DS 5220

Supervised Machine Learning and Learning Theory

Alina Oprea
Associate Professor, CCIS
Northeastern University

October 7 2019

Logistics

- HW 2 is due on Oct. 8
- Exams
  - Midterm: Monday, Oct. 28
  - Final exam: Wednesday, Dec. 4
- Project
  - Proposal due on Oct. 21; teams of 2-3
  - Project presentation on Dec. 9
  - Project report due on Dec. 10
  - Project ideas and datasets posted on Piazza
  - Example projects from DS 4400 posted on Piazza

Project Proposal

- Project Title
- Project Team
- Problem Description
  - What is the prediction problem you are trying to solve?
- Dataset
  - Link to data, brief description, number of records, feature dimensionality (at least 10K records)
- Approach and methodology
  - Normalization
  - Feature selection
  - Machine learning models you will try (at least 3)
  - Splitting into training and testing, cross validation
  - Language and packages you plan to use
- Metrics (how you will evaluate your models)

Outline

- Logistic regression
  - Classification based on probability
  - Gradient descent for logistic regression
- Evaluation metrics
  - Confusion matrix
  - ROC curves
- Lab for linear classification

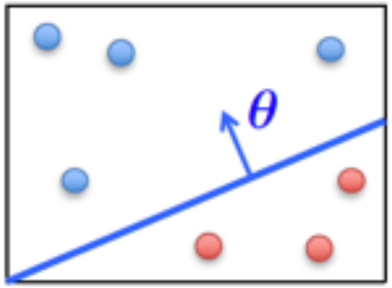
Review

- Regularization is a general method to avoid over-fitting
- Cross-validation should be performed to
  - Improve model generalization
  - Avoid over-fitting
  - Choose hyper parameters (k in kNN)
- Logistic regression is a linear classifier that predicts class probability
  - Classification based on probability; interpretability
  - MLE objective: Cross-entropy loss

Linear Classifiers

- **Linear classifiers**: represent decision boundary by hyperplane

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{x}^\mathsf{T} = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$



$h_\theta(x) = f(\theta^T x)$ linear function

- If $\theta^T x > 0$ classify 1
- If $\theta^T x < 0$ classify 0

All the points x on the hyperplane satisfy: $\theta^T x = 0$

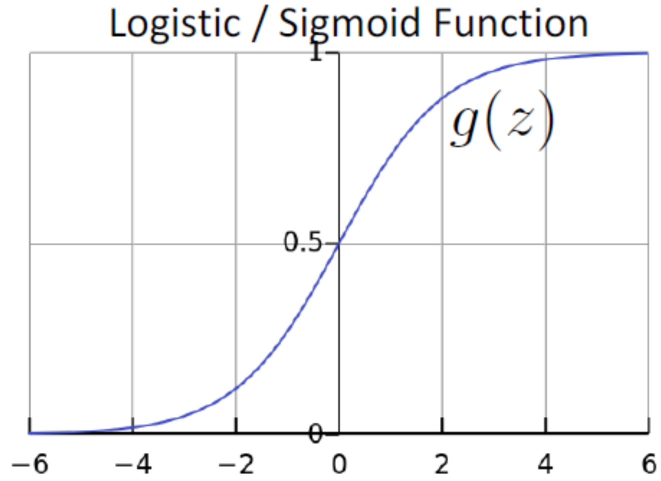
Logistic Regression

- Takes a probabilistic approach to learning discriminative functions (i.e., a classifier)
- $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ should give $P(Y = 1 | X; \theta)$
  - Want $0 \leq h_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq 1$
- Logistic regression model:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}\right)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}}}$$



Logistic Regression is a linear classifier!

Cross-Entropy Objective

$$\mathrm{P}(Y = y_i | X = x_i; \theta) = h_\theta(x_i)^{y_i}\left(1 - h_\theta(x_i)\right)^{1-y_i}$$

$$\theta_{MLE} = \mathrm{argmax}_\theta \sum_{i=1}^{N} \log P[Y = y_i | X = x_i; \theta]$$

$$= \mathrm{argmax}_\theta \sum_{i=1}^{N} y_i \log h_\theta(x_i) + (1 - y_i) \log\left(1 - h_\theta(x_i)\right)$$

Logistic regression objective

$$\min_\theta J(\theta)$$

$$J(\theta) = -\sum_{i=1}^{N} \left[y_i \log h_\theta(x_i) + (1 - y_i) \log\left(1 - h_\theta(x_i)\right)\right]$$

Gradient Descent for Logistic Regression

$$J(\theta) = -\sum_{i=1}^{N}[y_{\rm i}\log h_\theta(x_i) + (1-y_i)\log\left(1-h_\theta(x_i)\right)]$$

$$J(\theta) = -\sum_{i=1}^{n} C_i$$

Want $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update for j = 0 ... d

Computing Gradients

- Derivative of sigmoid
  - $g(z) = \frac{1}{1+e^{-z}}$; $g'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = g(z)(1 - g(z))$
- Derivative of hypothesis
  - $h_\theta(x) = g(\theta^T x) = g(\theta_j x_j + \sum_{k \neq j} \theta_k x_k)$
  - $\frac{\partial h_\theta(x)}{\partial \theta_j} = \frac{\partial g(\theta^T x)}{\partial \theta_j} x_j = g(\theta^T x)\big(1 - g(\theta^T x)\big) x_j$
- Derivation of C_i
  - $\frac{\partial C_i}{\partial \theta_j} = y_i \frac{1}{h_\theta(x_i)} \mathrm{g}(\theta^T x_i)\big(1 - \mathrm{g}(\theta^T x_i)\big) x_{ij}$ -

    $(1 - y_i) \frac{1}{1-h_\theta(x_i)} \mathrm{g}(\theta^T x_i)\big(1 - \mathrm{g}(\theta^T x_i)\big) x_{ij}$

    $= \big(y_i - h_\theta(x_i)\big) x_{ij}$

Gradient Descent for Logistic Regression

$$J(\theta) = -\sum_{i=1}^{N} [y_{\mathrm{i}} \log h_\theta(x_i) + (1 - y_i) \log\left(1 - h_\theta(x_i)\right)]$$

Want $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence (simultaneous update for j = 0 ... d)

$$\theta_0 \leftarrow \theta_0 - \alpha \sum_{i=1}^{N} (h_\theta(x_i) - y_i)$$

$$\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{N} (h_\theta(x_i) - y_i) x_{ij}$$

Gradient Descent for Logistic Regression

Want $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence (simultaneous update for j = 0 ... d)

$$\theta_0 \leftarrow \theta_0 - \alpha \sum_{i=1}^{N} (h_\theta(x_i) - y_i)$$

$$\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{N} (h_\theta(x_i) - y_i) x_{ij}$$

This looks IDENTICAL to Linear Regression!

