CPSC 340: Machine Learning and Data Mining

Linear Classifiers
Fall 2017
Admin

• Assignment 0+1:
  – Looked into remaining grade anomalies.

• Assignment 0+1:
  – Grades posted.

• Assignment 3:
  – Due Friday of next week (shorter, sorry about A2 length + midterm date).

• Midterm:
  – Can view your exam during instructor office hours next week, or after class this/next week.
Last Time: L1-Regularization

- We discussed **L1-regularization**:
  
  \[ f(w) = \frac{1}{2} \| Xw - y \|^2 + \lambda \| w \|_1 \]

  - Also known as “LASSO” and “basis pursuit denoising”.
  - Regularizes ‘w’ so we decrease our test error (like L2-regularization).
  - Yields sparse ‘w’ so it selects features (like L0-regularization).

- Properties:
  - It’s **convex and fast** to minimize (with “proximal-gradient” methods).
  - Solution is **not unique** (sometimes people do L2- and L1-regularization).
  - Usually includes “correct” variables but tends to yield **false positives**.
Ensemble Feature Selection

• In this case of L1-regularization, we want to reduce false positives.
  – Unlike L0-regularization, the non-zero $w_j$ are still “shrunk”.
    • “Irrelevant” variables are included, before “relevant” $w_j$ reach best value.

• We can also use ensemble methods for feature selection.
  – Usually designed to reduce false positives or reduce false negatives.

• A bootstrap approach to reducing false positives:
  – Apply the method to bootstrap samples of the training data.
  – Only take the features selected in all bootstrap samples.
Example: boostrapping plus L1-regularization ("BoLASSO").

- Reduces false positives.
- It’s possible to show it recovers “correct” variables with weaker conditions.
Part 3 Key Ideas: Linear Models, Least Squares

• Focus of Part 3 is **linear models**:  
  – Supervised learning where prediction is **linear combination of features**:

  \[ y_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id} \]

  \[ = w^T x_i \]

• Regression:
  – Target \( y_i \) is numerical, testing \( \hat{y}_i = y_i \) doesn’t make sense.

• Squared error:

  \[ \frac{1}{2} \sum_{i=1}^{n} (w^T x_i - y_i)^2 \quad \text{or} \quad \frac{1}{2} \| Xw - y \|^2 \]

  – Can find optimal ‘w’ by solving “normal equations”.

---

**Good fit that doesn’t exactly pass through any point.**
Part 3 Key Ideas: Gradient Descent, Error Functions

• For large ‘d’ we often use gradient descent:
  – Iterations only cost $O(nd)$.
  – Converges to a critical point of a smooth function.
  – For convex functions, it finds a global optimum.

• $L_1$-norm and $L_\infty$-norm errors:
  \[ ||\hat{x}_n - y||_1 \quad ||\hat{x}_n - y||_{\infty} \]
  – More/less robust to outliers.
  – Can apply gradient descent after smoothing with Huber or log-sum-exp.
Part 3 Key Ideas: Change of basis, Complexity Scores

• **Change of basis**: replaces features \( x_i \) with non-linear transforms \( z_i \):
  – Add a **bias variable** (feature that is always one).
  – **Polynomial basis**.
  – **Radial basis functions** (non-parametric basis).

• We discussed scores for choosing “true” model complexity.
  – Validation score vs. AIC/BIC.

• **Search and score** for feature selection:
  – Define a “score” like BIC, and do a “search” like **forward selection**.
Part 3 Key Ideas: Regularization

• **L0-regularization** (AIC, BIC):
  – Adds penalty on the number of non-zeros to select features.
    \[ f(w) = ||Xw - y||^2 + \lambda ||w||_0 \]

• **L2-regularization** (ridge regression):
  – Adding penalty on the L2-norm of ‘w’ to decrease overfitting:
    \[ f(w) = ||Xw - y||^2 + \frac{\lambda}{2} ||w||^2 \]

• **L1-regularization** (LASSO):
  – Adding penalty on the L1-norm decreases overfitting and selects features:
    \[ f(w) = ||Xw - y||^2 + \lambda ||w||_1 \]
Key Idea in Rest of Part 3

• The next few lectures will focus on:
  – Using linear models for classification and with discrete features.
  – Using linear models with really big datasets.
  – Connections between regression and probabilities.

