

INTRODUCTION TO DATA SCIENCE

JOHN P DICKERSON

Lecture #18 – 4/4/2017

CMSC320
Tuesdays & Thursdays
3:30pm – 4:45pm



COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

Mini-Project #3 will be out tonight!

- It is now due **April 18th**.
- ELMS link:
<https://myelms.umd.edu/courses/1218364/assignments/4389209>

Both Mini-Project #1 and #2 will be graded before your midterm.

- Finally got some ELMS stuff sorted out for the TAs ...
- Please fill in the Google Doc labeling your partners:
<https://docs.google.com/spreadsheets/d/1tODyzj29lr8TmrJhJ9hngA72YYewnp3mqWB8yI0R4c/edit#gid=0>



MIDTERM

In-class midterm next Thursday (4/13).

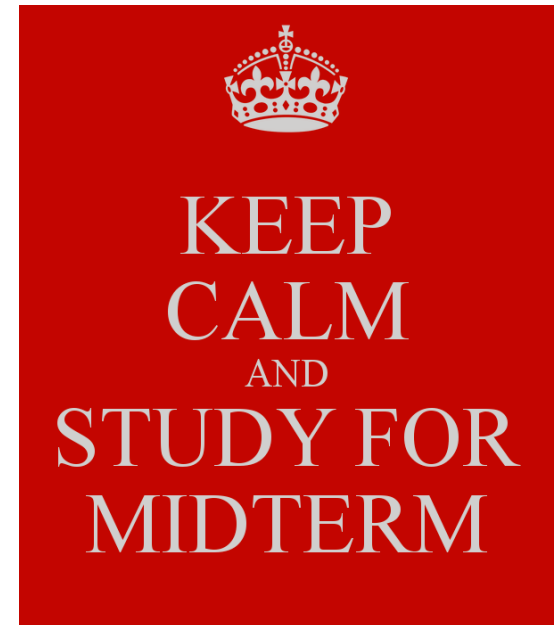
- People are starting to ask about this ...

Questions & clarifications:

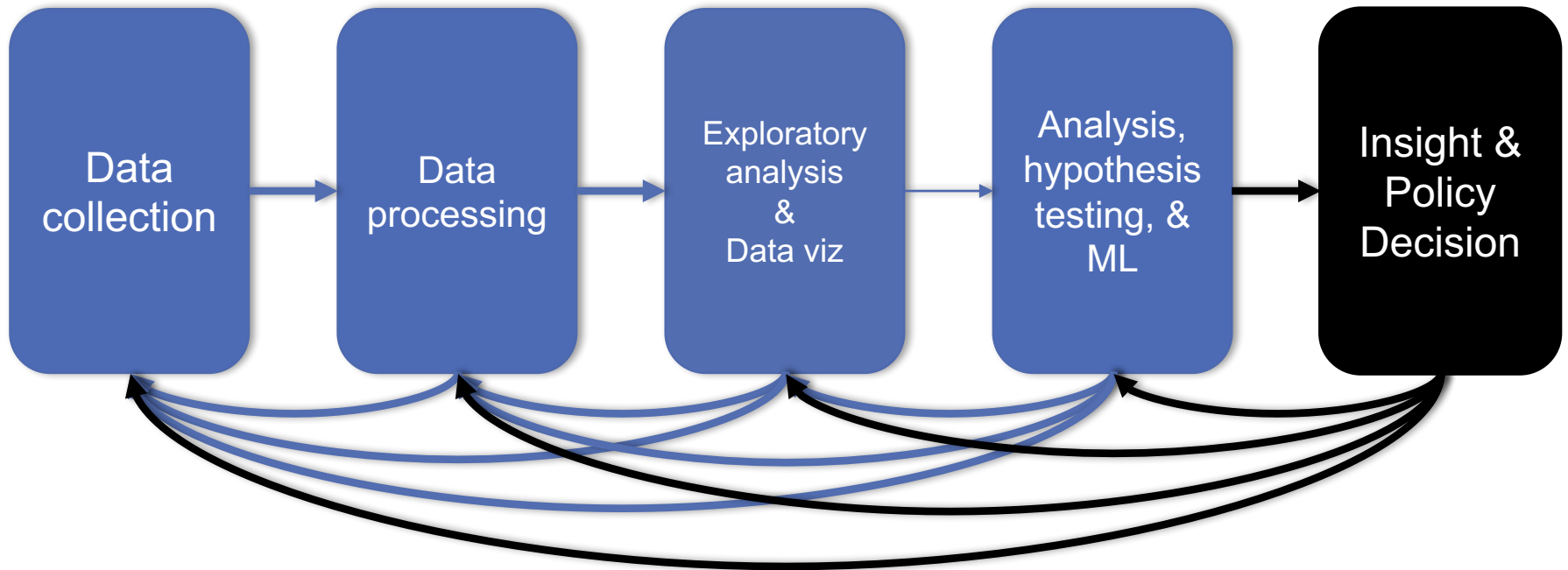
- Any material covered **after** this Thursday's class (4/6) will **not** be on the midterm.
- No programming questions.
- Some qualitative questions.
- Some “applied” quantitative questions – that is, no proofs.

Good ways to study:

- Lecture slides; trying stuff out in iPython; external links on the course website; asking TAs about gaps in knowledge



TODAY'S LECTURE

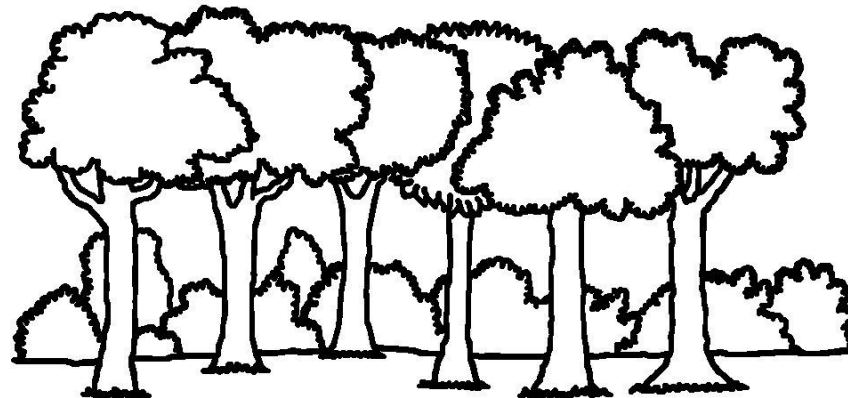


TODAY'S LECTURE

More nonlinear classification/regression methods

- Decision trees & random forests in Scikit-Learn
- K-Nearest Neighbors (KNN)
- Support Vector Machines (SVMs)

Thanks to: **Hector Corrada Bravo (UMD), Panagiotis Tsaparas (U of I), Oliver Schulte (SFU)**



DECISION TREES IN SCIKIT

```
from sklearn.datasets import load_iris
from sklearn import tree

# Load a common dataset, fit a decision tree to it
iris = load_iris()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)
```

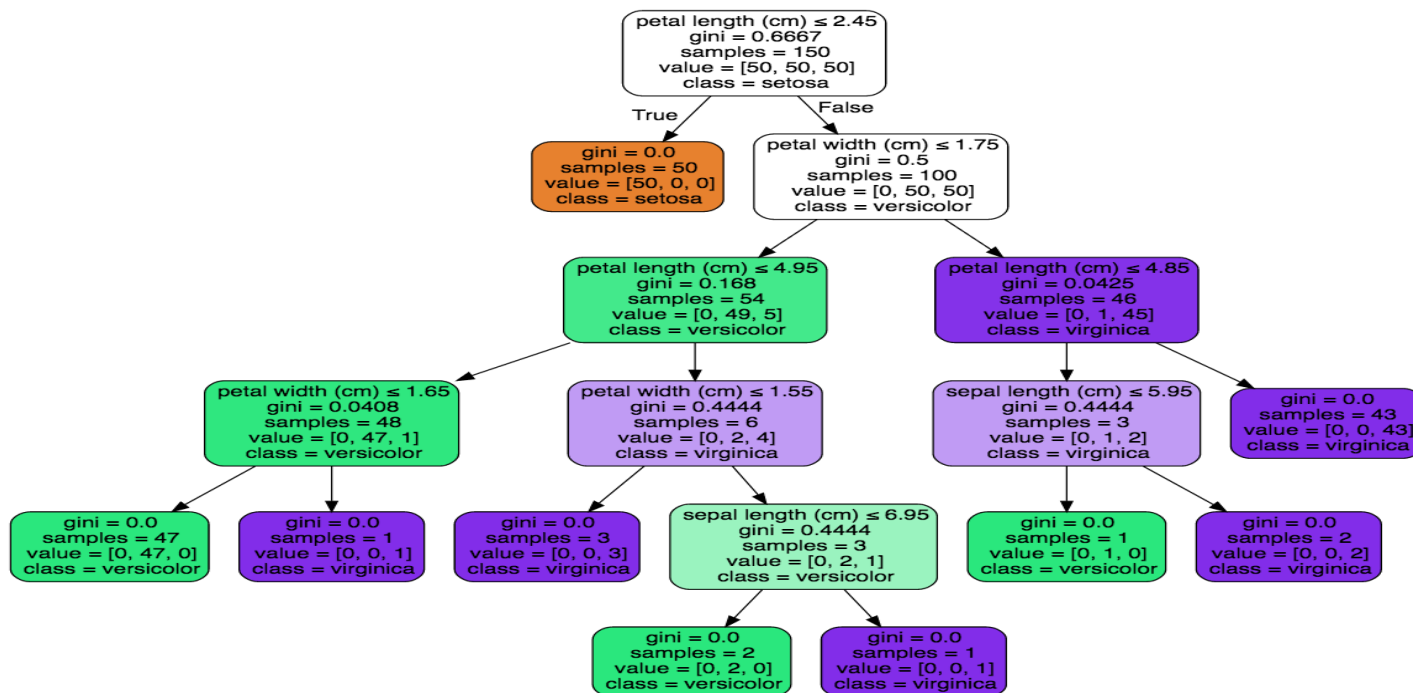
Trains a decision tree using default parameters (attribute chosen to split on either Gini or entropy, no max depth, etc)

```
# Predict most likely class
clf.predict([[2., 2.]])
```

```
# Predict PDF over classes (%training samples in leaf)
clf.predict_proba([[2., 2.]])
```

