Meta-learning Methods

CS771: Introduction to Machine Learning

Purushottam Kar

Announcements

- A few project proposals still not in! Please submit them by Sunday Nov 12 1159 IST
- Assignment 3 late submissions deadline cancelled! There is only one submission deadline now Nov 14 1159 IST
- Extra class Saturday Nov 11 6PM L16 (usual time, usual venue)
- Have started clearing Piazza doubts, will clear all of them
- Should expect Assignment 1 grades in a day or two.
- Assignment 2 grading already underway
- Project presentation/submission timeline to be released soon

Outline of today's discussion

- A few meta-learning methods
  - Techniques that can be applied to several learning models/tasks
- Model selection
- Ensemble methods
  - Bagging
  - Boosting
- Next up: Recommendation systems and learning with incomplete supervision

Model Selection

Many Myriad Models

- Every algorithm we have studied requires us to make high level decision about the design and structure of the algorithm
- Nearest neighbors: How many neighbors? What metric?
- Decision tree: How many levels? Binary/ternary splits?
- ML/MAP: Which prior (L1/L2)? What regularization constant?
- GMMs: How many components in the mixture
- (K)PCA: How many dimensions?
- SVMs: Which kernel? Which kernel hyperparameters?
- NNs: How many layers? How many nodes? Which activation fn?
- Which algo? NNs or SVMs or DTs or k-NN?
- Form a part of the inductive bias – can we lessen this bias?

Model Selection

- Let $\mathcal{M} = \{m_1, m_2, \dots, m_k, \dots\}$ be a set of models to choose from
- Depending on our setting, each m_i could represent
  - a regressor learnt using a regularization constant λ_i, or
  - a GMM learnt with n_i number of clusters, or
  - a NN learnt with L_i hidden layers, or
  - a kernel SVM learnt with Gaussian kernel with hyperparameter γ_i
- Some of the m_i could be kernel SVMs, others could be NNs etc
- How to choose the model that will perform the best on test?
- Notation
  - $\theta_i = \mathrm{TRAIN}(m_i, S)$ model m_i was trained on data S to get parameters θ_i
  - $v_i = \mathrm{TEST}(m_i, \theta_i, T)$ model m_i with parameters θ_i was tested on data T to get performance $v_i \in \mathbb{R}$ (misclassifn rate, residual, reconstruction error)

Model Selection

- Let $\mathcal{M} = \{m_1, m_2, \dots, m_k, \dots\}$ be a set of models to choose from
- Depending on our setting, each m_i could represent
  - a regressor learnt using a regularization constant λ_i, or
  - a GMM learnt with n_i number of clusters
  - a NN learnt with L_i hidden layers, or
  - a kernel SVM learnt with Gaussian kernel with hyperparameter γ_i
- Some of
- How to
- Notation
  - $\theta_i = \text{TRAIN}(m_i, S)$ model m_i was trained on data S to get parameters θ_i
  - $v_i = \text{TEST}(m_i, \theta_i, T)$ model m_i with parameters θ_i was tested on data T to get performance $v_i \in \mathbb{R}$ (misclassifn rate, residual, reconstruction error)

If $\mathcal{M}$ contains variants of the same model (e.g. all are DTs) then $\mathcal{M}$ is called a *model class*

Same model trained on different data gives different parameters θ_i

θ_i could be weights of regressor, cluster centers in GMM, network weights in NN

Held-out Validation

S

- Split training set S into 2 parts S_1, S_2 randomly
- Train each model on S_1, test on S_2. Choose model with best perf.

$$m^* = \arg\min_{m_i \in \mathcal{M}} \text{TEST}(m_i, \text{TRAIN}(m_i, S_1), S_2)$$

- Very efficient, widely used in practice 70-30, 80-20 splits popular
- Wastes data: S_2 never used in training
- Carries a risk of choosing an unfortunate split. If we are unlucky, S_2 may make a good model look bad and a bad model look good

Held-out Validation

- Split training set S into 2 parts S_1, S_2 randomly
- Train each model on S_1, test on S_2. Choose model with best perf.

