CS1138

Machine Learning

Lecture : Evaluation Metrics, Model Selection, Data Splits and Bias Variance Tradeoff

(Slide Credits: Sebastian Raschka, Andrew Ng)

Arpan Gupta

Overview

- Confusion Matrix
- Precision, Recall, F1-score
- Balanced Accuracy
- ROC Curve
- Model Selection / Data Splits
- Bias Variance Tradeoff

Find the accuracy

- Suppose there are 100 patients, 50 of them actually have a disease, while remaining 50 are healthy. An ML model (which pretends to classify the patients based on disease), just takes the input features of patients and outputs “healthy” for all the cases. (It actually doesn’t use the features, just outputs the same thing for all the patients). What is the accuracy of the “model”?

- Answer: **50%**

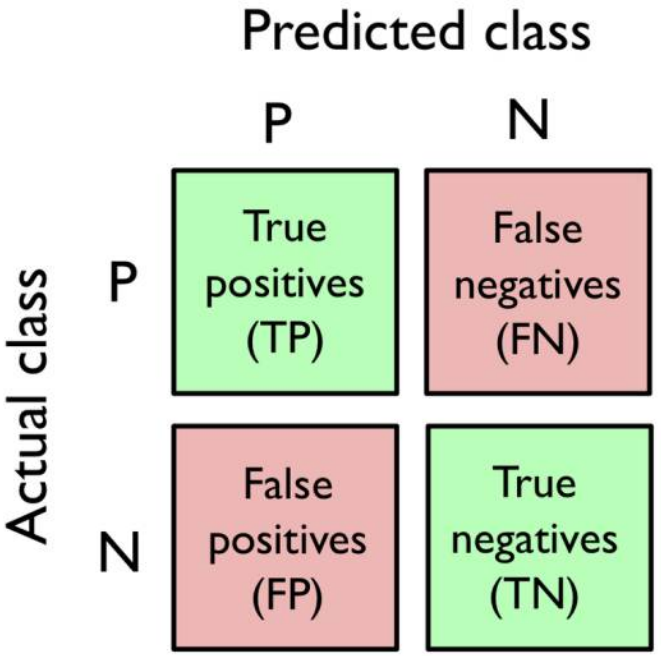
Find the accuracy

- Suppose there are 100 patients, 10 of them actually have a disease (say cancer), while remaining 90 are healthy. What is the accuracy when the same “model” is used?

- Answer: **90%**

  <u>Class Imbalance problem</u> arises. Therefore, our accuracy metric does not show the correct evaluation.

2x2 Confusion Matrix



Malignant Tumour:
Positive Class

Benign Tumour:
Negative Class

$$ERR = \frac{FP + FN}{FP + FN + TP + TN} = 1 - ACC$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

TP, TN, FP, FN

- True Positive (TP): #Samples of positive class that were correctly predicted. Malignant tumours correctly predicted.
- True Negative (TN): Benign tumours correctly predicted.
- False Positive (FP): Predicted malignant, but were actually benign.
- False Negative (FN): Predicted benign, but were actually malignant.

TPR, FPR, TNR, FNR

- TPR: ratio of correct +ve predictions to the total positives in the dataset.
- FPR: ratio of false positives to the total number of negatives in the dataset.
- TNR: ratio of correct -ve predictions to the total negatives in the dataset.
- FNR: ratio of false negatives to the total number of positves in the dataset.

False Positive Rate and False Negative Rate

* Relevant later for ROC

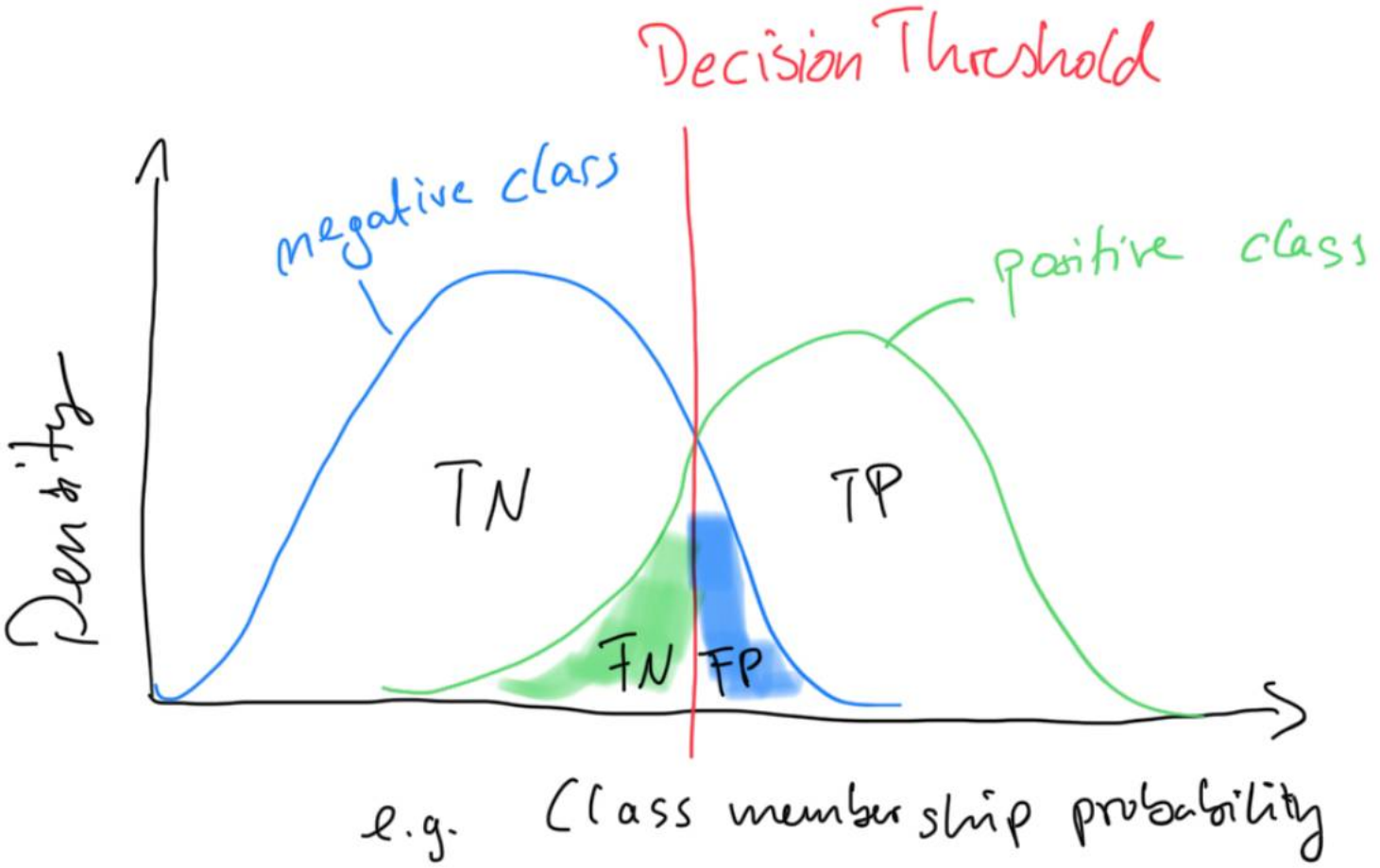
$$\mathrm{TPR}^{*} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

