Model Evaluation & Selection

APPLIED MACHINE LEARNING IN PYTHON

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science

Learning Objectives

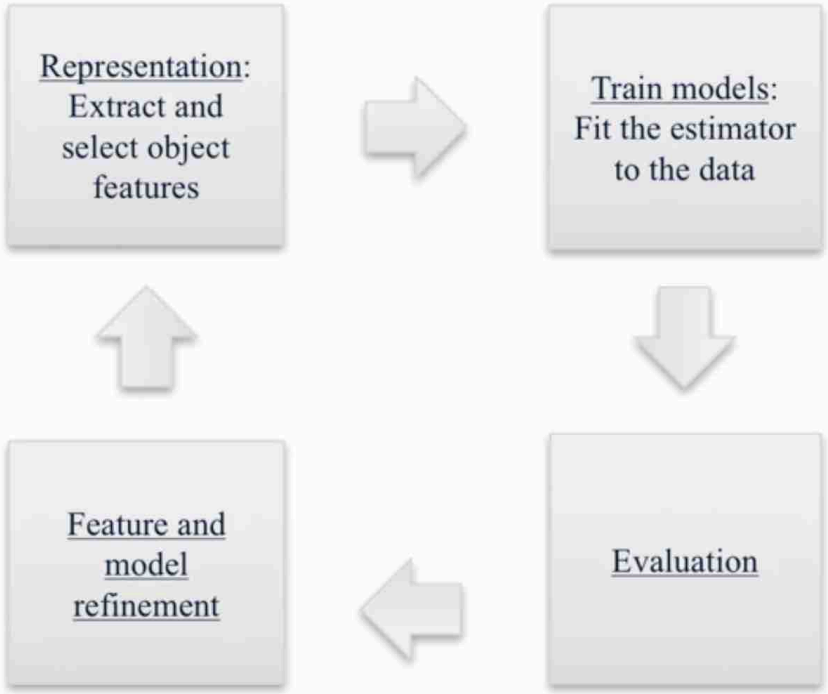
- Understand why accuracy only gives a partial picture of a classifier's performance.
- Understand the motivation and definition of important evaluation metrics in machine learning.

Learning Objectives

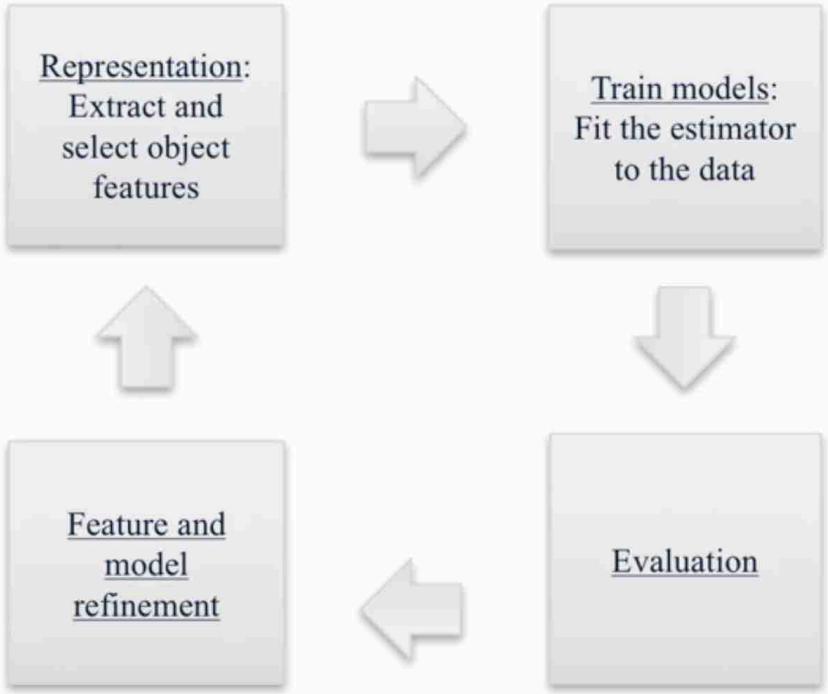
- Learn how to use a variety of evaluation metrics to evaluate supervised machine learning models.

- Learn about choosing the right metric for selecting between models or for doing parameter tuning.