$$m^* = \arg\min_{m_i \in \mathcal{M}} \text{TEST}(m_i, \text{TRAIN}(m_i, S_1), S_2)$$

- Very efficient, widely used in practice 70-30, 80-20 splits popular
- Wastes data: S_2 never used in training
- Carries a risk of choosing an unfortunate split. If we are unlucky, S_2 may make a good model look bad and a bad model look good

k-fold Cross Validation

S

- Split training set S into k parts $S_1, S_2, \ldots, S_k$ randomly ($k = 5$ popular)
- Train each model on all but S_j, test on S_j. Repeat for all $j = 1, \ldots, k$
- Choose mode with best average performance

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

k-fold Cross Validation

S_1	S_2	S_3	S_4	S_5

- Split training set S into k parts $S_1, S_2, \ldots, S_k$ randomly ($k = 5$ popular)
- Train each model on all but S_j, test on S_j. Repeat for all $j = 1, \ldots, k$
- Choose mode with best average performance

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

k-fold Cross Validation

S_1	S_2	S_3	S_4	S_5

- Split training set S into k parts $S_1, S_2, \ldots, S_k$ randomly ($k = 5$ popular)
- Train each model on all but S_j, test on S_j. Repeat for all $j = 1, \ldots, k$
- Choose mode with best average performance

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

k-fold Cross Validation

S_1	S_2	S_3	S_4	S_5

- Split training set S into k parts $S_1, S_2, \ldots, S_k$ randomly ($k = 5$ popular)
- Train each model on all but S_j, test on S_j. Repeat for all $j = 1, \ldots, k$
- Choose mode with best average performance

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

k-fold Cross Validation

S_1	S_2	S_3	S_4	S_5

- Split training set S into k parts $S_1, S_2, \ldots, S_k$ randomly ($k = 5$ popular)
- Train each model on all but S_j, test on S_j. Repeat for all $j = 1, \ldots, k$
- Choose mode with best average performance

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

k-fold Cross Validation

S_1	S_2	S_3	S_4	S_5

- Split training set S into k parts $S_1, S_2, \ldots, S_k$ randomly ($k = 5$ popular)
- Train each model on all but S_j, test on S_j. Repeat for all $j = 1, \ldots, k$
- Choose mode with best average performance

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

k-fold Cross Validation

S_1	S_2	S_3	S_4	S_5

- Split training set S into k parts $S_1, S_2, \ldots, S_k$ randomly ($k = 5$ popular)
- Train each model on all but S_j, test on S_j. Repeat for all $j = 1, \ldots, k$
- Choose mode with best average performance

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

k-fold Cross Validation

S_1	S_2	S_3	S_4	S_5

- Split training set S into k parts $S_1, S_2, \ldots$)
- Train each model on all but S_j, test on
- Choose mode with best average perfo ance

Extreme variant LOO (leave-one-out) If $|S| = n$, then $k = n$ i.e. every data point is a part

$$m^* = \arg\min_{m_i \in \mathcal{M}} \frac{1}{k} \sum_{j=1}^{k} \mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S \backslash S_j), S_j)$$

- Much more expensive but more reliable too
- Even if one part is "bad" there are other parts

Other techniques

- **Overlapping fold selection**: select k randomly chosen sets of size, say $0.3n$, $S_1, \ldots, S_k$. Train on $S \backslash S_k$, test on S_k. Choose best avg perf.
- **Bootstrap**: select n data points randomly with replacement and use as training set (may have repeated points). Use points never selected as validation set.
- **Structural risk minimization (SRM)**: Define a complexity term for each model $r(m_i)$ (# layers, clusters, magnitude of hyper param)
  - Prefers models that are less "complex" (see Occam's razor)

$$m^* = \arg\min_{m_i \in \mathcal{M}} \{\mathrm{TEST}(m_i, \mathrm{TRAIN}(m_i, S), S) + r(m_i)\}$$

- **Akaike/Bayesian information criteria (AIC, BIC)**: Counterpart to SRM in PML techniques. Replace test error with likelihood.

