Classification using Discriminative Models

Note – some material for the slides on Support Vector Machines came from:
- https://www.cs.utexas.edu/~mooney/cs391L/slides/svm.ppt
Classifiers

• “Classification” is the problem of identifying which of a set of categories (or classes) an observation belongs
  – Need training data containing observations (or instances) whose class membership is known
  – Typically observations are represented by feature vectors

• We need a model that relates the observation to the world state (in this case, the class)
  – A “generative” model represents the probability of observations, given the world, \( P(x|w) \)
  – A “discriminative” model represents the probability of the world, given the observation, \( P(w|x) \)
Popular Discriminative Classifiers

• **K-nearest neighbors**
  – The simplest possible discriminative classifier to implement.
  – Often effective, but for many training points, is slow and requires lots of memory.

• **Decision trees**
  – A feature is tested at each node. Very fast.
  – A variant is a forest of “random trees”.

• **Boosting**
  – The overall classification decision is made from the combined weighted decisions of a group of “weak” classifiers.
  – Effective when a large amount of training data is available.

• **Support vector machines**
  – Finds a small number of support vectors that straddle a separating hyperplane.
  – Among the best type of classifier with limited data, losing out to boosting or random trees only when large data sets are available.

• **Neural networks**
  – Uses “hidden units” between input and output.
  – It can be slow to train but is very fast to run.
K-nearest neighbors

- A query feature vector is classified by a majority vote of its $K$ nearest training vectors (distance measured in feature space)
- Exhaustively searching all the training vectors can be slow
- You can speed up search by organizing the training vectors into a data structure such as a “kd tree”

To classify the test (green) sample:
- If $k = 3$ it is assigned to the red class
- If $k = 5$ it is assigned to the blue class

KNN Example

• Fisher's iris data consists of measurements on the sepal length, sepal width, petal length, and petal width of 150 iris specimens. There are 50 specimens from each of three species.

• Here, we will just use 2 of the 4 feature dimensions (since it is easier to visualize)

• We also just use 2 of the 3 classes
Matlab’s knnsearch function

**Syntax**

```matlab
IDX = knnsearch(X,Y)
[IDX,D] = knnsearch(X,Y)
[IDX,D] = knnsearch(X,Y,'Name',Value)
```

**Description**

`IDX = knnsearch(X,Y)` finds the nearest neighbor in X for each point in Y. IDX is a column vector with m rows. Each row in IDX contains the index of nearest neighbor in X for the corresponding row in Y.

`[IDX,D] = knnsearch(X,Y)` returns an m-by-1 vector D containing the distances between each observation in Y and the corresponding closest observation in X. That is, D(i) is the distance between X(IDX(i),:) and Y(i,:).

`[IDX,D] = knnsearch(X,Y,'Name',Value)` accepts one or more optional comma-separated name-value pair arguments. Specify `Name` inside single quotes. knnsearch does not save a search object. To create a search object, use `createns`.

**Input Arguments**

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>An m-by-n numeric matrix. Rows of X correspond to observations and columns correspond to variables.</td>
</tr>
<tr>
<td>Y</td>
<td>An m-by-n numeric matrix of query points. Rows of Y correspond to observations and columns correspond to variables.</td>
</tr>
</tbody>
</table>

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'k'</td>
<td>Positive integer specifying the number of nearest neighbors in X for each point in Y. Default is 1. IDX and D are m-by-k matrices. D sorts the distances in each row in ascending order. Each row in IDX contains the indices of the k closest neighbors in X corresponding to the k smallest distances in D.</td>
</tr>
<tr>
<td>'IncludeTies'</td>
<td>A logical value indicating whether knnsearch includes all the neighbors whose distance values are equal to the kth smallest distance. If IncludeTies is true, knnsearch includes all these neighbors. In this case, IDX and D are m-by-1 cell arrays. Each row in IDX and D contains a vector with at least k numeric numbers. D sorts the distances in each vector in ascending order. Each row in IDX contains the indices of the closest neighbors corresponding to these smallest distances in D.</td>
</tr>
</tbody>
</table>
clear all
close all