• It may seem like we’re spending a lot of time on linear models.
  – Linear models are used a lot and are understandable.
    • ICBC only uses linear models for insurance estimates.
  – Linear models are also the building blocks for more-advanced methods.
    • “Latent-factor” models in Part 4 and “deep learning” in Part 5.
Motivation: Identifying Important E-mails

• How can we automatically identify ‘important’ e-mails?
  
  • A binary classification problem (“important” vs. “not important”).
    – Labels are approximated by whether you took an “action” based on mail.
    – High-dimensional feature set (that we’ll discuss later).

• Gmail uses regression for this binary classification problem.
Binary Classification Using Regression?

• Can we apply linear models for binary classification?
  – Set $y_i = +1$ for one class (“important”).
  – Set $y_i = -1$ for the other class (“not important”).

• At training time, fit a linear regression model:

  $y_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id}$

  $= w^T x_i$

• The model will try to make $w^T x_i = +1$ for “important” e-mails, and $w^T x_i = -1$ for “not important” e-mails.
Binary Classification Using Regression?

• Can we apply linear models for **binary classification**?
  – Set $y_i = +1$ for one class ("important").
  – Set $y_i = -1$ for the other class ("not important").

• **Linear model gives real numbers** like 0.9, -1.1, and so on.

• So to predict, we look at the **sign of** $w^T x_i$.
  – If $w^T x_i = 0.9$, predict $y_i = +1$.
  – If $w^T x_i = -1.1$, predict $y_i = -1$.
  – If $w^T x_i = 0.1$, predict $y_i = +1$.
  – If $w^T x_i = -100$, predict $y_i = -1$. 
Decision Boundary in 1D

\[ y_i = w^T x_i \]
Decision Boundary in 1D

We can interpret ‘w’ as hyperplane separating x into 2 half-spaces:
- Half-space where $w^T x_i > 0$ and half-space where $w^T x_i < 0$. 
A linear classifier would be linear function $y = w_0 + w_1 x_1 + w_2 x_2$ coming out of the page (the boundary is at $y=0$).

Or recall from multivariable calculus that a plane in d-dimensions is defined by its normal vector in d-dimensions, plus an intercept/offset.
Perceptron Algorithm

• One of the first “learning” algorithms was the “perceptron” (1957).
  – Searches for a ‘w’ such that sign(wᵀxᵢ) = yᵢ for all i.

• Perceptron algorithm:
  – Start with w⁰ = 0.
  – Go through examples in any order until you make a mistake predicting yᵢ.
    • Set wᵗ₊₁ = wᵗ + yᵢxᵢ.
  – Keep going through examples until you make no errors on training data.

• Intuition for step: if yᵢ = +1, “add more of xᵢ to w” so that wᵀxᵢ is larger.
  \[(w^{t+1})^T x_i = (w^t + x_i)^T x_i = (w^t)^T x_i + x_i^T x_i = \text{(old prediction)} + ||x_i||^2\]

• If a perfect classifier exists, this algorithm finds one in finite number of steps.
  – In this case we say the training data is “linearly separable”
The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used 20x20 cadmium sulfide photocells to produce a 400-pixel image. The main visible feature is a patchboard that allowed experimentation with different combinations of input features. To the right of that are arrays of potentiometers that implemented the adaptive weights.\footnote{213}
Can we just use least squares??

• Consider training by minimizing squared error with these $y_i$:

$$\ell(w) = \frac{1}{2} \| Xw - y \|^2$$

• If we predict $w^T x_i = +0.9$ and $y_i = +1$, error is small: $(0.9 - 1)^2 = 0.01$.
• If we predict $w^T x_i = -0.8$ and $y_i = +1$, error is big: $(-0.8 - 1)^2 = 3.24$.
• If we predict $w^T x_i = +100$ and $y_i = +1$, error is huge: $(100 - 1)^2 = 9801$.

• Least squares penalized for being “too right”.
  – +100 has the right sign, so the error should be zero.
Can we just use least squares??

• Least squares behaves weirdly when applied to classification:

![Diagram showing a linear regression model and the actual data points.]

- This is the linear regression model we want (a perfect classifier).
- This is what we actually get.

• Make sure you understand why the green line achieves 0 training error.
Can we just use least squares??

• What went wrong?
  – “Good” errors vs. “bad” errors.
Can we just use least squares??