$$\mathrm{FPR}^{*} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

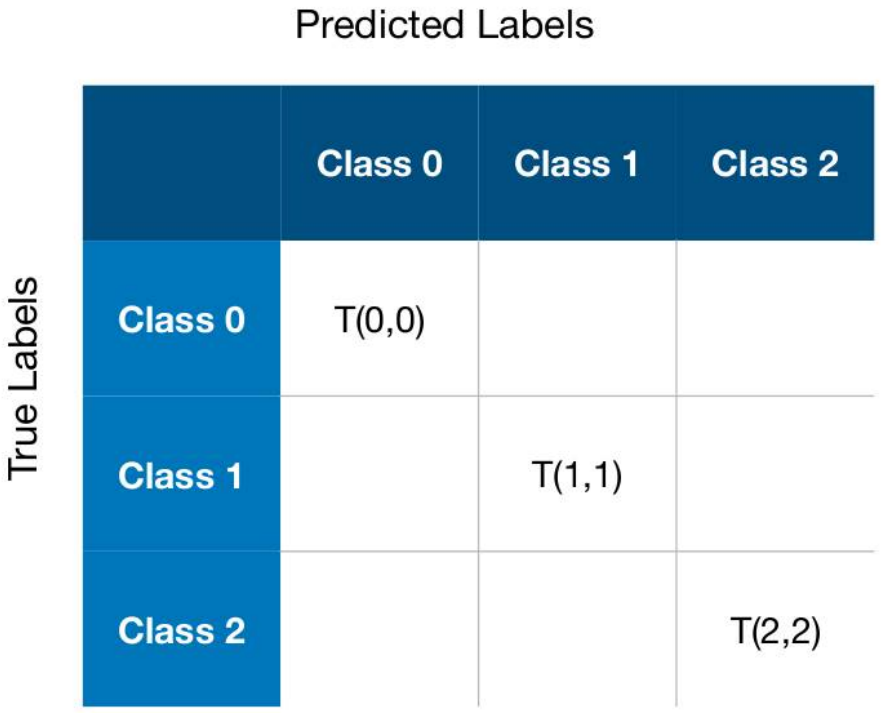
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

FPR Vs FNR


Decision Threshold
negative class
positive class
Density
TN
TP
FN
FP
e.g. Class membership probability

Confusion Matrix for multi-class setting



Confusions matrices are traditionally for binary class problems but we can be readily generalized it to multi-class settings

Precision and Recall

$$Precision = \frac{TP}{TP + FP}$$

#Correct +ve predictions

#Total +ve predictions

$$Recall = \frac{TP}{TP + FN}$$

#Correct +ve predictions

#Total +ve Ground Truth values

Recall is another name for “True Positive Rate (TPR)” or “sensitivity”.

F1-Score

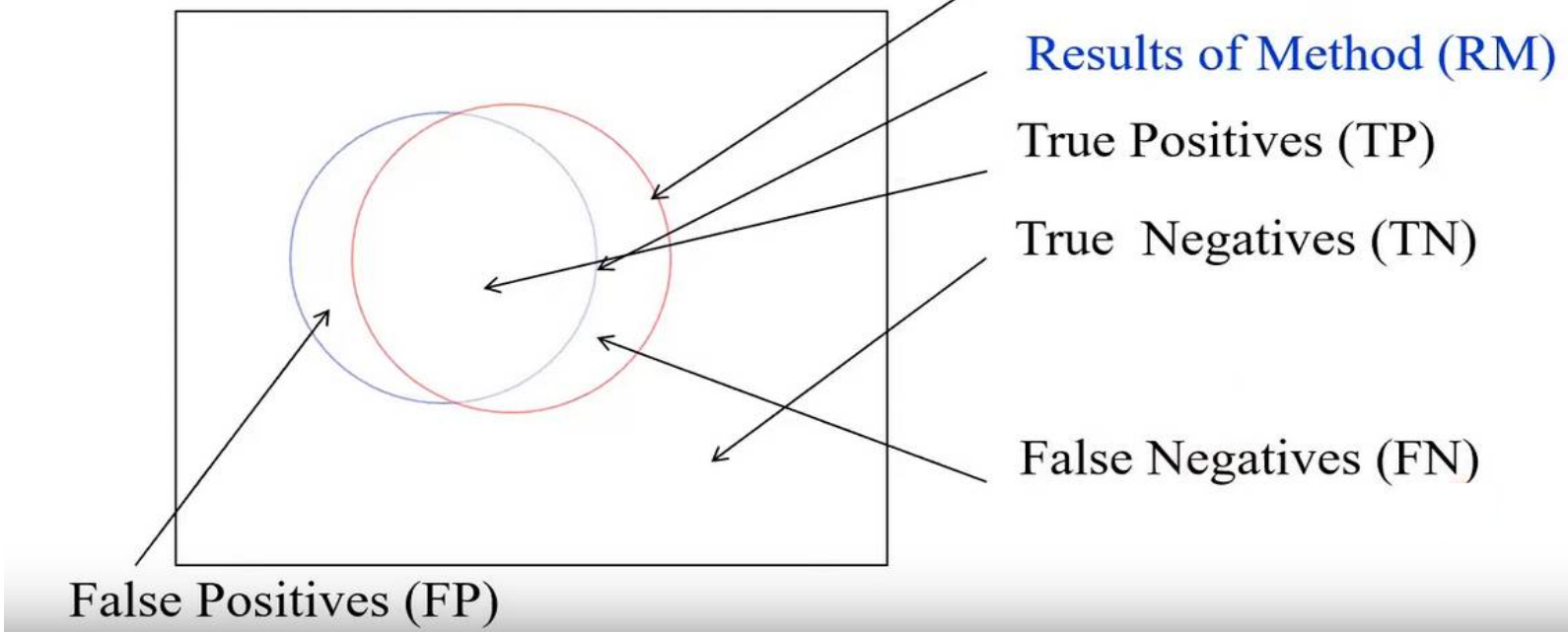
- It is the harmonic mean of Precision and Recall.

$$F1-Score=\frac{2\,x\,Precision\,x\,Recall}{Precision+Recall}$$

Evaluation Metrics

$$\text{precision} = \frac{GT \bigcap RM}{RM} = \frac{TP}{RM}$$

$$\text{recall} = \frac{GT \bigcap RM}{GT} = \frac{TP}{GT}$$



Find the precision recall and F1-score

- Suppose we want to classify 100 patients into having cancer or not. The expert doctors (after taking into account their features/lab reports etc.) have said that 5 have cancer, and rest of them do not have cancer. We build an ML model, which generates the outputs and we find the TP, TN, FP and FN values as follows. What is the precision and recall values of the same “model”?