Other techniques

- **Bandit Optimization**: useful when each $m \in \mathcal{M}$ corresponds to a hyperparameter. View model selection as an optimization problem

$$m^* = \arg\min_{m \in \mathcal{M}} f(\mathrm{m}) = \arg\min_{m \in \mathcal{M}} \mathrm{TEST}(m, \mathrm{TRAIN}(m, S), S)$$

  - However, getting "gradients" for the above objective function intractable
  - Hence cannot request for gradients or Hessians of f while optimizing it
  - Can only ask for $f(\cdot)$ values on specific models $m^1, m^2, \ldots$
  - Also known as zeroth-order optimization, derivative-free optimization
  - Bayesian optimization is an example of Bandit optimization
- **Bayesian Learning**: cast model selection as a learning problem!
  - Establish a prior over the model class $\mathcal{M}$ and a likelihood $\mathbb{P}[S \mid m]$
  - Perform model learning jointly with parameter learning

Bias-Variance

Bias-Variance

- Ultimate goal of ML is to do well on test data

Bias-Variance

- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance

Bias-Variance

- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance
  - **Bias**: your model class is too weak e.g. linear model for a very complex task
    Even the best trained linear model is pathetic

Bias-Variance

- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance
  - **Bias**: your model class is too weak e.g. linear model for a very complex task Even the best trained linear model is pathetic
  - **Variance**: your model class is strong but you could not train it properly e.g. NNs. The best trained NN is NP-hard to learn

Bias-Variance

- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance
  - **Bias**: your model class is too weak e.g. linear model for a very complex task Even the best trained linear model is pathetic
  - **Variance**: your model class is strong but you could not train it properly e.g. NNs. The best trained NN is NP-hard to learn
- Models with high variance usually are brittle too. Changing the training data even slightly changes the model parameters a lot

Bias-Variance

- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance
  - **Bias**: your model class is too weak e.g. linear model for a very complex task Even the best trained linear model is pathetic
  - **Variance**: your model class is strong but you could not train it properly e.g. NNs. The best trained NN is NP-hard to learn
- Models with high variance usually are brittle too. Changing the training data even slightly changes the model parameters a lot
- Usually models that are weak are easy to train very accurately

Bias-Variance

- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance
  - **Bias**: your model class is too weak e.g. linear model for a very complex task Even the best trained linear model is pathetic
  - **Variance**: your model class is strong but you could not train it properly e.g. NNs. The best trained NN is NP-hard to learn
- Models with high variance usually are brittle too. Changing the training data even slightly changes the model parameters a lot
- Usually models that are weak are easy to train very accurately
  - High bias, low variance