- What went wrong?
  - “Good” errors vs. “bad” errors.

\[ f(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2 \]

What happens if \( y_i = -1 \) and \( w^T x_i = -1000 \)?

This is the linear regression model we want (a perfect classifier).

#time> we See "vicodin"\n
"Bad" errors of the perfect linear classifier are HUGE.
Comparing Loss Functions

- $(w^T x_i - y_i)^2$
- "Error" or "loss" for predicting $w^T x_i$ when true label $y_i$ is $-1$.
- Big penalty for being "too right".
- "Bad" error: you should not penalize for putting $w^T x_i$ here.
- "Good" error: having $w^T x_i$ here is bad.
Thoughts on the previous (and next) slide

• We are now plotting the loss vs. the predicted $\mathbf{w}^\top \mathbf{x}_i$.
  – This is totally different from plotting in the data space ($y$ vs. $x$).

• The loss is a sum over training examples.
  – We're plotting the individual loss for a particular training example.
  – In the figure, this example has label $y_i = -1$ so the loss is centered at -1.
    (The plot would be mirrored in the case of $y_i = +1$.)
    • We only need to show one case or the other to get our point across.
  – Note that with regular linear regression the output $y_i$ could be any number and thus the parabola could be centred anywhere. But here we've restricted ourselves to $y_i=\{-1,+1\}$.

• (The next slide is the same as the previous one)
Comparing Loss Functions

\[(w^T x_i - y_i)^2\]

"Error" or "loss" for predicting \(w^T x_i\) when true label \(y_i\) is \(-1\).

Big penalty for being "too right".

"bad" error: you should not penalize for putting \(w^T x_i\) here.

"good" error: having \(w^T x_i\) here is bad.
Comparing Loss Functions

$\left( w^T x_i - y_i \right)^2$

"Error" or "loss" for predicting $w^T x_i$ when true label $y_i$ is $-1$.

Big penalty for being "too right".

Absolute error reduces but does not fix this issue.

"Bad" error: you should not penalize for putting $w^T x_i$ here.

"Good" error: having $w^T x_i$ here is bad.
Comparing Loss Functions

\[(w^T x_i - y_i)^2\]

"Error" or "loss" for predicting \(w^T x_i\) when true label \(y_i\) is -1.

Big penalty for being "too right".

Absolute error reduces but does not fix this issue.

"bad" error: you should not penalize for putting \(w^T x_i\) here.

"good" error: having \(w^T x_i\) here is bad.

What we want is the "0-1 loss".
0-1 Loss Function

• The **0-1 loss function** is the number of classification errors:
  – We can write using the L0-norm as $||\text{sign}(Xw) - y||_0$.
  – Unlike regression, in classification it’s reasonable that $\text{sign}(w^Tx_i) = y_i$.

• Unfortunately the **0-1 loss is non-convex** in ‘$w$’.
  – It’s easy to minimize if a perfect classifier exists (perceptron).
  – Otherwise, finding the ‘$w$’ minimizing 0-1 loss is a hard problem.
    – Gradient is zero everywhere so you don’t know “which way to go” in $w$-space.
    – Note this is NOT the same type of problem we had with using the squared loss.
      • We can minimize the squared error, it might giver a bad model.

• Next lecture we’ll introduce a **convex approximation to the 0-1 loss**.
Summary

• **Ensemble feature selection** reduces false positives or negatives.
• **Binary classification using regression:**
  – Encode using $y_i$ in {-1,1}.
  – Use $\text{sign}(w^T x_i)$ as prediction.
  – “Linear classifier” (a hyperplane splitting the space in half).
• **Perceptron algorithm:** finds a perfect classifier (if one exists).
• Least squares is a weird error for classification.
• **0-1 loss** is the ideal loss, but is non-smooth and non-convex.

• Next time: one of the best “out of the box” classifiers.
L1-Regularization as a Feature Selection Method

• Advantages:
  – Deals with conditional independence (if linear).
  – Sort of deals with collinearity:
    • Picks at least one of “mom” and “mom2”.
  – Very fast with specialized algorithms.

• Disadvantages:
  – Tends to give false positives (selects too many variables).

