Bias/Variance Trade-off, Some Practical Issues, Semi-supervised and Active Learning

Piyush Rai

Introduction to Machine Learning (CS771A)

November 13, 2018

Plan for today

- Bias/Variance Trade-off of Learning Algorithms
- Some basic guidelines for debugging ML algorithms
- What if training data and test data distributions are NOT the same
- How to utilize unlabeled data: Semi-supervised Learning
- How to selectively ask for the most informative data: Active Learning

Bias/Variance Trade-off

A Fundamental Decomposition

- Assume $\mathcal{F}$ to be a class of models (e.g., set of linear classifiers with some pre-defined features)
- Suppose we've learned a model $f \in \mathcal{F}$ learned using some (finite amount of) training data
- Can decompose the test error $\epsilon(f)$ of f as follows

$$\epsilon(f) = \underbrace{\left[\min_{f^* \in \mathcal{F}} \epsilon(f^*)\right]}_{\text{approximation error}} + \underbrace{\left[\epsilon(f) - \min_{f^* \in \mathcal{F}} \epsilon(f^*)\right]}_{\text{estimation error}}$$

- Here f^* is the best possible model in $\mathcal{F}$ assuming infinite amount of training data
- Approximation error: Error of best model f^* because of the model class $\mathcal{F}$ being too simple
- Estimation error: Error of learned model f (relative to f^*) because we only had finite training data

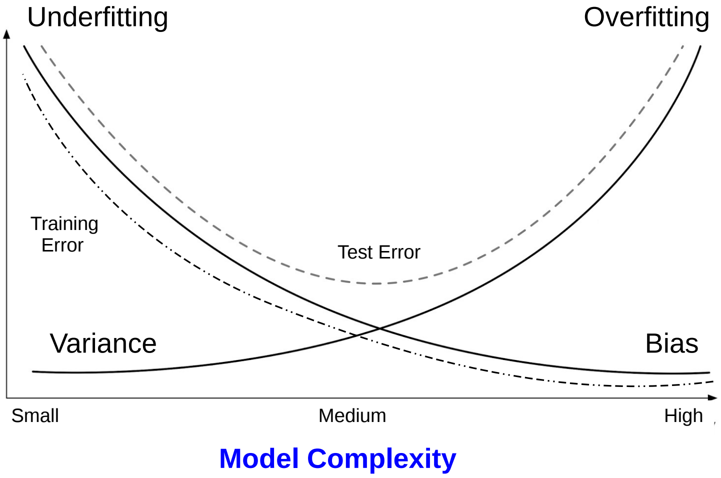
A Trade-off

- There is a trade-off between the two components of the error

$$\epsilon(f) = \underbrace{\left[\min_{f^* \in \mathcal{F}} \epsilon(f^*)\right]}_{\text{approximation error}} + \underbrace{\left[\epsilon(f) - \min_{f^* \in \mathcal{F}} \epsilon(f^*)\right]}_{\text{estimation error}}$$

- Can reduce the approximation error by making $\mathcal{F}$ more complex/richer, e.g.,
  - Replace the linear classifiers by a higher order polynomials
  - Add more features in the data
- However, making $\mathcal{F}$ richer will usually cause estimation error to increase since f can now overfit
  - f may give small error on some datasets, large error on some datasets
- Note: The above trade-off is popularly known as the bias/variance trade-off
  - Approximation error known as "bias" (high if the model is simple)
  - Estimation error known as "variance" (high if the model is complex)

A Visual Illustration of the Trade-off



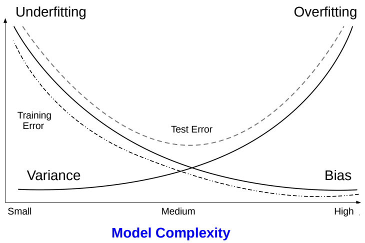
Debugging ML Algorithms

Debugging Learning Algorithms

- A notoriously hard problem in general
    - Note that code for ML algorithms is not procedural but data-driven
- What to do when our model (say logistic regression) isn't doing well on test data
    - Use more training examples to train the model?
    - Use a smaller number of features?
    - Introduce new features (can be combinations of existing features)?
    - Try tuning the regularization parameter?
    - Run (the iterative) optimizer longer, i.e., for more iterations?
    - Change the optimization algorithm (e.g., GD to SGD or Newton..)?
    - Give up and switch to a different model (e.g., SVM)?
- How to know what might be going wrong and how to debug?

