Machine Learning

- Up until now: how to reason in a model and how to make optimal decisions
- Machine learning: how to acquire a model on the basis of data / experience
  - Learning parameters (e.g. probabilities)
  - Learning structure (e.g. BN graphs)
  - Learning hidden concepts (e.g. clustering)

Machine Learning This Set of Slides

- Applications
- Naïve Bayes
- Main concepts
- Perceptron

Example: Spam Filter

- Input: email
- Output: spam/ham
- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ham”
  - Note: someone has to hand label all this data
  - Want to learn to predict labels of new, future emails
- Features: The attributes used to make the ham / spam decision
  - Words: FREE!
  - Text Patterns: $dd, CAPS
  - Non-text: SenderInContacts

Example: Digit Recognition

- Input: images / pixel grids
- Output: a digit 0-9
- Setup:
  - Get a large collection of example images, each labeled with a digit
  - Note: someone has to hand label all this data
  - Want to learn to predict labels of new, future digit images
- Features: The attributes used to make the digit decision
  - Pixels: (i,j)=ON
  - Shape Patterns: NumComponents,AspectRatio, NumLoops

Other Classification Tasks

- In classification, we predict labels y (classes) for inputs x
- Examples:
  - Spam detection (input: document, classes: spam / ham)
  - OCR (input: images, classes: characters)
  - Medical diagnosis (input: symptoms, classes: diseases)
  - Automatic essay grader (input: document, classes: grades)
  - Fraud detection (input: account activity, classes: fraud / no fraud)
  - Customer service email routing
  - … many more
- Classification is an important commercial technology!
Bayes Nets for Classification

- One method of classification:
  - Use a probabilistic model!
  - Features are observed random variables $F_i$
  - $Y$ is the query variable
  - Use probabilistic inference to compute most likely $Y$

$$y = \arg\max_y P(y|f_1 \ldots f_n)$$

- You already know how to do this inference

General Naïve Bayes

- A general naïve Bayes model:

\[
P(Y, F_1 \ldots F_n) = P(Y) \prod_i P(F_i|Y)
\]

- We only specify how each feature depends on the class
- Total number of parameters is linear in $n$

Inference for Naïve Bayes

- Goal: compute posterior over causes
  - Step 1: get joint probability of causes and evidence
  - Step 2: get probability of evidence
  - Step 3: renormalize

Naïve Bayes for Digits

- Simple version:
  - One feature $F_{ij}$ for each grid position $<i,j>$
  - Possible feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
  - Each input maps to a feature vector, e.g.

\[
\begin{aligned}
F_{0,0} &= 1 \\
F_{0,1} &= 0 \\
F_{0,2} &= 0 \\
F_{1,0} &= 1 \\
F_{1,1} &= 0 \\
\end{aligned}
\]

- Here: lots of features, each is binary valued

- Naïve Bayes model:

\[
P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{ij}|Y)
\]

- What do we need to learn?

A Digit Recognizer

- Input: pixel grids
- Output: a digit 0-9

Examples: CPTs

\[
P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{ij}|Y)
\]
Naïve Bayes for Text

- **Bag-of-Words Naïve Bayes:**
  - Predict unknown class label (spam vs. ham)
  - Assume evidence features (e.g., the words) are independent
  - Warning: subtly different assumptions than before!

- **Generative model**
  \[ P(Y; W_1 \ldots W_n) = P(Y) \prod_i P(W_i | Y) \]

- **Tied distributions and bag-of-words**
  - Usually, each variable gets its own conditional probability distribution \(P(F|Y)\)
  - In a bag-of-words model
    - Each position is identically distributed
    - All positions share the same conditional probs \(P(W|C)\)
  - Why make this assumption?
    - Word at position \(i\), not \(i^{th}\) word in the dictionary!

Example: Overfitting

- Posterior determined by relative probabilities (odds ratios):
  \[
  \frac{P(W|h) \cdot P(Y)}{P(W|s) \cdot P(Y)}
  \]
  | south-west | : inf |
  | nation     | : inf |
  | morally    | : inf |
  | nicely     | : inf |
  | extent     | : inf |
  | seriously  | : inf |
  | ...        | ...  |

  What went wrong here?