• Neither good nor bad:
  – Does not take small effects.
  – Says “gender” is relevant if we know “baby”.
  – Good for prediction if we want fast training and don’t care about having some irrelevant variables included.
“Elastic Net”: L2- and L1-Regularization

• To address non-uniqueness, some authors use L2- and L1-:

\[
\hat{\beta}(w) = \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda_2}{2} \|w\|^2 + \lambda_1 \|w\|_1
\]

• Called “elastic net” regularization.
  – Solution is sparse and unique.
  – Slightly better with feature dependence:
    • Selects both “mom” and “mom2”.

• Optimization is easier though still non-differentiable.
L1-Regularization Debiasing and Filtering

• To remove false positives, some authors add a debiasing step:
  – Fit ‘w’ using L1-regularization.
  – Grab the non-zero values of ‘w’ as the “relevant” variables.
  – Re-fit relevant ‘w’ using least squares or L2-regularized least squares.

• A related use of L1-regularization is as a filtering method:
  – Fit ‘w’ using L1-regularization.
  – Grab the non-zero values of ‘w’ as the “relevant” variables.
  – Run standard (slow) variable selection restricted to relevant variables.
    • Forward selection, exhaustive search, stochastic local search, etc.
Non-Convex Regularizers

- Regularizing $|w_j|^2$ selects all features.
- Regularizing $|w_j|$ selects fewer, but still has many false positives.
- What if we regularize $|w_j|^{1/2}$ instead?
  - Minimizing this objective would lead to fewer false positives.
    - Less need for debiasing, but it’s not convex and hard to minimize.
- There are many non-convex regularizers with similar properties.
  - L1-regularization is (basically) the “most sparse” convex regularizer.
Online Classification with Perceptron

- **Perceptron for online linear binary classification** [Rosenblatt, 1957]:
  - Start with \( w_0 = 0 \).
  - At time ‘t’ we receive features \( x_t \).
  - We predict \( \hat{y}_t = \text{sign}(w_t^T x_t) \).
  - If \( \hat{y}_t \neq y_t \), then set \( w_{t+1} = w_t + y_t x_t \).
    - Otherwise, set \( w_{t+1} = w_t \).  

(Slides are old so above I’m using subscripts of ‘t’ instead of superscripts.)

- **Perceptron mistake bound** [Novikoff, 1962]:
  - Assume data is *linearly-separable* with a “margin”:
    - There exists \( w^* \) with \( ||w^*|| = 1 \) such that \( \text{sign}(x_t^T w^*) = \text{sign}(y_t) \) for all ‘t’ and \( |x_t^T w^*| \geq \gamma \).
  - Then the **number of total mistakes is bounded**.
    - No requirement that data is IID.
Perceptron Mistake Bound

• Let’s **normalize each** \( x_t \) so that \( ||x_t|| = 1 \).
  – Length doesn’t change label.

• Whenever we make a mistake, we have \( \text{sign}(y_t) \neq \text{sign}(w_t^T x_t) \) and

\[
||w_{t+1}||^2 = ||w_t + y x_t||^2 = ||w_t||^2 + 2 y_t w_t^T x_t + 1 \leq ||w_t||^2 + 1 \leq ||w_{t-1}||^2 + 2 \leq ||w_{t-2}||^2 + 3.
\]

• So after ‘k’ errors we have \( ||w_t||^2 \leq k. \)
Perceptron Mistake Bound

• Let’s consider a solution $w^*$, so $\text{sign}(y_t) = \text{sign}(x_t^Tw^*)$.
• Whenever we make a mistake, we have:

$$||w_{t+1}|| = ||w_{t+1}|| ||w_*||$$
$$\geq w^T_{t+1}w_*$$
$$= (w_t + y_tx_t)^T w_*$$
$$= w^T_t w_* + y tx^T_t w_*$$
$$= w^T_t w_* + |x^T_t w_*|$$
$$\geq w^T_t w_* + \gamma.$$

• So after ‘$k$’ mistakes we have $| |w_t| | \geq \gamma k$. 
Perceptron Mistake Bound

• So our two bounds are $||w_t|| \leq \sqrt{k}$ and $||w_t|| \geq \gamma k$.

• This gives $\gamma k \leq \sqrt{k}$, or a maximum of $1/\gamma^2$ mistakes.
  – Note that $\gamma > 0$ by assumption and is upper-bounded by one by $||x|| \leq 1$.
  – After this ‘$k$’, under our assumptions we’re guaranteed to have a perfect classifier.