- However, the form of the model is very different:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}}}$$

Regularized Logistic Regression

$$J(\theta) = -\sum_{i=1}^{N}[y_\mathrm{i}\log h_\theta(x_i) + (1 - y_i)\log\left(1 - h_\theta(x_i)\right)]$$

- We can regularize logistic regression exactly as before:

$$\begin{aligned} J_{\text{regularized}}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \lambda \sum_{j=1}^{d} \theta_j^2 \\ &= J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2 \end{aligned}$$

L2 regularization

Classifier Evaluation

- Classification is a supervised learning problem
  - Prediction is binary or multi-class
- Classification techniques
  - Linear classifiers
    - Perceptron (online or batch mode)
    - Logistic regression (probabilistic interpretation)
  - Instance learners
    - kNN: need to store entire training data
- Cross-validation should be used for parameter selection and estimation of model error

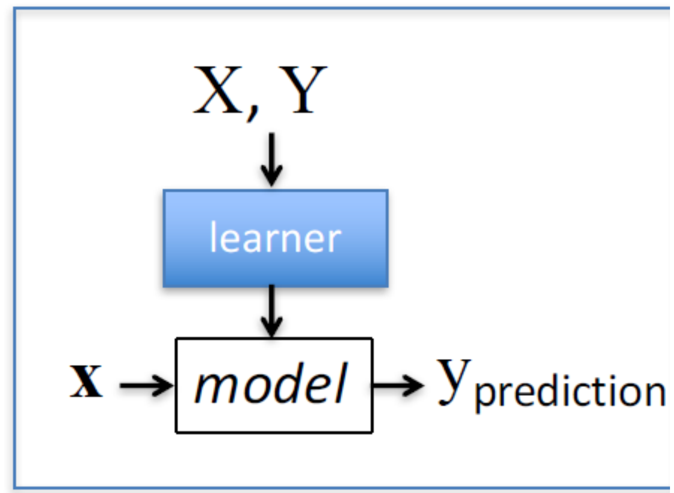
Evaluation of classifiers

Given: labeled training data $X, Y = \{\langle \boldsymbol{x}_i, y_i \rangle\}_{i=1}^{n}$

- Assumes each $\boldsymbol{x}_i \sim \mathcal{D}(\mathcal{X})$

Train the model:

model ← *classifier*.train(X, Y)



Apply the model to new data:

- Given: new unlabeled instance $\boldsymbol{x} \sim \mathcal{D}(\mathcal{X})$

$y_{\text{prediction}}$ ← *model*.predict($\mathbf{x}$)

Classification Metrics

$$\text{accuracy} = \frac{\#\ \text{correct predictions}}{\#\ \text{test instances}}$$

$$\text{error} = 1 - \text{accuracy} = \frac{\#\ \text{incorrect predictions}}{\#\ \text{test instances}}$$

- Training set accuracy and error
- Testing set accuracy and error

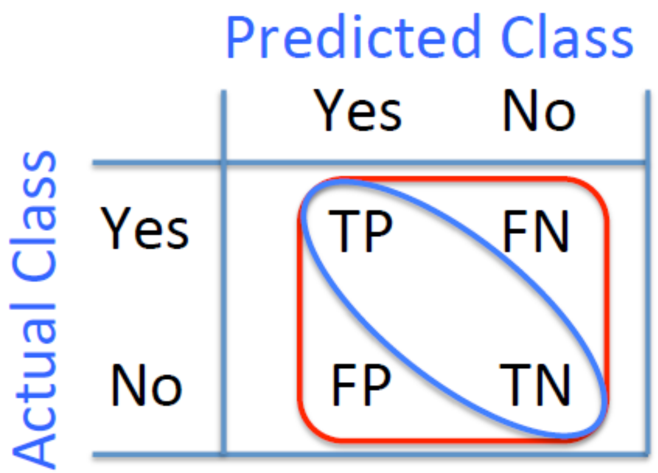
Confusion Matrix

Given a dataset of P positive instances and N negative instances:

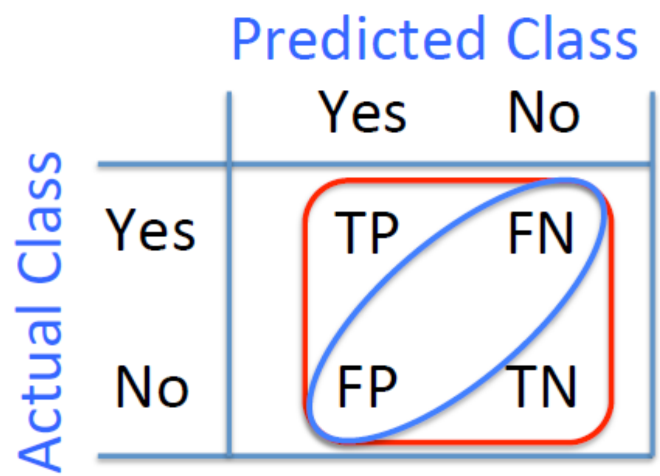
		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

Accuracy and Error

Given a dataset of P positive instances and N negative instances:



$$\text{accuracy} = \frac{TP + TN}{P + N}$$



$$\text{error} = 1 - \frac{TP + TN}{P + N}$$

$$= \frac{FP + FN}{P + N}$$

Confusion Matrix

- Given a dataset of P positive instances and N negative instances:

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

$$\text{accuracy} = \frac{TP + TN}{P + N}$$

- Imagine using classifier to identify positive cases (i.e., for information retrieval)

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

Probability that classifier predicts positive correctly

Probability that actual class is predicted correctly

Why One Metric is Not Enough

Assume that in your training data, Spam email is 1% of data, and Ham email is 99% of data

- Scenario 1
  - Have classifier always output HAM!
  - What is the accuracy? 99%
- Scenario 2
  - Predict one SPAM email as SPAM, all other emails as legitimate
  - What is the precision? 100%
- Scenario 3
  - Output always SPAM!
  - What is the recall? 100%

Precision & Recall

Precision

- the fraction of positive predictions that are correct
- P(is pos|predicted pos)

$$\text{precision} = \frac{TP}{TP + FP}$$

Recall

- fraction of positive instances that are identified
- P(predicted pos|is pos)

$$\text{recall} = \frac{TP}{TP + FN}$$

- You can get high recall (but low precision) by only predicting positive
- Recall is a non-decreasing function of the # positive predictions
- Typically, precision decreases as either the number of positive predictions or recall increases
- Precision & recall are widely used in information retrieval

F-Score

- Combined measure of precision/recall tradeoff

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

  - This is the harmonic mean of precision and recall
  - In the F_1 measure, precision and recall are weighted evenly
  - Can also have biased weightings that emphasize either precision or recall more (F_2 = 2 × recall; $F_{0.5}$ = 2 × precision)

- Limitations:
  - F-measure can exaggerate performance if balance between precision and recall is incorrect for application
    - Don't typically know balance ahead of time

A Word of Caution

- Consider binary classifiers A, B, C:

		A	.	B	.	C	.
		1	0	1	0	1	0
Predictions	1	0.9	0.1	0.8	0	0.78	0
	0	0	0	0.1	0.1	0.12	0.1

- Clearly A is useless, since it always predicts 1
- B is slightly better than C
  - less probability mass wasted on the off-diagonals
- But, here are the performance metrics:

Metric	A	B	C
Accuracy	0.9	0.9	0.88
Precision	0.9	1.0	1.0
Recall	1.0	0.888	0.8667
F-score	0.947	0.941	0.9286