  TP=3, TN = 92, FP=3, FN = 2

- Answer:

  Precision: 3 / (3+3) = 0.5 (or 50%)

  Recall: 3 / (3 + 2) = 0.6   (or 60%)

  F1-Score = 2 * 0.5 * 0.6 / (1.1) = 0.5454

Sensitivity and Specificity

- ***Sensitivity*(SEN)** measures the recovery rate of the Positives and complimentary, ***Specificity*(SPC)** measure the recovery rate of the Negatives.

- SEN = TPR = Recall = TP/P = TP/(TP+FN)
- SPC = TNR = TN/N = TN/(TN+FP)

Matthew's Correlation Coefficient

- Matthews correlation coefficient (MCC) was first formulated by Brian W. Matthews [1] in 1975 to assess the performance of protein secondary structure predictions
- The MCC can be understood as a specific case of a linear correlation coefficient (Pearson r) for a binary classification setting
- Considered as especially useful in unbalanced class settings
- The previous metrics take values in the range between 0 (worst) and 1 (best)
- The MCC is bounded between the range 1 (perfect correlation between ground truth and predicted outcome) and -1 (inverse or negative correlation) — a value of 0 denotes a random prediction.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

[1] Brian W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochima et Biophysica Acta (BBA) – Protein Structure, 402(2), 1975

Implementation : Confusion matrix

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from mlxtend.evaluate import confusion_matrix


pipe_knn = make_pipeline(StandardScaler(),
                         KNeighborsClassifier(n_neighbors=5))

pipe_knn.fit(X_train, y_train)

y_pred = pipe_knn.predict(X_test)

confmat = confusion_matrix(y_test, y_pred)

print(confmat)
```

```
[[71  1]
 [ 3 39]]
```

sklearn.metrics.confusion_matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

```
>>> tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
>>> (tn, fp, fn, tp)
(0, 2, 1, 1)
```

Precision, Recall, F1, MCC

```
from sklearn.metrics import accuracy_score, precision_score, \
                            recall_score, f1_score, matthews_corrcoef


print('Accuracy: %.3f' % accuracy_score(y_true=y_test, y_pred=y_pred))
print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
print('MCC: %.3f' % matthews_corrcoef(y_true=y_test, y_pred=y_pred))
```