VISUALIZING A DECISION TREE

```
from IPython.display import Image
dot_data = tree.export_graphviz(clf,
                                out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```



RANDOM FORESTS

Decision trees are very interpretable, but may be brittle to changes in the training data, as well as noise

Random forests are an ensemble method that:

- Resamples the training data;
- Builds many decision trees; and
- Averages predictions of trees to classify.

This is done through bagging and random feature selection



BAGGING

Bagging: **B**ootstrap **a**ggregation

Resampling a training set of size n via the **bootstrap**:

- Sample **with replacement** n elements

General scheme for random forests:

1. Create B bootstrap samples, $\{Z_1, Z_2, \dots, Z_B\}$
2. Build B decision trees, $\{T_1, T_2, \dots, T_B\}$, from $\{Z_1, Z_2, \dots, Z_B\}$

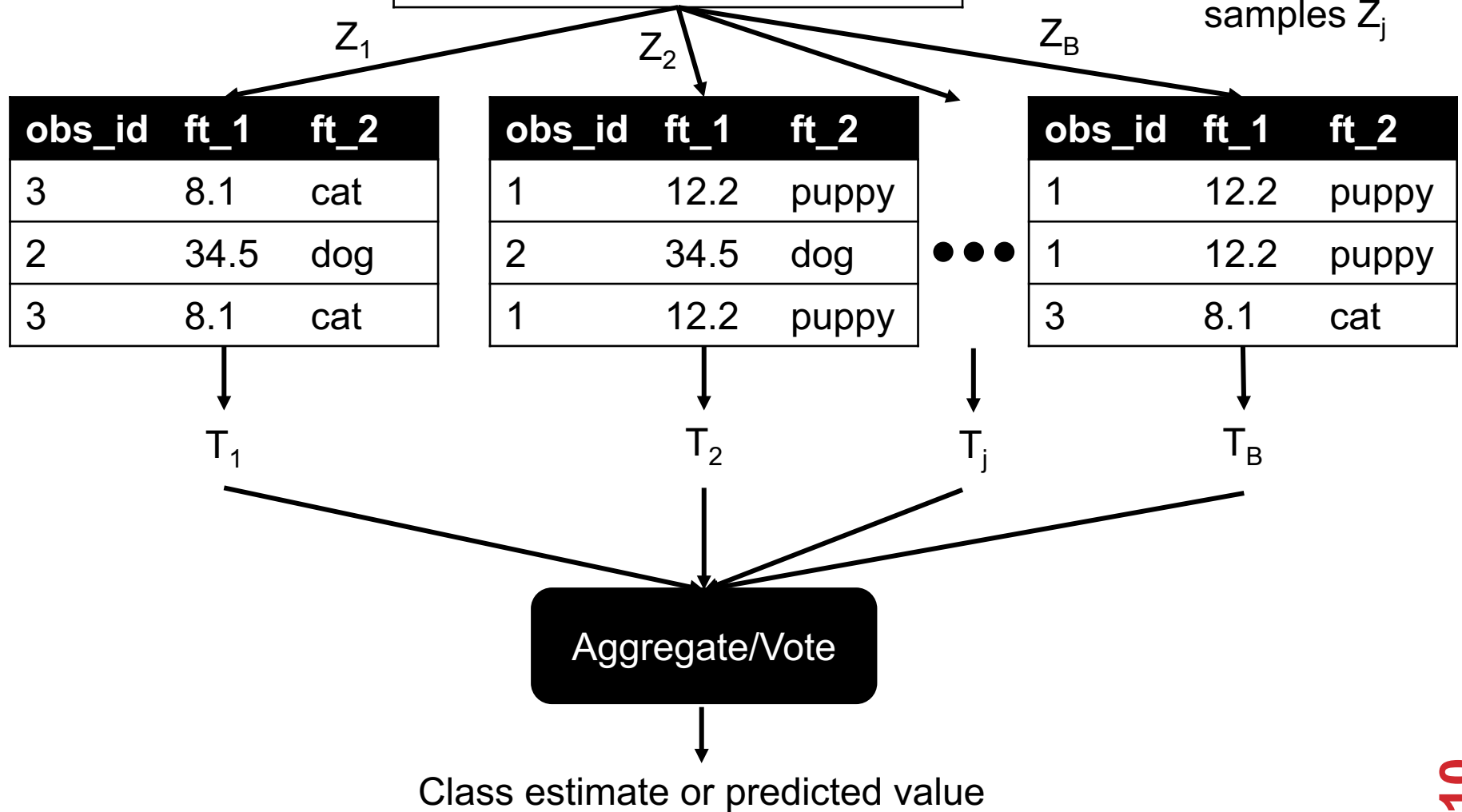
Classification/Regression:

1. Each tree T_j predicts class/value y_j
2. Return average $1/B \sum_{j=\{1, \dots, B\}} y_j$ for regression,
or majority vote for classification

Original training dataset (Z):

obs_id	ft_1	ft_2
1	12.2	puppy
2	34.5	dog
3	8.1	cat

B Bootstrap samples Z_j



RANDOM ATTRIBUTE SELECTION

We get some randomness via bootstrapping

- We like this! Randomness increases the bias of the forest slightly at a huge decrease in variance (due to averaging)

We can further reduce correlation between trees by:

1. For each tree, at every split point ...
2. ... choose a **random subset** of attributes ...
3. ... then split on the “best” (entropy, Gini) within only that subset

RANDOM FORESTS IN SCIKIT-LEARN

