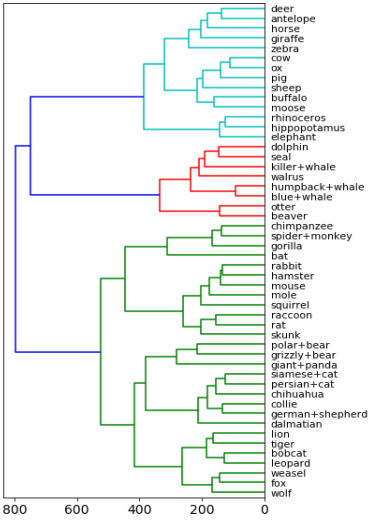
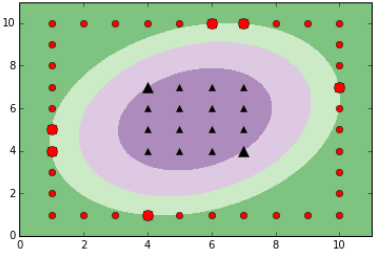
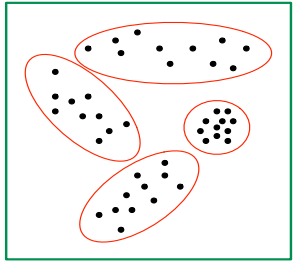
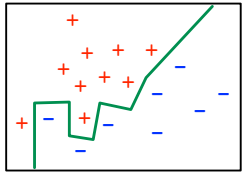
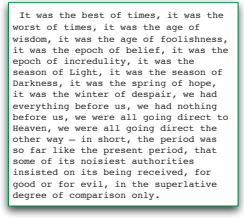
Fundamentals of Machine Learning

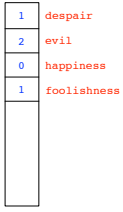


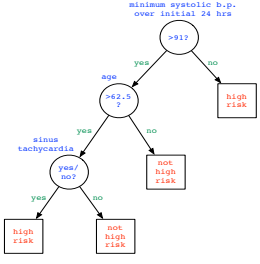


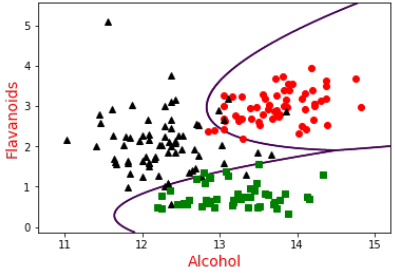








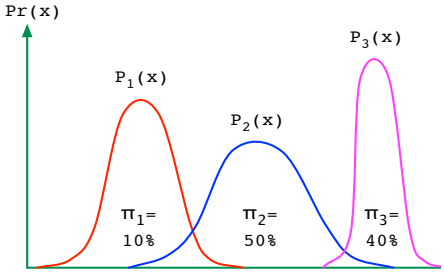


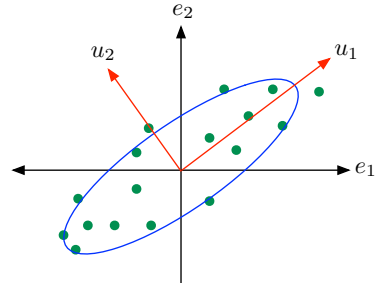


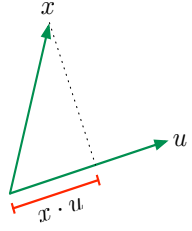
Nearest neighbor
Generative models
Least-squares regression
Ridge regression, Lasso
Logistic regression
Support vector machines

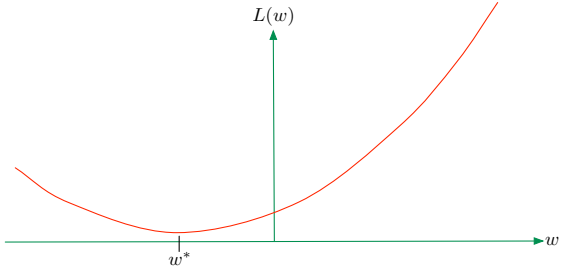
Kernel methods
Decision trees
Boosting and bagging
Random forests
k-means
Mixtures of Gaussians

