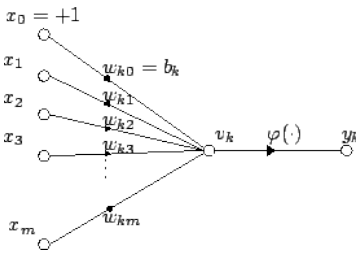
Modeling Neurons

- Walter Pitts and Warren McCulloch in 1943



Modeling Neurons

- Walter Pitts and Warren McCulloch in 1943



$$y_k = \phi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

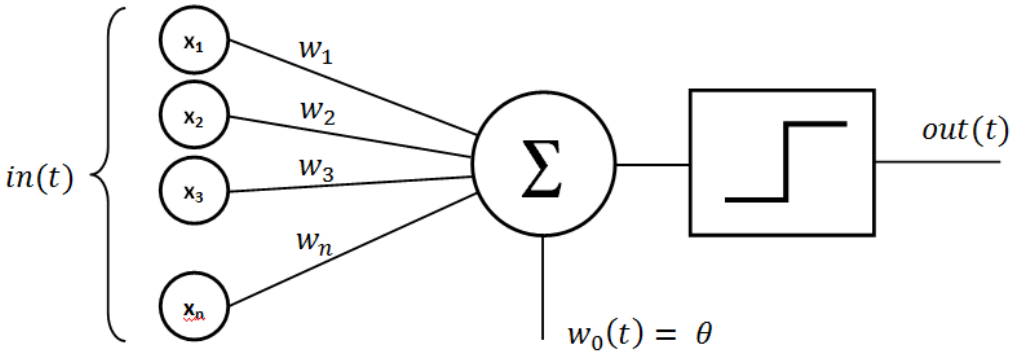
Perceptrons

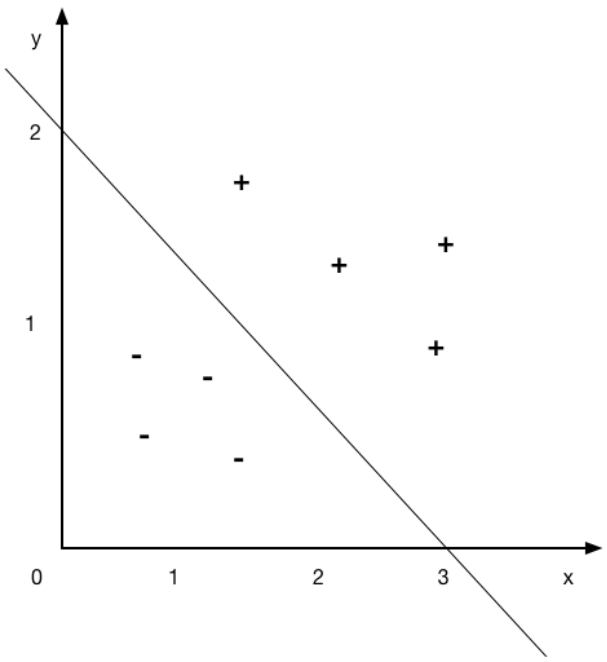
Perceptrons

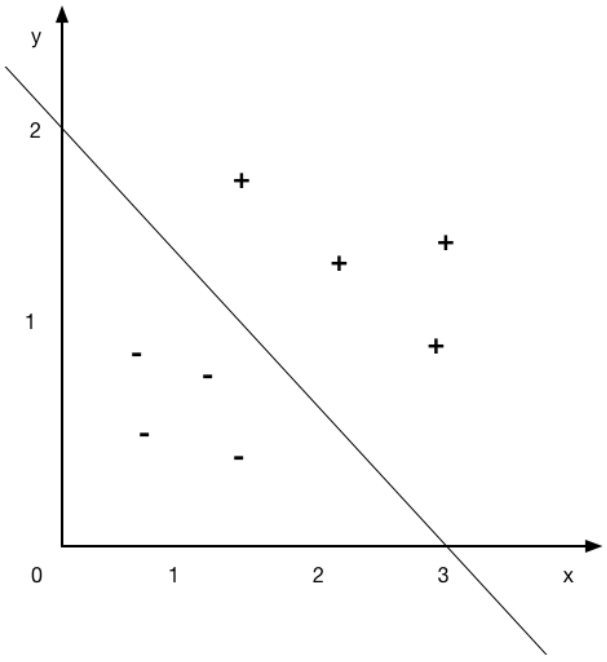
- Frank Rosenblatt in late 1950s

Perceptrons

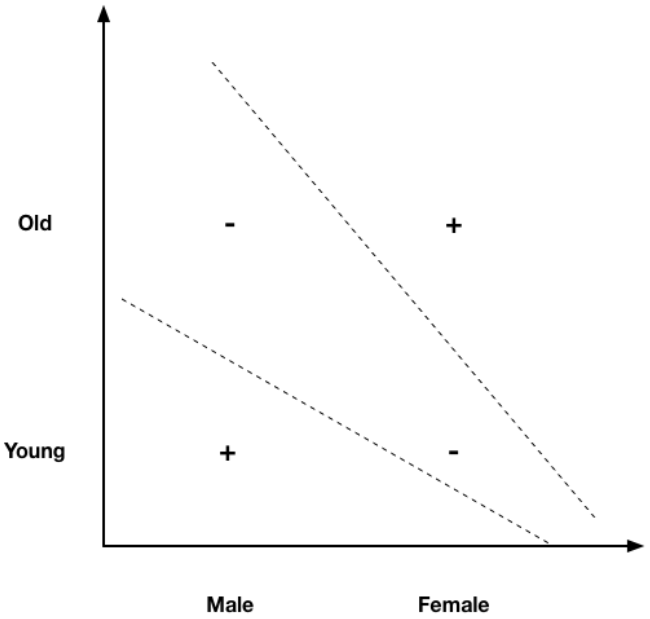
- Frank Rosenblatt in late 1950s

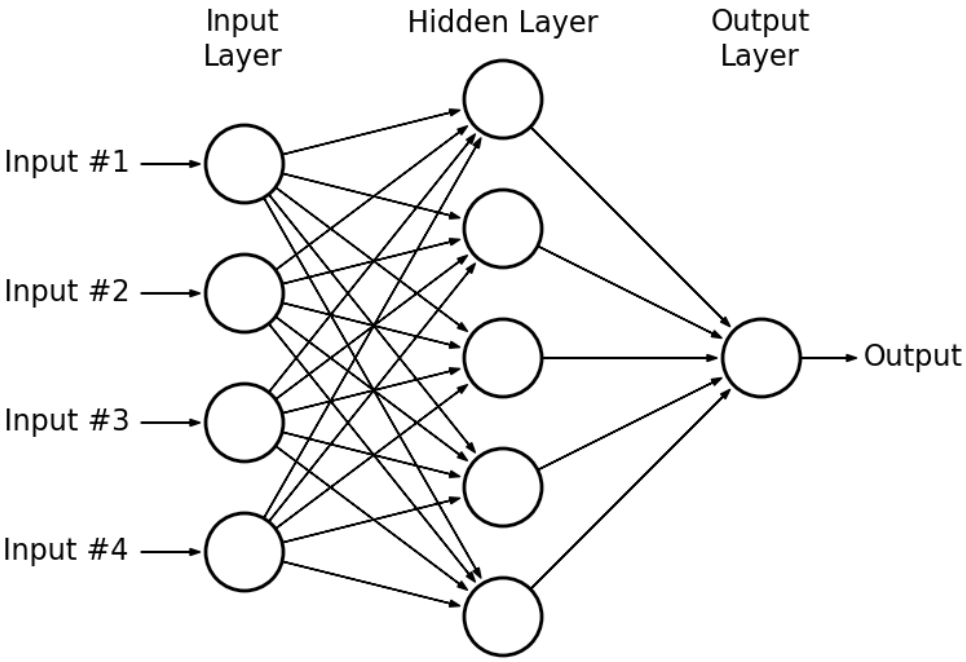






$$2x + 3y = 6$$


Old
-
+
Young
+
-
Male
Female


Input Layer
Hidden Layer
Output Layer
Input #1
Input #2
Input #3
Input #4
Output

http://neuralnetworksanddeeplearning.com/chap1.html