% Load Fisher's iris data set. This loads in
% meas(N,4) - feature vectors, each 4 dimensional
% species(150) - class names: 'versicolor', 'virginica', 'setosa'
load fisheriris

% Let's keep only two classes.
inds1 = find(strcmp(species,'versicolor'));
inds2 = find(strcmp(species, 'virginica'));

% Name them class 1 and class 2.
y = [ones(length(inds1),1); 2*ones(length(inds2),1)];

% We will just use 2 feature dimensions, since it is easier to visualize.
X = meas([inds1;inds2],3:4);
% However, when we do that there is a chance that some points will be
% duplicated (since we are ignoring the other features). If so, just keep
% the first point.
indicesToKeep = true(size(X,1),1);
for i=1:size(X,1)
    % See if we already have the ith point.
    if any((X(i,1)==X(1:i-1,1)) & (X(i,2)==X(1:i-1,2)))
        indicesToKeep(i) = false; % Skip this point
    end
end
allFeatureVectors = X(indicesToKeep, :);
allClasses = y(indicesToKeep);

numTotal = size(allFeatureVectors,1);

% Plot the vectors.
figure, hold on;
myColors = ['r', 'g'];

for j=1:numTotal
    plot(allFeatureVectors(j,1),allFeatureVectors(j,2), ...
         'Color', myColors(allClasses(j)), 'Marker', 'o');
end
% Ask user to pick a test vector with the mouse.
fprintf('Click a point on the graph to choose a test vector.');
pointTest = ginput(1);

% Display the test vector.
plot(pointTest(1),pointTest(2), ...
    'Color', 'k', 'Marker', '*');

% Find the K nearest neighbors to the test vector.
K = 5;
indicesNeighbors = knnsearch(allFeatureVectors, pointTest, ... 
    'k', K);

% Display the nearest neighbors to this test vector.
line(allFeatureVectors(indicesNeighbors,1), ...
    allFeatureVectors(indicesNeighbors,2), ...
    'color', [.5 .5 .5],'marker','o', ... 
    'linestyle','none','markersize',10);

% Get the classes of the nearest neighbors.
classesNeighbors = allClasses(indicesNeighbors);
estimatedClass = mode(classesNeighbors); % Get majority vote

% Display the estimated class.
line(pointTest(1),pointTest(2), ...
    'color',myColors(estimatedClass),'marker','o', ... 
    'linestyle','none','markersize',10);

fprintf('Test point is classified as %d
', estimatedClass);
Support Vector Machines

• A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane
  – A “hyperplane” is a subspace of one dimension less than its ambient space
  – In 3D, the hyperplane is a plane. In 2D, the hyperplane is a line.
  – We have training vectors $\mathbf{x}_i$ and associated labels $y_i \in \{-1, +1\}$

$$w^T \mathbf{x} + b = 0$$

To classify a new point:
$$f(\mathbf{x}) = \text{sign}(w^T \mathbf{x} + b)$$
Optimal Hyperplane

- Intuitively, a line is bad if it passes too close to the points because it will be noise sensitive.
- We should find the line passing as far as possible from all points.

![Diagram showing the optimal hyperplane and points](image-url)
Maximize the “margin”

• The “margin” $\rho$ is the minimum distance to the closest training examples
• The optimal separating hyperplane maximizes the margin of the training data

Distance from example $x_i$ to the separator is

$$r = \frac{w^T x_i + b}{\|w\|}$$

The training examples closest to the hyperplane are called “support vectors”
Support Vectors

• Once you have found the optimal hyperplane, only the support vectors matter; other training examples are ignorable
Linear SVM Mathematically

• Let training set \( \{(x_i, y_i)\}_{i=1..n} \), \( x_i \in \mathbb{R}^d \), \( y_i \in \{-1, 1\} \) be separated by a hyperplane with margin \( \rho \). Then for each training example \((x_i, y_i)\):

\[
\begin{align*}
    w^T x_i + b &\leq -\frac{\rho}{2} \quad \text{if } y_i = -1 \\
    w^T x_i + b &\geq \frac{\rho}{2} \quad \text{if } y_i = 1
\end{align*}
\]

\( \iff \)

\[
y_i (w^T x_i + b) \geq \frac{\rho}{2}
\]

• For every support vector \( x_s \) the above inequality is an equality. After rescaling \( w \) and \( b \) by \( \rho/2 \) in the equality, the distance between each \( x_s \) and the hyperplane is

\[
r = \frac{y_s (w^T x_s + b)}{\|w\|} = \frac{1}{\|w\|}
\]

• Then the margin can be expressed through (rescaled) \( w \) and \( b \) as:

\[
\rho = 2r = \frac{2}{\|w\|}
\]
Linear SVMs Mathematically (cont.)

- Then we can formulate the *quadratic optimization problem*:

  \[
  \text{Find } w \text{ and } b \text{ such that } \\
  \rho = \frac{2}{||w||} \text{ is maximized} \\
  \text{and for all } (x_i, y_i), i=1..n : \quad y_i(w^T x_i + b) \geq 1
  \]

  Which can be reformulated as:

  \[
  \text{Find } w \text{ and } b \text{ such that } \\
  \Phi(w) = ||w||^2 = w^T w \text{ is minimized} \\
  \text{and for all } (x_i, y_i), i=1..n : \quad y_i(w^T x_i + b) \geq 1
  \]
Solving the Optimization Problem

Find \( \mathbf{w} \) and \( b \) such that
\[
\Phi(\mathbf{w}) = \mathbf{w}^T \mathbf{w}
\]
is minimized
and for all \( (\mathbf{x}_i, y_i), i = 1..n \):
\[
y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1
\]

• Need to optimize a \textit{quadratic} function subject to \textit{linear} constraints.

• We solve by constructing a \textit{dual problem} where a \textit{Lagrange multiplier} \( \alpha_i \) is associated with every inequality constraint in the primal (original) problem:

Find \( \alpha_1...\alpha_n \) such that
\[
Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j
\]
is maximized and

1. \( \sum \alpha_i y_i = 0 \)
2. \( \alpha_i \geq 0 \) for all \( \alpha_i \)
The Optimization Problem Solution

• Given a solution $\alpha_1...\alpha_n$ to the dual problem, the solution to the primal is:

$$w = \sum \alpha_i y_i x_i$$
$$b = y_k - \sum \alpha_i y_i x_i^T x_k$$
for any $\alpha_k > 0$

• Each non-zero $\alpha_i$ indicates that corresponding $x_i$ is a support vector.

• Then the classifying function is (note that we don’t need $w$ explicitly):

$$f(x) = \sum \alpha_i y_i x_i^T x + b$$

• To classify, we just take the inner product between the test point $x$ and the support vectors $x_i$. 
Example

- Fisher’s iris data again
  - Use 2 of the 4 feature dimensions
  - Use 2 of the 3 classes

Classes: 'setosa' and 'versicolor'
Dimensions 1 and 2
Matlab’s “fitcsvm” function

**fitcsvm**

Train binary support vector machine classifier

**Syntax**

```matlab
SVMModel = fitcsvm(TBL,ResponseVarName)
SVMModel = fitcsvm(TBL,formula)
SVMModel = fitcsvm(TBL,Y)
SVMModel = fitcsvm(X,Y)
SVMModel = fitcsvm(__,Name,Value)
```

**Description**

- `SVMModel = fitcsvm(TBL,ResponseVarName)` returns a support vector machine classifier `SVMModel` trained using the sample data contained in a table (`TBL`). `ResponseVarName` is the name of the variable in `TBL` that contains the class labels for one- or two-class classification.

- `SVMModel = fitcsvm(TBL,formula)` returns an SVM classifier trained using the sample data contained in a table (`TBL`). `formula` is a formula string that identifies the response and predictor variables in `TBL` used for training.

- `SVMModel = fitcsvm(TBL,Y)` returns an SVM classifier trained using the predictor variables in table `TBL` and class labels in vector `Y`.