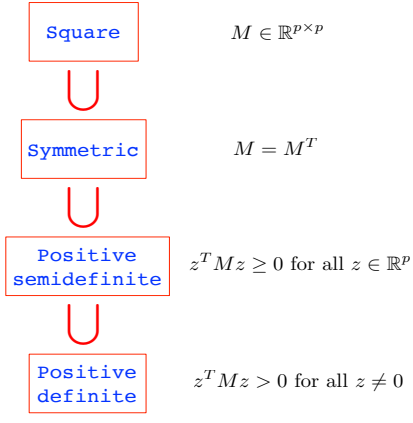
Hierarchical clustering
Principal component analysis
Singular value decomposition
Autoencoders
Deep learning











Probability

Probability spaces
Bayes' rule
Random variables
Mean and variance
Measuring dependence

Linear algebra

Matrices and vectors
Projections
Positive definiteness
Eigendecompositions
Spectral decomposition

Optimization

Gradient descent
Stochastic gradient descent
Convex optimization
Duality

Skills you will acquire

1. Familiarity with most widely-used ML methods
   - How they work
   - The kinds of data they are suited to
   - Their strengths and weaknesses
2. Adapting existing methods to a particular application
3. The foundational knowledge to keep pace with a fast-moving field

jupyter **kernel-perceptron** Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted

Code Validate

In [5]:
```
# RBF kernel function
def rbfk(x,z,sigma):
    tmp = np.linalg.norm(x-z)/sigma
    return np.exp(-tmp*tmp)
# Evaluate a classifier (alpha,x,y) on a new point x0
def evaluate(alpha,x,y,x0):
    value = 0.0
    for i in range(len(alpha)):
        if alpha[i] > 0:
            value = value + alpha[i] * y[i] * rbfk(x[i,:],x0,1.0)
    return np.sign(value)
# Now train an RBF kernel perceptron
working = True
alpha = np.zeros(n)
np.random.seed(None)
while working:
    working = False
    I = np.random.permutation(n)
    for i in range(n):
        j = I[i]
        if (evaluate(alpha,x,y,x[j,:]) != y[j]):
            working = True
            alpha[j] = alpha[j] + 1
```

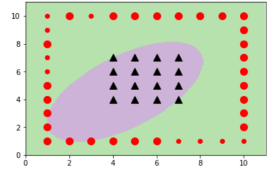
In [3]:
```
delta = 0.05
xx, yy = np.meshgrid(np.arange(xmin, xmax, delta), np.arange(ymin, ymax, delta))
grid = np.c_[xx.ravel(), yy.ravel()]
gn, gp = grid.shape
z = np.zeros(gn)
for i in range(gn):
    z[i] = evaluate(alpha,x,y,grid[i,:])

# Put the result into a color plot
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z, cmap=plt.cm.PRGn, vmin=-3, vmax=3)

# Identify the support vectors
sv = (alpha > 0)
notsv = np.logical_not(sv)

# Plot also the training points
plt.plot(x[(y==1)*notsv,0], x[(y==1)*notsv,1], 'ro')
plt.plot(x[(y==1)*sv,0], x[(y==1)*sv,1], 'ro', markersize=10)
plt.plot(x[(y==-1)*notsv,0], x[(y==-1)*notsv,1], 'k^')
plt.plot(x[(y==-1)*sv,0], x[(y==-1)*sv,1], 'k^', markersize=10)

plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
plt.show()
```



jupyter **cluster-awa** Last Checkpoint: 32 minutes ago (autosaved)

