Classification

- Automatically make a decision about inputs
  - Example: document → category
  - Example: image of digit → digit
  - Example: image of object → object type
  - Example: query + webpages → best match
  - Example: symptoms → diagnosis

- Three main ideas
  - Representation as feature vectors / kernel functions
  - Scoring by linear functions
  - Learning by optimization

Some Definitions

- Inputs: \( \mathbf{x}_i \) (close the ____)
- Candidate set: \( \mathcal{Y}(\mathbf{x}) \) (door, table, ...)
- Candidates: \( \mathbf{y} \) (table)
- True outputs: \( \mathbf{y}_i^* \) (door)
- Feature vectors: \( f(x, y) \) \([0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]\)

Feature Vectors

- Example: web page ranking (not actually classification)
  - \( x_i = \text{“Apple Computers”} \)
  - \( f_i(\text{Apple Computers}) = [0.3 \ 5 \ 0 \ 0 \ 0 \ 0 \ 0] \)
  - \( f_i(\text{Apple Logo}) = [0.8 \ 4 \ 2 \ 1 \ 0 \ 0 \ 0] \)
Block Feature Vectors

- Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates.

\[ x \quad \rightarrow \quad \text{...win the election...} \]

\[ f(x) = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \]

\[ f(\text{SPORTS}) = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]
\[ f(\text{POLITICS}) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \]
\[ f(\text{OTHER}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \]

Non-Block Feature Vectors

- Sometimes the features of candidates cannot be decomposed in this regular way.

Example: a parse tree's features may be the productions present in the tree:

\[ f(\text{SPORTS}) = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \]
\[ f(\text{POLITICS}) = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \]

- Different candidates will thus often share features.
- We'll return to the non-block case later.

Linear Models

In a linear model, each feature gets a weight \( w \).

\[ f(\text{POLITICS}) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \]
\[ w = \begin{bmatrix} 1 & 1 & -1 & -2 & 1 & -1 & 1 & -2 & -1 & -1 \end{bmatrix} \]

We score hypotheses by multiplying features and weights:

\[ \text{score}(y, w) = w^T f(y) \]

\[ \text{score}(\text{SPORTS}, w) = 1 \times 1 + 1 \times 1 = 2 \]

Linear Models: Decision Rule

- The linear decision rule:

\[ \text{prediction}(x) = \arg \max_{y \in Y(x)} w^T f(y) \]

\[ \text{score}(\text{SPORTS}, w) = 1 \times 1 + 1 \times 1 = 2 \]
\[ \text{score}(\text{POLITICS}, w) = (-2) \times 1 + (-1) \times 1 = -3 \]

\[ \text{prediction}(x) = \text{POLITICS} \]

- We've said nothing about where weights come from.

Binary Classification

Important special case: binary classification

- Classes are \( y = +1/-1 \)

\[ f(x, -1) = -f(x, +1) \]
\[ f(x) = 2f(x, +1) \]

- Decision boundary is a hyperplane

\[ w^T f(x) = 0 \]

\[ +1 = \text{SPAM} \]

\[ -1 = \text{HAM} \]
Multiclass Decision Rule

- If more than two classes:
  - Highest score wins
  - Boundaries are more complex
  - Harder to visualize

\[ \text{prediction}(x_i, w) = \arg \max_{y \in \mathcal{Y}} w^T f_i(y) \]
- There are other ways: e.g. reconcile pairwise decisions

Learning Classifier Weights

- Two broad approaches to learning weights
- Generative: work with a probabilistic model of the data, weights are \( \log \) local conditional probabilities
  - Advantages: learning weights is easy, smoothing is well-understood, backed by understanding of modeling
- Discriminative: set weights based on some error-related criterion
  - Advantages: error-driven, often weights which are good for classification aren’t the ones which best describe the data

We’ll mainly talk about the latter for now

How to pick weights?

- Goal: choose “best” vector \( w \) given training data
  - For now, we mean “best for classification”
- The ideal: the weights which have greatest test set accuracy / F1 / whatever
  - But, don’t have the test set
  - Must compute weights from training set
- Maybe we want weights which give best training set accuracy?
  - Hard discontinuous optimization problem
  - May not (does not) generalize to test set
  - Easy to overfit

Though, min-error training for MT does exactly this.

Minimize Training Error?

- A loss function declares how costly each mistake is
  \[ l_i(y) = l(x, y_i) \]
  - E.g. 0 loss for correct label, 1 loss for wrong label
  - Can weight mistakes differently (e.g. false positives worse than false negatives or Hamming distance over structured labels)

We could, in principle, minimize training loss:

\[ \min_w \sum_i l_i \left( \arg \max_{y \in \mathcal{Y}} w^T f_i(y) \right) \]
- This is a hard, discontinuous optimization problem

Linear Models: Perceptron

- The perceptron algorithm
  - Iteratively processes the training set, reacting to training errors
  - Can be thought of as trying to drive down training error
- The (online) perceptron algorithm:
  - Start with zero weights \( w \)
  - Visit training instances one by one
    - Try to classify
    - If correct, no change!
    - If wrong: adjust weights
      \[ w_i \leftarrow w + f_i(y_i) \]
      \[ w_i \leftarrow w - f_i(y_i) \]
Example: “Best” Web Page

\[ w = [1 \ 2 \ 0 \ 0 \ \ldots] \]
\[ x_i = "Apple Computers" \]
\[ f_i(x) = [0.3 \ 5 \ 0 \ 0 \ \ldots] \quad w^T f = 10.3 \]
\[ f_i(x) = [0.6 \ 4 \ 2 \ 1 \ \ldots] \quad w^T f = 8.8 \]
\[ w \leftarrow w + f(x_i^*) - f(\hat{y}) \]
\[ w = [1.5 \ 1 \ 2 \ 1 \ \ldots] \]

Perceptrons and Separability

- A data set is separable if some parameters classify it perfectly
- Convergence: if training data separable, perceptron will separate (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability

Examples: Perceptron

- Separable Case

\[ \text{Examples: Perceptron} \]

- Non-Separable Case

Issues with Perceptrons

- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining isn’t the typically discussed source of overfitting, but it can be important
- Regularization: if the data isn’t separable, weights often thrash around
  - Averaging weight vectors over time can help (averaged perceptron)
- Mediocre generalization: finds a “barely” separating solution

Problems with Perceptrons

- Perceptron “goal”: separate the training data
  \[ \forall i, y_i \neq y^i \quad w^T f_i(y^i) \geq w^T f_i(y) \]
  - This may be an entire feasible space
  - Or it may be impossible
**Objective Functions**

- What do we want from our weights?
  - Depends!
  - So far: minimize (training) errors:
    \[
    \sum_i \text{step} \left( w^T f_i(y_i) - \max_{y \neq y_i} w^T f_i(y) \right)
    \]
- This is the “zero-one loss”
  - Discontinuous, minimizing is NP-complete
  - Not really what we want anyway
- Maximum entropy and SVMs have other objectives related to zero-one loss

**Margin**

**Linear Separators**

- Which of these linear separators is optimal?

**Classification Margin (Binary)**

- Distance of \(x_i\) to separator is its margin \(m_i(y)\)
- Examples closest to the hyperplane are support vectors
- Margin \(\gamma\) of the separator is the minimum \(m\)

**Classification Margin**

- For each example \(x_i\) and possible mistaken candidate \(y\), we avoid that mistake by a margin \(m_i(y)\) (with zero-one loss)
  \[
  m_i(y) = w^T f_i(y_i) - w^T f_i(y) \]
- Margin \(\gamma\) of the entire separator is the minimum \(m\)
  \[
  \gamma = \min \left( w^T f_i(y_i) - \max_{y \neq y_i} w^T f_i(y) \right) \]
- It is also the largest \(\gamma\) for which the following constraints hold
  \[
  \forall i, y \quad w^T f_i(y_i) \geq w^T f_i(y) + \gamma f_i(y) \]

**Maximum Margin**

- Separable SVMs: find the max-margin \(w\)
  \[
  \max_{||w||=1} \gamma \quad \ell_i(y) = \begin{cases} 0 & \text{if } y = y_i^+ \\ 1 & \text{if } y = y_i^- \end{cases} \]
  \[
  \forall i, y \quad w^T f_i(y_i) \geq w^T f_i(y) + \gamma f_i(y) \]
- Can stick this into Matlab and (slowly) get an SVM
- Won’t work (well) if non-separable
Why Max Margin?