```
from sklearn.ensemble import RandomForestClassifier

# Train a random forest of 10 default decision trees
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = RandomForestClassifier(n_estimators=10)
clf = clf.fit(X, Y)
```

Can we get even more random?!

Extremely randomized trees (`ExtraTreesClassifier`)
do bagging, random attribute selection, but also:

1. At each split point, choose random splits
2. Pick the best of those random splits

Similar bias/variance performance to RFs, but can
be faster computationally

1.21 
GIGAWATTS



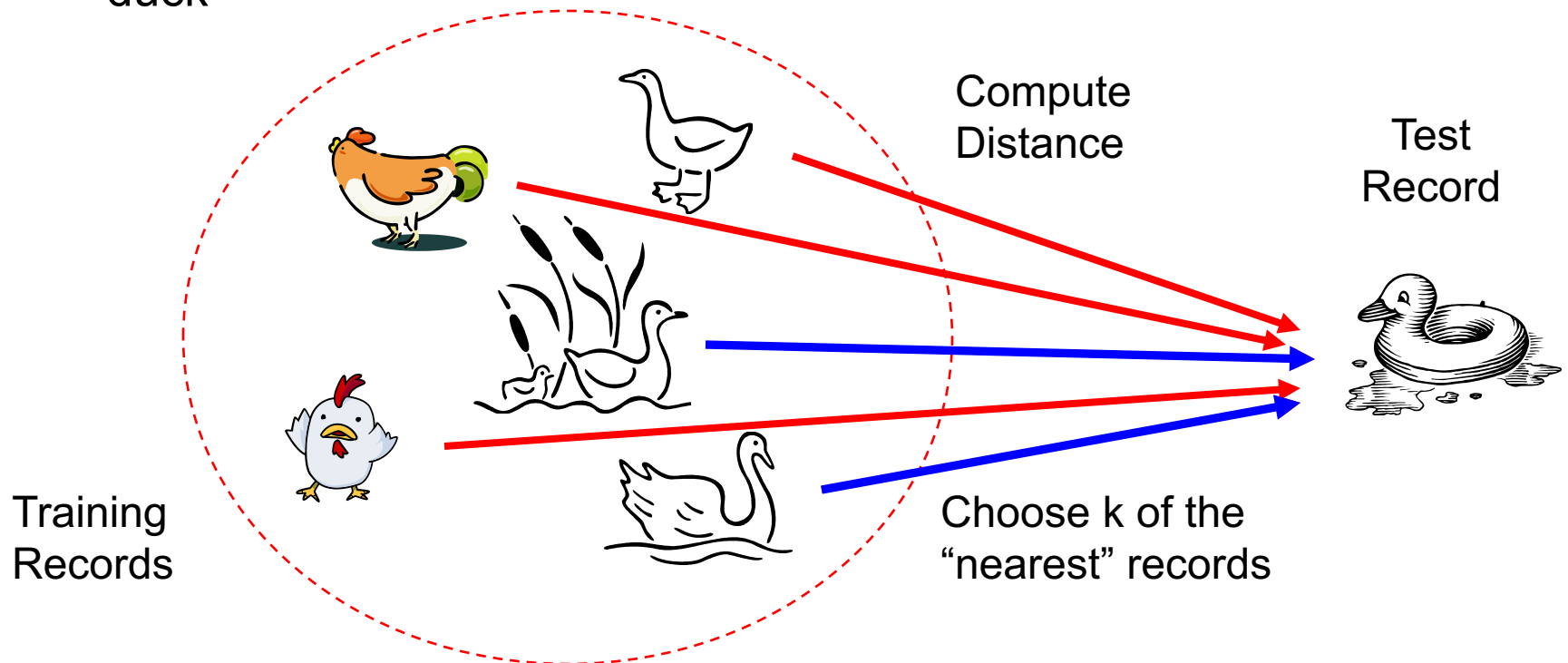


K-NEAREST NEIGHBORS

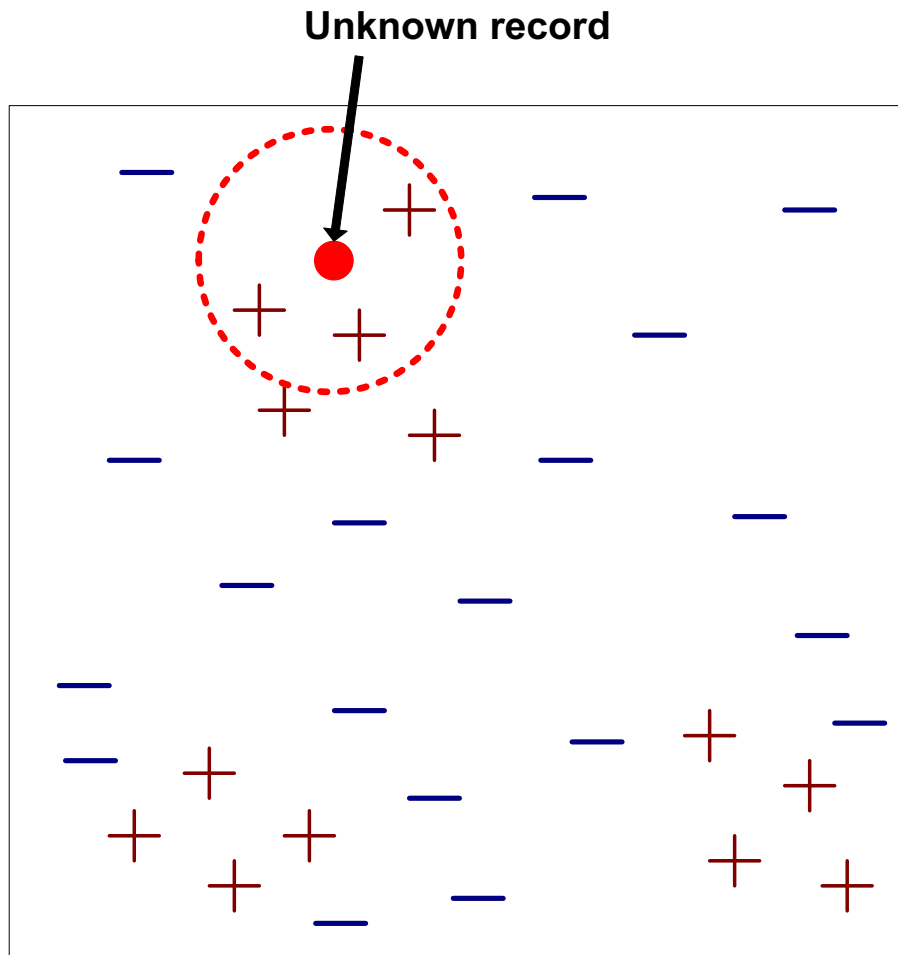
NEAREST NEIGHBOR CLASSIFIERS

Basic idea:

- If it walks like a duck, quacks like a duck, then it's probably a duck

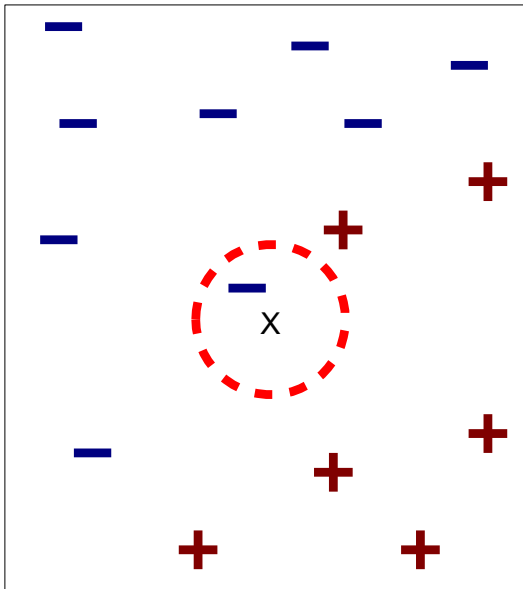


NEAREST-NEIGHBOR CLASSIFIERS

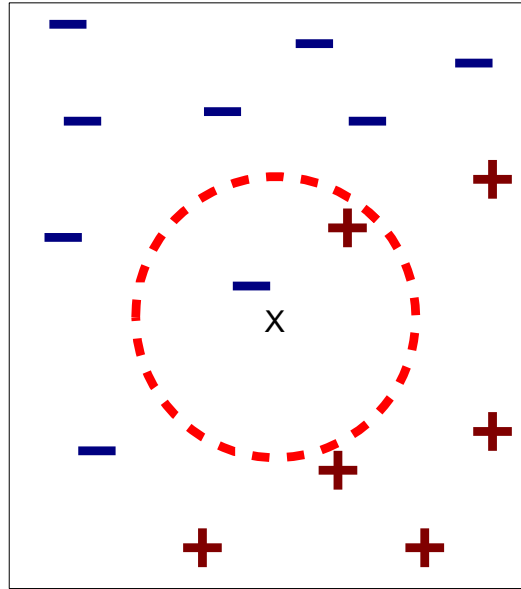


- Requires three things
 - The set of stored records
 - **Distance Metric** to compute distance between records
 - The value of k , the **number of nearest neighbors** to retrieve
- To classify an unknown record:
 - **Compute distance** to other training records
 - Identify k nearest neighbors
 - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

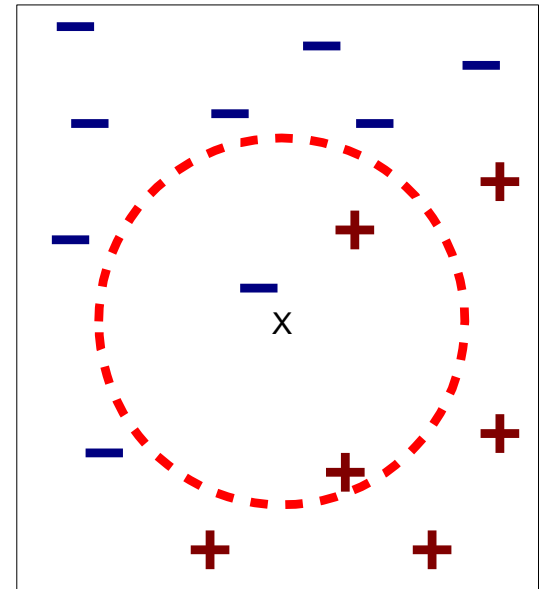
DEFINITION OF NEAREST NEIGHBOR



(a) 1-nearest neighbor



(b) 2-nearest neighbor

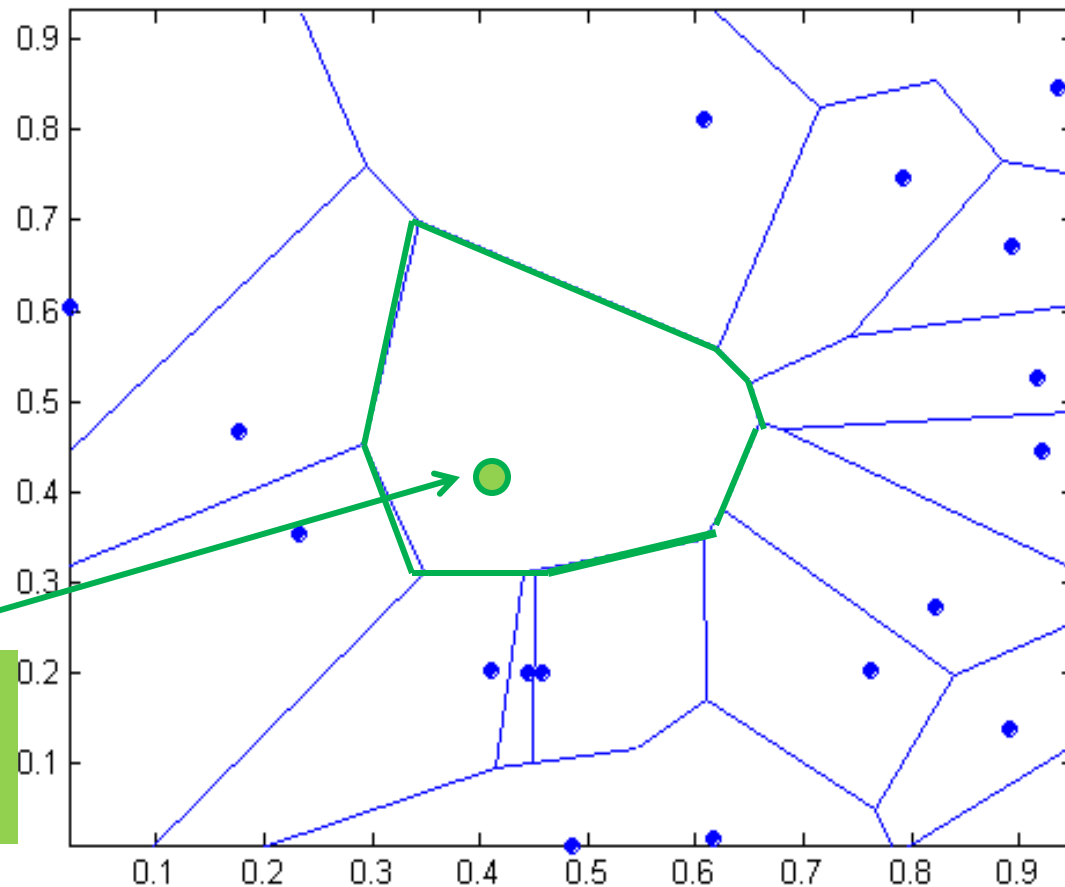


(c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distances to x

1-NEAREST NEIGHBOR

Voronoi Diagram defines the classification boundary



The area takes the class of the green point

NEAREST NEIGHBOR CLASSIFICATION

Compute distance between two points:

- Euclidean distance

$$d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$$

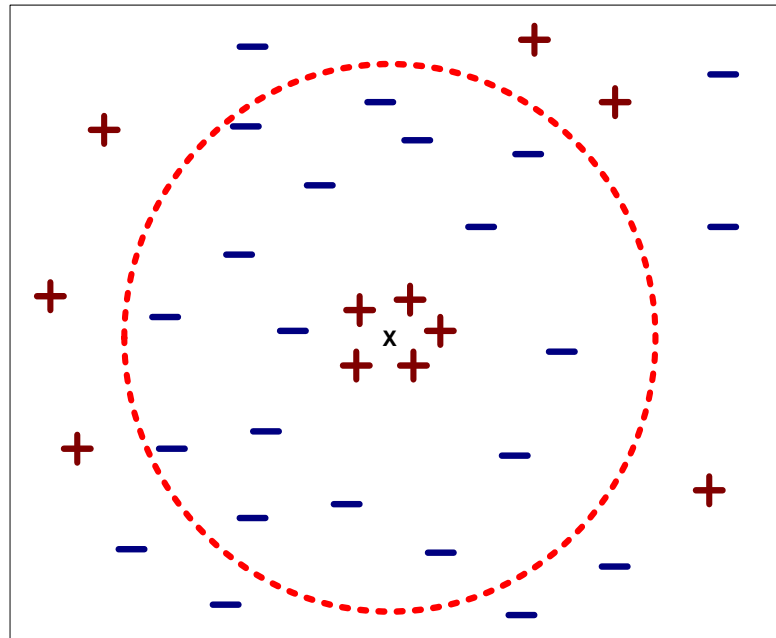
Determine the class from nearest neighbor list

- Take the majority vote of class labels among the k-nearest neighbors
- Weight the vote according to distance
 - E.g., weight factor $w = 1/d^2$

NEAREST NEIGHBOR CLASSIFICATION...

Choosing the value of k :

- If k is too small, sensitive to noise points
- If k is too large, neighborhood may include points from other classes



NEAREST NEIGHBOR CLASSIFICATION...

Scaling issues

- **Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes**
- **Example:**
 - height of a person may vary from 1.5m to 1.8m
 - weight of a person may vary from 90lb to 300lb
 - income of a person may vary from \$10K to \$1M

Standardize variables, like in Mini-Project #2.

NEAREST NEIGHBOR CLASSIFICATION...

Problem with Euclidean measure:

- High dimensional data
 - **The curse of dimensionality** – data becomes sparse relative to the total volume of the space, distance metrics “lose meaning”
- Can produce counter-intuitive results

1 1 1 1 1 1 1 1 1 1 1 0

VS

1 0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1 1 1 1 1

0 0 0 0 0 0 0 0 0 0 0 1

$d = 1.4142$

$d = 1.4142$

Solution: Normalize the vectors to unit length

NEAREST NEIGHBOR CLASSIFICATION...

k-NN classifiers are lazy learners

- It does not build models explicitly
- Unlike **eager learners** such as decision tree induction and rule-based systems

Classifying unknown records are relatively expensive

- Naïve algorithm: $O(n)$
- Need for structures to retrieve nearest neighbors fast
 - The **Nearest Neighbor Search** problem
- CMSC420 covers spatial data structures extensively

NEAREST NEIGHBOR SEARCH

Two-dimensional **kd-trees**:

- A data structure for answering nearest neighbor queries in \mathbb{R}^2

kd-tree construction algorithm

- Select the **x** or **y** dimension (alternating between the two)
- Partition the space into two with a line passing from the median point
- Repeat recursively in the two partitions as long as there are enough points
- Can quickly query the tree for nearest neighbors by finding an incumbent best and pruning large chunks of the tree away

K-NN: ADVANTAGES

Simple technique that is easily implemented

Building model is cheap

Extremely flexible classification scheme

Well suited for:

- Multi-modal classes
- Records with multiple class labels

Can sometimes be the best method

- Michihiro Kuramochi and George Karypis, Gene Classification using Expression Profiles: A Feasibility Study, International Journal on Artificial Intelligence Tools. Vol. 14, No. 4, pp. 641-660, 2005
- K nearest neighbor outperformed SVM for protein function prediction using expression profiles

K-NN: DISADVANTAGES

Classifying unknown records are relatively expensive

- Requires distance computation of k -nearest neighbors
- Computationally intensive, especially when the size of the training set grows

Accuracy can be severely degraded by the presence of:

- Noisy or irrelevant features
- High-dimensional space
- Choosing the wrong distance metric
- Choosing the wrong k

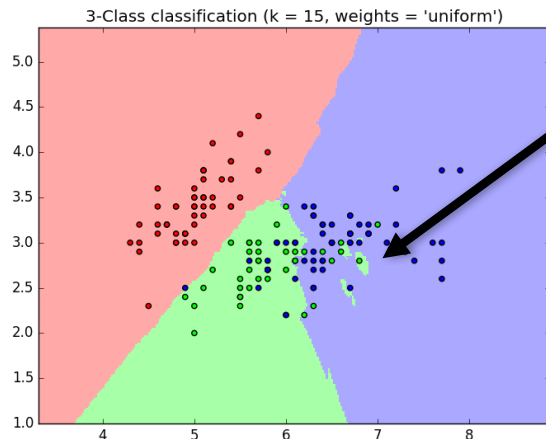
KNN CLASSIFICATION IN SCIKIT-LEARN

```
from sklearn import neighbors, datasets

# Load a common dataset, fit a 15-NN classifier to it
iris = datasets.load_iris()
X = iris.data[:, :2] # take the first two features
y = iris.target

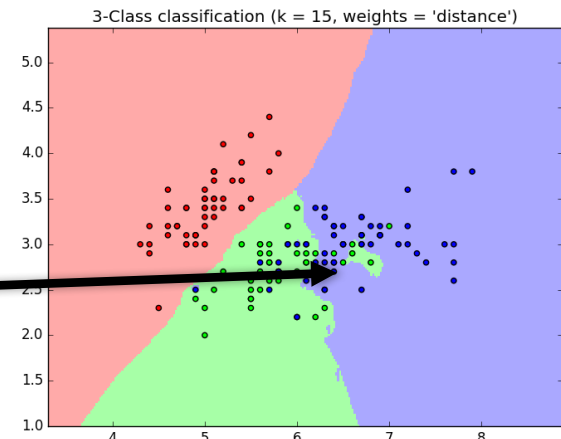
clf = neighbors.KNeighborsClassifier(
    15, weights='uniform')

clf.fit(X, y)
```



Uniform
weights

$1/d^2$
weights



LOCAL REGRESSION

Basic Idea: To predict a target value y for data point x , apply interpolation/regression to the neighborhood of x .

Simplest version: connect the dots.

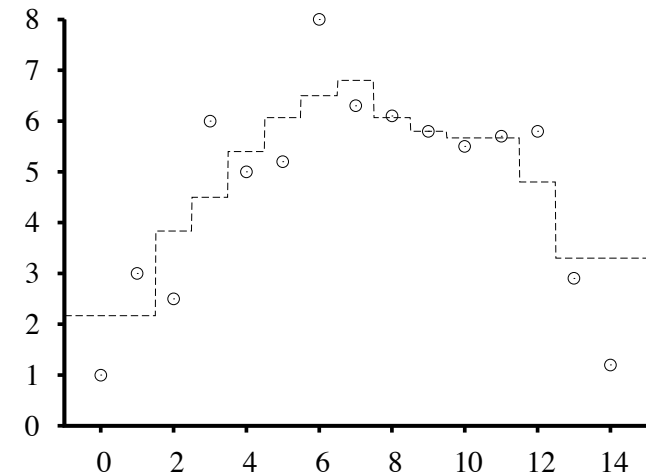
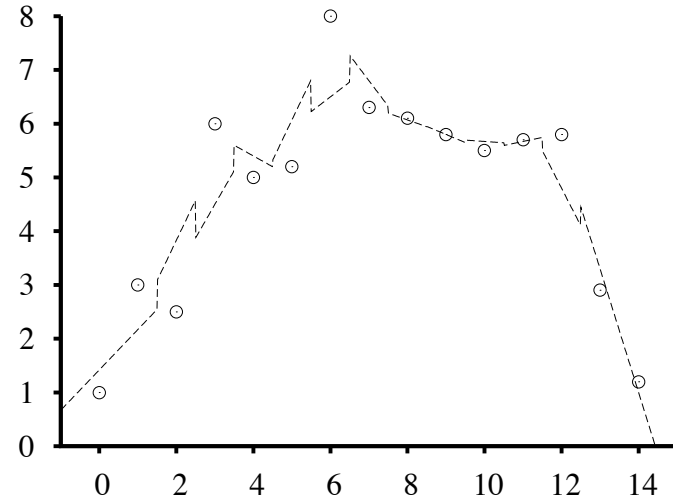


K-NEAREST NEIGHBOR REGRESSION

Connect the dots uses $k = 2$, fits a line.

Ideas for $k = 5$.

- Fit a line using linear regression.
- Predict the average target value of the k points.



LOCAL REGRESSION WITH KERNELS

Spikes in regression prediction come from in-or-out nature of neighborhood

Instead, **weight** examples as function of the distance

A **homogenous kernel function** maps the distance between two vectors to a number, usually in a nonlinear way.

$$k(x, x') = k(\text{distance}(x, x'))$$

Example: The quadratic kernel

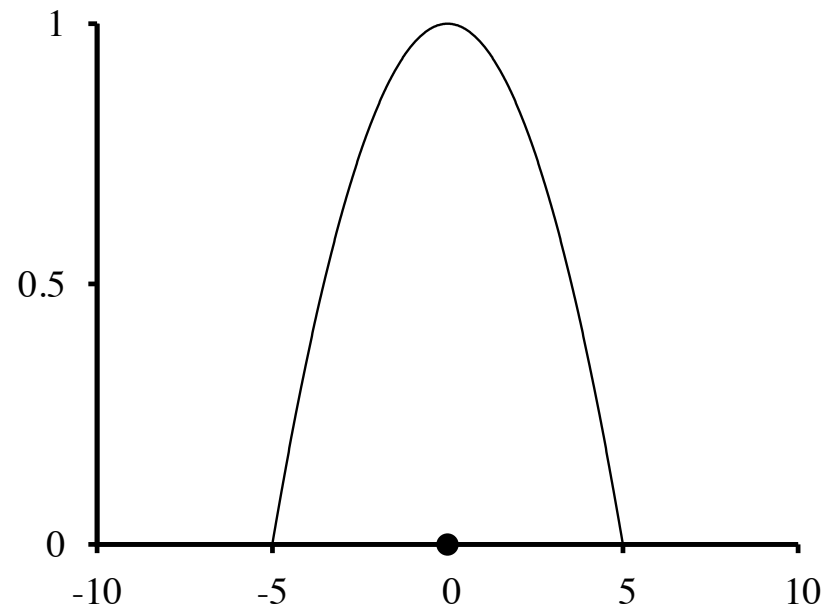
THE QUADRATIC KERNEL

$$k(d) = \max\left\{0, 1 - \left(\frac{2d}{k}\right)^2\right\}$$

$k = 5$

Let query point be $x = 0$.

Plot $k(0, x') = k(|x'|)$.



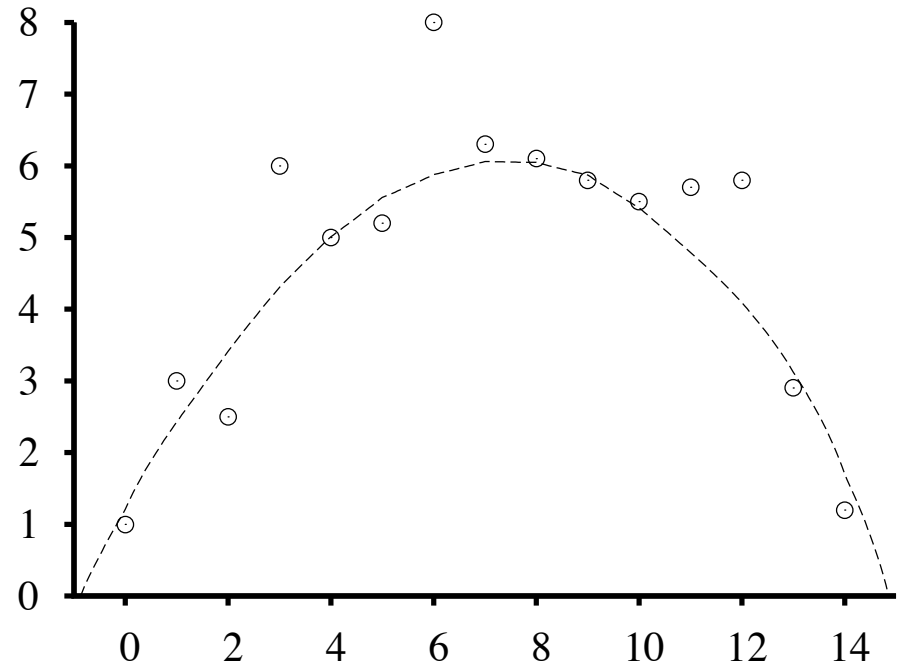
KERNEL REGRESSION

For each query point \mathbf{x}_q , prediction is made as weighted linear sum:

$$y(\mathbf{x}_q) = \mathbf{w} \cdot \mathbf{x}_q.$$

To find weights, solve the following regression on the k -nearest neighbors:

$$w^* = \operatorname{argmin}_w \sum_j k(\operatorname{dist}(\mathbf{x}_q, \mathbf{x}_j))(t_j - \mathbf{w} \cdot \mathbf{x}_j)^2$$



KNN REGRESSION IN SCIKIT-LEARN

```
from sklearn.neighbors import KNeighborsRegressor

# Basic KNN regression in Scikit (interpolation)
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]

neigh = KNeighborsRegressor( n_neighbors=2 )
neigh.fit(X, y)

print(neigh.predict([[1.5]]))
```

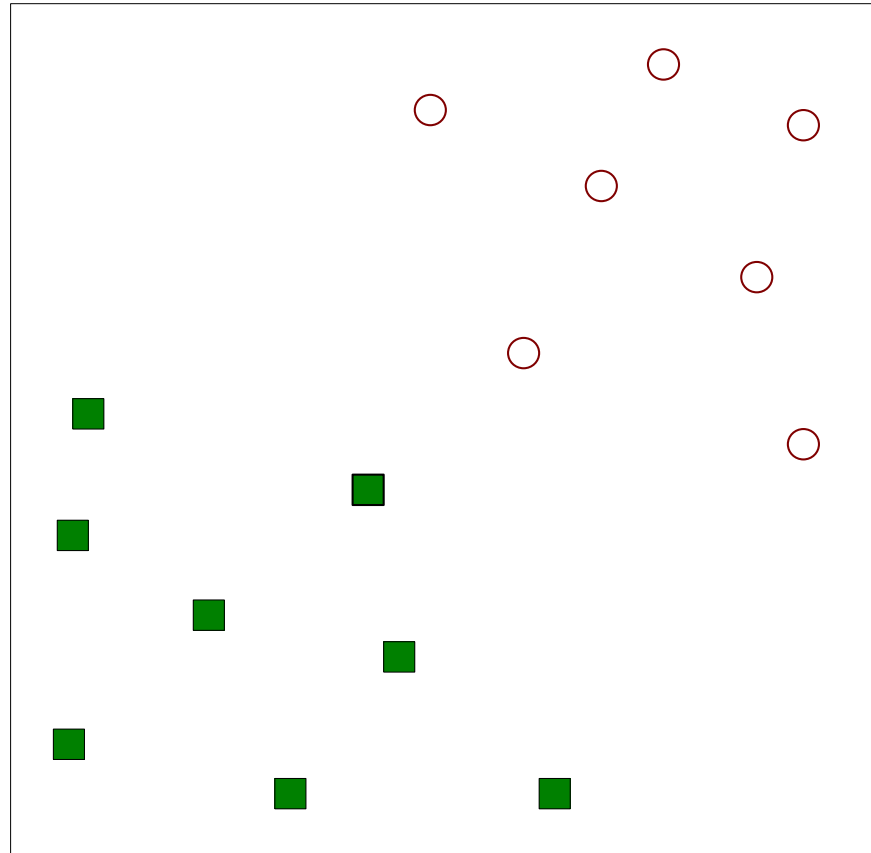
```
[ 0.5]
```

Also provides a variety of distance metrics, backing algorithms to find nearest neighbors, weight functions (down-weight points based on distance), etc.



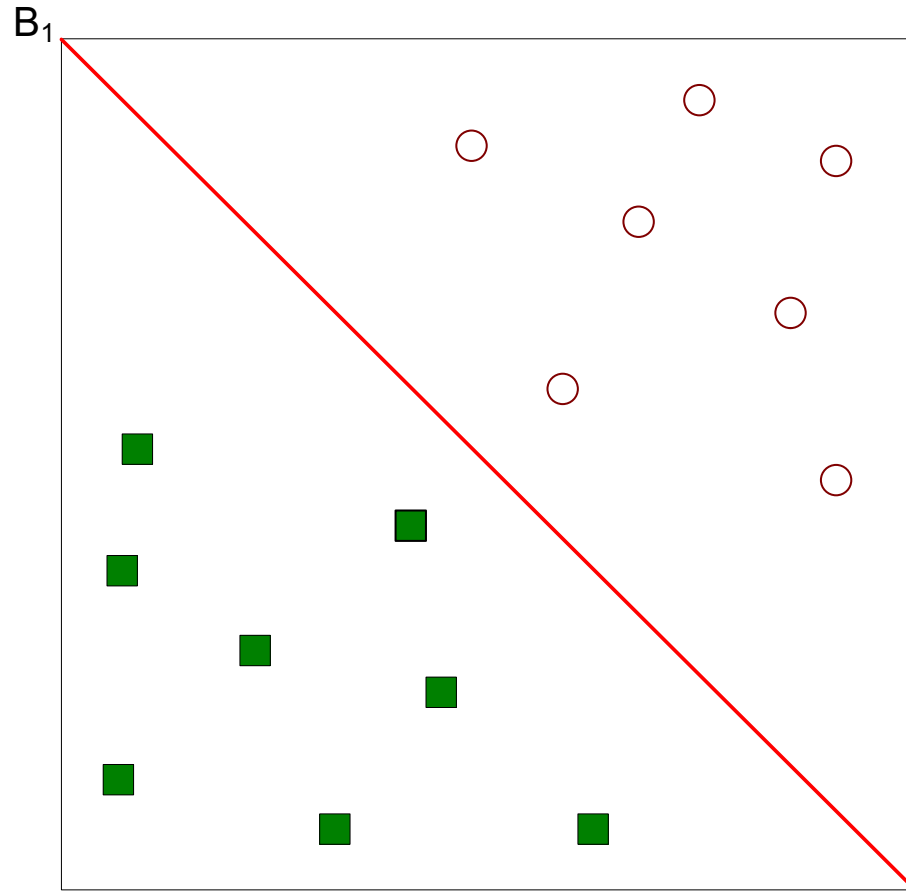
SUPPORT VECTOR MACHINES

SUPPORT VECTOR MACHINES (SVM)



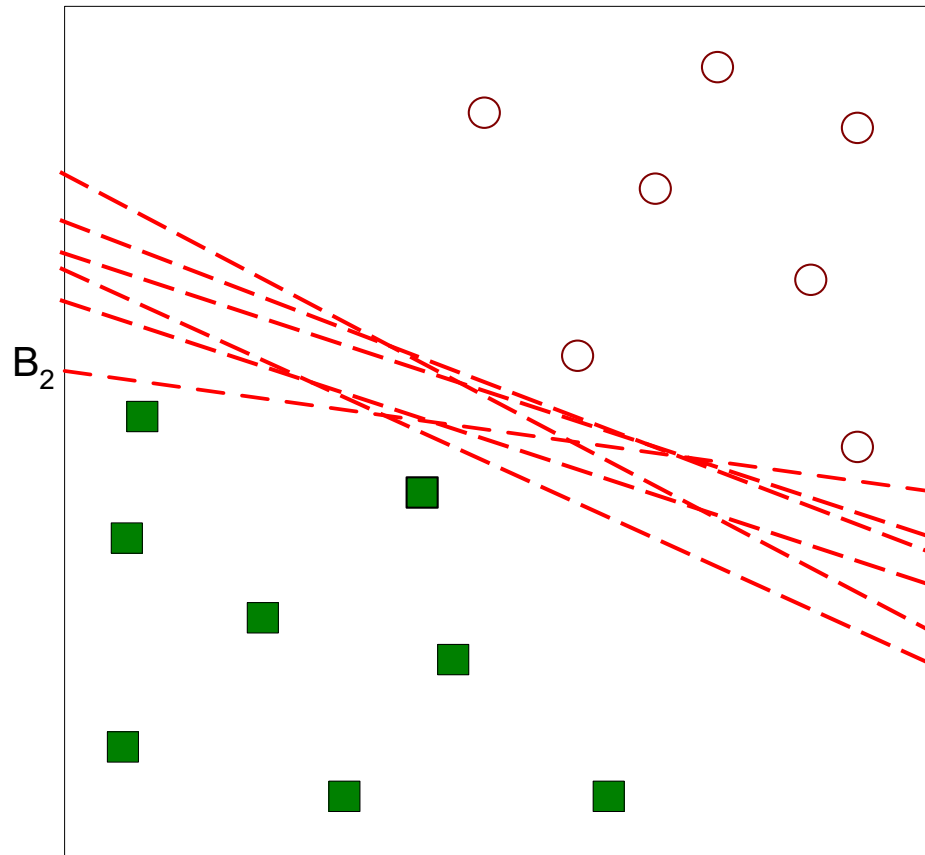
Find a linear hyperplane (decision boundary) that will separate the data

SUPPORT VECTOR MACHINES



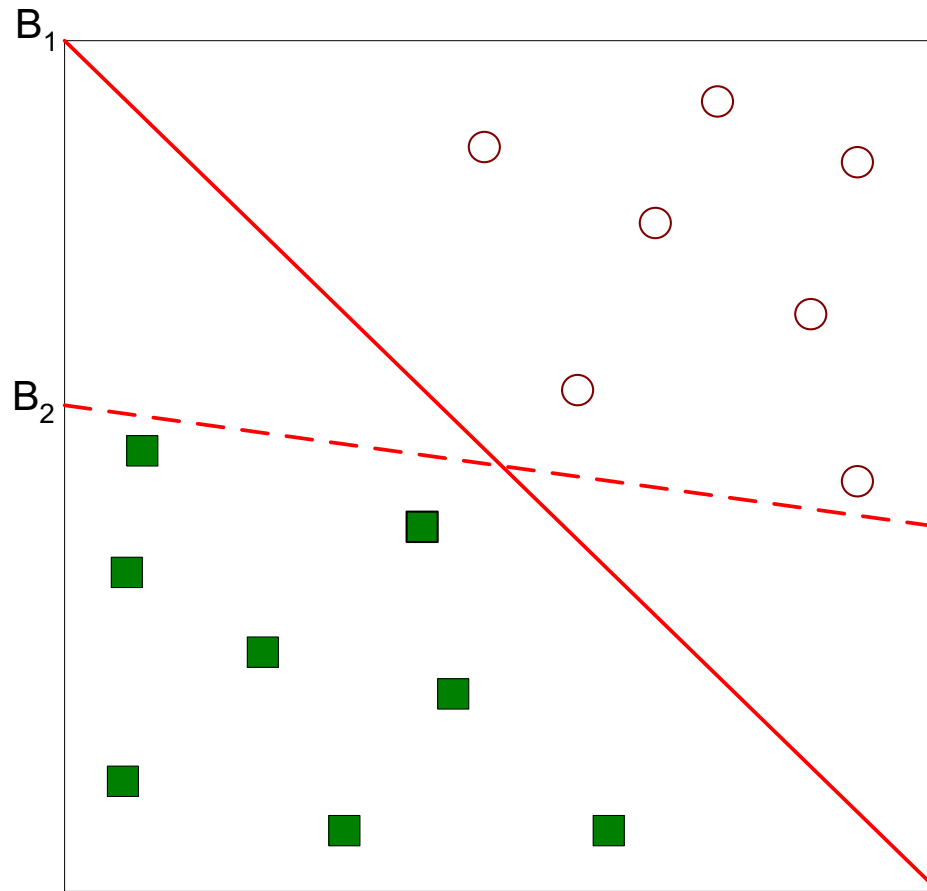
One possible solution

SUPPORT VECTOR MACHINES



Other possible solutions

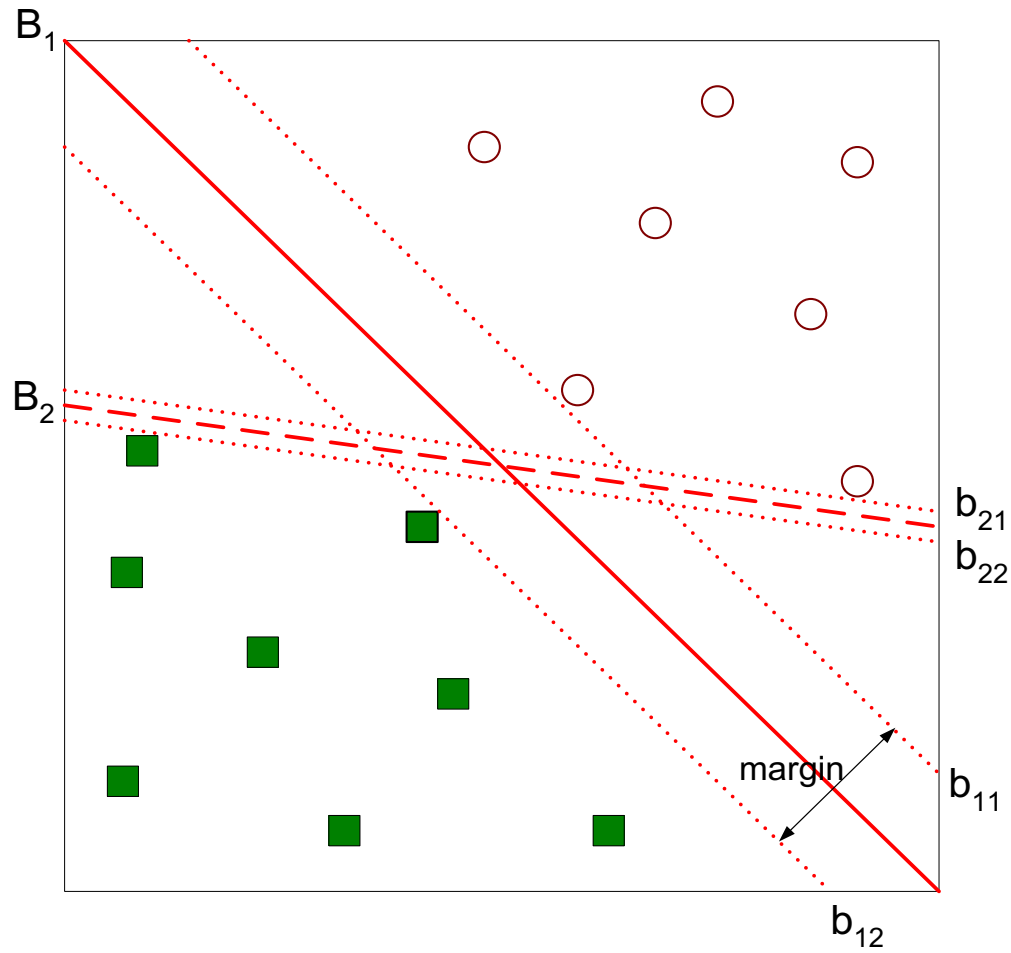
SUPPORT VECTOR MACHINES



Which one is better? B_1 or B_2 ? ????????????

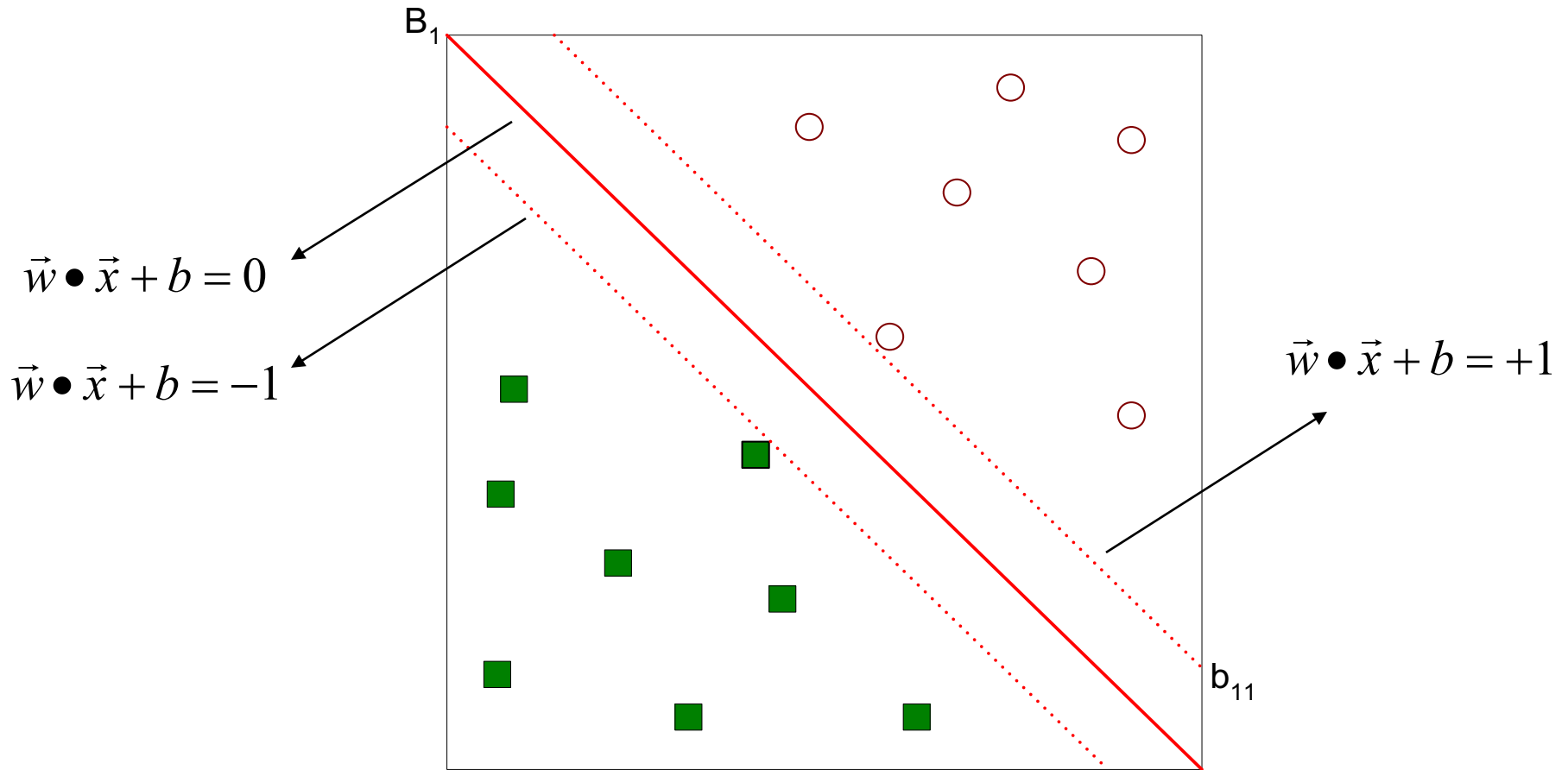