http://colah.github.io/posts/2015-08-Backprop/

http://cs231n.github.io/optimization-2/

https://github.com/dennybritz/nn-from-scratch


```
# Create and activate new virtual environment (optional)
virtualenv venv
source venv/bin/activate
# Install requirements
pip install -r requirements.txt
# Start the notebook server
jupyter notebook
```

```
# Create and activate new virtual environment (optional)
virtualenv venv
source venv/bin/activate
# Install requirements
pip install -r requirements.txt
# Start the notebook server
jupyter notebook
```

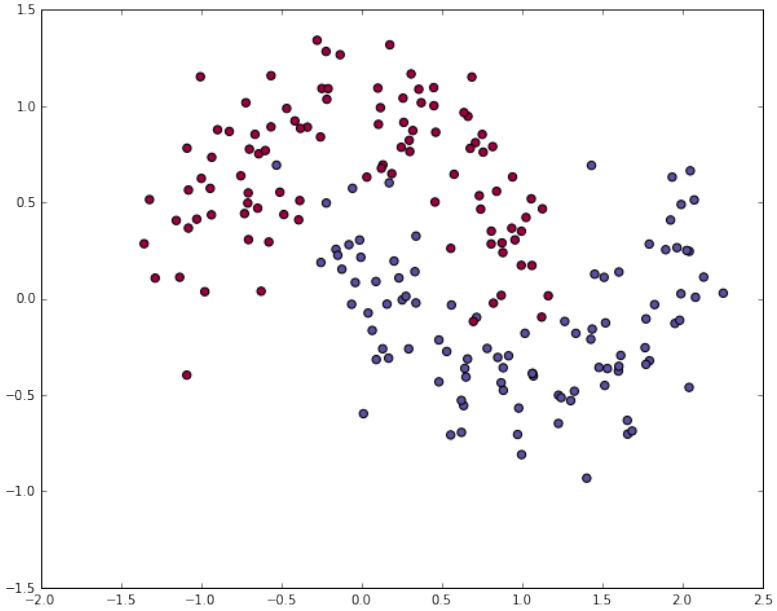
```
# Package imports
import matplotlib.pyplot as plt
import numpy as np
import sklearn
import sklearn.datasets
import sklearn.linear_model
import matplotlib
```



```
# Create and activate new virtual environment (optional)
virtualenv venv
source venv/bin/activate
# Install requirements
pip install -r requirements.txt
# Start the notebook server
jupyter notebook
```

```
# Package imports
import matplotlib.pyplot as plt
import numpy as np
import sklearn
import sklearn.datasets
import sklearn.linear_model
import matplotlib
```


```
# Generate a dataset and plot it
np.random.seed(0)
X, y = sklearn.datasets.make_moons(200, noise=0.20)
plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=plt.cm.Spectral)
```




```
# Train the logistic regeression classifier
clf = sklearn.linear_model.LogisticRegressionCV()
clf.fit(X, y)
```

```
# Train the logistic regeression classifier
clf = sklearn.linear_model.LogisticRegressionCV()
clf.fit(X, y)
```