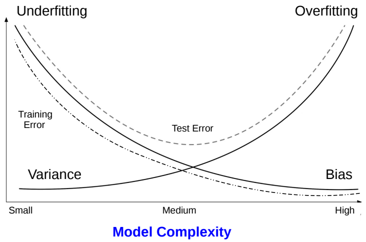
High Bias or High Variance?

- The bad performance (low accuracy on test data) could be due either
  - High Bias (Underfitting)
  - High Variance (Overfitting)
- Looking at the training and test error can tell which of the two is the case



- High Bias: Both training and test errors are large
- High Variance: Small training error, large test error (and huge gap)

Some Guidelines for Debugging Learning Algorithms

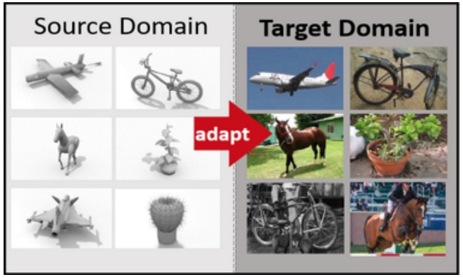


- If the model has high bias (underfitting) then the model class is weak
  - Adding more training data won't help
  - Instead, try making the model class richer, or add more features (makes the model richer)
- If the model has high variance (overfitting) then the model class is sufficiently rich
  - Adding more training data can help (overfitting might be due to limited training data)
  - Using a simpler model class or fewer features or regularization can help

When Train and Test Distributions Differ

When Train and Test Conditions are Different..

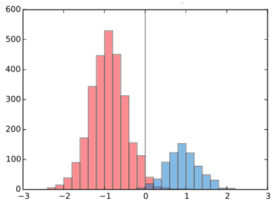
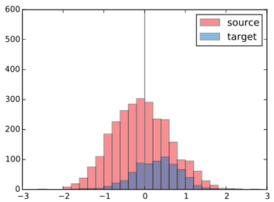
- Training and test inputs typically assumed to be drawn from same distribution, i.e., $p_{tr}(\boldsymbol{x}) = p_{te}(\boldsymbol{x})$
- However, this is rarely true in real-world applications of ML where $p_{tr}(\boldsymbol{x}) \neq p_{te}(\boldsymbol{x})$



- So the training and test data are from different "domains". How can we "adapt"?
- Known as "domain adaptation" (also "covariate-shift", since features $\boldsymbol{x}$ are also called covariates)

Handling Domain Differences: Some Approaches

- Learning a transformation that makes the training and test data distributions "close"

- Use an "importance weighted" correction of the loss function defined on the training data

$$\sum_{n=1}^{N} \frac{p_{te}(\boldsymbol{x}_n)}{p_{tr}(\boldsymbol{x}_n)} \ell(y_n, f(\boldsymbol{x}_n))$$

.. basically, assign a weight to each term of the loss function, depending on how "close" the training example $(\boldsymbol{x}_n, y_n)$ is from the test distribution

Importance-Weighted Correction: A Closer Look

- Note that, since we care about test data, ideally, we would like to learn f by minimizing

$$\mathbb{E}_{p_{te}(\boldsymbol{x})}[\ell(y, f(\boldsymbol{x}))] = \int p_{te}(\boldsymbol{x})\ell(y, f(\boldsymbol{x}))d\boldsymbol{x}$$