Classifiers can be tuned

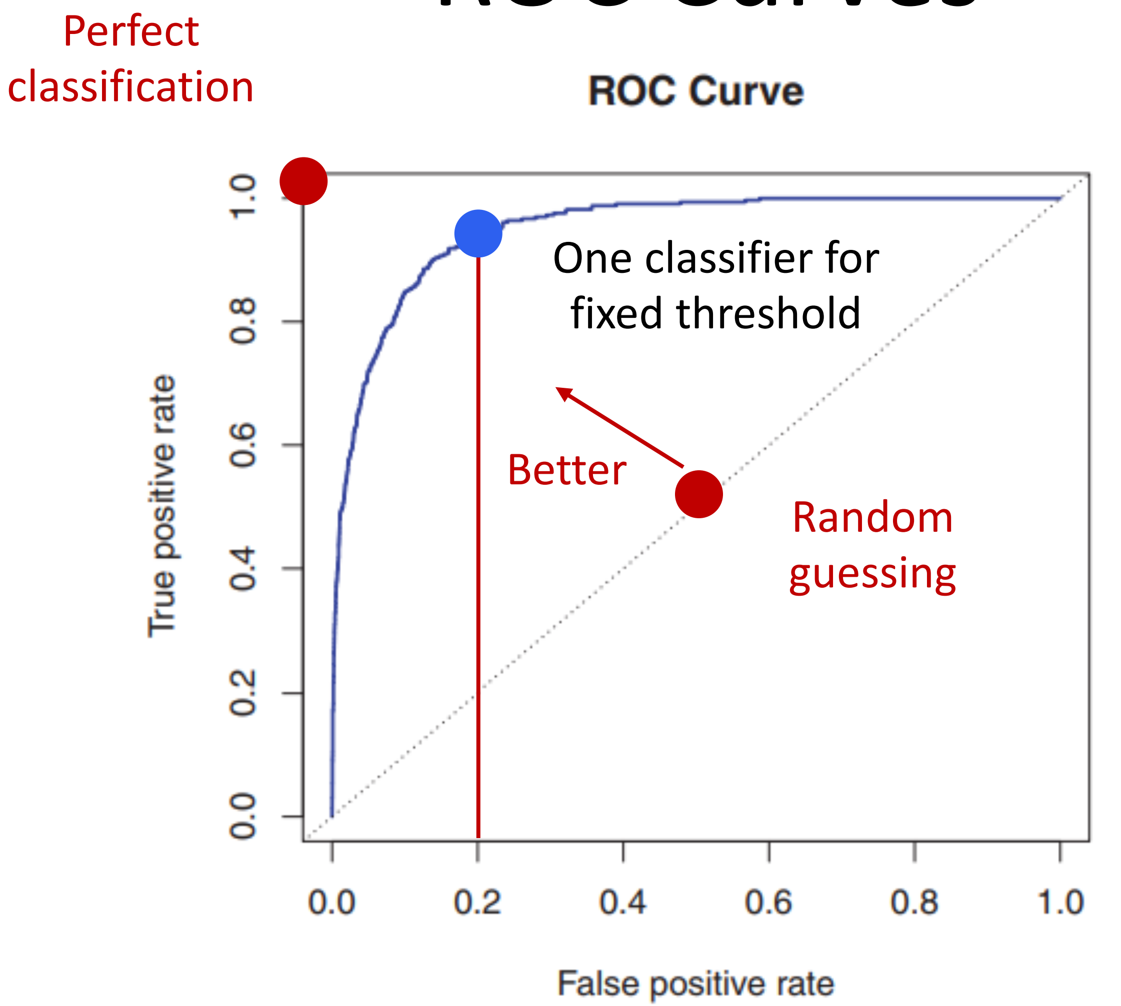
- Logistic regression sets by default the threshold at 0.5 for classifying positive and negative instances
- Some applications have strict constraints on false positives (or other metrics)
  - Example: very low false positives in security (spam)
- Solution: choose different threshold

Probabilistic model $h_{\theta(x)} = \mathrm{P}[y = 1 | x; \theta]$

- Predict y = 1 if $h_{\theta}(x) \geq$ T
- Predict y = 0 if $h_{\theta}(x) <$ T

Higher T, lower FP
Lower T, lower FN

ROC Curves

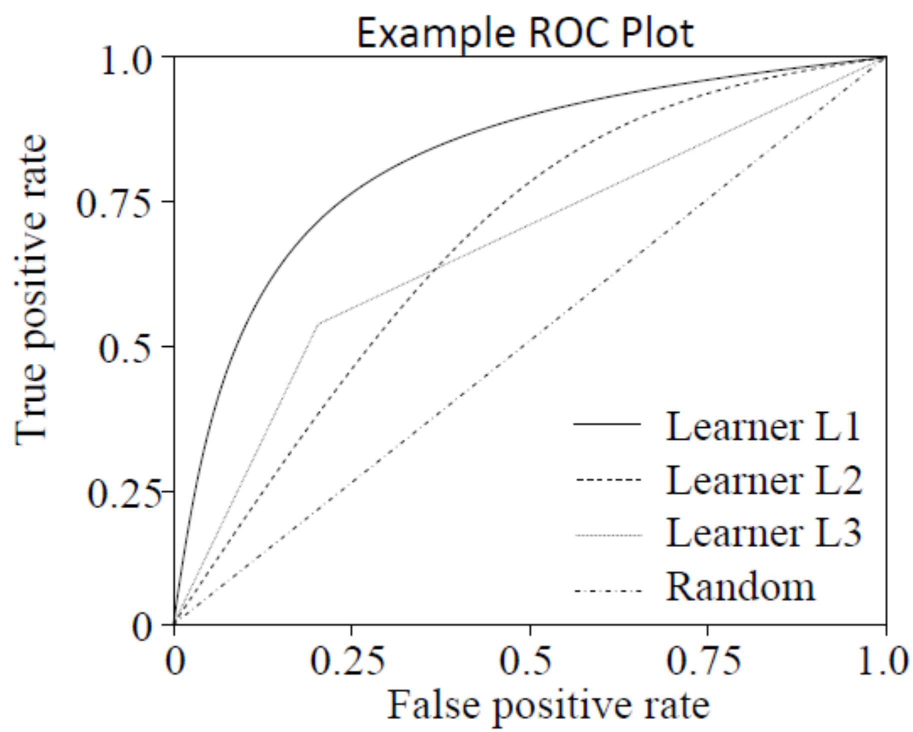


- Receiver Operating Characteristic (ROC)
- Determine operating point (e.g., by fixing false positive rate)

