Machine Learning
Outline

- A bit of vocabulary
- Example: Linear Regression
- Types of learning algorithms
- A few random points
- Some problems and ways to ameliorate them
- Resources & References
- A deeper look at a few algorithms
Preliminaries

- **Labels**
  - aka Targets or Classes
  - e.g. Topic, Discourse Relation, Part of Speech

- **Features**
  - Describe data
  - Feature values might be binary, categorical, real valued
  - **Feature Vector**
    - An indexed list of features defining an example
    - Very machine friendly
    - e.g. for topic labeling a document, the feature vector may include a list of binary values indicating whether or not words occurred in the document
  - **Feature interactions**
    - If features are not independent we have interactions
    - Sometimes defined as an entirely new feature
    - In predicting hospital stays the composite feature ("age" + "sex") is much more powerful than the individual features "age" and "sex"

- **Typical Learning Algorithm**
  - Attempts to find a function which maps features to labels.
    - \( \Phi(\{\text{features}\}) \rightarrow \text{Label} \)
  - Uses training examples to tune the function

- **Test, Training, and Development Sets**
  - **Training set**: used to build a model
  - **Development/Validation set**: used to evaluate model and suggest tweaks. Can be treated as a part of the training set not used for actually building the model
  - **Test set**: used for final evaluation. In an ideal world should not be touched or even looked at before the final evaluation.
  - Using the test set for training or parameter tuning is very bad and will frequently result in grossly overestimating the effectiveness of your algorithm. Using it as a development set has similar effect.
  - Generally, examples in each set should be sampled from the same population and all examples should be independent and identically distributed (IID)
  - **n-fold cross-validation** involves repeated splitting the training set into training and validation sets. (will elaborate later)
Example: Linear Regression

- Assume the output is a linear combination of the features with some offset

\[ y = \lambda_0 + \lambda_1 x_1 + \lambda_2 x_2 + \ldots + \lambda_n x_n = \sum_{i=1}^{n} \lambda_i x_i = \Lambda \cdot X \]

- As a feature's value increases, the model's output always goes up if \( \lambda > 0 \) or down if \( \lambda < 0 \). Never up to a point, then down.
- The values of \( \lambda \) encode how much influence each feature has and in what direction.
- It is common to normalize the features using Z-Values (aka Z-Scores, Standard Scores, etc.)
  - \( z = (x - \mu)/\sigma \)
  - Especially useful when you try to make sense of the model.
Example: Linear Regression
Example: Linear Regression
Example: Linear Regression
Example: Linear Regression
Some Important Distinctions For Machine Learning Algorithms
Classification and Regression

• Classification
  • Predictions are from a finite set of classes (labels)
  • An important subclass is binary classifiers where the set of classes is of size=2
  • Evaluation metrics include accuracy, precision, recall, F measures

• Regression
  • Predictions are real numbers
  • Often interested in the effect of individual features
  • Evaluation metrics include mean square error
Online and Offline

- **Offline (Batch)**
  - Build model on a fixed training set

- **Online**
  - Build and update model based on a stream of examples
Supervised and Unsupervised

• **Supervised learning**
  - Uses labeled examples to train a model which is able to predict the labels for new (unlabeled) examples
  - Example algorithms include Naive bayes, Decision Trees, Linear Regression, Support Vector Machines (SVMs)

• **Unsupervised learning**
  - Tries to find patterns in unlabeled data
  - Example algorithms include K-Means Clustering, Latent Dirichlet Allocation (LDA)

• **Semi-supervised learning**
  - Uses a relatively small number of labels to seed the algorithm or explain the results
Generative and Discriminative Models

• **Generative Models**
  - Trained model can be used to generate examples similar to training input (in addition to predicting labels given the feature vectors)
  - Attempts to predict the joint probability of label (y) and input (x): $P(y,x)$
  - Examples include n-gram, Naive Bayes, Hidden Markov Models

• **Discriminative Models**
  - Build a non-invertible function.
  - Attempts to predict the conditional probability of label given input: $P(y|x)$
  - Examples include linear regression, SVMs
### Parametric and Non-Parametric

#### Parametric Models
- Models have a finite number of parameters
  - The number of parameters is frequently a function of the number of features
  - e.g. coefficients in linear regression
- Relatively compact models
  - Easy to store on disk or in memory
- Requires training (time)

