Lecture 7: feature selection, combination of methods
Topics

● Feature selection
  – filter methods
  – wrapper methods

● Combination of methods
  – greedy sequential fitting
  – voting methods: bagging, boosting
A bit more general view

- A filtering approach
  - is generic, i.e., not optimized for any specific classifier
  - may sacrifice classification accuracy
  - modular

- A wrapper approach
  - is always tailored to a specific classifier
  - may lead to better accuracy as a result
Example: feature pruning

- The goal here is to remove non-informative features

- Definitions:
  1. \( \{\phi_1(x), \ldots, \phi_m(x)\} \) is the set of possible word detectors,
  2. \( \Phi(x) = [\phi_{i_1}(x), \ldots, \phi_{i_m'}(x)]^T \) is our current feature vector (current set of word detectors),
  3. \( \Phi^{-i}(x) \) is the current feature vector without \( \phi_i(x) \) component

- We’d like to remove any uninformative word detectors from the current feature vector \( \Phi(x) \)

What is “uninformative”? 
Feature pruning cont’d

• A word detector $\phi_{ik}(x)$ is uninformative if it doesn’t help predict the label, i.e., if

$$\hat{P}(y|\Phi^{-ik}(x)) \approx \hat{P}(y|\Phi(x))$$

for all labels $y = 0, 1$ and documents $x$.

• If the probabilities here are estimated using a specific classifier such as Naive Bayes, then this is a wrapper approach. Otherwise we are dealing with a filtering method.
Feature pruning with Naive Bayes (wrapper)

• A word detector \( \phi_{ik}(x) \) is uninformative if it doesn’t help predict the label, i.e., if

\[
P(y|\Phi^{-ik}(x), \hat{\theta}) \approx P(y|\Phi(x), \hat{\theta})
\]

for all labels \( y = 0, 1 \) and documents \( x \).

• These probabilities are now computed from the Naive Bayes model:

\[
P(\Phi(x)|y, \hat{\theta}) = \prod_{j=1}^{m'} P(\phi_{ij}(x)|y, \hat{\theta}_k)
\]

\[
P(y|\Phi(x), \hat{\theta}) = \frac{P(\Phi(x)|y, \hat{\theta})\tilde{P}(y)}{\sum_{y'=0,1} P(\Phi(x)|y', \hat{\theta})\tilde{P}(y')}
\]

for each document \( x \).

• Note that the “expert” models \( P(\phi_{i}(x)|y, \hat{\theta}_k) \) need not be recomputed during the feature search.
Another Wrapper approach

- Let’s look at the document classification task again, now with a logistic regression model. We have $m$ possible binary word detectors $\{\phi_1(x), \ldots, \phi_m(x)\}$ and

$$P(y = 1|x, w) = g(w_0 + w_1\phi_1(x) + \ldots + w_m\phi_m(x))$$

when all features are included.

- We’d like to find a small subset of features that lead to good classification.

- We can
  1. Greedily add features
  2. Find relevant features using regularization
Greedy selection of features

1. Find $k$ for which

$$P(y = 1|x, w) = g(w_0 + w_k \phi_k(x))$$

yields the best classifier

2. Find $k'$ for which

$$P(y = 1|x, w) = g(w_0 + w_k \phi_k(x) + w_{k'} \phi_{k'}(x))$$

yields the best classifier. $w_0$, $w_k$ and $w_{k'}$ are all reoptimized in the context of each $k'$ that we try to add

3. ...

• When/how do we stop?
Wrapper example: regularization

\[ P(y = 1|x, w) = g(w_0 + w_1 \phi_1(x) + \ldots + w_m \phi_m(x)) \]

- We can introduce a regularization penalty that tries to set the weights to zero unless they are “useful”

\[ J(w; C) = \sum_{t=1}^{n} \log P(y_t|x_t, w) - C \sum_{i=1}^{m} |w_i| \]

where \{ (x_1, y_1), \ldots, (x_n, y_n) \} is our training set. Note that \( w_0 \) is not penalized.