- `SVMModel = fitcsvm(X,Y)` returns an SVM classifier trained using the predictors in the matrix `X` and class labels in vector `Y` for one- or two-class classification.

- `SVMModel = fitcsvm(__,Name,Value)` returns a support vector machine classifier with additional options specified by one or more `Name,Value` pair arguments, using any of the previous syntaxes. For example, you can specify the type of cross validation, the cost for misclassification, or the type of score transformation function.

---

Once you have a trained classifier `cl`, use it to predict the class of a new sample `x` using:

```
[class,score] = predict(cl,x);
```
clear all
close all

% Load Fisher's iris data set. This loads in
% meas(N,4) - feature vectors, each 4 dimensional
% species{150} - class names: 'versicolor', 'virginica', 'setosa'
load fisheriris

% Let's keep only two classes.
indices1 = find(strcmp(species,'setosa'));
indices2 = find(strcmp(species, 'versicolor'));

% Name them class 1 and class 2.
y = [ones(length(indices1),1); 2*ones(length(indices2),1)];

% We will just use 2 feature dimensions, since it is easier to visualize.
X = meas([indices1;indices2],1:2);
% However, when we do that there is a chance that some points will be
% duplicated (since we are ignoring the other features). If so, just keep
% the first point.
indicesToKeep = true(size(X,1),1);
for i=1:size(X,1)
    % See if we already have the ith point.
    if any((X(i,1)==X(1:i-1,1)) & (X(i,2)==X(1:i-1,2)))
        indicesToKeep(i) = false; % Skip this point
    end
end
allFeatureVectors = X(indicesToKeep, :);
allClasses = y(indicesToKeep);

numTotal = size(allFeatureVectors,1);
% Plot the vectors.
figure, hold on;
myColors = ['r', 'g'];
for j=1:numTotal
    plot(allFeatureVectors(j,1),allFeatureVectors(j,2), ... 
    'Color', myColors(allClasses(j)), 'Marker', 'o'); 
end

% Train SVM classifier.
cl = fitcsvm(allFeatureVectors,allClasses, ... 
    'KernelFunction', 'linear', ... % 'rbf', 'linear', 'polynomial' 
    'BoxConstraint', 1, ... % Default is 1 
    'ClassNames', [1,2]);

% Predict scores over the grid
[d] = 0.02;
[x1Grid,x2Grid] = meshgrid(min(allFeatureVectors(:,1)):d:max(allFeatureVectors(:,1)),... 
    min(allFeatureVectors(:,2)):d:max(allFeatureVectors(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
[~,scores] = predict(cl,xGrid);

% Plot the data and the decision boundary
figure;
h(1:2) = gscatter(allFeatureVectors(:,1),allFeatureVectors(:,2),allClasses,'rb','.'); 
hold on
h(3) = plot(allFeatureVectors(cl.IsSupportVector,1),allFeatureVectors(cl.IsSupportVector,2),'ko');
contour(x1Grid,x2Grid,reshape(scores(:,2),size(x1Grid)),[0 0],'k');
hold off

• To visualize the classifier, predict the class of a dense set of points on a grid.
• Then plot the contour between -1 and +1
Resulting Classifier

>> cl

c1 =

ClassificationSVM
   ResponseName: 'Y'
   CategoricalPredictors: []
   ClassNames: [1 2]
   ScoreTransform: 'none'
   NumObservations: 83
   Alpha: [18x1 double]
   Bias: -5.0000
   KernelParameters: [1x1 struct]
   BoxConstraints: [83x1 double]
   ConvergenceInfo: [1x1 struct]
   IsSupportVector: [83x1 logical]
   Solver: 'SMO'

- How many support vectors?
- What is $\alpha_i$, $b$?

\[ f(x) = \sum \alpha_i y_i x_i^T x + b \]
Soft Margin Classification

• What if the training set is not linearly separable?
  – Try Fisher data 'virginica' vs 'versicolor', dimensions 2,3 or 3,4
• *Slack variables* $\xi_i$ can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*. 
Soft Margin Classification Mathematically