#### Non-Parametric Models
- Basic idea: Let the data define the model
  - Does not attempt to impose some predefined model or distribution on the problem
- Few assumptions are made
- Sometimes more difficult to interpret
- Requires that training data be kept on-hand (i.e. in memory)
- Requires little to no training time, but applying the model may require more time
A few more random things
Bayes Rule

\[ P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \]
Plate Notation

- Used for describing simple models and relationships
- aka plate model or plate diagram
- Example from Wikipedia’s Latent Dirichlet Allocation (LDA) article
  - N words in M documents with K topics.
  - Arrows indicate influence.
  - Note that the N plate nests in the M plate so there are N*M instances of W and Z (actual words and topics for the words respectively)
  - Words are known and observed (circle filled in)
  - $\alpha$ and $\beta$ are outside of the plates and thus there is exactly one value of each for all documents, topics, and words. These are parameters for Dirichlet distributions.
  - $\theta$ is the distribution over topics for each document. I probably would have let the K plate overlap the M plate and include $\theta$, but I could be wrong?
  - $\phi$ represents the distribution over topics.
- Koller’s PGM Course has a lecture that describes these well (see references)
Evaluation Metrics

- Already saw Accuracy, Precision, Recall, and F

- Sum of square errors
  - Other forms include Root Mean Square Error (RMSE)
  - Useful for regression
  - Margin is how far off the prediction is from the actual value (see figure stolen from PRML)
  - SSE is the sum of the square of the margins.
  - Margins are squared for a few reasons:
    - It makes them always non-negative
    - It weights really bad mistakes more heavily than slightly bad mistakes
    - It is a differentiable convex function which means it is easy to work with and use in certain proofs
  - RMSE is a normalized version of the SSE
Gradient Descent

- Suppose we have some surface and want to find a minimum
- Assume we can compute the value and slope at any given point
- To find the minimum we merely need pick a point, calculate the slope at that point and walk downhill
- Note that convex surfaces have exactly one minimum so if we find it we know we are at the global minimum
- Stochastic Gradient Descent involves using each training example in an Online fashion and calculating the slope relative to the example at hand. The reason for doing this is that the training set may be too huge to calculate the true slope in any reasonable amount of time
- Gradient Descent is used for minimizing error
- Newton's Method (the distracting animation at right) is a quick way of finding the root of the derivative which is the minimum for some other function
- Images from Wikipedia
Receiver Operating Characteristic (ROC) Curves

- Not all that common
- Illustrate tradeoffs between precision and recall
- Requires regression or some confidence measure
- The idea is that you rank predictions based on confidence and sort from most confident to least. You then step through this list plotting the ROC curve as you go. You move up if the prediction is correct and move right if the prediction is wrong
- This yields a monotonically increasing line
- The Upper left corner is good. The diagonal line represents random guessing.
- The bottom left corner indicates you labeled everything negative. This means you never incorrectly labeled something as positive (yay!) but also never correctly labeled something that actually was positive (boo!)
- Conversely the top right corner means you labeled everything positive and thus your True Positive rate is perfect! But your False Positive rate is equally high (bad)
- Sometimes the result skews one way or another.
- One metric for evaluating model performance is the area under the ROC curve (actually the area between the plotted line and the diagonal). More is better.
- Example taken from Wikipedia's article on ROC curves
MapReduce

- A framework for handling many situations in Computer Science but especially useful for ML
- First break your problem into numerous identical pieces. Next apply (map) a function to each piece. And finally combine the results (reduce).
- Encourages the distribution of work for both the map and reduce steps over many nodes or machines.
- Various tools out there such as Hadoop

Highly contrived example:
- Suppose we wanted to gather unigram counts from a large corpus with 100,000,000 documents.
- We could define a function which takes a single document and counts its unigrams.
- We would then Map this function onto the corpus
- The Reduce step would merely add up the results.
Ensembles

- Combine many classifiers to build a really good one
- Very, very powerful!
- Bootstrap Aggregating (bagging) trains each model on a subset of the feature space before combine their predictions
- Boosting (covered later) updates weights assigned to training examples before training the next model
- Random Forests build many decision trees and combine their predictions
- Gradient Boosted Machines try to predict their mistakes on the previous iteration. (A personal favorite at the moment. R has a good implementation)
Issues and Solutions
Overfitting

The Problem:

- Overfitting occurs when you build a model that handles training data very well but fails to perform as well on test data.
- This is frequently because the algorithm begins to model the inherent noise in the data, mistaking it for meaningful information.
- Complex models have more degrees of freedom and are thus more prone to overfitting than simpler models.