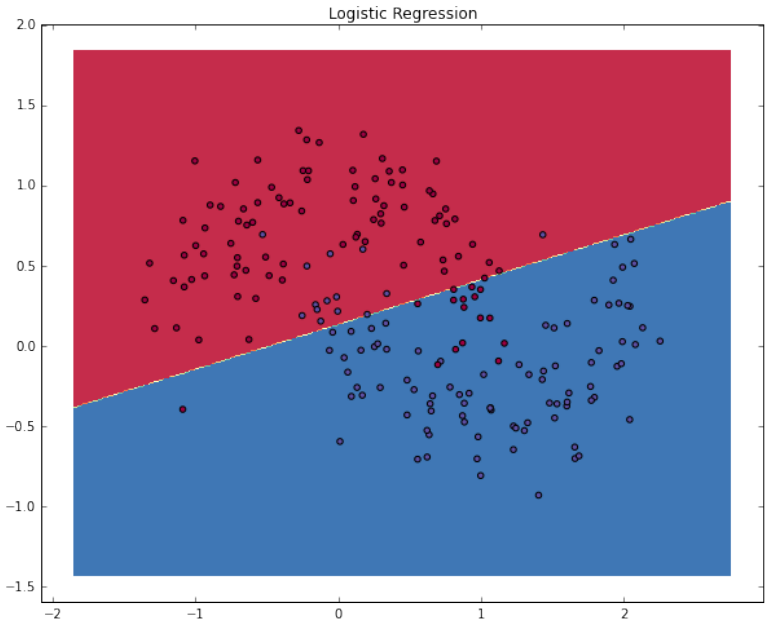
```
# Helper function to plot a decision boundary.
# If you don't fully understand this function don't worry, it just generates the c
def plot_decision_boundary(pred_func):
    # Set min and max values and give it some padding
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.01
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Predict the function value for the whole gid
    Z = pred_func(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
```

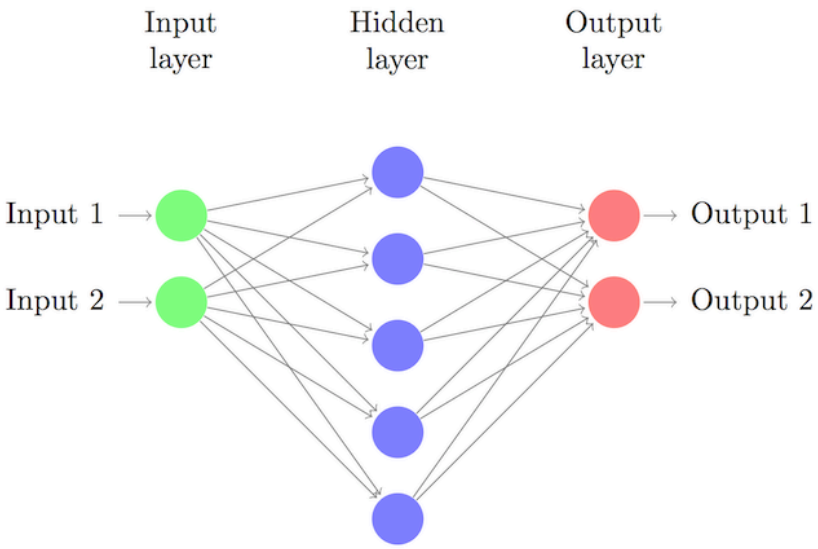


```
# Train the logistic regeression classifier
clf = sklearn.linear_model.LogisticRegressionCV()
clf.fit(X, y)
```

```
# Helper function to plot a decision boundary.
# If you don't fully understand this function don't worry, it just generates the c
def plot_decision_boundary(pred_func):
    # Set min and max values and give it some padding
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.01
    # Generate a grid of points with distance h between them
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Predict the function value for the whole gid
    Z = pred_func(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
```


```
# Plot the decision boundary
plot_decision_boundary(lambda x: clf.predict(x))
plt.title("Logistic Regression")
```