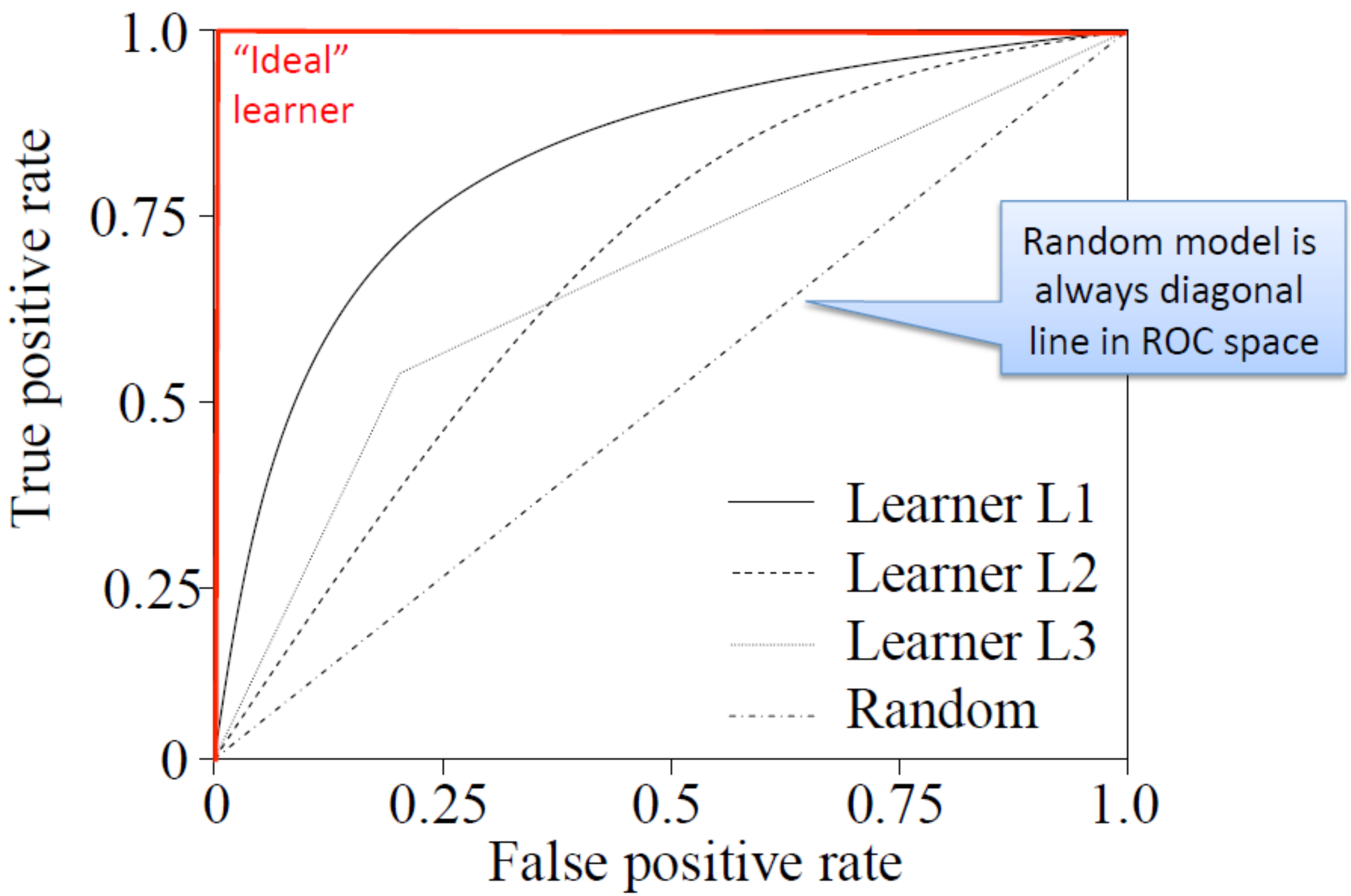
Performance Depends on Threshold

Predict positive if $\mathrm{P}(y = 1 \mid \mathbf{x}) > \theta$, otherwise negative

- Number of TPs and FPs depend on threshold θ
- As we vary θ, we get different (TPR, FPR) points