Solutions:

- Use regularization to punish complexity.
- Use a validation set to test the model.
- Build an ensemble by combining many models (possibly built using cross-validation).
- Use Priors to indicate what is expected and reasonable.
- Add more data!
Overfitting In Pictures (From: PRML)
Overfitting In Pictures (From: PRML)
HUGE Feature Space

The Problem:

- In NLP we sometimes have millions of features!
- 100,000 words => 100,000 unigram features, up to $10^{10}$ possible bigrams, it gets scary after that...
- Some algorithms do not scale well with the feature space

Solutions:

- Choose features that occur at least n times
  - Crude but simple
  - Other methods are generally preferred
- Feature Selection
  - Use various techniques to see which features are useful and toss out the rest
  - Might be done explicitly before even building a model
  - Might be part of the regularization step for overfitting
- Dimensionality Reduction
  - Collapse features into groups or weighted combinations of features
  - (Useful technique for other purposes)
  - Singular Value Decomposition is a popular method. (Used in Latent Semantic Indexing (LSI))
Missing or Zero Values For Features

The Problem:

- A word \((w)\) never occurred with label \((L)\) in the training set and thus we might assume \(P(w,L) = P(L|w) = P(w|L) = 0\)
- Some algorithms multiply or divide these probabilities
  - 0 times anything is still 0
  - Anything divided by 0 is a problem
- We don't know that this pair can't occur, only that it is rare enough that in our limited sample (training set) we didn't observe any instances

Solutions:

- Smoothing
  - Additive/Add-One/Laplace Smoothing: add one to every pair of (feature, label)
    - Instead of \(0/N\) we have \((0+1)/(N+V)\) where \(V\) is the total number of features.
    - This effectively deals with the issues involving 0 but is kind of dumb and there are much better methods
  - Kneser-Ney Smoothing
    - Basically: How many other labels is this feature associated with?
      - If many, it isn't surprising to see it in this novel situation. If few this new situation is surprising so \(P(w,L)\) should remain near 0.
  - Use dimensionality reduction techniques to spread probability mass intelligently.
Useful Resources/References

- **Pattern Recognition and Machine Learning by Christopher Bishop**
  - Good book
  - Not a gentle introduction but quite good if you are comfortable with the math and willing to take time to understand the details
  - PDFs can be found on the web (no idea if legal)
- **Online NLP Class by Dan Jurafsky and Chris Manning**
  - [www.nlp-class.org](http://www.nlp-class.org)
  - Very good online course
  - Video lectures available via the "preview" button
  - Also relevant are similar courses: "Probabilistic Graphical Models" by Daphne Koller and "Machine Learning" by Andrew Ng
  - When they update the courses you'll need to look around a bit to find the older videos.
- **Wikipedia**
  - Very useful
  - Sometimes a little cryptic when it assumes you already know the material or language
- **Courses at UCSC**
  - Machine Learning (CMPS 142/242, 290C, a bit in AI:140/240 and other places)
  - Data Mining and related courses (ISM 209, 210, 245, 260?)
Selected Algorithms
Naïve Bayes

- Naive Bayes is built around the assumption that all features are independent conditionally independent given the class
  - This assumption is almost never valid
  - When it is valid the model is essentially as good as you can hope to do.
- Simple to train and relatively easy to apply
- Surprisingly effective
- An incredibly simple form of a Bayesian Network
- A good baseline but you can do better
Naïve Bayes