Logistic Regression
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2


Input layer
Hidden layer
Output layer
Input 1
Input 2
Output 1
Output 2

```
num_examples = len(X) # training set size
nn_input_dim = 2 # input layer dimensionality
nn_output_dim = 2 # output layer dimensionality

# Gradient descent parameters (I picked these by hand)
epsilon = 0.01 # learning rate for gradient descent
reg_lambda = 0.01 # regularization strength
```



```
num_examples = len(X) # training set size
nn_input_dim = 2 # input layer dimensionality
nn_output_dim = 2 # output layer dimensionality

# Gradient descent parameters (I picked these by hand)
epsilon = 0.01 # learning rate for gradient descent
reg_lambda = 0.01 # regularization strength
```


```
# Helper function to evaluate the total loss on the dataset
def calculate_loss(model):
    W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']
    # Forward propagation to calculate our predictions
    z1 = X.dot(W1) + b1
    a1 = np.tanh(z1)
    z2 = a1.dot(W2) + b2
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    # Calculating the loss
    corect_logprobs = -np.log(probs[range(num_examples), y])
    data_loss = np.sum(corect_logprobs)
    # Add regulatization term to loss (optional)
    data_loss += reg_lambda/2 * (np.sum(np.square(W1)) + np.sum(np.square(W2)))
    return 1./num_examples * data_loss
```

```
# Helper function to predict an output (0 or 1)
def predict(model, x):
    W1, b1, W2, b2 = model['W1'], model['b1'], model['W2'], model['b2']
    # Forward propagation
    z1 = x.dot(W1) + b1
    a1 = np.tanh(z1)
    z2 = a1.dot(W2) + b2
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    return np.argmax(probs, axis=1)
```



```
# This function learns parameters for the neural network and returns the model.
# - nn_hdim: Number of nodes in the hidden layer
# - num_passes: Number of passes through the training data for gradient descent
# - print_loss: If True, print the loss every 1000 iterations
def build_model(nn_hdim, num_passes=20000, print_loss=False):

    # Initialize the parameters to random values. We need to learn these.
    np.random.seed(0)
    W1 = np.random.randn(nn_input_dim, nn_hdim) / np.sqrt(nn_input_dim)
    b1 = np.zeros((1, nn_hdim))
    W2 = np.random.randn(nn_hdim, nn_output_dim) / np.sqrt(nn_hdim)
    b2 = np.zeros((1, nn_output_dim))

    # This is what we return at the end
    model = {}

    # Gradient descent. For each batch...
    for i in xrange(0, num_passes):

        # Forward propagation
        z1 = X.dot(W1) + b1
        a1 = np.tanh(z1)
        z2 = a1.dot(W2) + b2
        exp_scores = np.exp(z2)
        probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

        ...
```

```
        ...

        # Backpropagation
        delta3 = probs
        delta3[range(num_examples), y] -= 1
        dW2 = (a1.T).dot(delta3)
        db2 = np.sum(delta3, axis=0, keepdims=True)
        delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))
        dW1 = np.dot(X.T, delta2)
        db1 = np.sum(delta2, axis=0)

        # Add regularization terms (b1 and b2 don't have regularization terms)
        dW2 += reg_lambda * W2
        dW1 += reg_lambda * W1

        # Gradient descent parameter update
        W1 += -epsilon * dW1
        b1 += -epsilon * db1
        W2 += -epsilon * dW2
        b2 += -epsilon * db2

        # Assign new parameters to the model
        model = { 'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}

        # Optionally print the loss.
        # This is expensive because it uses the whole dataset, so we don't want to
        if print_loss and i % 1000 == 0:
          print "Loss after iteration %i: %f" %(i, calculate_loss(model))

    return model
```



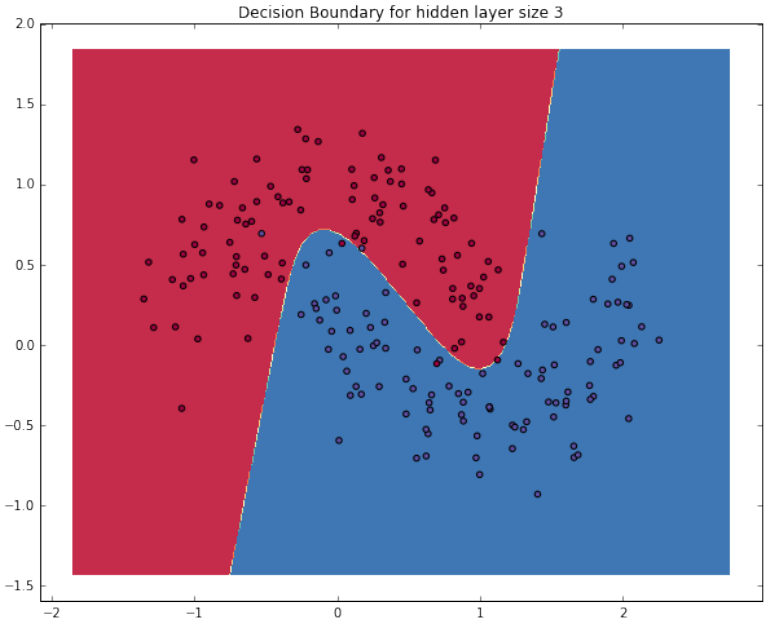
```
# Build a model with a 3-dimensional hidden layer
model = build_model(3, print_loss=True)

Loss after iteration 0: 0.432387
Loss after iteration 1000: 0.068947
Loss after iteration 2000: 0.068936
Loss after iteration 3000: 0.071218
Loss after iteration 4000: 0.071253
Loss after iteration 5000: 0.071278
Loss after iteration 6000: 0.071293
Loss after iteration 7000: 0.071303
Loss after iteration 8000: 0.071308
Loss after iteration 9000: 0.071312
Loss after iteration 10000: 0.071314
Loss after iteration 11000: 0.071315
Loss after iteration 12000: 0.071315
Loss after iteration 13000: 0.071316
Loss after iteration 14000: 0.071316
Loss after iteration 15000: 0.071316
Loss after iteration 16000: 0.071316
Loss after iteration 17000: 0.071316
Loss after iteration 18000: 0.071316
Loss after iteration 19000: 0.071316
```