• The old formulation:

\[
\begin{align*}
\text{Find } w \text{ and } b \text{ such that } \\
\Phi(w) &= w^T w \text{ is minimized} \\
\text{and for all } (x_i, y_i), i=1..n : y_i (w^T x_i + b) \geq 1
\end{align*}
\]

• Modified formulation incorporates slack variables:

\[
\begin{align*}
\text{Find } w \text{ and } b \text{ such that } \\
\Phi(w) &= w^T w + C \sum \xi_i \text{ is minimized} \\
\text{and for all } (x_i, y_i), i=1..n : y_i (w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0
\end{align*}
\]

• Parameter $C$ can be viewed as a way to control overfitting: it “trades off” the relative importance of maximizing the margin and fitting the training data.

*In Matlab’s “fitcsvm”, this is the “BoxConstraint” parameter*
Non-linear SVMs

• Datasets that are linearly separable with some noise work out great:

• But what are we going to do if the dataset is just too hard?

• How about... mapping data to a higher-dimensional space:
Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:

\[ \Phi: \mathbf{x} \rightarrow \phi(\mathbf{x}) \]
The “Kernel Trick”

• The linear classifier uses the inner product $K(x_i,x_j) = x_i^T x_j$

• If every datapoint is mapped into high-dimensional space via some transformation $\Phi$: $x \rightarrow \phi(x)$, the inner product becomes: $K(x_i,x_j) = \phi(x_i)^T \phi(x_j)$

• A kernel function is a function that is equivalent to an inner product in some feature space.

• Example:
  – Consider 2-dimensional vectors $x = [x_1, x_2]$; let $K(x_i,x_j) = (1 + x_i^T x_j)^2$,
  – We can write $K(x_i,x_j) = \phi(x_i)^T \phi(x_j)$
  – where $\phi(x) = [1, x_1^2, \sqrt{2} x_1 x_2, x_2^2, \sqrt{2} x_1, \sqrt{2} x_2]$  

• Thus, a kernel function implicitly maps data to a high-dimensional space (without the need to compute each $\phi(x)$ explicitly).
Examples of Kernel Functions

• Linear: \( K(x_i, x_j) = x_i^T x_j \)

• Polynomial of power \( p \): \( K(x_i, x_j) = (1 + x_i^T x_j)^p \)

• Gaussian (radial-basis function): \( K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \)

• The higher-dimensional space still has \textit{intrinsic} dimensionality \( d \), but linear separators in it correspond to \textit{non-linear} separators in original space.
Example

- Using “radial basis function” (rbf) kernel in “fitcsvm”:

![Input data](image1)

![Classification boundary, and support vectors](image2)
Multiple Classes

- The support vector machine is fundamentally a two-class classifier. However, we often have more than two classes.

- There are several approaches to handle $K>2$ classes
  - “One-versus-the-rest”
    - Construct $K$ separate SVMs, where the $k$th model is trained using data from class $C_k$ as the positive examples and data from the remaining $K-1$ classes are the negative examples.
    - To classify a point, choose the classifier with highest score
  - “One-versus-one”
    - Train $K(K-1)/2$ different 2-class SVMs on all possible pairs of classes
    - To classify a point, choose the classifier with highest number of votes
Multiple Classes

• The support vector machine is fundamentally a two-class classifier. However, we often have more than two classes.

• There are several approaches to handle $K>2$ classes

• “One-versus-the-rest”
  – Construct $K$ separate SVMs, where the $k$th model is trained using data from class $C_k$ as the positive examples and data from the remaining $K-1$ classes are the negative examples.
  – To classify a point, choose the classifier with highest score

• “One-versus-one”
  – Train $K(K-1)/2$ different 2-class SVMs on all possible pairs of classes
  – To classify a point, choose the classifier with highest number of votes
Matlab’s “fitcecoc” function

**fitcecoc**
Fit multiclass models for support vector machines or other classifiers.

**Syntax**

```matlab
Md1 = fitcecoc(tbl,ResponseVarName)
Md1 = fitcecoc(tbl,formula)
Md1 = fitcecoc(tbl,Y)
Md1 = fitcecoc(X,Y)
Md1 = fitcecoc(__,Name,Value)
```

**Description**

`Md1 = fitcecoc(tbl,ResponseVarName)` returns a full, trained error-correcting output codes (ECOC) multiclass model using the predictors in table `tbl` and the class labels in `tbl.ResponseVarName`. `fitcecoc` uses $K(K-1)/2$ binary support vector machine (SVM) models using the one-versus-one coding design, where $K$ is the number of unique class labels (levels). `Md1` is a `ClassificationECOC` model.