```
Accuracy: 0.965
Precision: 0.975
Recall: 0.929
F1: 0.951
MCC: 0.925
```

https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_2_pre-recall-f1.ipynb

Balanced Accuracy / Average Per-Class (APC) Accuracy

$$ACC = \frac{T}{n}$$

Correct predictions

Total samples in dataset

Balanced Accuracy / Average Per-Class (APC) Accuracy

Predicted Labels

True Labels	Class 0	Class 1	Class 2
Class 0	3	0	0
Class 1	7	50	12
Class 2	0	0	18

$$ACC = \frac{3 + 50 + 18}{90} \approx 0.79$$

$$APC\ ACC = \frac{83/90 + 71/90 + 78/90}{3} \approx 0.86$$

Balanced Accuracy / Average Per-Class (APC) Accuracy

Predicted Labels

True Labels	Class 0	Neg Class
Class 0	3	0
Neg Class	7	80

Predicted Labels

True Labels	Class 1	Neg Class
Class 1	50	19
Neg Class	0	21

Predicted Labels

True Labels	Class 2	Neg Class
Class 2	18	0
Neg Class	12	60

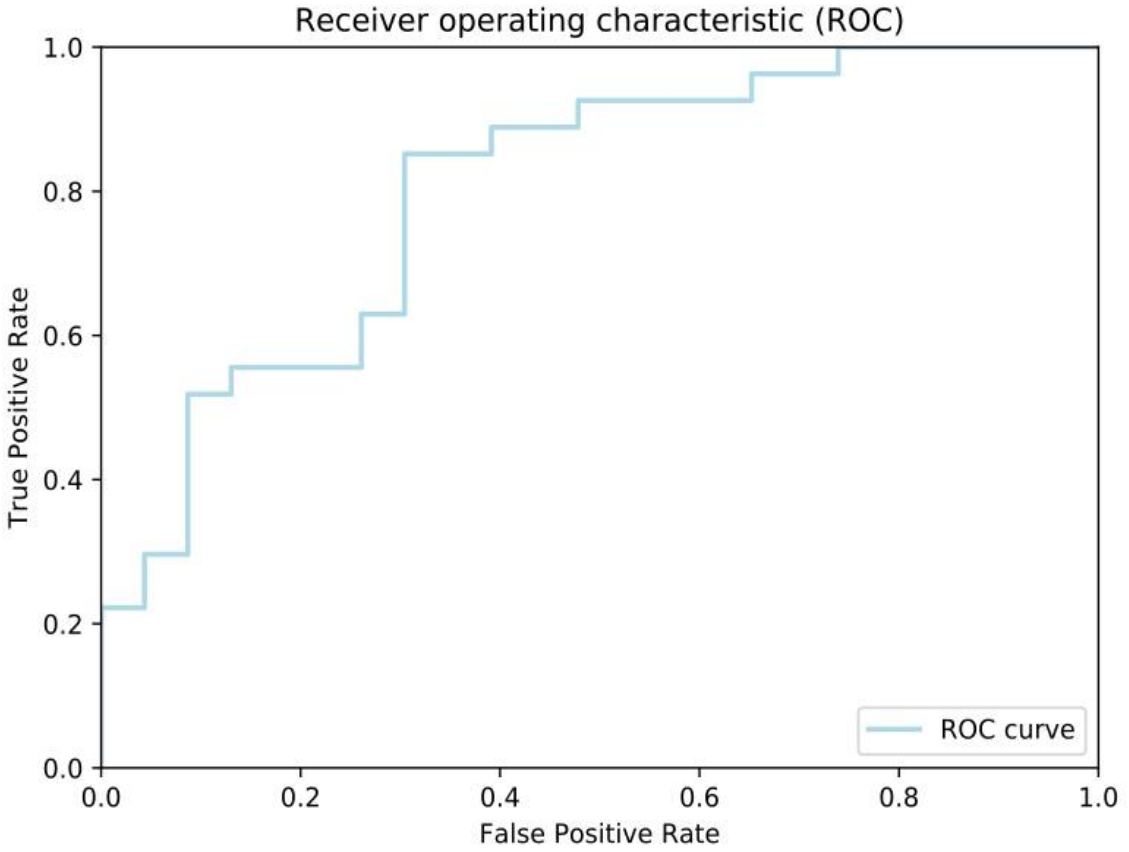
Predicted Labels

True Labels	Class 0	Class 1	Class 2
Class 0	3	0	0
Class 1	7	50	12
Class 2	0	0	18

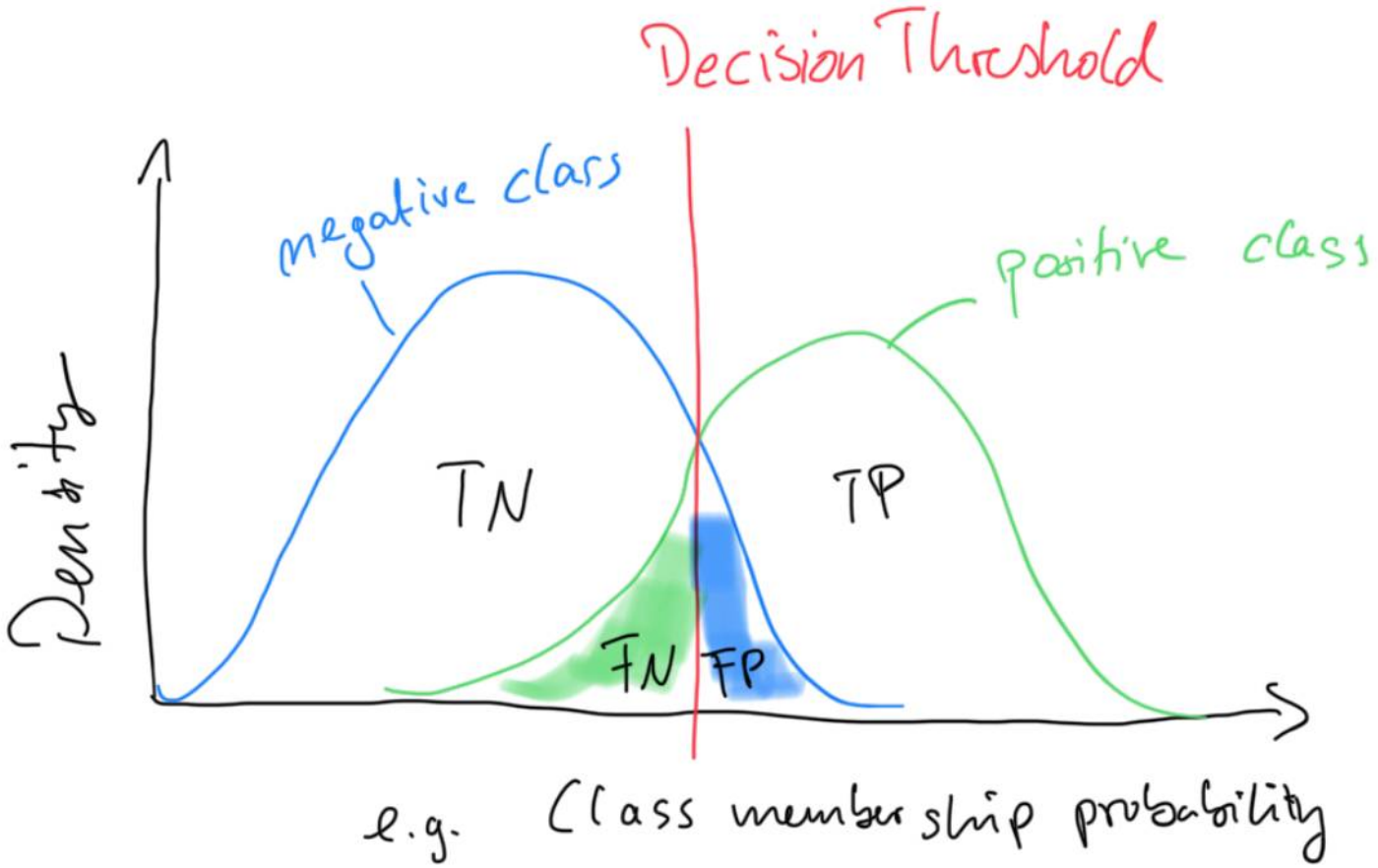
$$APC\ ACC = \frac{83/90 + 71/90 + 78/90}{3} \approx 0.86$$

Receiver Operating Characteristic Curve (ROC Curve)

- Trade-off between True Positive Rate(TPR) and False Positive Rate(FPR)
- ROC can be plotted by changing the prediction threshold

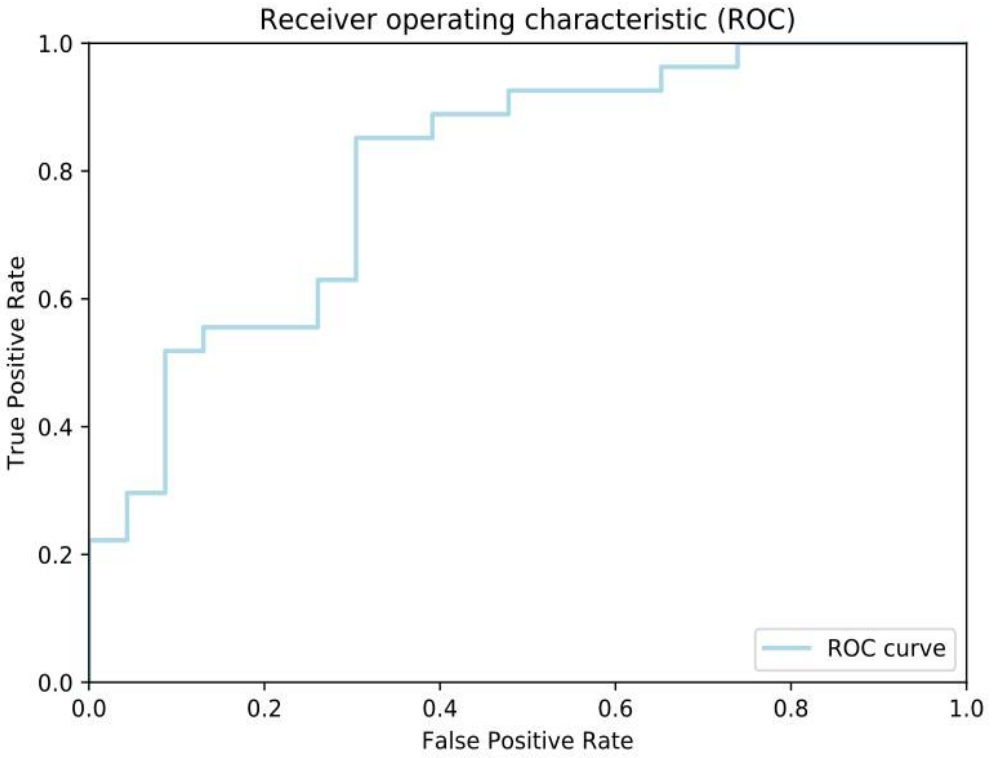


RECAP: False Positive Rate and False Negative Rate

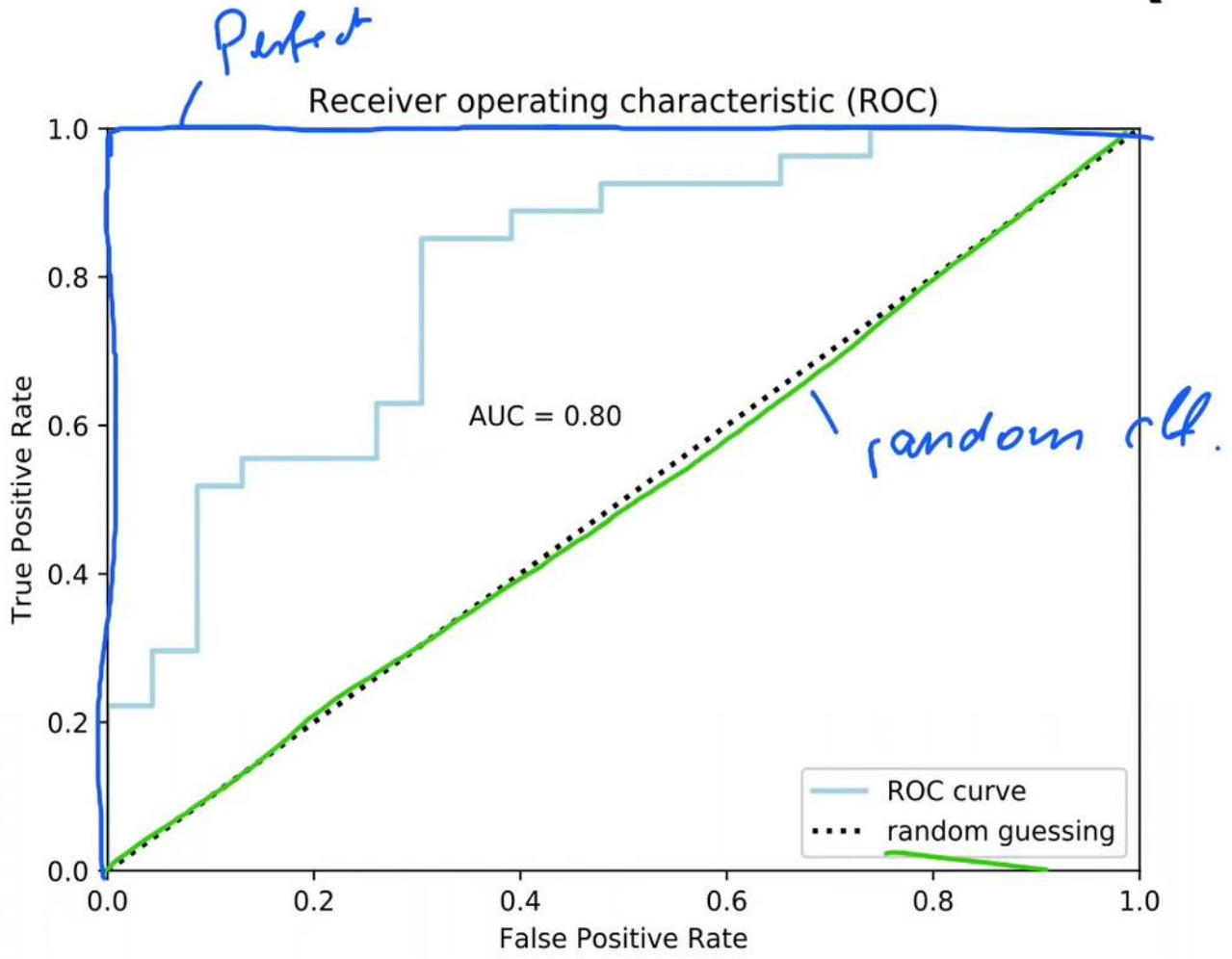


Receiver Operating Characteristic curve (ROC curve)

- ?.? = Perfect Prediction
- ?.? = Random Prediction



ROC Area Under the Curve (AUC)