Bias-Variance

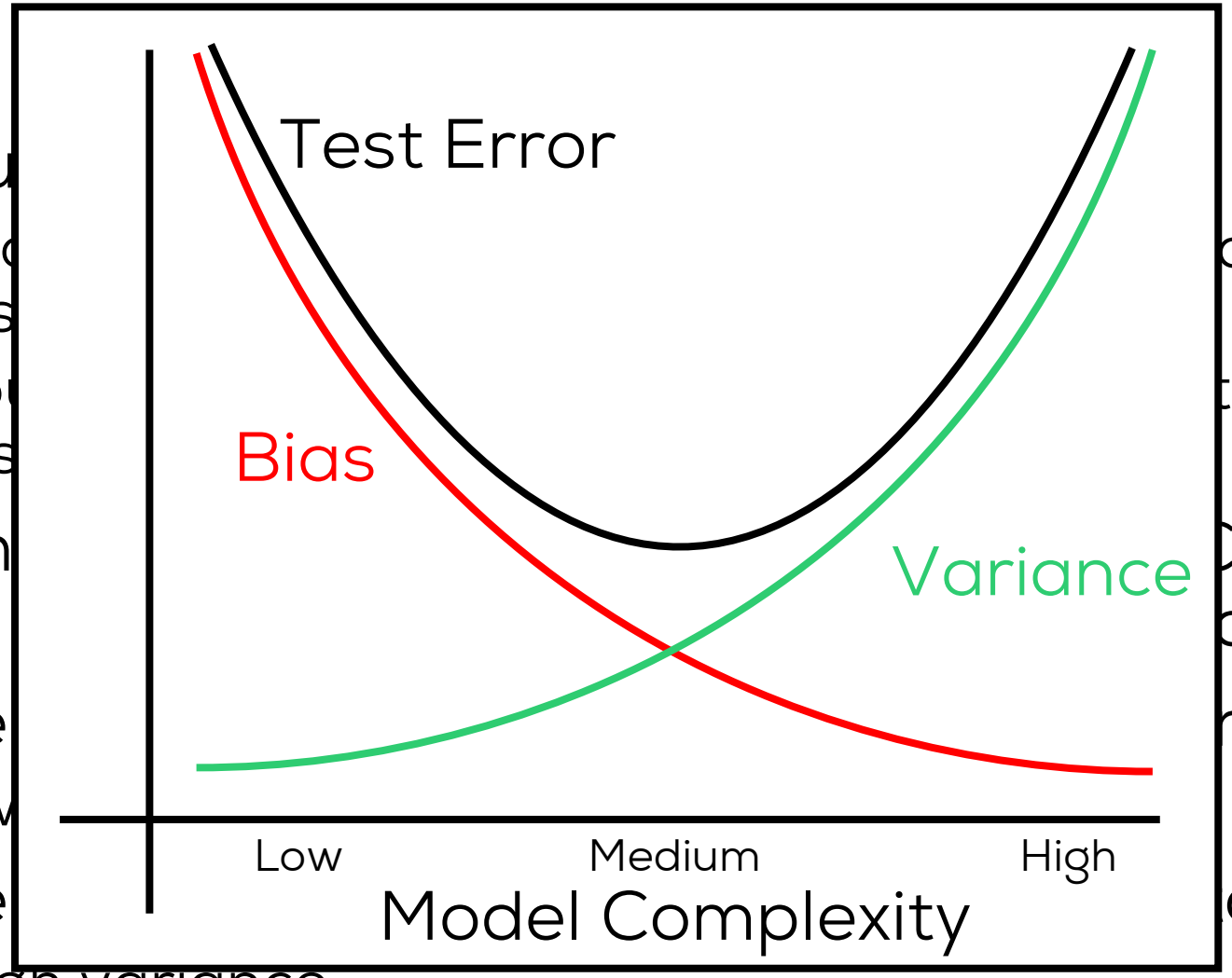
- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance
  - **Bias**: your model class is too weak e.g. linear model for a very complex task Even the best trained linear model is pathetic
  - **Variance**: your model class is strong but you could not train it properly e.g. NNs. The best trained NN is NP-hard to learn
- Models with high variance usually are brittle too. Changing the training data even slightly changes the model parameters a lot
- Usually models that are weak are easy to train very accurately
  - High bias, low variance
- Usually models that are strong are more difficult to train too

Bias-Variance

- Ultimate goal of ML is to do well on test data
- Two main sources of bad test performance
  - **Bias**: your model class is too weak e.g. linear model for a very complex task Even the best trained linear model is pathetic
  - **Variance**: your model class is strong but you could not train it properly e.g. NNs. The best trained NN is NP-hard to learn
- Models with high variance usually are brittle too. Changing the training data even slightly changes the model parameters a lot
- Usually models that are weak are easy to train very accurately
  - High bias, low variance
- Usually models that are strong are more difficult to train too
  - Low-bias, high variance

Bias-Variance

- Ultimate goal
- Two main sou
  - **Bias**: your m a very complex task Even the bes
  - **Variance**: yo train it properly e.g. NNs. The bes
- Models with h Changing the training data ameters a lot
- Usually mode ry accurately
  - High bias, lo
- Usually mode o train too
  - Low-bias, high variance



In Practice

In Practice

- Low training error but high test error

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error
  - You may have underfit. Increase model class complexity to decrease bias

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error
  - You may have underfit. Increase model class complexity to decrease bias
  - Adding more data cannot decrease bias. Your model class just sucks

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error
  - You may have underfit. Increase model class complexity to decrease bias
  - Adding more data cannot decrease bias. Your model class just sucks
- Low training error and low test error

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error
  - You may have underfit. Increase model class complexity to decrease bias
  - Adding more data cannot decrease bias. Your model class just sucks
- Low training error and low test error
  - er … exactly what are you complaining about?

