Advanced Learning Algorithms-II

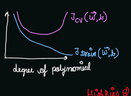
Friday, March 10, 2023 4:20 PM

Large errors in prediction
- → get more training examples
- → try smaller sets of features
- → try adding additional features
- → try adding polynomial features
- → try decreasing λ
- → try increasing λ

cross validation set (dev set, validation set)

High Bias → underfit
J_{train} is high
J_{CV} is also high

High Variance → overfit
J_{train} is low
J_{CV} is high

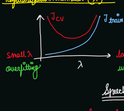


High Bias (underfit)
J_{train} will be high
($J_{train} \approx J_{CV}$)

High Variance (overfit)
J_{train} is low
$J_{CV} >> J_{train}$

High Bias & High Variance
J_{train} is high
$J_{CV} >> J_{train}$

Regularization Parameter (λ)



Speech Recognition

Human level performance → 10.6% ↑0.2%
training error J_{train} → 10.7% ↑ 4% high variance
Cross validation error J_{CV} → 14.8%

High Bias → In learning algorithm has high bias, getting more data will not help much by itself.

High Variance → If an algorithm suffers from high variance then getting more data is likely to help.

Debugging a learning algorithm

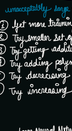
Regularized Linear Regression

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

unacceptably large error in prediction
1. Get more training examples → fix high variance
2. Try smaller set of features → fix high variance
3. Try getting additional features → fix high bias
4. Try adding polynomial features (x_1^2, x_2^2, x_1x_2 etc.) → fix high bias
5. Try decreasing λ → fix high bias
6. Try increasing λ → fix high variance

Neural Networks and Bias/Variance

Large neural networks are low bias machines



A large neural network will usually do better or as well than a smaller one so long as regularization is chosen appropriately.

Machine learning Development Process

Iterative loop of ML Development

Adding Data
→ Add more data of everything (e.g. "Honeypot" project)
→ Add more data of the types where error analysis indicated it might help

Data Augmentation

Augmentation: modifying an existing training example to create new training example.

Transfer Learning (Option 1 also work)
1. Supervised Pretraining
2. Fine Tuning

Option
1. Only train output layer parameters
2. Train all parameters (more data)

Full cycle of Machine learning project

Scope Project (Define project) → Collect Data (Define and collect data) → Train Model (Training, error analysis & iterative improvement) → Deploy in production (Deploy, monitor & maintain system)

Deployment

Skewed Datasets

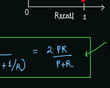
		1	0
Predicted class	1	TP	FP
	0	FN	TN
		Actual class	

Precision = # true positives (TP) / # predicted positives (TP+FP)

Recall = # true positives (TP) / # actual positives (TP+FN)

"high precision & high recall"

Trade off between Precision & Recall



F_1 score

$$F_1 \text{ score} = \frac{1}{\frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)} = \frac{2PR}{P+R}$$

Harmonic mean

Decision Trees

"cat classification example"

Decision Tree Learning

Decision 1: How to choose what feature to split on at each node?
→ Maximize purity (or minimize impurity)

Decision 2: When do you stop splitting?
→ When a node is 100% one class
→ When splitting a node will result in a tree exceeding a maximum depth
→ When improvements in purity score are below a threshold
→ When # examples in a node is below a threshold

Measuring Purity

Entropy → measurement of impurity of data. {C,C,C,D,D,D}

P_1 = fraction of examples that are cat (C) → $P_1 = 3/6$ → $H(P_1) = 1$

$P_0 = 1 - P_1$

Entropy

$$H(P_1) = -P_1\log_2(P_1) - P_0\log_2(P_0)$$
$$= -P_1\log_2(P_1) - (1-P_1)\log_2(1-P_1)$$

{ note: "0 log(0)" = 0 }

Choosing a split

$P_1^{root} = 5/10 = 0.5$ {C,C,C,D,D,C,D,C,D,D}

Ear Shape: $P_1^{left} = 4/5$, $w^{left} = 5/10$; $P_1^{right} = 1/5$, $w^{right} = 5/10$