- However, we don't have labeled data from the test domain. :-(What can we do now?
- Let's re-express the above in terms of the loss on training data

$$\begin{aligned}
\int p_{te}(\boldsymbol{x})\ell(y, f(\boldsymbol{x}))d\boldsymbol{x} &= \int p_{tr}(\boldsymbol{x})\frac{p_{te}(\boldsymbol{x})}{p_{tr}(\boldsymbol{x})}\ell(y, f(\boldsymbol{x}))d\boldsymbol{x} \\
&= \mathbb{E}_{p_{tr}(\boldsymbol{x})}\left[\frac{p_{te}(\boldsymbol{x})}{p_{tr}(\boldsymbol{x})}\ell(y, f(\boldsymbol{x}))\right] \\
&\approx \sum_{n=1}^{N} \frac{p_{te}(\boldsymbol{x}_n)}{p_{tr}(\boldsymbol{x}_n)}\ell(y_n, f(\boldsymbol{x}_n)) \qquad \text{(uses only training data from } p_{tr}\text{)}
\end{aligned}$$

- In order to do this, we need to estimate the two densities $p_{tr}(\boldsymbol{x})$ and $p_{te}(\boldsymbol{x})$ (can assume some form for these and estimate them using unlabeled data from both domains)

Semi-supervised Learning

Labeled vs Unlabeled Data

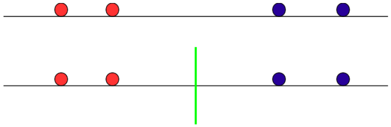
- Supervised Learning models require labeled data
- Learning a reliable model usually requires plenty of labeled data
- Labeled Data: Expensive and Scarce (someone has to do the labeling)
  - Often labeling is very difficult too (e.g., in speech analysis or NLP problems)
- Unlabeled Data: Abundant and Free/Cheap
  - E.g., can easily crawl the web and download webpages/images

Learning with Labeled+Unlabeled Data

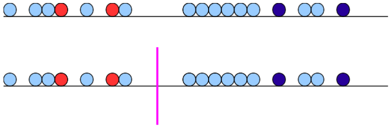
- Usually such problems come in one of the following two flavors
- Semi-supervised Learning
  - Training set containslabeled data $\mathcal{L} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{L}$ and unlabeled data $\mathcal{U} = \{\boldsymbol{x}_j\}_{j=L+1}^{L+U}$ (usually $U \gg L$)
  - Note: In some cases, $\mathcal{U}$ is also our test data (this is called transductive setting)
  - Inductive setting is more standard: Want to learn a classifier for future test data
- Semi-Unsupervised Learning
  - We are given unlabeled data $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}$
  - Additionally, we're given supervision in form of some constraints on data (e.g., points $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ belong to the same cluster) or "labels" of some points.
  - Want to learn an unsupervised learning model combining both data sources
- Here, we will focus on Semi-supervised Learning (SSL)

Why/How Might Unlabeled Data Help?

- Red: + 1, Dark Blue: -1

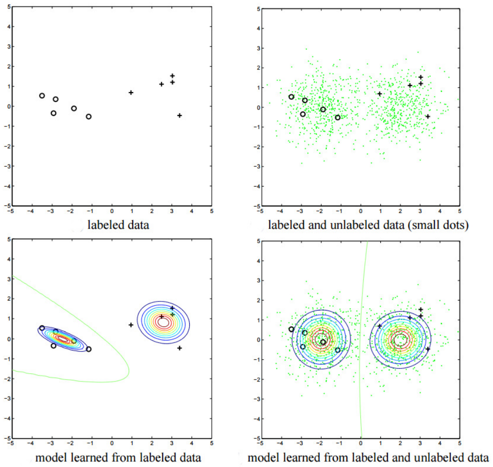


- Let's include some additional unlabeled points (Light Blue points)



- Assumption: Examples from the same class are clustered together
- Assumption: Decision boundary lies in the region where data has low density

Why/How Might Unlabeled Data Help?