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error
  - You may have underfit. Increase model class complexity to decrease bias
  - Adding more data cannot decrease bias. Your model class just sucks
- Low training error and low test error
  - er … exactly what are you complaining about?
- High training error and low test error

In Practice

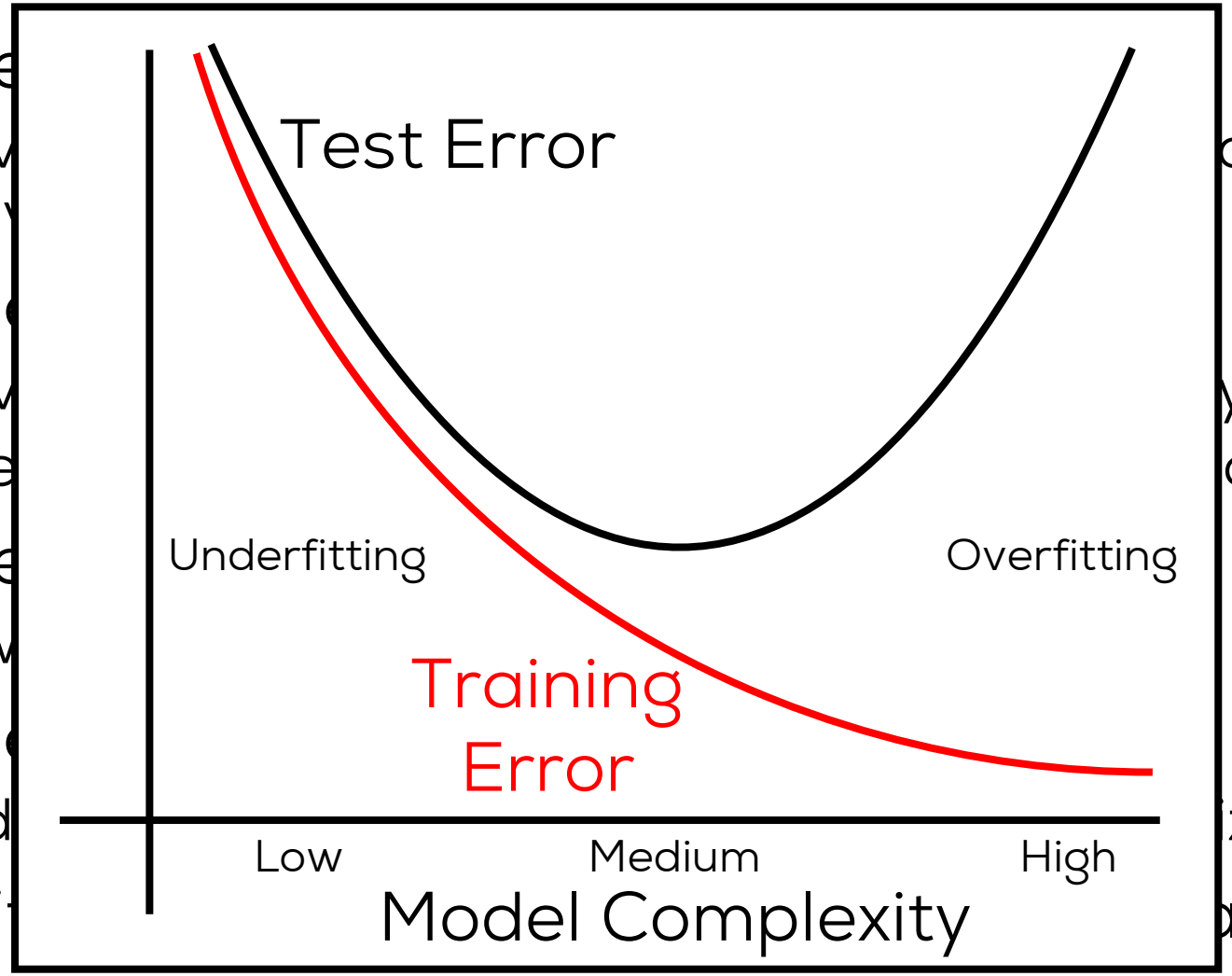
- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error
  - You may have underfit. Increase model class complexity to decrease bias
  - Adding more data cannot decrease bias. Your model class just sucks
- Low training error and low test error
  - er … exactly what are you complaining about?
- High training error and low test error
  - Maybe you did early stopping which acted as a regularizer