`Md1 = fitcecoc(tbl,formula)` returns an ECOC model using the predictors in table `tbl` and the class labels defined by the formula. The formula identifies the response and predictor variables in `tbl` used for training.

`Md1 = fitcecoc(tbl,Y)` returns an ECOC model using the predictors in table `tbl` and the class labels in vector `Y`.

`Md1 = fitcecoc(X,Y)` returns a full, trained ECOC model using the predictors `X` and the class labels `Y`.

`Md1 = fitcecoc(__,Name,Value)` returns an ECOC model with additional options specified by one or more `Name,Value` pair arguments, using any of the previous syntaxes.

For example, specify different binary learners, a different coding design, or to cross-validate. It is good practice to cross-validate using the `KFold` `Name,Value` pair argument. The cross-validation results determine how well the model generalizes.
Example

- Fisher’s iris data again
  - Use 2 of the 4 feature dimensions
  - Use all 3 classes

Dimensions 2 and 3
clear all
close all

% Load Fisher's iris data set. This loads in
%   meas(N,4) - feature vectors, each 4 dimensional
%   species{150} - class names: 'versicolor', 'virginica', 'setosa'
load fisheriris

% Get the indices of the classes.
indices1 = find(strcmp(species,'setosa'));
indices2 = find(strcmp(species,'versicolor'));
indices3 = find(strcmp(species,'virginica'));

% Name them class 1, class 2 and class 3.
y = [ones(length(indices1),1); 2*ones(length(indices2),1); 3*ones(length(indices2),1)];

% We will just use 2 feature dimensions, since it is easier to visualize.
X = meas([indices1;indices2;indices3],2:3);
% However, when we do that there is a chance that some points will be
% duplicated (since we are ignoring the other features). If so, just keep
% the first point.
indicesToKeep = true(size(X,1),1);
for i=1:size(X,1)
    % See if we already have the ith point.
    if any((X(i,1)==X(1:i-1,1)) & (X(i,2)==X(1:i-1,2)))
        indicesToKeep(i) = false;  % Skip this point
    end
end
allFeatureVectors = X(indicesToKeep, :);
allClasses = y(indicesToKeep);

numTotal = size(allFeatureVectors,1);
% Plot the vectors.
figure, hold on;
myColors = ['r', 'g', 'b'];

for j=1:numTotal
    plot(allFeatureVectors(j,1),allFeatureVectors(j,2), ...
         'Color', myColors(allClasses(j)), 'Marker', '*');
end

% Train SVM classifier for multiple classes.
cl = fitcecoc(allFeatureVectors,allClasses);

% Visualize by classifying all points on the grid.
d = 0.1;
[x1Grid,x2Grid] = meshgrid(min(allFeatureVectors(:,1)):d:max(allFeatureVectors(:,1)),... 
                          min(allFeatureVectors(:,2)):d:max(allFeatureVectors(:,2)));
xGrid = [x1Grid(:),x2Grid(:)];
[classes,~] = predict(cl,xGrid);
figure, hold on;
for j=1:length(classes)
    plot(xGrid(j,1), xGrid(j,2), 'Color', myColors(classes(j)), 'Marker', '.');
end
Resulting Classifier

>> cl

cl =

ClassificationECOC
   ResponseName: 'Y'
   CategoricalPredictors: []
   ClassNames: [1 2 3]
   ScoreTransform: 'none'
   BinaryLearners: {3x1 cell}
   CodingName: 'onevsone'

Look at cl.CodingMatrix to see the combinations of single class classifiers