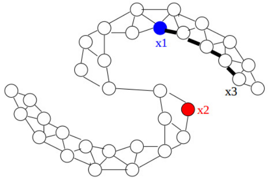
Knowledge of the input distribution helps even if the data is unlabeled

Some Basic Assumptions used in SSL

- Nearby points (may) have the same label
  - Smoothness assumption
- Points in the same cluster (may) have same label
  - Cluster assumption
- Decision boundary lies in areas where data has low density
  - Low-density separation
- Note: Generative models already take some of these assumptions into account (recall HW3 problem on semi-supervised learning using EM)

SSL using Graph-based Regularization

- Based on smoothness assumption
- Requires constructing a graph between all the examples (labeled/unlabeled)



- The graph can be constructed using several ways
  - Connecting every example with its top k neighbors (labeled/unlabeled)
  - Constructing an all-connected weighted graph
  - Weighted case: weight of edge connecting examples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$

$$a_{ij} = \exp(-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/\sigma^2)$$

.. where $\mathbf{A}$ is the $(L+U)\times(L+U)$ matrix of pairwise similarities

SSL using Graph-based Regularization

- Suppose we want to learn a function f using labeled+unlabeld data $\mathcal{L} \cup \mathcal{U}$
- Suppose f_i denotes the prediction on example $\boldsymbol{x}_i$
    - Similar examples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ (thus high a_{ij}) should have similar f_i and f_j
    - Graph-based regularization optimizes the following objective:

$$\min_{\boldsymbol{f}} \underbrace{\sum_{i \in \mathcal{L}} \ell(y_i, f_i)}_{\text{loss term}} + \underbrace{\lambda R(f)}_{\text{usual regularizer}} + \underbrace{\gamma \sum_{i,j \in \mathcal{L},\mathcal{U}} a_{ij}(f_i - f_j)^2}_{\text{graph-based regularizer}}$$

SSL using EM based Generative Classification

- Suppose data $\mathcal{D}$ is labeled $\mathcal{L} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{L}$ and unlabeled $\mathcal{U} = \{\boldsymbol{x}_j\}_{j=L+1}^{L+U}$
- Assume the following model

$$p(\mathcal{D}|\theta) = \underbrace{\prod_{i=1}^{L} p(\boldsymbol{x}_i, y_i|\theta)}_{\text{labeled}} \underbrace{\prod_{j=L+1}^{L+U} p(\boldsymbol{x}_j|\theta)}_{\text{unlabeled}} = \prod_{i=1}^{L} p(\boldsymbol{x}_i, y_i|\theta) \prod_{j=L+1}^{L+U} \sum_{y_j} p(\boldsymbol{x}_j, y_j|\theta)$$

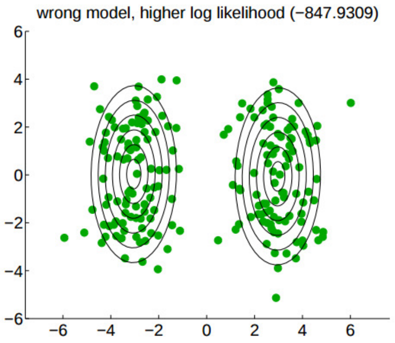
- The unknowns are $\{y_j\}_{j=L+1}^{L+U}$ (latent variables) and θ (parameters)
- We can use EM to estimate the unknowns
  - Given $\theta = \hat{\theta}$, E step will compute expected labels for unlabeled examples

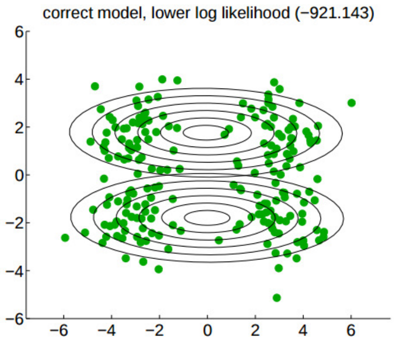
$$\mathbb{E}[y_j] = +1 \times P(y_j = +1|\hat{\theta}, \boldsymbol{x}_j) + (-1) \times P(y_j = -1|\hat{\theta}, \boldsymbol{x}_j)$$

  - M step can then perform standard MLE for re-estimating the parameters θ
- A fairly general framework for semi-supervised learning. Can be used for different types of data (by choosing the appropriate $p(\boldsymbol{x}|y)$ distribution)

Things can go wrong..

If assumptions are not appropriate for the data (e.g., incorrectly specified class conditional distributions)



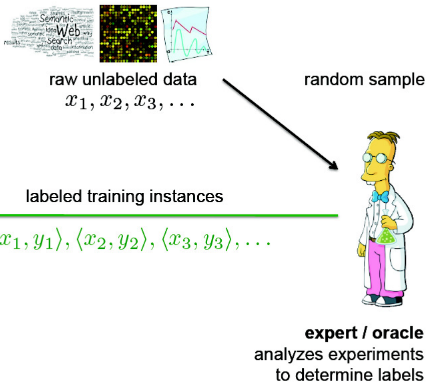


Thus need to be careful/flexible about the choice of class conditional distributions

Active Learning

Passive vs Active Learning

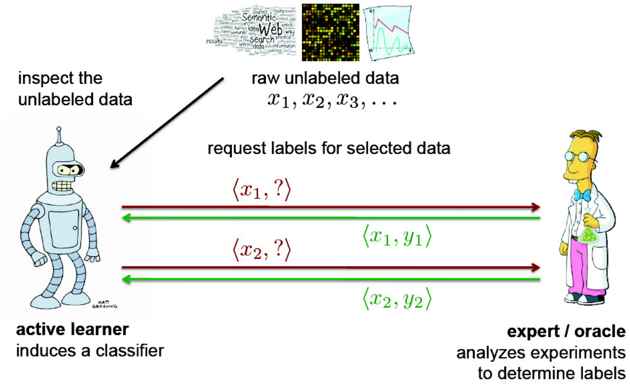
Standard supervised learning assumes a “passive” learner



Called “passive” because the learner doesn’t have any control over the labeled data it gets to learn

Active Learning

An "active" learner can specifically request labels of "hard" examples that are most useful for learning

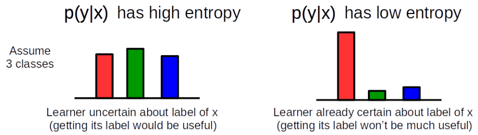


raw unlabeled data
$x_1, x_2, x_3, \ldots$
inspect the unlabeled data
request labels for selected data
$\langle x_1, ? \rangle$
$\langle x_1, y_1 \rangle$
$\langle x_2, ? \rangle$
$\langle x_2, y_2 \rangle$
active learner
induces a classifier
expert / oracle
analyzes experiments to determine labels

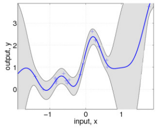
How to quantify the "hardness" or "usefulness" of an unlabeled example?

Active Learning

- Various ways to define the usefulness/hardness of an unlabeled example $\boldsymbol{x}$
- A probabilistic approach can use distribution of y given $\boldsymbol{x}$ to quantify usefulness of $\boldsymbol{x}$, e.g.,
  - In classification, can look at the entropy of the distribution $p(y|\boldsymbol{x},\boldsymbol{w})$ or posterior predictive $p(y|\boldsymbol{x})$



    .. and choose example(s) for which the distribution of y given $\boldsymbol{x}$ has the largest entropy
  - Look at the variance of the posterior predictive $p(y|\boldsymbol{x})$. E.g. for regression,



    .. and choose choose example(s) for which variance of posterior predictive is the largest
  - .. many other ways to utilize the information in $p(y|\boldsymbol{x})$