How do you define better? ????????????

SUPPORT VECTOR MACHINES



Find hyperplane **maximizes** the margin $\rightarrow B_1$ is better than B_2

SUPPORT VECTOR MACHINES



$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x} + b \leq -1 \end{cases}$$

$$\text{Margin} = \frac{2}{\|\vec{w}\|^2}$$

SUPPORT VECTOR MACHINES

We want to maximize: $\text{Margin} = \frac{2}{\|\vec{w}\|^2}$

Which is equivalent to minimizing: $L(w) = \frac{\|\vec{w}\|^2}{2}$

But subject to the following constraints:

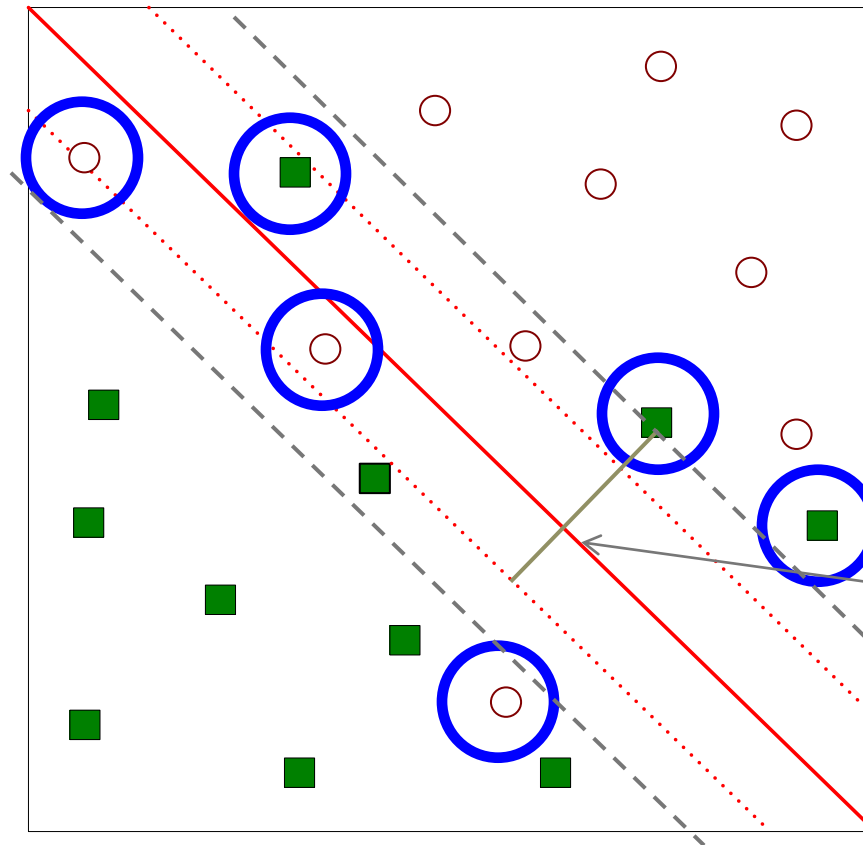
$$\begin{aligned}\vec{w} \cdot \vec{x}_i + b &\geq 1 \text{ if } y_i = 1 \\ \vec{w} \cdot \vec{x}_i + b &\leq -1 \text{ if } y_i = -1\end{aligned}$$

This is a **constrained optimization problem**

- Numerical approaches to solve it (e.g., quadratic programming)