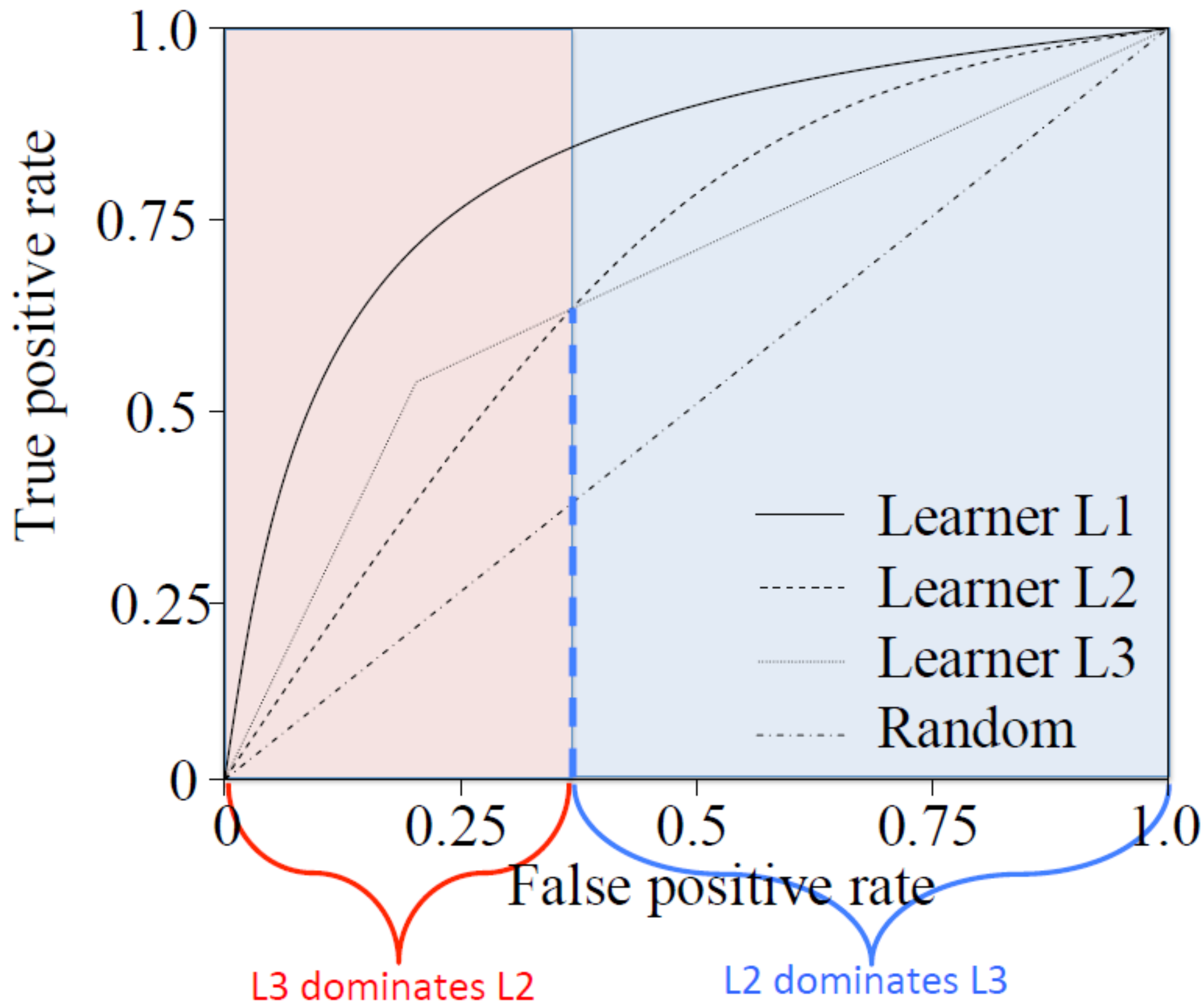


ROC Curve

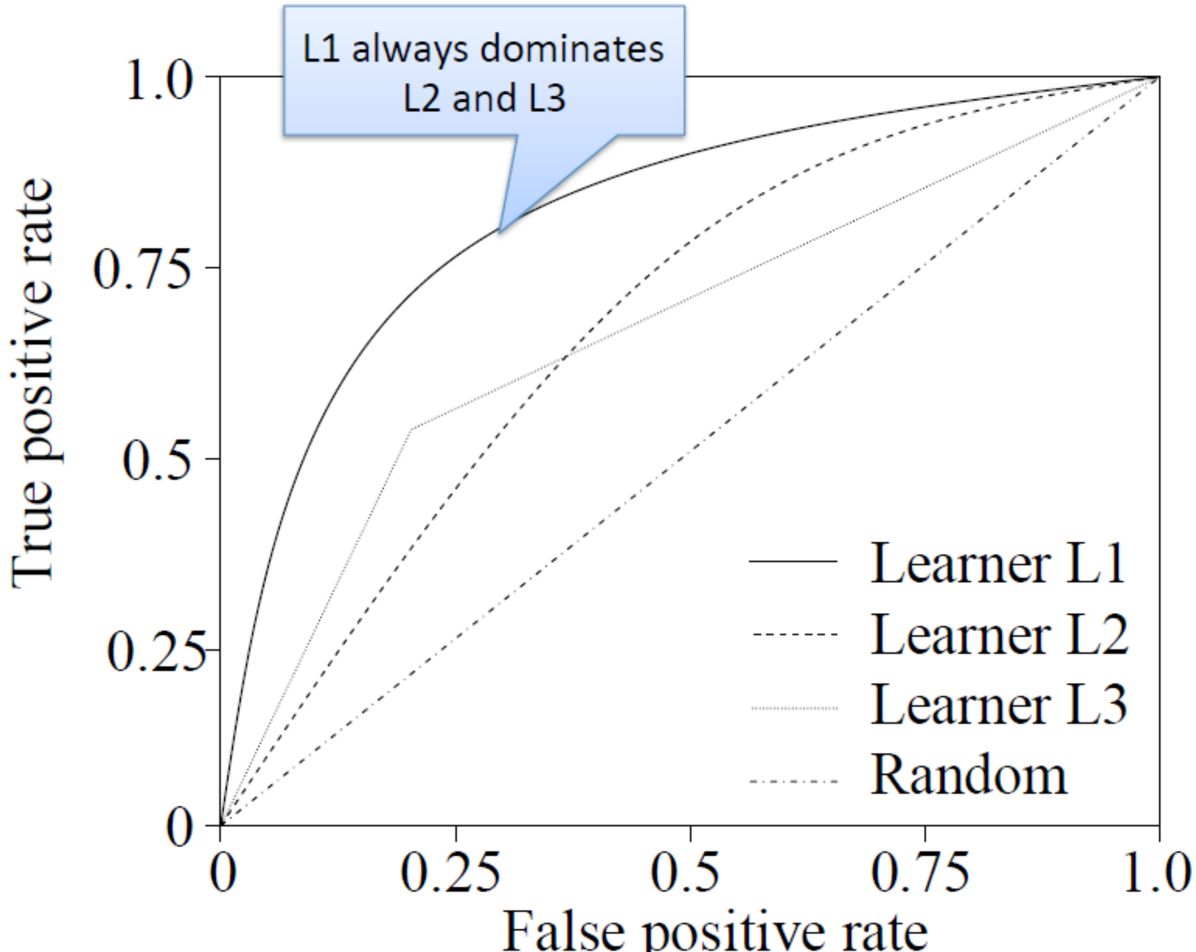


1.0
0.75
0.5
0.25
0
True positive rate
“Ideal” learner
Random model is always diagonal line in ROC space
Learner L1
Learner L2
Learner L3
Random
0
0.25
0.5
0.75
1.0
False positive rate

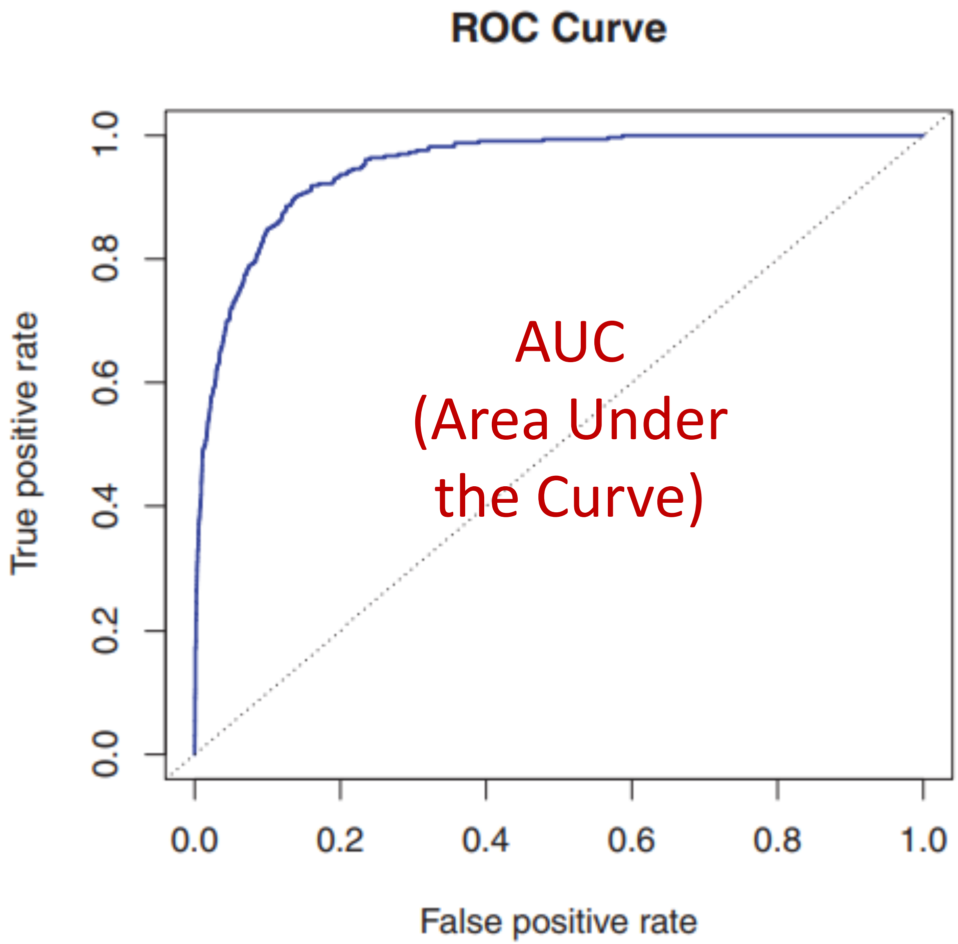
ROC Curve


1.0
0.75
0.5
0.25
0
True positive rate
0
0.25
0.5
0.75
1.0
False positive rate
Learner L1
Learner L2
Learner L3
Random
L3 dominates L2
L2 dominates L3

ROC Curve


L1 always dominates L2 and L3
1.0
0.75
0.5
0.25
0
True positive rate
0
0.25
0.5
0.75
1.0
False positive rate
Learner L1
Learner L2
Learner L3
Random