Generalization and Overfitting

- Relative frequency parameters will overfit the training data!
  - Just because we never saw a 3 with pixel (15, 15) on during training doesn’t mean we won’t see it at test time
  - Unlikely that every occurrence of “minute” is 100% spam
  - What about all the words that don’t occur in the training set at all?
  - In general, we can’t go around giving unseen events zero probability

  As an extreme case, imagine using the entire email as the only feature
  - Would get the training data perfect (if deterministic labeling)
  - Wouldn’t generalize at all
  - Just making the bag-of-words assumption gives us some generalization, but isn’t enough

Estimation: Smoothing

- Problems with maximum likelihood estimates:
  - If I flip a coin once, and it’s heads, what’s the estimate for \(P(\text{heads})\)?
  - What if I flip 10 times with 8 heads?
  - What if I flip 10M times with 8M heads?

- Basic idea:
  - We have some prior expectation about parameters (here, the probability of heads)
  - Given little evidence, we should skew towards our prior
  - Given a lot of evidence, we should listen to the data

\[
\hat{\theta}_{ML} = \arg \max_{\theta} P(X|\theta) = \frac{\text{count}(x)}{\text{total samples}}
\]

- In Bayesian statistics, we think of the parameters as just another random variable, with its own distribution

\[
\hat{\theta}_{MAP} = \arg \max_{\theta} P(\theta|X) = \frac{\arg \max_{\theta} P(X|\theta)P(\theta)}{P(X)} \quad ???
\]

\[
= \arg \max_{\theta} P(X|\theta)P(\theta)
\]
Estimation: Laplace Smoothing

- Laplace’s estimate:
  - Pretend you saw every outcome once more than you actually did
  \[
P_{LAP}(x) = \frac{c(x) + 1}{\sum_c c(x) + 1}
  \]
  \[
P_{ML}(X) = \frac{c(x) + 1}{N + |X|}
  \]
  - Can derive this as a MAP estimate with Dirichlet priors (see cs281a)

Estimation: Linear Interpolation

- In practice, Laplace often performs poorly for P(X|Y):
  - When |X| is very large
  - When |Y| is very large

- Another option: linear interpolation
  - Also get P(X) from the data
  - Make sure the estimate of P(X|Y) isn’t too different from P(X)
  \[
P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha) \hat{P}(x)
  \]
  - What if \( \alpha \) is 0? 1?

Real NB: Smoothing

- For real classification problems, smoothing is critical

- New odds ratios:
  - \[
P(W|ham) / P(W|spam) = \frac{c(x,y) + k}{c(y) + k|X|}
  \]
  - \[
P(W|spam) / P(W|ham) = \frac{c(x,y) + k}{c(y) + k|X|}
  \]

Do these make more sense?

Tuning on Held-Out Data

- Now we’ve got two kinds of unknowns
  - Parameters: the probabilities P(Y|X), P(Y)
  - Hyperparameters, like the amount of smoothing to do: k, α

- Where to learn?
  - Learn parameters from training data
  - Must tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters, train and test on the held-out data
    - Choose the best value and do a final test on the held-out data

Important Concepts

- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held-out set
  - Test set

- Features: attribute-value pairs which characterize each x
  - Experimentation cycle
    - Learn parameters (e.g. model probabilities) on training set
    - (Tune hyperparameters on held-out set)
    - Compute accuracy of test set
    - Very important: never “peek” at the test set

- Evaluation
  - Accuracy: fraction of instances predicted correctly
  - Overfitting and generalization
    - Want a classifier which does well on test data
    - Overfitting: fitting the training data very closely, but not generalizing well
Generative vs. Discriminative

- Generative classifiers:
  - E.g. naïve Bayes
  - A probabilistic model with evidence variables
  - Query model for class variable given evidence

- Discriminative classifiers:
  - No generative model, no Bayes rule, often no probabilities at all!
  - Try to predict the label \( Y \) directly from \( X \)
  - Robust, accurate with varied features
  - Loosely: mistake driven rather than model driven

Binary Linear Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to \( Y = +1 \)
  - Other corresponds to \( Y = -1 \)

- Dot product \( w \cdot f \) positive means the positive class

Binary Perceptron Update

- Start with zero weights
- For each training instance:
  - Classify with current weights
  - If correct (i.e., \( y = y^* \)), no change!
  - If wrong: adjust the weight vector
    by adding or subtracting the feature vector. Subtract if \( y^* \) is -1.