Represent / Train / Evaluate / Refine Cycle



Represent / Train / Evaluate / Refine Cycle



Evaluation

- Different applications have very different goals
- Accuracy is widely used, but many others are possible, e.g.:
  - *User satisfaction (Web search)*
  - *Amount of revenue (e-commerce)*
  - *Increase in patient survival rates (medical)*

Evaluation

- It's very important to choose evaluation methods that match the goal of your application.
- Compute your selected evaluation metric for multiple different models.
- Then select the model with 'best' value of evaluation metric.

Accuracy with Imbalanced Classes

- Suppose you have two classes:
  - *Relevant (R): the positive class*
  - *Not_Relevant (N): the negative class*
- Out of 1000 randomly selected items, on average
  - *One item is relevant and has an R label*
  - *The rest of the items (999 of them) are not relevant and labelled N.*
- Recall that:

$$\text{Accuracy} = \frac{\#\text{correct predictions}}{\#\text{total instances}}$$

Accuracy with Imbalanced Classes

- You build a classifier to predict relevant items, and see that its accuracy on a test set is 99.9%.
- Wow! Amazingly good, right?
- For comparison, suppose we had a "dummy" classifier that didn't look at the features at all, and always just blindly predicted the most frequent class (i.e. the negative N class).

Accuracy with Imbalanced Classes

- Assuming a test set of 1000 instances, what would this dummy classifier's accuracy be?

- Answer:

$$\text{Accuracy}_{\text{DUMMY}} = 999 / 1000 = 99.9\%$$

Dummy classifiers completely ignore the input data!

- Dummy classifiers serve as a sanity check on your classifier's performance.
- They provide a null metric (e.g. null accuracy) baseline.
- Dummy classifiers should not be used for real problems

Dummy classifiers completely ignore the input data!

- Some commonly-used settings for the `strategy` parameter for DummyClassifier in scikit-learn:
  - *most_frequent : predicts the most frequent label in the training set.*
  - *stratified : random predictions based on training set class distribution.*
  - *uniform : generates predictions uniformly at random.*
  - *constant : always predicts a constant label provided by the user.*
    - *A major motivation of this method is F1-scoring, when the positive class is in the minority.*

What if my classifier accuracy is close to the null accuracy baseline?

This could be a sign of:

- Ineffective, erroneous or missing features
- Poor choice of kernel or hyperparameter
- Large class imbalance

Dummy Regressors

`strategy` **parameter options:**

- *mean* : predicts the mean of the training targets.
- *median* : predicts the median of the training targets.
- *quantile* : predicts a user-provided quantile of the training targets.
- *constant* : predicts a constant user-provided value.

Binary Prediction Outcomes

	Predicted negative	**Predicted** positive
True negative	TN	FP
True positive	FN	TP

Label 1 = positive class (class of interest)

Label 0 = negative class (everything else)

TP = true positive

FP = false positive (Type I error)

TN = true negative

FN = false negative (Type II error)

Confusion Matrix for Binary Prediction Task

	Predicted negative	Predicted positive
True negative	TN = 356	FP = 51
True positive	FN = 38	TP = 5

N = 450

- Every test instance is in exactly one box (integer counts).
- Breaks down classifier results by error type.
- Thus, provides more information than simple accuracy.
- Helps you choose an evaluation metric that matches project goals.
- Not a single number like accuracy.. but there are many possible metrics that can be derived from the confusion matrix.

In [6]:
```
dummy_majority.score(X_test, y_test)
```

Out[6]:
```
0.9044444444444445
```

In [7]:
```
svm = SVC(kernel='linear', C=1).fit(X_train, y_train)
svm.score(X_test, y_test)
```

Out[7]:
```
0.97777777777777775
```

Confusion matrices

binary (two-class) confusion matrix

In [8]:
```
from sklearn.metrics import confusion_matrix

dummy_majority = DummyClassifier(strategy = 'most_frequent').fit(X_train, y_train)

y_majority_predicted = dummy_majority.predict(X_test)
confusion = confusion_matrix(y_test, y_majority_predicted)

print('Most frequent class (dummy classifier)\n', confusion)
```

```
Most frequent class (dummy classifier)
 [[407   0]
 [ 43   0]]
```

In [ ]:

binary (two-class) confusion matrix

In [8]:
```
from sklearn.metrics import confusion_matrix

dummy_majority = DummyClassifier(strategy = 'most_frequent').fit(X_train, y_train)

y_majority_predicted = dummy_majority.predict(X_test)
confusion = confusion_matrix(y_test, y_majority_predicted)

print('Most frequent class (dummy classifier)\n', confusion)
```

```
Most frequent class (dummy classifier)
 [[407   0]
 [ 43   0]]
```

In [9]:
```
dummy_classprop = DummyClassifier(strategy='stratified').fit(X_train, y_train)

y_classprop_predicted = dummy_classprop.predict(X_test)
confusion = confusion_matrix(y_test, y_classprop_predicted)

print('Random class-proportional prediction (dummy classifier)\n',
      confusion)
```

```
Random class-proportional prediction (dummy classifier)
 [[360  47]
 [ 41   2]]
```

In [ ]:

```
Most frequent class (dummy classifier)
 [[407   0]
 [ 43   0]]
```

In [9]:
```
dummy_classprop = DummyClassifier(strategy='stratified').fit(X_train, y_train)

y_classprop_predicted = dummy_classprop.predict(X_test)
confusion = confusion_matrix(y_test, y_classprop_predicted)

print('Random class-proportional prediction (dummy classifier)\n',
      confusion)
```

```
Random class-proportional prediction (dummy classifier)
 [[360  47]
 [ 41   2]]
```

In [10]:
```
svm = SVC(kernel='linear', C=1).fit(X_train, y_train)
svm_predicted = svm.predict(X_test)
confusion = confusion_matrix(y_test, svm_predicted)

print('Support vector machine classifier (linear kernel, C=1)\n',
      confusion)
```

```
Support vector machine classifier (linear kernel, C=1)
 [[402   5]
 [  5  38]]
```

In [ ]:

```
 [  5  38]]
```

In [11]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression().fit(X_train, y_train)
lr_predicted = lr.predict(X_test)
confusion = confusion_matrix(y_test, lr_predicted)

print('Logistic regression classifier (default settings)\n',
      confusion)
```

```
Logistic regression classifier (default settings)
 [[401   6]
 [  6  37]]
```

In [12]:
```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
tree_predicted = dt.predict(X_test)
confusion = confusion_matrix(y_test, tree_predicted)