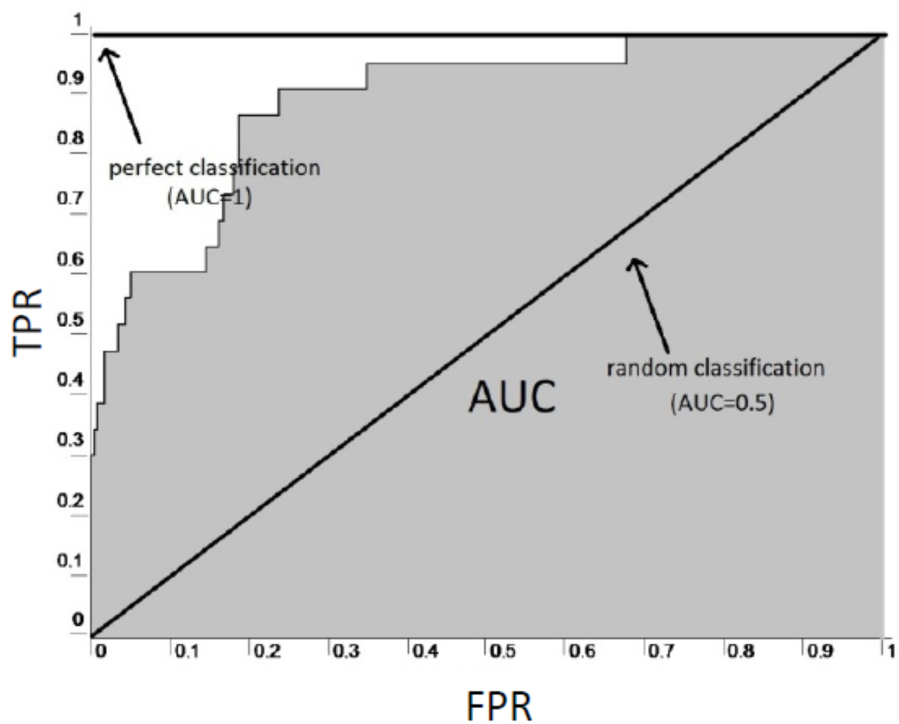
ROC Curves

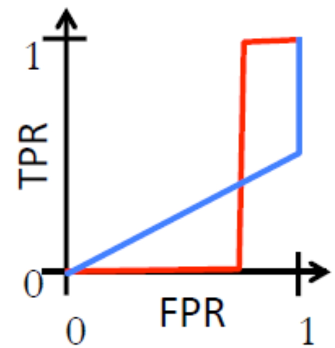


- Another useful metric: Area Under the Curve (AUC)
- The closest to 1, the better!

Area Under the ROC Curve

- Can take area under the ROC curve to summarize performance as a single number
  - Be cautious when you see only AUC reported without a ROC curve; AUC can hide performance issues

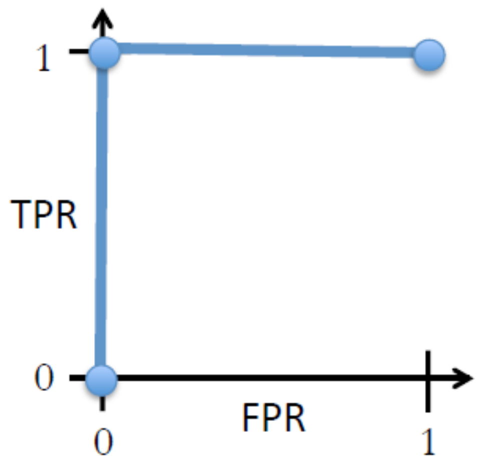




Same AUC, very different performance

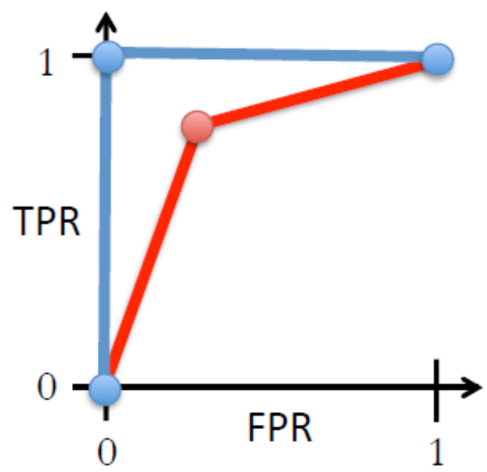
ROC Example

i	y_i	$p(y_i = 1 \mid \mathbf{x}_i)$	$h(\mathbf{x_i} \mid \theta = 0)$	$h(\mathbf{x_i} \mid \theta = 0.5)$	$h(\mathbf{x_i} \mid \theta = 1)$
1	1	0.9	1	1	0
2	1	0.8	1	1	0
3	1	0.7	1	1	0
4	1	0.6	1	1	0
5	1	0.5	1	1	0
6	0	0.4	1	0	0
7	0	0.3	1	0	0
8	0	0.2	1	0	0
9	0	0.1	1	0	0
			$TPR = 5/5 = 1$ $FPR = 4/4 = 1$	$TPR = 5/5 = 1$ $FPR = 0/4 = 0$	$TPR = 0/5 = 0$ $FPR = 0/4 = 0$



ROC Example

i	y_i	$p(y_i = 1 \mid \mathbf{x}_i)$	$h(\mathbf{x_i} \mid \theta = 0)$	$h(\mathbf{x_i} \mid \theta = 0.5)$	$h(\mathbf{x_i} \mid \theta = 1)$
1	1	0.9	1	1	0
2	1	0.8	1	1	0
3	1	0.7	1	1	0
4	1	0.6	1	1	0
5	1	**0.2**	1	**0**	0
6	0	**0.6**	1	**1**	0
7	0	0.3	1	0	0
8	0	0.2	1	0	0
9	0	0.1	1	0	0
			$TPR = 5/5 = 1$	$TPR = 4/5 = 0.8$	$TPR = 0/5 = 0$
			$FPR = 4/4 = 1$	$FPR = 1/4 = 0.25$	$FPR = 0/4 = 0$



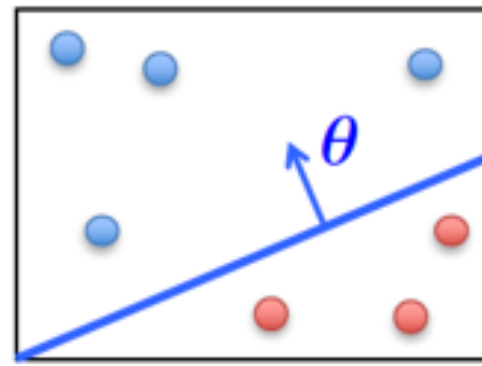
Linear models

- Perceptron

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x})$$

- Logistic regression

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\intercal \boldsymbol{x}}}$$



- LDA

$$Max_k \;\; \delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

LDA vs Logistic Regression

- Logistic regression computes directly $\Pr[Y = 1|X = x]$ by assuming sigmoid function
  - Uses Maximum Likelihood Estimation
  - Discriminative Model
- LDA uses Bayes Theorem to estimate it
  - Estimates mean, co-variance, and prior from training data
  - Generative model
  - Assumes Gaussian distribution for $f_k(x) = \Pr[X = x|Y = k]$
- Which one is better?
  - LDA can be sensitive to outliers
  - LDA works well for Gaussian distribution
  - Logistic regression is more complex to solve, but more expressive