Example Exercise --- Which Category is Chosen?

- "win the vote"

Multiclass Linear Decision Rule

- If we have multiple classes:
  - A weight vector for each class:
    \( w_y \)
  - Score (activation) of a class \( y \):
    \( w_y \cdot f(x) \)
  - Prediction highest score wins
    \( y = \arg \max_y w_y \cdot f(x) \)

- \( w \) = multiclass where the negative class has weight zero

Multiclass Linear Decision Rule

- If we have multiple classes:
  - A weight vector for each class:
    \( w_y \)
  - Score (activation) of a class \( y \):
    \( w_y \cdot f(x) \)
  - Prediction highest score wins
    \( y = \arg \max_y w_y \cdot f(x) \)
Learning Multiclass Perceptron

- Start with zero weights
- Pick up training instances one by one
- Classify with current weights
  \[ y = \arg\max_y w_y \cdot f(x) \]
  \[ y^{*} = \arg\max_y \sum_i w_{yi} \cdot f_i(x) \]
- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer
  \[ w_y = w_y - f(x) \]
  \[ w_{y^{*}} = w_{y^{*}} + f(x) \]

Example

"win the vote"
"win the election"
"win the game"

Properties of Perceptrons

- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability
  \[ \text{mistakes} < \frac{k}{\delta^2} \]

Examples: Perceptron

- Separable Case

Problems with the Perceptron

- Noise: if the data isn’t separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)
- Mediocre generalization: finds a ‘barely’ separating solution
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting

Fixing the Perceptron

- Idea: adjust the weight update to mitigate these effects
- MIRA*: choose an update size that fixes the current mistake...
  \[ \min_{w} \frac{1}{2} \sum_y ||w_y - w_{y^{*}}||^2 \]
  \[ w_{y^{*}} \cdot f(x) \geq w_y \cdot f(x) + 1 \]
  \[ w_y = w_y^{*} - \tau f(x) \]
  \[ w_{y^{*}} = w_{y^{*}} + \tau f(x) \]
* Margin Infused Relaxed Algorithm
Minimum Correcting Update

\[
\min \frac{1}{2} \sum \|w_y - w'_y\|^2 \\
\text{s.t. } w_y \cdot f \geq w'_y \cdot f + 1
\]

\[
w_y = w'_y - \tau f(x) \\
w_y = w'_y + \tau f(x)
\]

\[\min \|\tau f\|^2 \\
\text{s.t. } w_y \cdot f \geq w'_y \cdot f + 1
\]

\[\tau = \frac{(w'_y - \tau f) \cdot f + 1}{2f \cdot f}
\]

Min not \(\tau\neq 0\), or would not have made an error, so \(\min\) will be where equality holds

Maximum Step Size

- In practice, it’s also bad to make updates that are too large
- Example may be labeled incorrectly
- You may not have enough features
- Solution: cap the maximum possible value of \(\tau\) with some constant \(C\)

\[\tau^* = \min \left(\frac{(w'_y - w''_y) \cdot f + 1}{2f \cdot f}, C\right)
\]

- Corresponds to an optimization that assumes non-separable data
- Usually converges faster than perceptron
- Usually better, especially on noisy data

Extension: Web Search

- Information retrieval:
  - Given information needs, produce information
  - Includes, e.g. web search, question answering, and classic IR
- Web search: not exactly classification, but rather ranking

Feature-Based Ranking

\[x = \text{"Apple Computers"}
\]

\[f(x) = [0.3 \ 5 \ 0 \ 0 \ldots]
\]

\[f(x) = [0.8 \ 4 \ 2 \ 1 \ldots]
\]

Now features depend on query and webpage.
E.g.: "times word1 in query occurs, "times word2 in query occurs, "times all words in query occur in sequence, page rank

Perceptron for Ranking

- Inputs \(x\)
- Candidates \(y\)
- Many feature vectors: \(f(x, y)\)
- One weight vector: \(w\)
  - Prediction:
    \[y = \arg \max_y w \cdot f(x, y)
    \]
  - Update (if wrong):
    \[w = w + f(x, y^*) - f(x, y)
    \]

Classification: Comparison

- Naïve Bayes
  - Builds a model training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)
- Perceptrons / MIRA:
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate