Introduction to Machine Learning

CMSC 422

MARINE CARPUAT
marine@cs.umd.edu

End of semester logistics

Final Exam

- Wednesday May 16th, 10:30am – 12:30pm, CSIC 3117
- closed book, 1 double-sided page of notes
- cumulative, with a focus on topics not covered on midterm
  - bias and adaptation
  - linear models, gradient descent
  - probabilistic models
  - unsupervised learning (PCA)
  - neural networks and deep learning
  - kernels, SVMs

End of semester logistics

- Course evals

https://www.CourseEvalUM.umd.edu

- See piazza for practice problems

What you should know: Bias and how to deal with it

- What is the impact of data selection bias on machine learning systems?
- How to address train/test mismatch
  - Unsupervised adaptation
    - Using auxiliary classifier
  - Supervised adaptation
    - Feature augmentation
- Overfitting/Underfitting

T/F sample from last homework:
Incorrect reasonings are highlighted
Corrections are marked in red

Statement:
When training on sample data, it's important to always make sure that the data is relatively homogenous so that the model can converge.

Reasoning:
False, Homogenous data can lead to a train-test mismatch. When the model is tested in the real world, natural variation could lead to incorrect and unpredictable results.
Think about domain adaptation.

Statement:
Overfit models tend to have low variance and high bias

Reasoning:
False, The opposite is true; they have higher variance and lower bias.

Statement:
When implementing neural networks, more hidden units will always lead to better training and test performance.

Reasoning:
False, Like most other classifiers, by increasing the number of hidden units will cause it to better fit the training data. However, with too many hidden features the function will over-fit the data and test performance will decline.

What you should know: Linear Models

- What are linear models?
  - a general framework for binary classification
  - how optimization objectives are defined
    - loss functions and regularizers
  - separate model definition from training algorithm (Gradient Descent)

What you should know: Gradient Descent

- Gradient descent
  - a generic algorithm to minimize objective functions
  - what are the properties of the objectives for which it works well?
  - subgradient descent (ie what to do at points where derivative is not defined)
  - why choice of step size, initialization matter

Statement:
Gradient Descent works well to minimize any objective function.

Reasoning:
False, Gradient Descent only works well for convex functions.

Statement:
Gradient descent points you in the direction that maximizes your loss.

Reasoning:
True this is why you need to do – del W(w) because it helps minimize loss
The gradient of the loss function with respect to the parameters w points in the direction of maximum increase for the loss.

Statement:
Choice of step size is not important when doing gradient descent. One can choose a random number as the step size.

Reasoning:
False, the statement is incorrect because if the step size is too big, it will "overshoot" the convex, too small will take too much time.

Statement:
Batch gradient descent is an optimization algorithm that calculates derivatives with respect to small subsets of training data.

Reasoning:
False, Batch gradient descent optimizes by computing derivatives over ALL training data by definition.

Statement:
Small changes to w, b can have a large impact on the value of the objective function for zero/one loss.

Reasoning:
True, Let a positive example with w*x + b = -0.0000001. An increase in b by 0.00000011 will decrease the error rate by 1 since it would classify this example correctly. An increase of 0.00000009 will have no effect even though the difference in change of b between these two cases is only 1E-8.

What you should know: Probabilistic Models

- The Naïve Bayes classifier
  - Conditional independence assumption
  - How to train it?
  - How to make predictions?
  - How does it relate to other classifiers we know?

- Fundamental Machine Learning concepts
  - iid assumption
  - Bayes optimal classifier
  - Maximum Likelihood estimation
  - Generative story

What you should know: PCA

- Principal Components Analysis
  - Goal: Find a **projection** of the data onto directions that **maximize variance** of the original data set
  - PCA **optimization objectives** and resulting **algorithm**
  - Why this is useful!

Statement:
It is possible that two different Principal Components are parallel with each other.

Reasoning:
False, Principal Components are orthogonal directions that capture most of the variance in the data.

Statement:
PCA finds vectors such that they minimize reconstruction error

Reasoning:
True, because PCA selects the vectors that have the most variance with respect to the original data.

Statement:
Gradient descent can be useful in unsupervised learning.
Reasoning:
False, gradient descent needs a loss function and error rate to descend on so it cannot be used with unsupervised learning due to the lack of error. In fact, reconstruction error could be used as loss function. So gradient descent is used very often in unsupervised learning.

What you should know: Neural Networks

- What are Neural Networks?
  - Multilayer perceptron
- How to make a prediction given an input?
  - Forward propagation: Matrix operations + non-linearities
- Why are neural networks powerful?
  - Universal function approximators!
- How to train neural networks?
  - The backpropagation algorithm
    - How to step through it, and how to derive update rules

What you should know: Deep Learning

- Why training deep networks is challenging
  - Computationally expensive, vanishing gradient
- Practical techniques for training deep networks
  - Computational graph
  - Stochastic gradient descent
  - Momentum
  - Weight decay

Statement:
Neural Networks can be Universal Function Approximators

Reasoning:
True, Two layer neural networks and larger are Universal Function Approximators

Statement:
Suppose you are given a two-layer neural network, in which the hidden layer contains k neurons. True or false: any real-valued, continuous function $f(x)$, where x is an n-dimensional vector, can be approximated arbitrarily well by this network.