- Why do this? Various arguments:
  - Solution depends only on the boundary cases, or support vectors (but remember how this diagram is broken!)
  - Solution robust to movement of support vectors
  - Sparse solutions (features not in support vectors get zero weight)
  - Generalization bound arguments
  - Works well in practice for many problems

Max Margin / Small Norm

- Reformulation: find the smallest \( w \) which separates data
  \[ \max \frac{1}{\|w\|^2} \quad \forall i, y \quad w^T f_i(x_i) \geq w^T f_i(y) + \gamma \xi_i(y) \]
- \( \gamma \) scales linearly in \( w \), so if \( \|w\| \) isn’t constrained, we can take any separating \( w \) and scale up our margin
  \[ \gamma = \min_{\|w\| \leq 1} \frac{1}{\|w\|^2} \quad \forall i, y \quad w^T f_i(x_i) \geq w^T f_i(y) + \xi_i(y) \]
- Instead of fixing the scale of \( w \), we can fix \( \gamma = 1 \)
  \[ \min \frac{1}{2}\|w\|^2 \quad \forall i, y \quad w^T f_i(x_i) \geq w^T f_i(y) + \xi_i(y) \]

Soft Margin Classification

- What if the training set is not linearly separable?
- Slack variables \( \xi \) can be added to allow misclassification of difficult or noisy examples, resulting in a soft margin classifier

Maximum Margin

- Non-separable SVMs
  - Add slack to the constraints
  - Make objective pay (linearly) for slack:
    \[ \min \frac{1}{2}\|w\|^2 + C \sum \xi_i \quad \forall i, y \quad w^T f_i(x_i) + \xi_i \geq w^T f_i(y) + \xi_i(y) \]
- \( C \) is called the capacity of the SVM – the smoothing knob
- Learning:
  - Can still stick this into Matlab if you want
  - Constrained optimization is hard; better methods!
  - We’ll come back to this later

Notice other choices of how to penalize slacks!
Linear Models: Maximum Entropy

- Maximum entropy (logistic regression)
  - Use the scores as probabilities:
    \[ P(y|x, w) = \frac{\exp(w^T f(x))}{\sum_y \exp(w^T f(y))} \]
  - Make normalization
  - Maximize the (log) conditional likelihood of training data
    \[ L(w) = \log \prod_i P(y_i|x_i, w) = \sum_i \log \left( \frac{\exp(w^T f(y_i))}{\sum_y \exp(w^T f(y))} \right) \]
    \[ = \sum_i \left( w^T f(y_i) - \log \sum_y \exp(w^T f(y)) \right) \]

Maximum Entropy II

- Motivation for maximum entropy:
  - Connection to maximum entropy principle (sort of)
  - Might want to do a good job of being uncertain on noisy cases...
    - ... in practice, though, posteriors are pretty peaked
  - Regularization (smoothing)
    \[ \min_{w} \sum_i \left( w^T f_i(y_i) - \log \sum_y \exp(w^T f_i(y)) \right) - \lambda ||w||^2 \]
    \[ \min_{w} \lambda ||w||^2 - \sum_i \left( w^T f_i(y_i) - \log \sum_y \exp(w^T f_i(y)) \right) \]

Maximum Entropy

Loss Comparison

Log-Loss

- If we view maxent as a minimization problem:
  \[ \min_{w} \frac{1}{2} ||w||^2 + \sum_i \left( -w^T f_i(y) + \log \sum_y \exp(w^T f_i(y)) \right) \]
- This minimizes the “log loss” on each example
  \[ -\left( w^T f_i(y) - \log \sum_y \exp(w^T f_i(y)) \right) = -\log P(y_i|x, w) \]
- One view: log loss is an upper bound on zero-one loss

Remember SVMs...

- We had a constrained minimization
  \[ \min_{w} \frac{1}{2} ||w||^2 + C \sum \epsilon_i \quad \forall i, \ y_i = w^T f_i(x) + \xi \]
  \[ \epsilon_i \geq w^T f_i(x) + \xi \]
- ...but we can solve for \( \xi \)
  \[ \forall i, \ y_i \geq w^T f_i(x) + \xi \]
  \[ \epsilon_i = \max \left( w^T f_i(x) + \xi \right) - w^T f_i(x) \]
- Giving
  \[ \min_{w} \frac{1}{2} ||w||^2 + C \sum \epsilon_i \left( \max \left( w^T f_i(x) + \epsilon \right) - w^T f_i(x) \right) \]
Hinge Loss

- Consider the per-instance objective:
  \[ \min_w \: \frac{1}{2} \|w\|^2 + \sum_i \max(0, 1 - y_i(x_i^T w)) \]

- This is called the "hinge loss"
  - Unlike maxent/log loss, you stop gaining objective once the true label wins by enough
  - You can start from here and derive the SVM objective
  - Can solve directly with subgradient decent (e.g. Pegasos: Shalev-Shwartz et al 07)

Max vs “Soft-Max” Margin

- SVMs:
  \[ \min_w \: \frac{1}{2} \|w\|^2 + \sum_i \max(0, 1 - y_i(x_i^T w)) \]

- Maxent:
  \[ \min_w \: \frac{1}{2} \|w\|^2 - \sum_i \left( y_i \log \frac{e^{x_i^T w}}{\sum_j e^{x_j^T w}} \right) \]

- You can make this zero
  - But not this one

Loss Functions: Comparison

- Zero-One Loss
  \[ \sum_i \max(w^T f_i(x_i) - \max_j w^T f_j(y_j)) \]

- Hinge
  \[ \sum_i \left( y_i (x_i^T w) - \max_j (x_i^T w_j) \right) \]

- Log
  \[ \sum_i \left( w^T f_i(x_i) - \log \sum_j e^{x_i^T w_j} \right) \]

Separators: Comparison

Conditional vs Joint Likelihood

Example: Sensors

- Raining
  - Reality: \( P(+,+,r) = 3/8 \) \( P(+,+,s) = 1/8 \)
  - Predictions: \( P(r|+,+) = 9/10 \) \( P(s|+,+) = 1/10 \)

- Sunny
  - Reality: \( P(-,-,r) = 1/8 \) \( P(-,-,s) = 3/8 \)
  - Predictions: \( P(r|-,-) = 9/10 \) \( P(s|-,-) = 1/10 \)
Example: Stoplights

**Reality**

<table>
<thead>
<tr>
<th>Lights Working</th>
<th>Lights Broken</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(g,r,w) = 3/7</td>
<td>P(r,g,w) = 3/7</td>
</tr>
<tr>
<td>P(r,r,b) = 1/7</td>
<td></td>
</tr>
</tbody>
</table>

**NB Model**

<table>
<thead>
<tr>
<th>Working?</th>
<th>NS</th>
<th>EW</th>
</tr>
</thead>
</table>

**NB FACTORS:**

- P(w) = 6/7
- P(r|w) = 1/2
- P(g|w) = 1/2
- P(b) = 1/7
- P(r|b) = 1
- P(g|b) = 0

**Example: Stoplights**

- What does the model say when both lights are red?
  - P(b,r,r) = (1/7)(1)(1) = 1/7 = 4/28
  - P(w,r,r) = (6/7)(1/2)(1/2) = 6/28 = 6/28
  - P(w|r,r) = 6/10!
- We’ll guess that (r,r) indicates lights are working!

- Imagine if P(b) were boosted higher, to 1/2:
  - P(b,r,r) = (1/2)(1)(1) = 1/2 = 4/8
  - P(w,r,r) = (1/2)(1/2)(1/2) = 1/8 = 1/8
  - P(w|r,r) = 1/5!
- Changing the parameters bought accuracy at the expense of data likelihood