ROC Curve Demo

https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#auc_and_roc_for_choosing_model_and_threshold

Model Selection and Data Splits

Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{m}\theta_j^2\right]$$

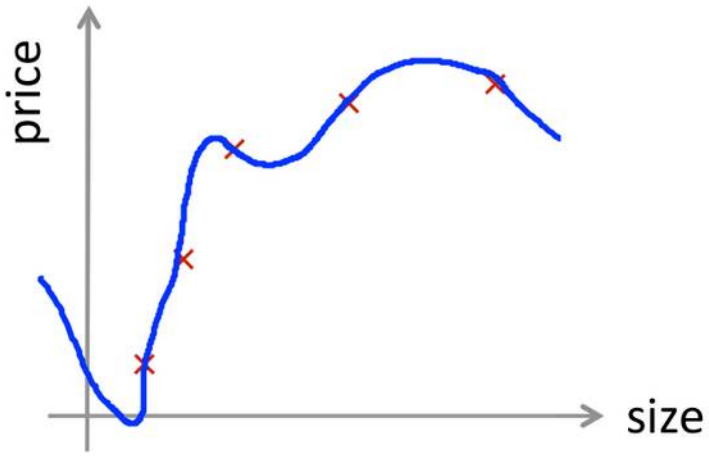
However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- Getting more training examples.
- Try smaller set of features.
- Try getting additional features.
- Try getting polynomial features (x_1^2, x_2^2, x_1x_2, etc.)
- Try decreasing λ
- Try increasing λ

Evaluating a hypothesis

- Suppose we train a model and we get a very low training error. But it fails to generalize on the new unseen examples.
- One reason can be that the hypothesis is overfitting
- How do we know that it is overfitting?

Evaluating your hypothesis



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Fails to generalize to new examples not in training set.

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}

Evaluating your hypothesis

Dataset:

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

70%

Training set

30%

Test set

$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
$\vdots$
$(x^{(m)}, y^{(m)})$

$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
$\vdots$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = no. of test example

$(x_{test}^{(i)}, y_{test}^{(i)})$

Random splits ensure unbiased evaluation estimate.

Andrew Ng

Evaluation on test set

- Linear Regression:

Compute test set error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)} \right)^2$$

- Logistic Regression:

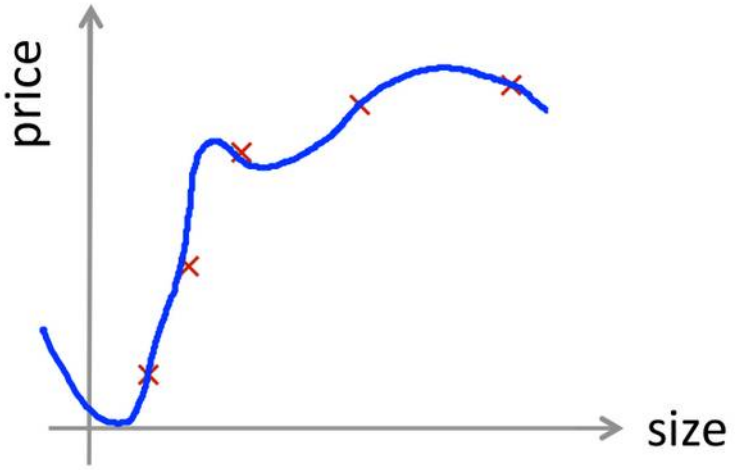
Compute test set error:

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log h_\theta(x_{test}^{(i)})$$

Model selection and training/validation and test sets

- If we want to decide what degree of polynomial features to use for the hypothesis while using logistic regression, or what should be the value of the regularization parameter λ.
- These are known as model selection problems.

Overfitting example



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Once parameters $\theta_0, \theta_1, \ldots, \theta_4$ were fit to some set of data (training set), the error of the parameters as measured on that data (the training error $J(\theta)$) is likely to be lower than the actual generalization error.

Model selection

1. $h_\theta(x) = \theta_0 + \theta_1 x$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$

$\vdots$

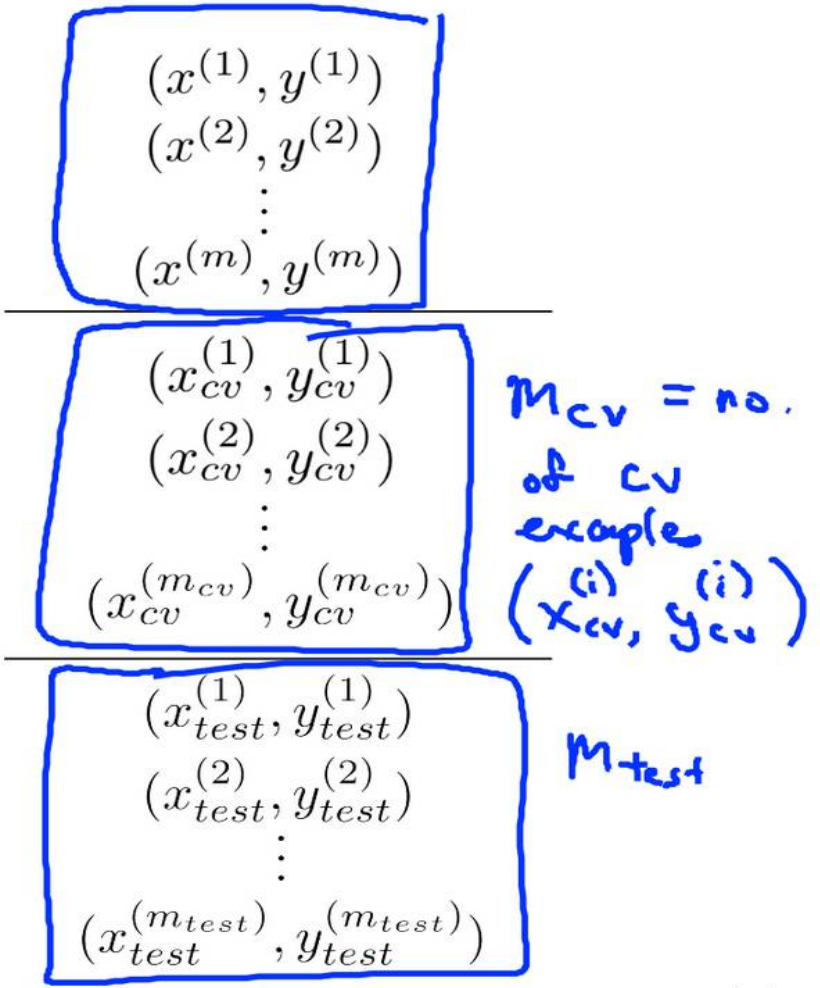
10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$

- Suppose, we train 10 models with different degrees of polynomial **(d)**, and evaluate on the test set. Finally, report the model with the lowest error.
- This will not be a fair estimate of the model. It is likely to be an optimistic estimate of the generalization error i.e., our extra parameter of **d** is fit to the test set.
- Therefore, we need a Cross Validation (or simply validation) set for model selection.

Evaluating your hypothesis

Dataset:

Size	Price	