- Our goal is to find $P(C|X_1,\ldots,X_n)$
- From Bayes' Theorem we know:
  \[ P(C|X) = \frac{P(X|C)P(C)}{P(X)} \]
- $P(X)$ doesn't matter since it is without regard to the class $C$.
- $P(X_1,\ldots,X_n|C) = P(X_1|C)P(X_2,\ldots,X_n|C,X_1)$
  \[ = P(X_1|C)P(X_2|C,X_1)P(X_3,\ldots,X_n|C,X_1,X_2) = \ldots \]
- The conditional independence assumption lets us simplify this.
  - Note that $P(X_2|C,X_1)=P(X_2|C)$ since $X_2$ is independent of $X_1$ when $C$ is in there.
  - Thus we have: $P(X_1,\ldots,X_n|C) = P(X_1|C)P(X_2|C)\ldots P(X_n|C)$
- Plugging the result back in yields: $P(C|X) = a\cdot P(C)\cdot \prod P(X_i|C)$
- We can easily find all quantities in that formula (except $a=1/P(X)$, but that doesn't matter). And thus we have the Naïve Bayes model!
- Implementation note: $\prod P(X_i|C)$ is obscenely small and risks underflow on computers, so in practice we apply the monotonic and reversible log() function which lets us add instead of multiply without affecting the final result.
Support Vector Machines are hugely popular in NLP and other areas.

In its simplest form it is a supervised binary classifier.

The basic idea is that each feature is a dimension and each example (again defined by its features) is located at some point in this high-dimensional space. The goal of a SVM is to draw a straight line (hyperplane) which separates the two classes with maximum margin.

Margin is the distance from the hyperplane to the nearest examples (supports)

The "trick" to being able to draw a straight line/hyperplane is that SVMs make use of extra dimensions and non-linear transformations.

The model has a few parameters but largely depends on a relatively small number of training examples aka the supports.

Hard-margin means no errors are allowed.

Soft-margin means you can make some errors.

Problems with SVMs

- Difficult to interpret the model
- Margins are not probabilities and aren't meant to encode how likely an example is to be in one class or another, though in practice it works okay.
- Choosing the right parameters and kernel requires testing or experience
- I understand they are not great for small datasets, but don't quote me on that.
SVM Examples (From PRML)
Boosting

- Uses many weak learners to produce a strong learner
- Train multiple models in sequence
- Each iteration weight hard training examples more and easy examples less.
  - Difficulty is defined by whether the classifier got the example right
- Note that most algorithms weight features but boosting weights training examples as well
- In the end the classifiers each get a weighted vote when predicting
- Boosting can use arbitrary base classifiers.
- Figure from PRML

\[
Y_M(x) = \text{sign} \left( \sum_{m} \alpha_m y_m(x) \right)
\]
Boosting (PRML)
K-Means

- Unsupervised
- Attempts to find $K$ clusters in the dataset
  - Choosing the correct $K$ can be difficult or imprecise
  - Elbow Method
- Algorithm:
  - Randomly select $K$ examples or points ($C_1, C_2, \ldots, C_k$). Call these points centroids.
  - Loop until the centroids don't change:
    - Relabel all examples based on which centroid they are closest to.
    - Update the locations of the centroids to the center of all their examples
- Optionally repeat whole process to find optimum clusters.
- Output clusters
K-Means Example

Actual Labels

my kmeans
K-Means Example
K-Means Example
K-Means Example
K-Means Example

Actual Labels

my kmeans
K-Means Example
K-Means Example
K-Means Example

Actual Labels

my kmeans
K-Means Example
K-Means Example
K-Means Example

Actual Labels vs my kmeans comparison.
K Nearest Neighbor (KNN)

- Non-parametric supervised learning algorithm
- As usual, we want to use the training data to predict the label for some unlabeled example
- Find the K nearest (most similar) examples from the training set and label the new instance with the label that gets the most votes
- Expensive in both time and space
- Not commonly used except for teaching
KNN (PRML Again)
KNN (PRML Again)
Decision Trees

- Decision trees build a set of rules
- The models they produce are typically very easy to interpret (if you don't let them get too complex - which you should avoid doing due to overfitting anyways)
- They can easily handle categorical, numerical, and missing data
- They are not as smooth as many other models
  - Small changes in feature values can lead to big changes in predictions if the features cross a rule’s threshold
  - Variants which resolve this include random forests and decision tree based GBMs
- If not properly pruned and cared for, they will horribly overfit your training data!
- Nowadays people don't seem to care whether they can understand the models and other algorithms perform a bit better, so decision trees aren't as popular as they once were.
Decision Trees
TODO

- SVD (LSI)
- Bayesian Network
- HMM
- CRF
- Logistic Regression
- Neural networks
- More Stats
- Iterative ML Process
- Tools
  - Weka
  - Mallet
  - R
  - ...

Fin