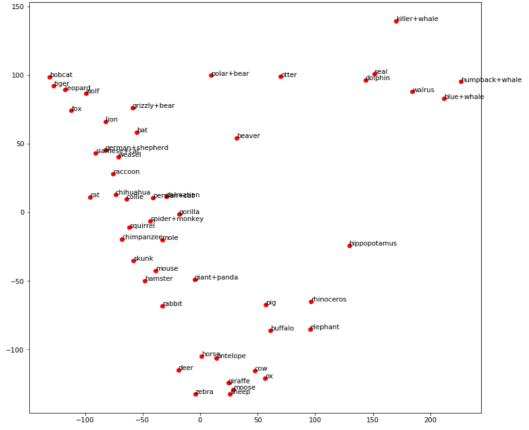
File Edit View Insert Cell Kernel Navigate Widgets Help

Code Validate

```
['ox', 'buffalo', 'pig', 'cow']
['skunk', 'rat', 'weasel', 'raccoon']
['killer+whale', 'blue+whale', 'humpback+whale', 'seal', 'otter', 'walrus', 'dolphin']
['spider+monkey', 'gorilla', 'chimpanzee']
['dalmatian', 'persian+cat', 'german+shepherd', 'collie']
['siamese+cat', 'chihuahua']
```

In [4]:
```
# Now visualize using PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
z = pca.fit_transform(x)
```

In [6]:
```
import matplotlib.pyplot as plt
from pylab import rcParams
rcParams['figure.figsize'] = 12, 12
plt.plot(z[:,0], z[:,1], 'ro')
for i in range(50):
    plt.annotate(names[i], xy=(z[i,0],z[i,1]))
plt.show()
```



Nearest neighbor classification

Topics we'll cover

1. What is a classification problem?
2. The training set and test set
3. Representing data as vectors
4. Distance in Euclidean space
5. The 1-NN classifier
6. Training error versus test error
7. The error of a random classifier

The problem we'll solve today

Given an image of a handwritten digit, say which digit it is.

 $\Longrightarrow$ 3

Some more examples:

The machine learning approach

Assemble a data set:

The MNIST data set of handwritten digits:

- **Training set** of 60,000 images and their labels.
- **Test set** of 10,000 images and their labels.

And let the machine figure out the underlying patterns.

Nearest neighbor classification

Training images $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(60000)}$
Labels $y^{(1)}, y^{(2)}, y^{(3)}, \ldots, y^{(60000)}$ are numbers in the range $0 - 9$

How to **classify** a new image x?

- Find its nearest neighbor amongst the $x^{(i)}$
- Return $y^{(i)}$

The data space

How to measure the distance between images?

MNIST images:

- Size 28×28 (total: 784 pixels)
- Each pixel is grayscale: 0-255

Stretch each image into a vector with 784 coordinates:

- Data space $\mathcal{X} = \mathbb{R}^{784}$
- Label space $\mathcal{Y} = \{0, 1, \ldots, 9\}$

The distance function

Remember Euclidean distance in two dimensions?

$z = (3, 5)$

$x = (1, 2)$

Euclidean distance in higher dimension

Euclidean distance between 784-dimensional vectors x, z is

$$\|x - z\| = \sqrt{\sum_{i=1}^{784} (x_i - z_i)^2}$$

Here x_i is the ith coordinate of x.

Nearest neighbor classification

Training images $x^{(1)}, \ldots, x^{(60000)}$, labels $y^{(1)}, \ldots, y^{(60000)}$

To classify a new image x:

- Find its nearest neighbor amongst the $x^{(i)}$ **using Euclidean distance in** $\mathbb{R}^{784}$
- Return $y^{(i)}$

How accurate is this classifier?

Accuracy of nearest neighbor on MNIST

Training set of 60,000 points.

- What is the error rate on training points? **Zero.**
  In general, **training error** is an overly optimistic predictor of future performance.
- A better gauge: separate test set of 10,000 points.
  Test error = fraction of test points incorrectly classified.
- What test error would we expect for a *random classifier*?
  (One that picks a label $0 - 9$ at random?) **90%.**
- Test error of nearest neighbor: **3.09%.**

Examples of errors

Test set of 10,000 points:

- 309 are misclassified
- Error rate 3.09%

Examples of errors:

Query 

NN 

Improving the performance of nearest neighbor

Recall: nearest neighbor on MNIST

- Images of handwritten digits, represented as vectors in $\mathbb{R}^{784}$.
- Labels $0 - 9$
- Training set: 60,000 points; test set: 10,000 points

Test error of nearest neighbor using Euclidean distance: 3.09%

Examples of errors:

Query 

NN 

Ideas for improvement: (1) k-NN (2) better distance function.