- The selection of non-zero weights here is carried out *jointly*, not individually

- Why should this regularization penalty work at all?
Wrapper example: regularization cont’d

- The effect of the regularization penalty on feature selection depends on its derivative at $w \approx 0$

\[ J(w; C) = \sum_{t=1}^{n} \log P(y_t|x_t, w) - C \sum_{i=1}^{m} |w_i| \]

- How are we dealing with redundant features?
Topics

- Combining multiple methods
  - greedy sequential fitting
  - voting methods: bagging, boosting
Combination of multiple methods

- Why would we want to generate and combine multiple methods rather than use a single method?
  - decomposition into simpler subproblems, modularity
  - multiple “weak” methods can be combined into a single “strong” method
  - robustness

- We have to
  - estimate the component methods in a modular way
  - find an appropriate combination rule
  - worry about generalization
Combination of regression methods

- We want to combine multiple “weak” regression methods into a single “strong” method.

- Suppose we are given a training set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and a family of simple regression methods (components) such as
  \[ f(x; \theta) = w \phi_k(x) \]
  where $\theta = \{k, w\}$ (the parameters specify a single basis function as well as the associated weight).

- Basic forward fitting idea: sequentially fit new components to the residuals.

  Step 1: $\hat{\theta}_1 \leftarrow \arg\min_{\theta} \sum_{i=1}^{n} (y_i - f(x_i; \theta))^2$

  Step 2: $\hat{\theta}_2 \leftarrow \arg\min_{\theta} \sum_{i=1}^{n} (y_i - f(x_i; \hat{\theta}_1) - f(x_i; \theta))^2$

  Step 3: $\ldots$
Forward fitting cont’d

Simple family: \( f(x; \theta) = w\phi_k(x) \), \( \theta = \{k, w\} \)

Step 1: \( \hat{\theta}_1 \leftarrow \arg\min_\theta \sum_{i=1}^{n} (y_i - f(x_i; \theta))^2 \)

Step 2: \( \hat{\theta}_2 \leftarrow \arg\min_\theta \sum_{i=1}^{n} (y_i - f(x_i; \hat{\theta}_1) - f(x_i; \theta))^2 \)

Step 3: …

- The resulting combined regression method

\[
\hat{f}(x) = f(x; \hat{\theta}_1) + \ldots + f(x; \hat{\theta}_m)
\]

has much lower (training) error.

- How many components? Reuse?
Combination of classifiers

- Suppose we are given a training set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of examples and $(\pm 1)$ labels and a family of component classifiers such as decision stumps:

$$h(x; \theta) = \text{sign}(w_1 x_k - w_0)$$

where $\theta = \{k, w_1, w_0\}$.

Each decision stump pays attention to only a single component of the input vector.
Bagging

- We can combine classifiers to ensure more robust predictions (classifications)

- Given a set of \( n \) training examples and labels, repeat
  1. resample (with replacement) a smaller training set of \( n' < n \) examples
  2. train a new classifier (decision stump) \( h(x; \hat{\theta}) \) based on the smaller training set

- The resulting combined classifier is obtained by voting

\[
\hat{h}(x) = \text{sign} \left( \frac{1}{m} \sum_{k=1}^{m} h(x; \hat{\theta}_k) \right)
\]
Beyond Bagging: reweighting training examples

- The component classifiers should concentrate more on training examples that are difficult to classify correctly
- We can tune the classifiers towards harder examples by reweighting the training examples (small margin ⇒ large weight)

**Example:** suppose we already have \( h(x; \hat{\theta}_1), \ldots, h(x; \hat{\theta}_m) \). We train the next component classifier \( h(x; \theta_{m+1}) \) on a reweighted training set

\[
\text{Weight } p(i) \text{ on } (x_i, y_i): \quad p(i) \propto \exp \left\{ - y_i \sum_{k=1}^{m} h(x_i; \hat{\theta}_k) \right\}
\]

where examples with small or negative classification margins (difficult examples) will have larger weights
Boosting

- A Boosting algorithm sequentially estimates and combines classifiers by reweighting training sets (concentrating on the harder examples)
  - each component classifiers is presented with a slightly different problem
- AdaBoost preliminaries:
  a) Training set \((x_1, y_1), \ldots, (x_n, y_n)\) with binary \(\pm 1\) labels \(y_i\).
  b) A set of “weak” binary \((\pm 1)\) classifiers \(h(x; \theta)\) such as decision stumps

\[
h(x; \theta) = \text{sign}(w_1 x_k - w_0)
\]

where \(\theta = \{k, w_1, w_0\}\).
  c) Initially all weights are equal: \(p(i) = 1/n\).