SUPPORT VECTOR MACHINES

What if the problem is not linearly separable?



Apply some sort of penalty

$$\frac{\xi_i}{\|w\|}$$

SUPPORT VECTOR MACHINES

What if the problem is not linearly separable?

- Introduce **slack** variables
- Need to minimize:

$$L(w) = \frac{\|\vec{w}\|^2}{2} + C \left(\sum_{i=1}^N \xi_i \right)$$

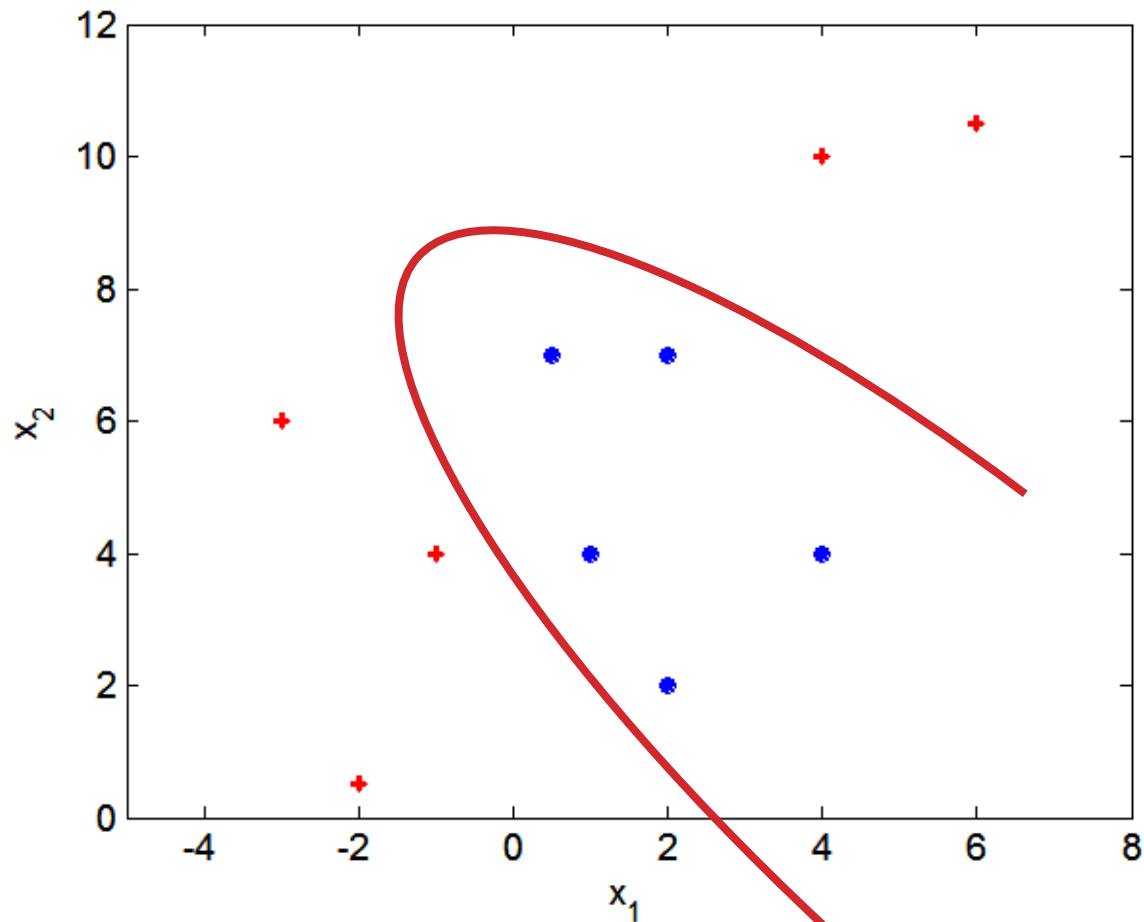
- Subject to:

$$\vec{w} \cdot \vec{x}_i + b \geq 1 - \xi_i \text{ if } y_i = 1$$

$$\vec{w} \cdot \vec{x}_i + b \leq -1 + \xi_i \text{ if } y_i = -1$$

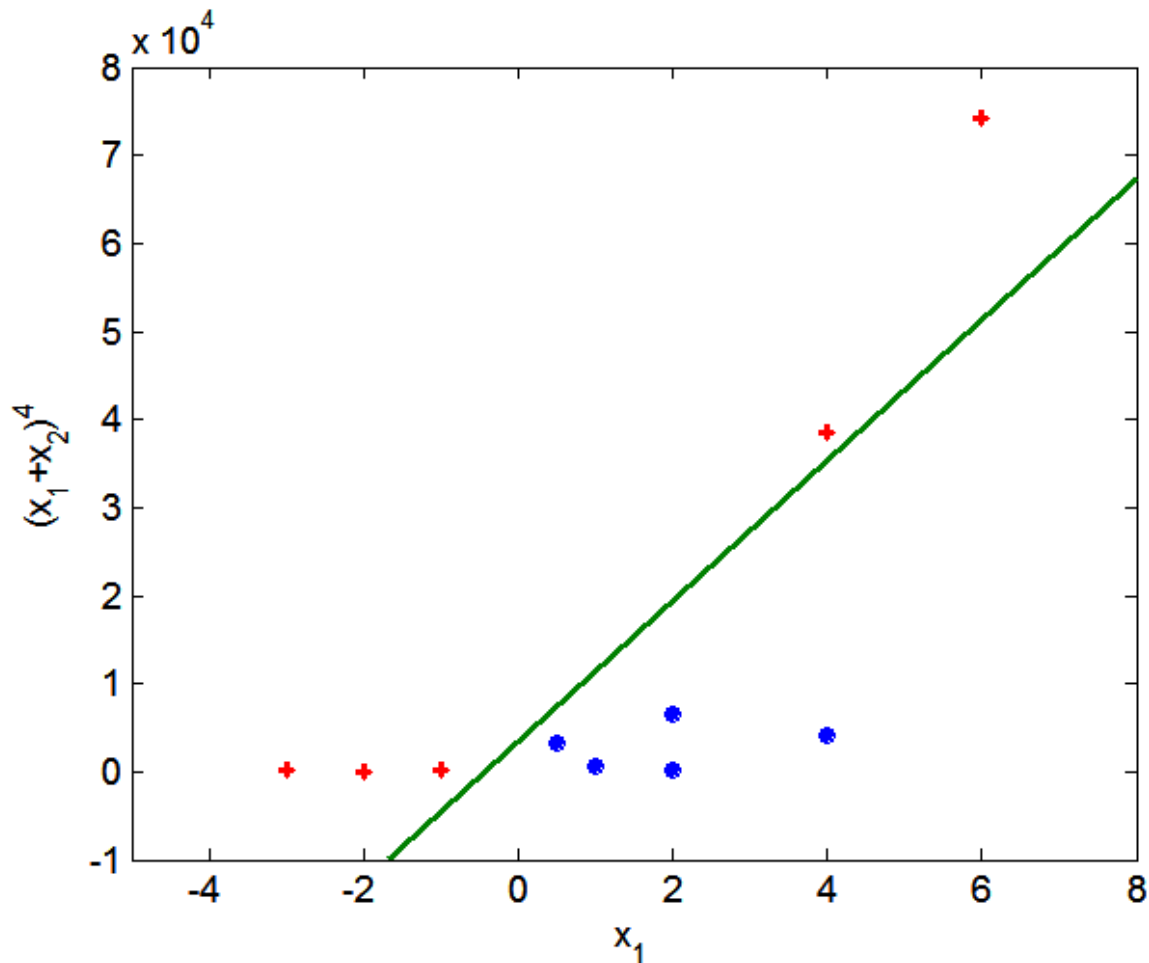
NONLINEAR SUPPORT VECTOR MACHINES

What if the decision boundary is not linear?



NONLINEAR SUPPORT VECTOR MACHINES

Transform data into higher dimensional space



SVMS IN SCIKIT-LEARN

```
from sklearn import svm

# Fit a default SVM classifier to fake data
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape=None, degree=3, gamma='auto',
kernel='rbf', max_iter=-1, probability=False,
random_state=None, shrinking=True, tol=0.001,
verbose=False)
```

Lots of defaults used for hyperparameters – can use cross validation (last class, 3/30) to search for good ones

MODEL SELECTION IN SCIKIT-LEARN

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

# ... Load some raw data into X and y ...
# Split the dataset in two equal parts
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.5, random_state=0)
```

```
# Pick values of hyperparameters you want to consider
tuned_parameters = [{'kernel': ['rbf'],
                      'gamma': [1e-3, 1e-4],
                      'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'],
                      'C': [1, 10, 100, 1000]}
                    ]
```

MODEL SELECTION IN SCIKIT-LEARN

```
# Perform a complete grid search + cross validation
# for each of the hyperparameter vectors
clf = GridSearchCV(SVC(C=1),
                  tuned_parameters,
                  cv=5,
                  scoring='precision')
clf.fit(X_train, y_train)
```

```
# Now that you've selected good hyperparameters via CV,
# and trained a model on your training data, get an
# estimate of the "true error" on your test set
y_true, y_pred = y_test, clf.predict(X_test)
print(classification_report(y_true, y_pred))
```

NEXT CLASS:
HYPOTHESIS TESTING