K-nearest neighbor classification

To classify a new point:

- Find the k nearest neighbors in the training set.
- Return the most common label amongst them.

MNIST:

k	1	3	5	7	9	11
Test error (%)	3.09	2.94	3.13	3.10	3.43	3.34

In real life, there's no test set. How to decide which k is best?

Cross-validation

How to estimate the error of k-NN for a particular k?

10-**fold cross-validation**

- Divide the training set into 10 equal pieces.
  Training set (call it S): 60,000 points
  Call the pieces $S_1, S_2, \ldots, S_{10}$: 6,000 points each.
- For each piece S_i:
  - Classify each point in S_i using k-NN with training set $S - S_i$
  - Let $\epsilon_i =$ fraction of S_i that is incorrectly classified
- Take the average of these 10 numbers:

$$\text{estimated error with } k\text{-NN} \;=\; \frac{\epsilon_1 + \cdots + \epsilon_{10}}{10}$$

Another improvement: better distance functions

The Euclidean (ℓ_2) distance between these two images is very high!

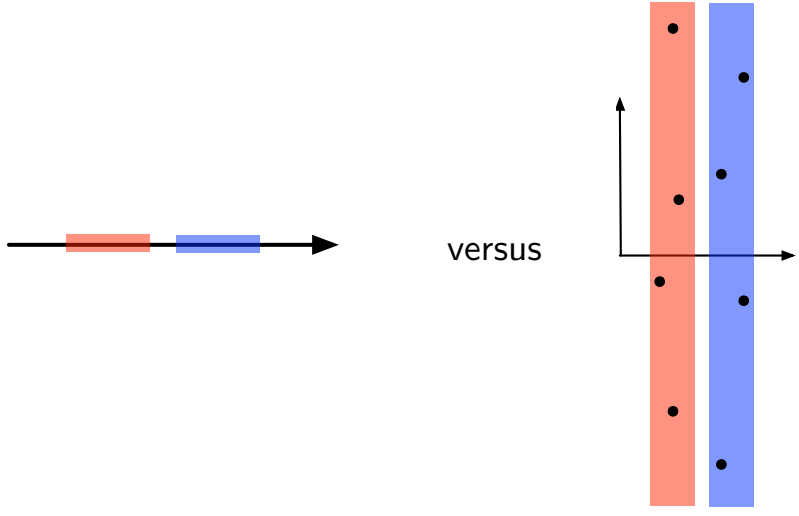
Much better idea: distance measures that are invariant under:

- Small translations and rotations. e.g. *tangent distance*.
- A broader family of natural deformations. e.g. *shape context*.

Test error rates:	ℓ_2	tangent distance	shape context
	3.09	1.10	0.63

Related problem: feature selection

Feature selection/reweighting is part of picking a distance function.
And, one noisy feature can wreak havoc with nearest neighbor!



Algorithmic issue: speeding up NN search

Naive search takes time $O(n)$ for training set of size n: slow!

Luckily there are data structures for speeding up nearest neighbor search, like:

1. Locality sensitive hashing
2. Ball trees
3. K-d trees

These are part of standard Python libraries for NN, and help a lot.

Useful distance functions for machine learning

Topics we'll cover

1. L_p norms
2. Metric spaces

Measuring distance in $\mathbb{R}^m$

Usual choice: **Euclidean distance**:

$$\|x - z\|_2 = \sqrt{\sum_{i=1}^{m} (x_i - z_i)^2}.$$

For $p \geq 1$, here is ℓ_p **distance**:

$$\|x - z\|_p = \left(\sum_{i=1}^{m} |x_i - z_i|^p \right)^{1/p}$$

- $p = 2$: Euclidean distance
- ℓ_1 distance: $\|x - z\|_1 = \sum_{i=1}^{m} |x_i - z_i|$
- ℓ_∞ distance: $\|x - z\|_\infty = \max_i |x_i - z_i|$

Example 1

Consider the all-ones vector $(1, 1, \ldots, 1)$ in $\mathbb{R}^d$.
What are its ℓ_2, ℓ_1, and ℓ_∞ length?