2104	400	Training set (60%)
1600	330	
2400	369	
1416	232	
3000	540	
1985	300	
1534	315	Cross validation (CV) set (20%)
1427	199	
1380	212	test set (20%)
1494	243	



Andrew Ng

Train/validation/test error

Training error:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

Train / Val / Test splits: Points to note

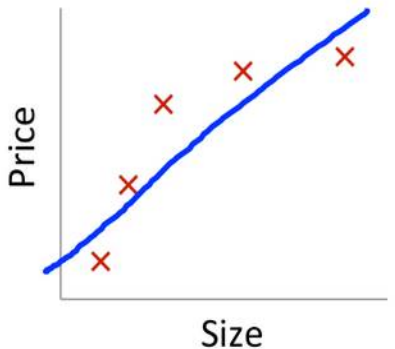
- The validation set helps in choosing model hyperparameters (such as degree of polynomial, or regularization parameter λ), while the training set will determine the parameters (Θ).
- To have an **unbiased estimate** of the performance, use the test set once on the trained model. Do not make any decisions based on the accuracy that you get on the test set.
- E.g., You may test the new models on a weekly basis in production environment, but if you revert to an older model (considering that in the current week the performance was worse), then the estimate will not remain unbiased.

Train / Val / Test splits: Points to note

- Generally the train val test splits are in the ratio 60%:20%:20. But it is quite arbitrary and depends on the dataset size and error improvement that we want to achieve.
- If the dataset is small, so we may want more data to be put in the training set. In this case, we may use **k-fold cross validation**. This divides the training data into k equal parts (k ranging from 1 to 10), train the model on k-1 folds and test on the remaining 1 fold. This is done for all the folds and validation accuracy is obtained by averaging all the k accuracy values.