Linear Classifier Lab

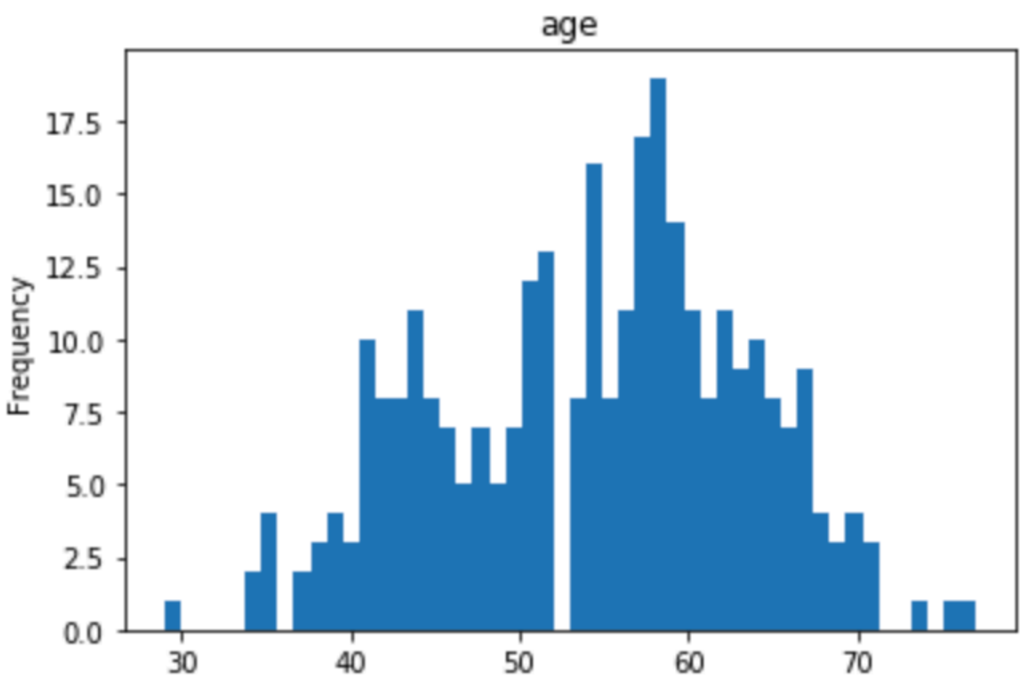
```
data = pd.read_csv('heart.csv')
data = data.dropna()
x_columns = data.columns != 'target'
data = utils.shuffle(data)
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
215	43	0	0	132	341	1	0	136	1	3.0	1	0	3	0
145	70	1	1	156	245	0	0	143	0	0.0	2	0	2	1
190	51	0	0	130	305	0	1	142	1	1.2	1	0	3	0
90	48	1	2	124	255	1	1	175	0	0.0	2	2	2	1
166	67	1	0	120	229	0	0	129	1	2.6	1	2	3	0

https://www.kaggle.com/ronitf/heart-disease-uci

Lab, cont.

```
import matplotlib.pyplot as plt

for column in ('age', 'chol', 'trestbps'):
    plt.hist(data[column], bins=50)
    plt.gca().set(title=column, ylabel='Frequency')
    plt.show()
```



Lab, Logistic Regression

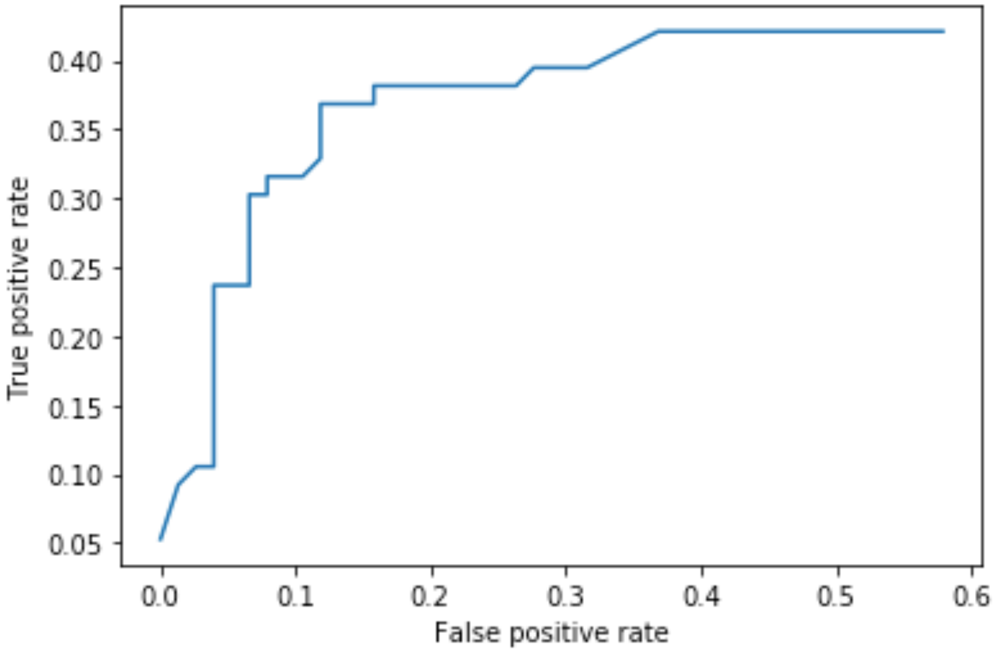
```
split = int(len(data) * 3/4)
x, y = data.loc[:, data.columns != 'target'], data['target']
x_train, x_test = x.iloc[:split], x.iloc[split:]
y_train, y_test = y.iloc[:split], y.iloc[split:]

logit_model = sm.Logit(y_train, x_train)
result = logit_model.fit()
print(result.summary2())
```

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	0.0221	0.0222	0.9955	0.3195	-0.0214	0.0656
sex	-1.8552	0.5683	-3.2643	0.0011	-2.9691	-0.7413
cp	0.8517	0.2224	3.8290	0.0001	0.4158	1.2877
trestbps	-0.0208	0.0113	-1.8346	0.0666	-0.0431	0.0014
chol	-0.0036	0.0042	-0.8532	0.3935	-0.0118	0.0046
fbs	0.5359	0.6673	0.8031	0.4219	-0.7720	1.8439
restecg	0.3930	0.4229	0.9292	0.3528	-0.4359	1.2219
thalach	0.0318	0.0100	3.1650	0.0016	0.0121	0.0514
exang	-0.6497	0.4920	-1.3203	0.1867	-1.6140	0.3147
oldpeak	-0.5583	0.2725	-2.0492	0.0404	-1.0923	-0.0243
slope	1.0992	0.4634	2.3721	0.0177	0.1910	2.0074
ca	-0.7973	0.2362	-3.3754	0.0007	-1.2603	-0.3344
thal	-1.0147	0.3628	-2.7973	0.0052	-1.7257	-0.3038

Metrics for LR

```
predictions = result.predict(x_test)
print_metrics(predictions, y_test)
plot_roc(predictions)
```