In Practice

- Low training error but high test error
  - You may have overfit. Add more data or simplify the model class (or both) to decrease variance
- High training error and high test error
  - You may have underfit. Increase model class complexity to decrease bias
  - Adding more data cannot decrease bias. Your model class just sucks
- Low training error and low test error
  - er … exactly what are you complaining about?
- High training error and low test error
  - Maybe you did early stopping which acted as a regularizer
- May need to iterate the above two a bit to reach a sweet spot

In Practice

- Low training e
  - You may hav odel class (or both) to decrease
- High training
  - You may hav y to decrease bias
  - Adding more ass just sucks
- Low training e
  - er … exactly
- High training
  - Maybe you d zer
- May need to i a sweet spot



Learning Ensembles

What are ensembles and why learn them?

- For sake of simplicity, focus only on binary classification only
- Similar techniques apply to regression, multi-label classfn too
- I wish to learn a classifier using a model class $\mathcal{M}$ (say linear or NN)
- Why?
  - I have a nice implementation to learn from $\mathcal{M}$ – don't wanna waste it
  - Model class $\mathcal{M}$ has very low variance which is nice
  - Prediction using $\mathcal{M}$ is very cheap
  - But $\mathcal{M}$ has very high bias ☹ – what do I do?
- Or
  - Model class $\mathcal{M}$ is very powerful and has very low bias which is nice
  - All my friends use models from $\mathcal{M}$ and I have FoMO issues
  - But $\mathcal{M}$ has very high variance ☹ - what do I do?
- Ensembles to the rescue!!

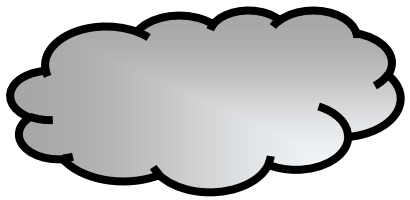
Voting Ensemble

- Works even when training is not in our hands or not from single $\mathcal{M}$
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

wikipedia.org

Voting Ensemble

- Works even when training is not in our hands or not from single $\mathcal{M}$
- Suppose we have 5 sources to answer "Will it rain tomorrow?"



Y

N

Y

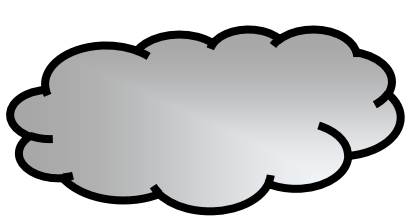
Y

Y

N

Voting Ensemble

- Works even when training is not in our hands or not from single $\mathcal{M}$
- Suppose we have 5 sources to answer "Will it rain tomorrow?"



Y
N
Y
Y
Y
N

Voting Ensemble

- Works even when training is not in our hands or not from single $\mathcal{M}$
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

wikipedia.org

Voting Ensemble

No individual network gets more than 66% correct predictions but if we take a majority vote, 100% correct

- Works even when training is n $\mathcal{M}$
- Suppose we have 5 sources to answer "Will it rain tomorrow?"