Reasoning:
False, The universal approximation theorem states that given f, and for each \epsilon, there exists a two-layer neural network (with finitely many hidden neurons) which approximates f. It does not state that a fixed neural network can approximate f arbitrarily well. The point is that even a two-layer network is extremely powerful, but in order to approximate a complicated function, we may still need to add a lot of neurons.

Statement:
As an activation function for neural networks, the hyperbolic tangent function is preferred over the sign function.

T

Reasoning:
The hyperbolic tangent function is differentiable while the sign function is not.

Statement:
When training will an artificial neural network will always converge to its optimal solution?

Reasoning:
False, when following gradient decent the algorithm is guaranteed to converge to a local minimum of the cost function but not always the global minimum.

Statement:

With respect to Neural nets:

$$y = \sum_i v_i \cdot \tanh(w_i \cdot x)$$

Reasoning:

False, this is the formula for y_hat, not y. I.E., this is how you calculate the predicted value of a vector sample. Only for a two layer neural network.

What you should know: Kernels

- Kernel functions
  - What they are, why they are useful, how they relate to feature combination
- Kernelized perceptron
  - You should be able to derive it and implement it

What you should know: SVMs

- What are Support Vector Machines
  - Hard margin vs. soft margin SVMs
- How to train SVMs
  - Which optimization problem we need to solve
- Geometric interpretation

  - What are support vectors and what is their relation with parameters **w**,b?
- How do SVM relate to the general formulation of linear classifiers
- Why/how can SVMs be kernelized

Statement:
Maximizing the margin of a linear classifier allows for better generalization.

Reasoning:
True, Maximizing the margin provides a decision boundary that equally splits the feature space.
Smaller ||w||.

Statement:
Support vectors are those that lie precisely 1 unit away from the maximum margin decision boundary.

Reasoning:
True, these points are called the support vectors because they "support" the decision boundary. (definition in the ciml book) Not always 1 unit.

Statement:
If you remove one of the support vectors from the training set for a support vector machine, the size of the maximum margin decreases.

Reasoning:
False, The maximum size of the margin either stays the same or increases, as support vectors keep the margin from expanding.

Statement:
Kernel methods can only be used with support vector machines.

Reasoning:
False, Kernel methods can be used anywhere, not just in SVMs. Only when the computations involve inner products of examples only.

Statement:
A benefit of using feature mappings is that they make linear classifiers computationally inexpensive to train.

Reasoning:
False, Since feature mappings are used to map vectors to higher dimensions, the mapped vectors contain more feature combinations and because of this are much more computationally expensive to train. In fact, the kernel method will not explicitly compute the feature mappings, therefore the computation is not necessarily more expensive.

Statement:
A kernel that computes a quadratic expansion will finish much faster than a kernel that computes a cubic expansion.

Reasoning:
False, the kernel computation for both will be very similar because the kernel implicitly computes the expanded dot product instead of expanding each data point.

Machine Learning

- Paradigm: “Programming by example”
  - Replace ``human writing code'' with ``human supplying data''
- Most central issue: generalization
  - How to abstract from ``training'' examples to ``test'' examples?

Course Goals

- By the end of the semester, you should be able to
  - Look at a problem
  - Identify if ML is an appropriate solution
  - If so, identify what types of algorithms might be applicable
  - Apply those algorithms

- This course is not
  - A survey of ML algorithms
  - A tutorial on ML toolkits such as Weka, TensorFlow, …

Key ingredients needed for learning

- Training vs. test examples
  - Memorizing the training examples is not enough!
  - Need to generalize to make good predictions on test examples
- Inductive bias
  - Many classifier hypotheses are plausible
  - Need assumptions about the nature of the relation between examples and classes

Machine Learning as Function Approximation

Problem setting

- Set of possible instances X
- Unknown target function $f: X \rightarrow Y$
- Set of function hypotheses $H = \{h \mid h: X \rightarrow Y\}$

Input

- Training examples $\{(x^{(1)}, y^{(1)}), \ldots (x^{(N)}, y^{(N)})\}$ of unknown target function f

Output

- Hypothesis $h \in H$ that best approximates target function f

Formalizing Induction

- Given
  - a loss function l
  - a sample from some unknown data distribution D

- Our task is to compute a function f that has low expected error over D with respect to l.

$$\mathbb{E}_{(x,y)\sim D}\{l(y, f(x))\} = \sum_{(x,y)} D(x,y)l(y, f(x))$$

Beyond 422…

- Many relevant courses in machine learning and applied machine learning in CS@UMD
  - Artificial Intelligence (CMSC421), Robotics (CMSC498F), Language (CMSC289J , CMSC470), Vision (CMSC 426), …
- Experiment with tools and datasets
  - weka, scikit-learn, vowpal wabbit, theano, pyTorch, tensorflow…
  - kaggle…
- Keep up to date on cutting-edge machine learning
  - Attend research seminars in the department (e.g., go.umd.edu/cliptalks)
  - Talking Machines podcast

Beyond 422…

- Many opportunities to create new high impact applications with ML

- But there is a gap between theory and practice
  - With real data, Fairness, Accountability, Transparency, Privacy are key concerns
  - "To make great products, do machine learning like the great software engineer you are, not the great machine learning expert you aren't" -Martin Zinkevich's Best practices for ML engineering