```
TPR: 0.88
FPR: 0.20
TNR: 0.80
FNR: 0.12
```



Plot Metrics and ROC

```
def plot_roc(test_predictions):
    tprs, fprs = [], []
    for thresh in range(0, 100, 1):
        predicted_labels = np.array(list(map(int, (test_predictions > thresh / 100))))
        tpr = sum((y_test == 1) & (predicted_labels == True)) / len(y_test)
        fpr = sum((y_test == 0) & (predicted_labels == True)) / len(y_test)
        tprs.append(tpr)
        fprs.append(fpr)

    plt.figure().add_subplot(111, xlabel="False positive rate", ylabel="True positive rate")
    plt.plot(fprs, tprs)
    plt.show()

def print_metrics(y_pred, y_true):
    y_pred = np.array(list(map(int, (y_pred > .5))))
    print("TPR: %.2f" % (sum((y_true == 1) & (y_pred == 1)) / sum(y_true == 1)))
    print("FPR: %.2f" % (sum((y_true == 0) & (y_pred == 1)) / sum(y_true == 0)))
    print("TNR: %.2f" % (sum((y_true == 0) & (y_pred == 0)) / sum(y_true == 0)))
    print("FNR: %.2f" % (sum((y_true == 1) & (y_pred == 0)) / sum(y_true == 1)))
```

Lab LDA

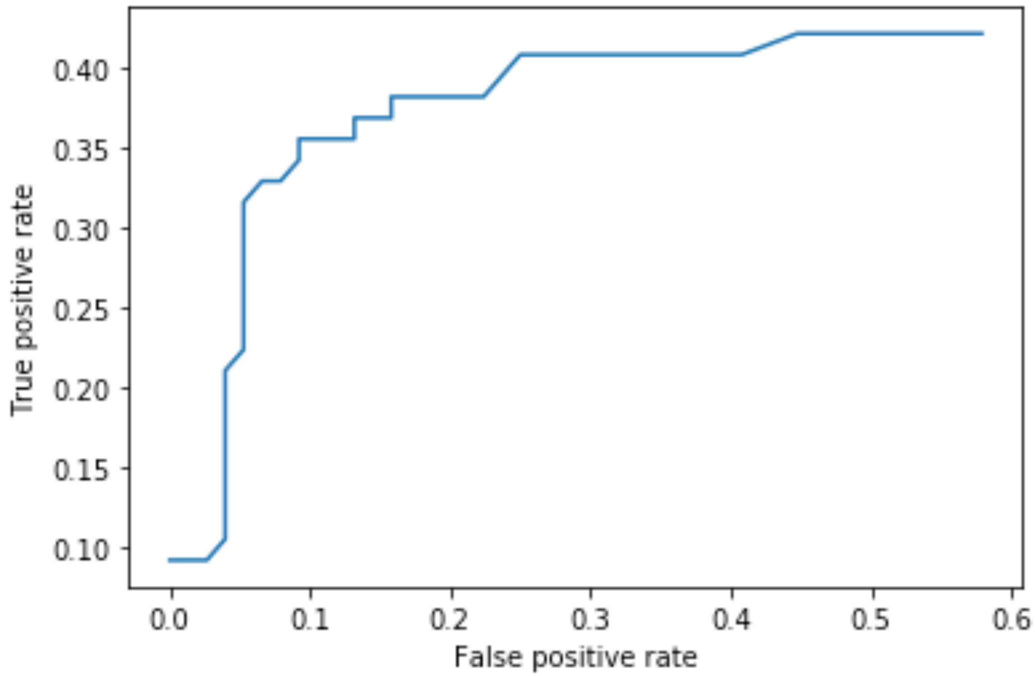
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis()
lda.fit(x_train, y_train)
print('Priors:')
print(lda.priors_)
print('Means:')
print(lda.means_)
print('Coefficients:')
print(lda.coef_)
print('Test Accuracy:')
print(lda.score(x_test, y_test))
```

```
Priors:
[0.41409692 0.58590308]
Means:
[[5.70744681e+01 8.19148936e-01 4.78723404e-01 1.34882979e+02
  2.49031915e+02 1.27659574e-01 4.36170213e-01 1.40021277e+02
  5.21276596e-01 1.62446809e+00 1.18085106e+00 1.24468085e+00
  2.57446809e+00]
 [5.24060150e+01 5.48872180e-01 1.36090226e+00 1.29548872e+02
  2.45052632e+02 1.27819549e-01 5.93984962e-01 1.59195489e+02
  1.35338346e-01 5.84962406e-01 1.64661654e+00 3.30827068e-01
  2.12030075e+00]]
Coefficients:
[[-5.12655671e-03 -1.65128336e+00  9.42708811e-01 -1.63429905e-02
  -8.26945654e-05  3.61220910e-01  6.53320414e-01  2.61543171e-02
  -1.10225766e+00 -5.26885663e-01  9.83938578e-01 -1.00983532e+00
  -1.16829536e+00]]
Test Accuracy:
0.8026315789473685
```

LDA Metrics

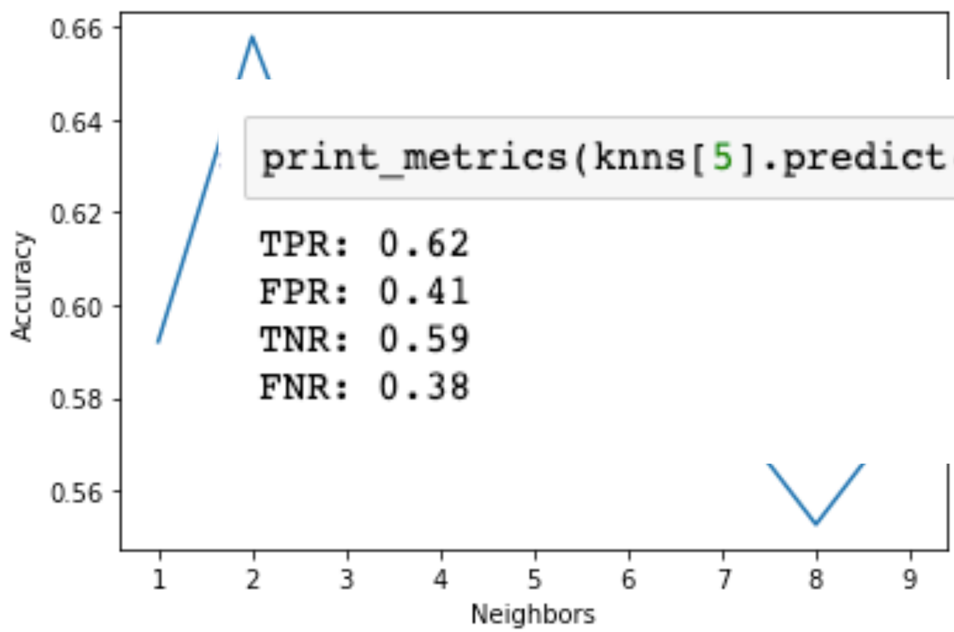
```
predictions = lda.predict_proba(x_test)[:,1]
print_metrics(predictions, y_test)
plot_roc(predictions)
```

```
TPR: 0.91
FPR: 0.27
TNR: 0.73
FNR: 0.09
```



Lab kNN

```python
from sklearn.neighbors import KNeighborsClassifier
accuracies = []
neighbors = list(range(1, 10))
knns = []
for n in neighbors:
    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(x_train, y_train)
    knns.append(knn)
    accuracies.append(knn.score(x_test, y_test))
plt.figure().add_subplot(111, xlabel="Neighbors", ylabel="Accuracy")
plt.plot(neighbors, accuracies)
plt.show()
```




```python
print_metrics(knns[5].predict(x_test), y_test)
```

```
TPR:  0.62
FPR:  0.41
TNR:  0.59
FNR:  0.38
```

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
- Linear classification lab designed by Yuxuan Wang
- Thanks!