- **Leave-One-Out CV** is an extreme case of k-fold CV, where we train on m-1 samples and test on one sample (k = m = training set size).

Bias Variance Tradeoff

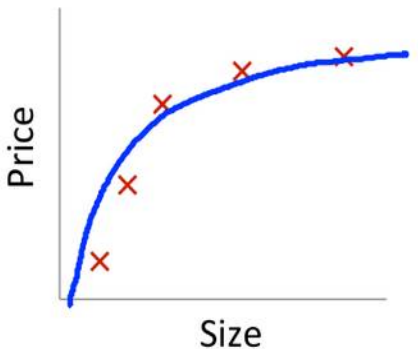
Bias/variance



$\theta_0 + \theta_1 x$

High bias
(underfit)

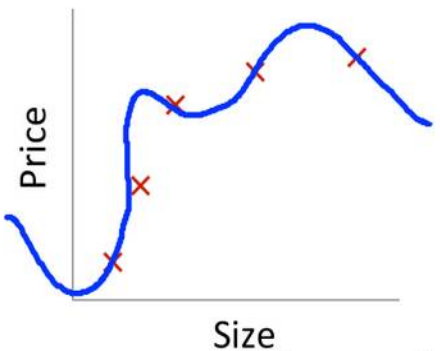
$d = 1$



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"

$d = 2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

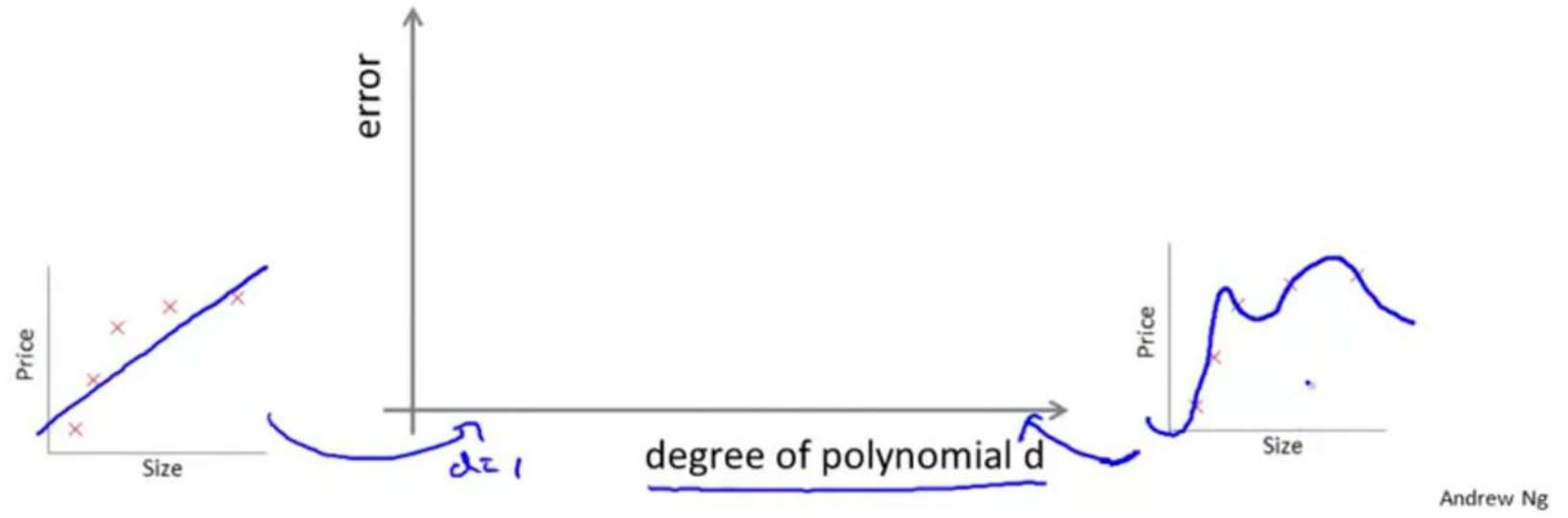
High variance
(overfit)

$d = 4$

Bias/variance

Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

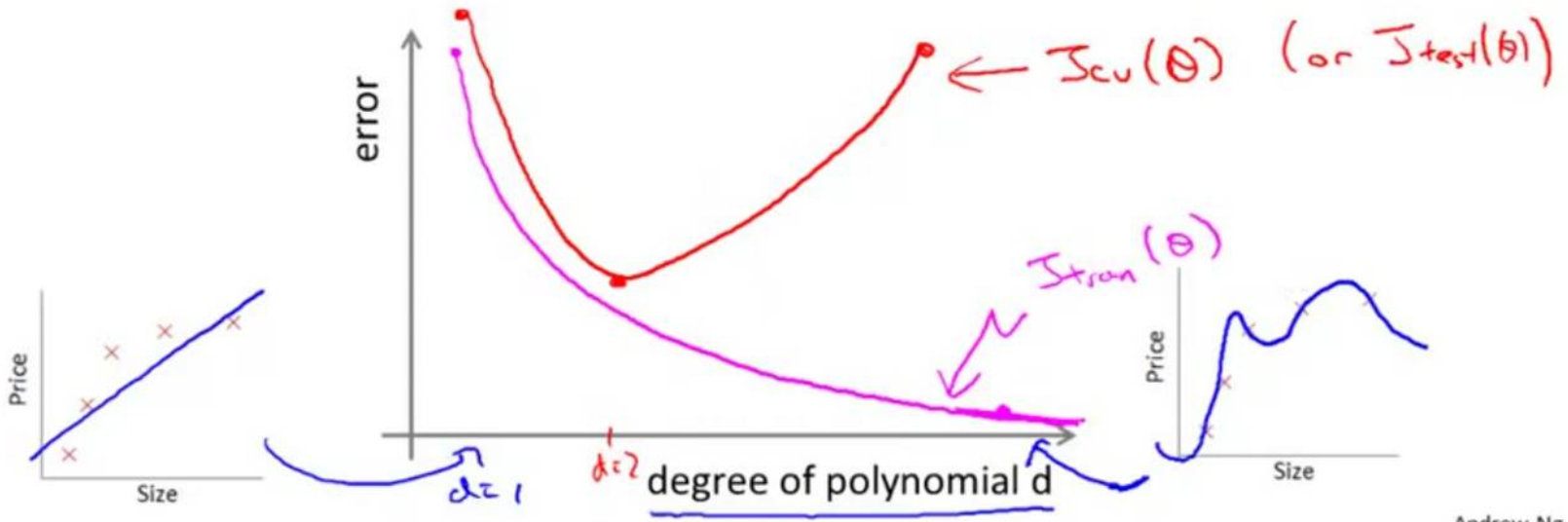
Cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$



Bias/variance

Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

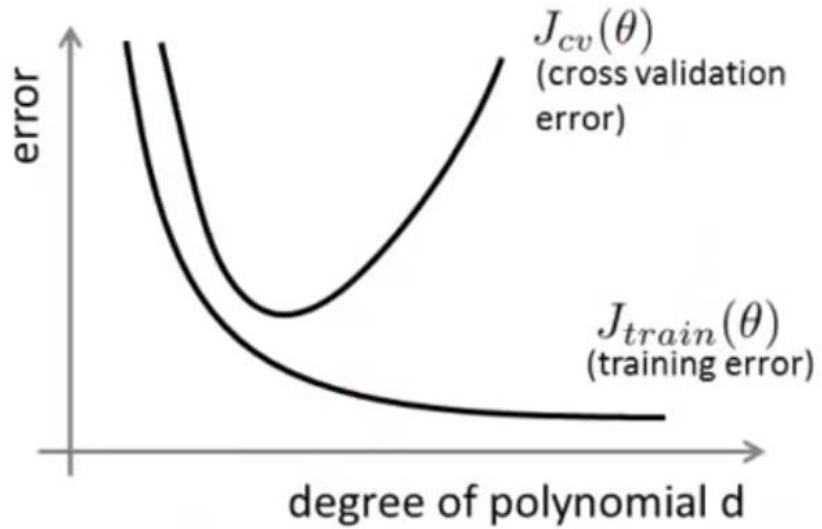
Cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$ (or $J_{test}(\theta)$)



Andrew Ng

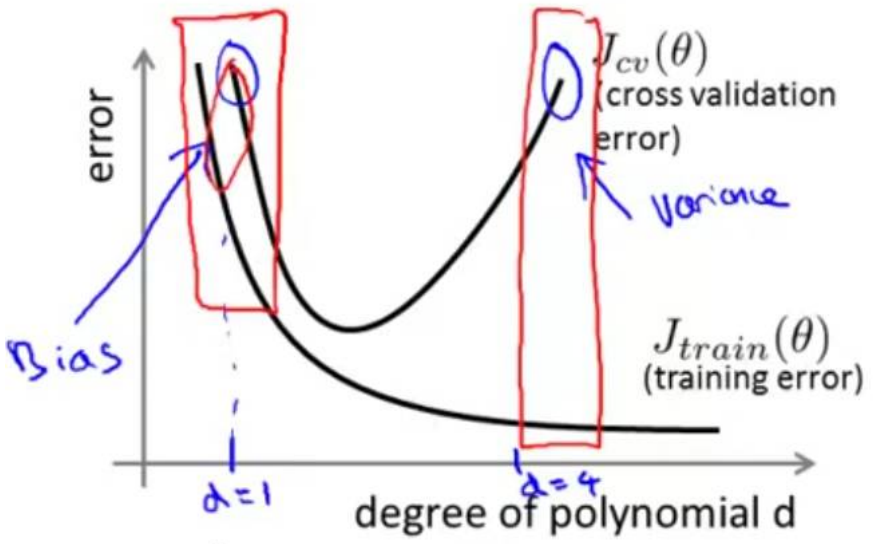
Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):

$J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):

$J_{train}(\theta)$ will be low

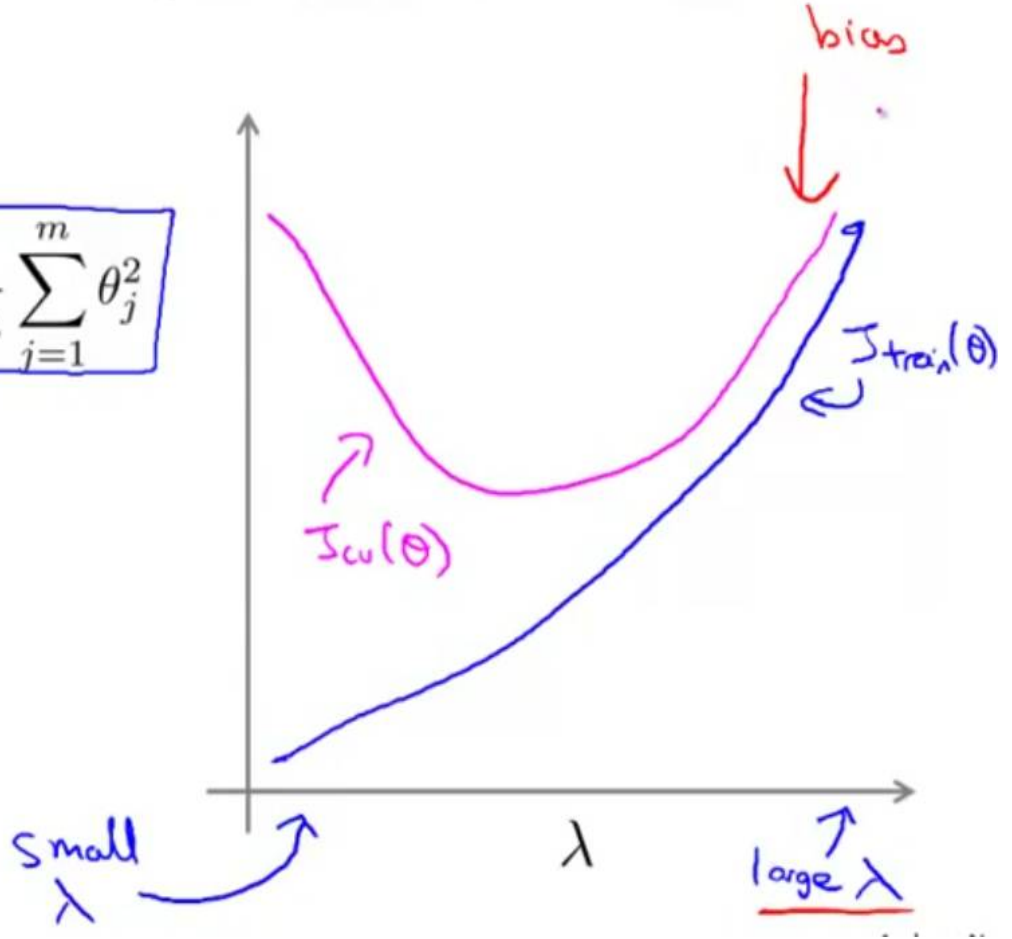
$J_{cv}(\theta) \gg J_{train}(\theta)$

Bias/variance as a function of the regularization parameter λ

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



References

- Sebastian Raschka – Slides + Code
  - https://github.com/rasbt/stat451-machine-learning-fs20/tree/master
- Andrew Ng: Coursera course on Introduction to Machine Learning
- Stanford CS229 (Andrew Ng).

End of Lecture