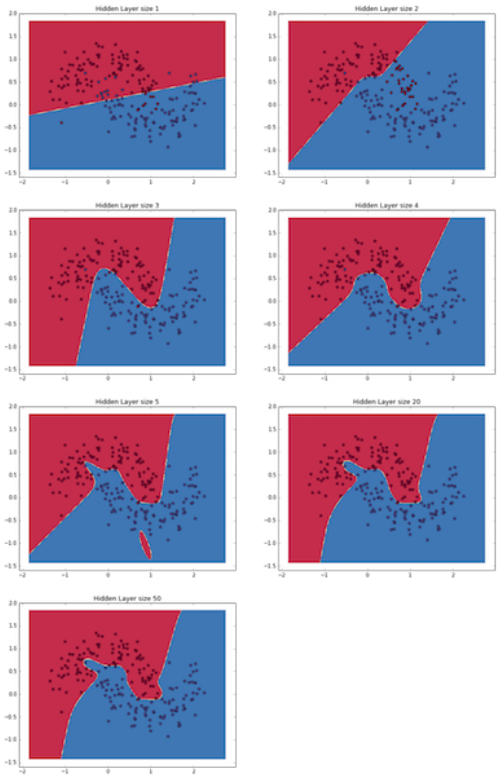
```
# Build a model with a 3-dimensional hidden layer
model = build_model(3, print_loss=True)

Loss after iteration 0: 0.432387
Loss after iteration 1000: 0.068947
Loss after iteration 2000: 0.068936
Loss after iteration 3000: 0.071218
Loss after iteration 4000: 0.071253
Loss after iteration 5000: 0.071278
Loss after iteration 6000: 0.071293
Loss after iteration 7000: 0.071303
Loss after iteration 8000: 0.071308
Loss after iteration 9000: 0.071312
Loss after iteration 10000: 0.071314
Loss after iteration 11000: 0.071315
Loss after iteration 12000: 0.071315
Loss after iteration 13000: 0.071316
Loss after iteration 14000: 0.071316
Loss after iteration 15000: 0.071316
Loss after iteration 16000: 0.071316
Loss after iteration 17000: 0.071316
Loss after iteration 18000: 0.071316
Loss after iteration 19000: 0.071316
```

```
# Plot the decision boundary
plot_decision_boundary(lambda x: predict(model, x))
plt.title("Decision Boundary for hidden layer size 3")
```

```
plt.figure(figsize=(16, 32))
hidden_layer_dimensions = [1, 2, 3, 4, 5, 20, 50]
for i, nn_hdim in enumerate(hidden_layer_dimensions):
    plt.subplot(5, 2, i+1)
    plt.title('Hidden Layer size %d' % nn_hdim)
    model = build_model(nn_hdim)
    plot_decision_boundary(lambda x: predict(model, x))
plt.show()
```


Questions
brian@bosatsu.net
@bsletten
bsletten