CNN	BBC	NBC NEWS	दूरदर्शन	ALJAZEERA	
N	Y	N	Y	Y	Y
N	Y	N	N	Y	N
Y	Y	N	Y	Y	Y
N	Y	Y	N	Y	Y
Y	N	Y	Y	N	Y
Y	N	N	Y	N	N

Voting Ensemble

- Receive K pre-trained classifiers $f_1, f_2, \ldots, f_K$ s.t. $f_i: \mathcal{X} \to \{-1, +1\}$
- Construct a new classifier $\hat{f}_{\text{MAJ}}$ such that for any $x \in \mathcal{X}$

$$\hat{f}_{\text{MAJ}}(x) = \text{sign}\left(\sum_{k=1}^{K} f_k(x)\right)$$

- Hope that mistakes of one will be corrected by others
- Stacking: interpret $[f_1(x), f_2(x), \ldots, f_K(x)]$ as a K-dimensional vector and learn a new classifier over these "features"
- This is not expected to do well in general. If the classifiers were not trained properly, they may synchronize their mistakes
- Fixing this issue leads to useful techniques bagging and boosting

Bagging – Bootstrap AGGregatING

- Has variance reduction properties – works for any model
- Given a training set S with n data points
  - Sample n data points with replacement from S call this S_1
  - Repeat this K times to obtain K bagged datasets $S_1, S_2, \ldots, S_K$
  - Learn a model $f_i: \mathcal{X} \to \{-1,1\}$ using dataset S_i (maybe the same algo)
  - Predict a new point $x \in \mathcal{X}$ using $\hat{f}_{\text{MAJ}}(x) = \text{sign}\left(\sum_{k=1}^{K} f_i(x)\right)$
- Can show that only about 63% of S lands up in any S_i
- This means all S_i have sufficient diff
- Even if we have a high variance method that overfits, it will overfit to substantially different sets
- Overall variance reduction effect

Bagging – Bootstrap AGGregatING

- Has variance reduction properties – works f
- Given a training set S with n data points
  - Sample n data points with replacement from S a
  - Repeat this K times to obtain K bagged datasets $S_1, S_2, \ldots, S_K$
  - Learn a model $f_i: \mathcal{X} \rightarrow \{-1,1\}$ using dataset S_i (maybe the same algo)
  - Predict a new point $x \in \mathcal{X}$ using $\hat{f}_{\text{MAJ}}(x) = \text{sign}\left(\sum_{k=1}^{K} f_i(x)\right)$
- Can show that only about 63% of S lands up in any S_i
- This means all S_i have sufficient diff
- Even if we have a high variance method that overfits, it will overfit to substantially different sets
- Overall variance reduction effect

Usually all f_i are from the same model class $\mathcal{M}$ e.g. DTs

Bagging

- Note that the K models are trained independently
  - Allows for massive parallelization of learning algorithms
- We may train the same kind of model in these K iterations e.g. train K decision trees of the same/different depth
- Can be used to enforce regularization without the explicit use of regularizers – useful in models where modifying code is tricky
- Seen to perform variance reduction
- Does not reduce bias because bagging usually applied to powerful models where bias is small to begin with
- Two popular algorithms: random forests and dropout, have their genesis in bagging

Random Forests

- A collection of decision trees is called a decision forest
- Let us fix the ID3 algorithm to learn a decision tree
- Let us have training data $S = \{(\mathbf{x}^i, y^i)\}_{i=1,\dots,n}$ with $x^i \in \mathbb{R}^d, y^i \in \{-1,1\}$
- Random forests learn K decision trees
- First, bagging is done to get datasets $S_1, \dots, S_K$
- Next, "feature bagging" is done
  - Sample K subsets of $[d]$ - $F_1, F_2, \dots F_K$, each of size d' where each is chosen randomly **without** replacement (note bagging was done with replacement)
  - Typically $d' \sim \sqrt{d}$ for DTs
- Learn the k-th DT on dataset S_k using only the features in F_k
- Intuition: if some feature is really good, every tree will use it and then all trees will behave similarly so restrict available features

Dropout

- Dropout for NNs can be seen as an attempt to perform bagging and feature bagging at a ridiculous scale
- If the NN has N nodes then dropout wishes to train all 2^N possible subnetworks as the $K = 2^N$ models
- However, whereas in random forests, different trees usually have very different parameter values, dropout wants all these 2^N subnetworks to share the parameters (edge weights)
- Two subnetworks that contain the same edge must have the same weight on that edge
- Intractable to execute explicitly which is why dropout does this approximately.
- At every time step t a random subnetwork trained using a mini-batch of data – mini-batch meant to approximate a bagged set

Please give your Feedback

http://tinyurl.com/ml17-18afb