Example 2

In $\mathbb{R}^2$, draw all points with:

1. ℓ_2 length 1
2. ℓ_1 length 1
3. ℓ_∞ length 1

Metric spaces

Let $\mathcal{X}$ be the space in which data lie.

A distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a **metric** if it satisfies these properties:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Example 1

$\mathcal{X} = \mathbb{R}^m$ and $d(x, y) = \|x - y\|_p$

Check:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Example 2

$\mathcal{X} = \{\text{strings over some alphabet}\}$ and $d = \text{edit distance}$

Check:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

A non-metric distance function

Let p, q be probability distributions on some set $\mathcal{X}$.

The **Kullback-Leibler divergence** or **relative entropy** between p, q is:

$$d(p, q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}.$$

A host of prediction problems

Topics we'll cover

1. Machine learning versus algorithms
2. A taxonomy of prediction problems
3. Roadmap for the course

Machine learning versus Algorithms

A central goal of both fields:

develop procedures that exhibit a desired input-output behavior.

- **Algorithms**: input-output mapping can be precisely defined.
  Input: Graph G, two nodes u, v in the graph.
  Output: Shortest path from u to v in G

- **Machine learning**: mapping cannot easily be made precise.
  Input: Picture of an animal.
  Output: Name of the animal.

Instead, provide examples of (input,output) pairs.
Ask the machine to *learn* a suitable mapping itself.

Prediction problems: inputs and outputs

Basic terminology:

- The input space, $\mathcal{X}$.
  E.g. 32×32 RGB images of animals.
- The output space, $\mathcal{Y}$.
  E.g. Names of 100 animals.

x:

y: "bear"

After seeing a bunch of examples (x, y), pick a mapping

$$f : \mathcal{X} \to \mathcal{Y}$$

that accurately recovers the input-output pattern of the examples.

Categorize prediction problems by the type of **output space**:
(1) discrete, (2) continuous, or (3) probability values

Discrete output space: classification

Binary classification

E.g., Spam detection
$\mathcal{X} = \{\text{email messages}\}$
$\mathcal{Y} = \{\text{spam}, \text{not spam}\}$

Multiclass

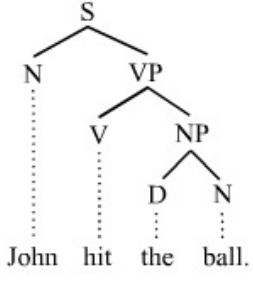
E.g., News article classification
$\mathcal{X} = \{\text{news articles}\}$
$\mathcal{Y} = \{\text{politics}, \text{business}, \text{sports}, \ldots\}$

Structured outputs

E.g., Parsing
$\mathcal{X} = \{\text{sentences}\}$
$\mathcal{Y} = \{\text{parse trees}\}$

$x =$ "John hit the ball"

$y =$



Continuous output space: regression

- Pollution level prediction
  Predict tomorrow's air quality index in my neighborhood
  $\mathcal{Y} = [0, \infty)$ (< 100: okay, > 200: dangerous)

- Insurance company calculations
  What is the expected life expectancy of this person?
  $\mathcal{Y} = [0, 120]$

What are suitable predictor variables ($\mathcal{X}$) in each case?

Probability estimation

$\mathcal{Y} = [0, 1]$ represents **probabilities**

Example: Credit card transactions

- $x =$ details of a transaction
- $y =$ probability this transaction is fraudulent

Why not just treat this as a binary classification problem?

Roadmap for the course

1. Solving prediction problems
   Classification, regression, probability estimation
2. Representation learning
   Clustering, projection, dictionary learning, autoencoders
3. Deep learning