Face Shape: $P_1^{left} = 4/7$, $w^{left} = 7/10$; $P_1^{right} = 1/3$, $w^{right} = 3/10$

{C,C,C,C,D} {D,D,D,C}

$P_1 = 4/5 = 0.8$ $P_1 = 1/5 = 0.2$

$H(0.8) = 0.72$ $H(0.2) = 0.72$

$H(0.5) - \left(\frac{5}{10}H(0.8) + \frac{5}{10}H(0.2)\right)$

$\Rightarrow 0.28$ {Information gain}

$$\text{Information gain} = H(P_1^{root}) - \left(w^{left}H(P_1^{left}) + w^{right}H(P_1^{right})\right)$$

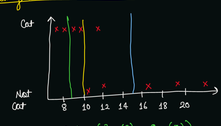
Decision Tree Learning
1. Start with all examples at the root node
2. Calculate Information gain for all possible features, and pick the one with the highest information gain
3. Split dataset according to selected feature & create left and right branches of the tree.
4. Keep repeating splitting process until stopping criteria is met:
   - → When a node is 100% one class
   - → When splitting a node will result in the tree exceeding a maximum depth
   - → Information gain from additional splits is less than threshold
   - → When # examples in a node is below a threshold.

Recursive splitting

One hot encoding

If a categorical feature can take on k values, create k features (0 or 1 valued).

Splitting on a continuous variable

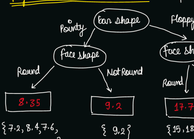


$H(0.5) - \left(\frac{2}{10}H\left(\frac{2}{2}\right) + \frac{8}{10}H\left(\frac{3}{8}\right)\right) = 0.24$

$H(0.5) - \left(\frac{4}{10}H\left(\frac{4}{4}\right) + \frac{6}{10}H\left(\frac{1}{6}\right)\right) = 0.61$ (max information gain)

$H(0.5) - \left(\frac{7}{10}H\left(\frac{5}{7}\right) + \frac{3}{10}H\left(\frac{0}{3}\right)\right) = 0.40$

Regression with Decision Trees "predicting weight"



{7.2, 8.4, 7.6, 10.2} {9.2} {25, 18, 20} {8.8, 11}

choosing a split?

Classification → minimize average weighted entropy

Regression → minimize average weighted variance

Tree Ensembles

using multiple decision trees → to get them to vote.

① Trees are highly sensitive to small changes of the data

Sampling with Replacement → to generate new training set for tree ensembles.

Tokens:

Sampling with replacement:

Random Forest Algorithm "Bagged Decision Trees"

Given a training example of size m

For $b = 1$ to B:
- use sampling with replacement to create new training set of size m
- Train a decision tree on the new dataset

Randomizing the feature choice (Random Forest Algorithm)

At each node, when choosing a feature to use to split, if n features are available pick a random subset of $k < n$ features & allow the algorithm to only choose from that subset of features.

large n, $k = \sqrt{n}$

XGBoost

Boosted Trees intuition

Given training set of size m

For $b = 1$ to B:
- use sampling with replacement to create a new training set of size m
- But instead of picking from all the examples with equal ($1/m$) probability, make it more likely to pick misclassified examples from the previously trained trees.
- Train a decision tree on the new dataset

XGBoost (eXtreme Gradient Boosting)
- → Open source implementation of Boosted Trees
- → Fast efficient implementation
- → Good choice of default splitting criteria & criteria for when to stop splitting
- → Built in regularization to prevent overfitting
- → Highly competitive algorithm for ML competitions

```
from xgboost import XGBClassifier/ XGBRegressor

model= XGBClassifier()/ XGBRegressor()

model.fit(X_train, y_train)
y_pred= model.predict(X_test)
```

Decision Trees vs Neural Networks

Decision Trees & Tree Ensembles
- → Work well on tabular (structured) data
- → Not recommended for unstructured data (images, audio, text)
- → Fast
- → Small decision trees may be human interpretable

Neural Networks
- → Works well on all types of data, including tabular (structured) and unstructured data
- → May be slower than a decision tree
- → Works with transfer learning
- → When building a system of multiple models working together, it might be easier to string together multiple neural networks