print('Decision tree classifier (max_depth = 2)\n',
      confusion)
```

```
Decision tree classifier (max_depth = 2)
 [[400   7]
 [ 17  26]]
```

In [ ]:

Confusion Matrices & Basic Evaluation Metrics

APPLIED MACHINE LEARNING IN PYTHON

Kevyn Collins-Thompson

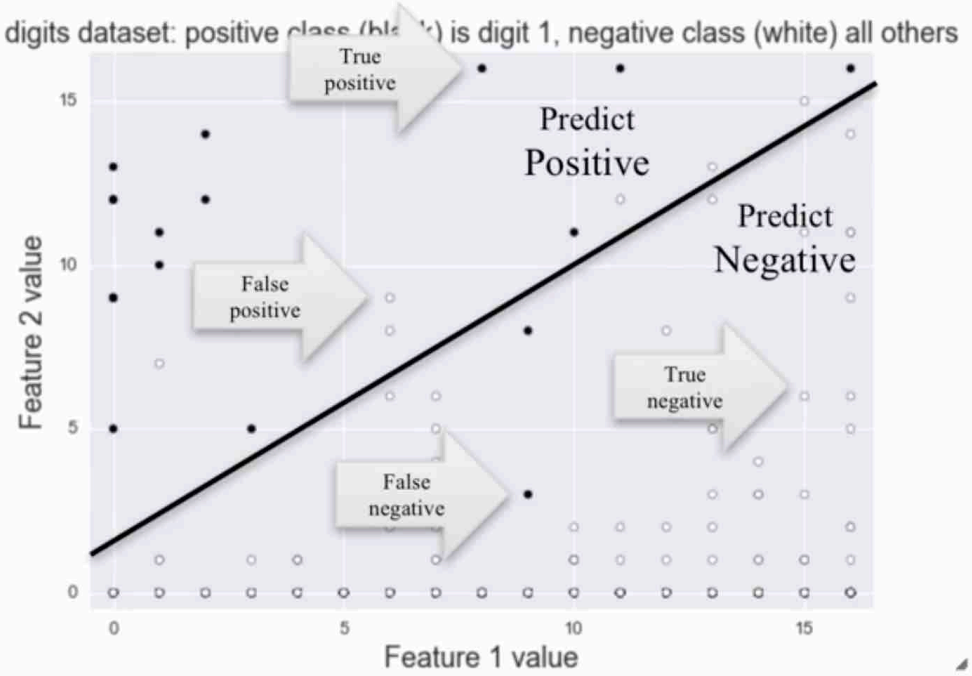
Associate Professor of Information and Computer Science

Confusion matrix for binary prediction task

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$
	Predicted negative	Predicted positive	

Always look at the confusion matrix for your classifier.

Visualization of Different Error Types



TN = 429	FP = 6
FN = 2	TP = 13

Accuracy: for what fraction of all instances is the classifier's prediction correct (for either positive or negative class)?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$
	Predicted negative	Predicted positive	

$$\text{Accuracy} = \frac{TN+TP}{TN+TP+FN+FP}$$

$$= \frac{400+26}{400+26+17+7}$$

$$= 0.95$$

Classification error (1 – Accuracy): for what fraction of all instances is the classifier's prediction incorrect?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$
	Predicted negative	Predicted positive	

$$\text{ClassificationError} = \frac{FP + FN}{TN + TP + FN + FP}$$

$$= \frac{7+17}{400+26+17+7}$$

$$= 0.060$$

Recall, or True Positive Rate (TPR): what fraction of all positive instances does the classifier correctly identify as positive?

	Predicted negative	Predicted positive	
True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$

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

$$= \frac{26}{26+17}$$

$$= 0.60$$

Recall is also known as:
- True Positive Rate (TPR)
- Sensitivity
- Probability of detection

Precision: what fraction of positive predictions are correct?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$
	Predicted negative	Predicted positive	

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

$$= \frac{26}{26+7}$$

$$= 0.79$$

False positive rate (FPR): what fraction of all negative instances does the classifier incorrectly identify as positive?

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
			$N = 450$
	Predicted negative	Predicted positive	

$$FPR = \frac{FP}{TN+FP}$$

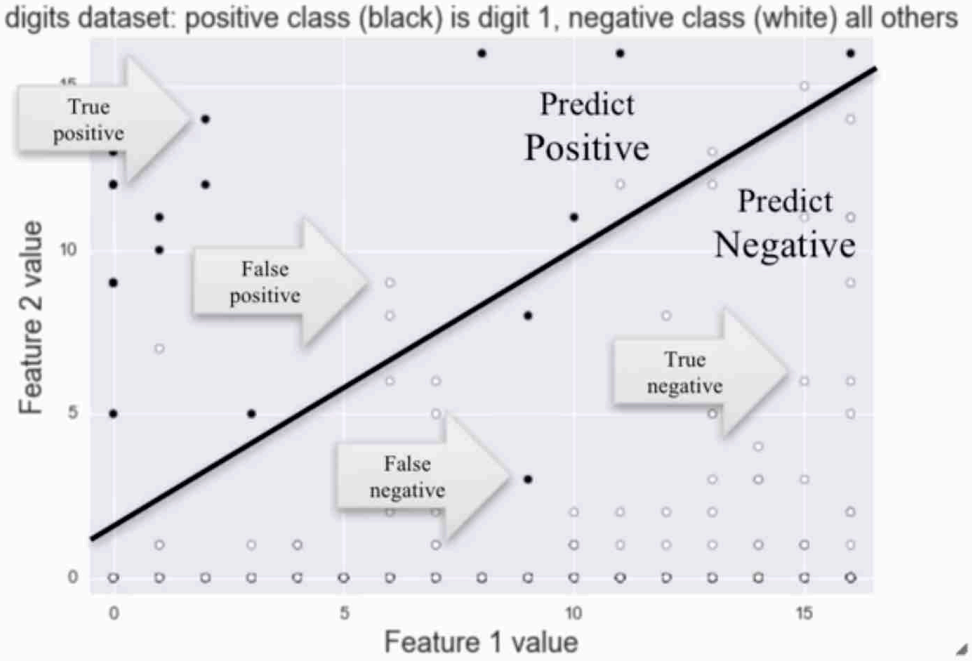
$$= \frac{7}{400+7}$$

$$= 0.02$$

False Positive Rate is also known as:

- Specificity

The Precision-Recall Tradeoff

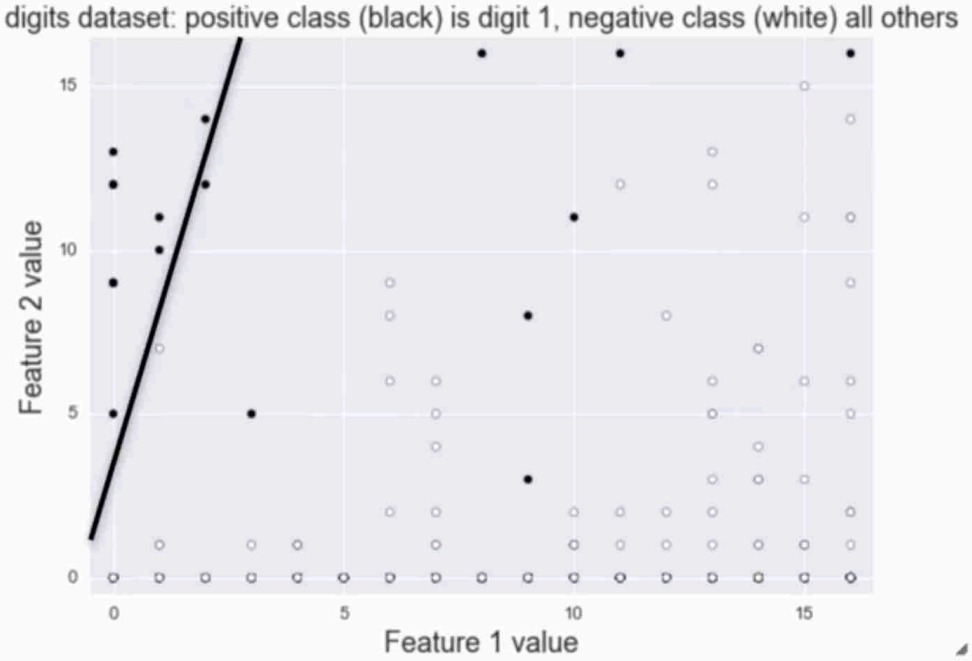


TN = 429	FP = 6
FN = 2	TP = 13

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{13}{19} = 0.68$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{13}{15} = 0.87$$

High Precision, Lower Recall

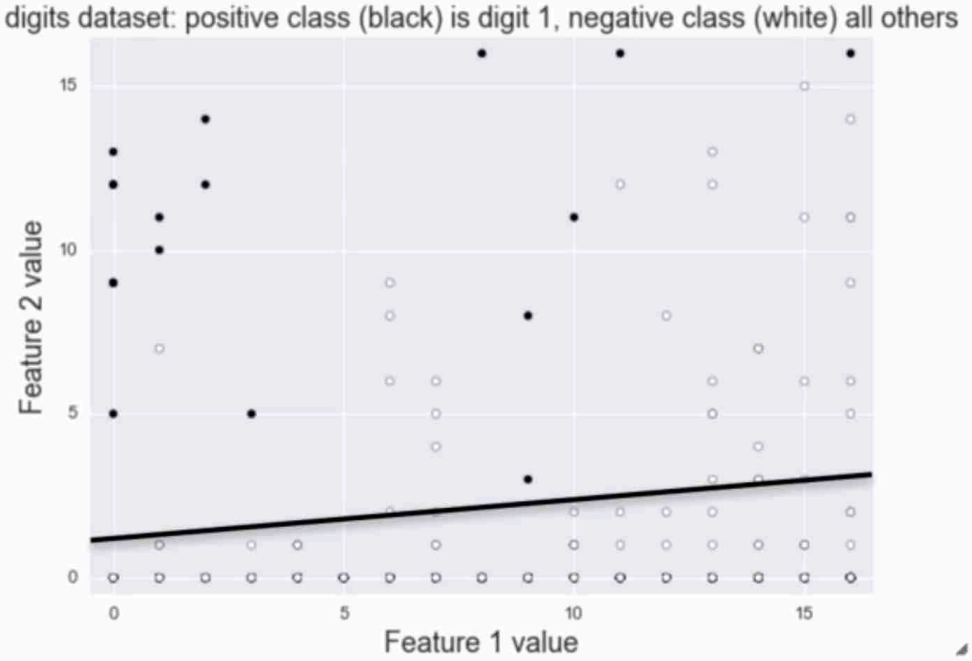


TN = 435	FP = 0
FN = 8	TP = 7

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{7}{7} = 1.00$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{7}{15} = 0.47$$

Low Precision, High Recall



TN = 408	FP = 27
FN = 0	TP = 15

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{15}{42} = 0.36$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{15}{15} = 1.00$$

There is often a tradeoff between precision and recall

- **Recall-oriented machine learning tasks:**
  - Search and information extraction in legal discovery
  - Tumor detection
  - Often paired with a human expert to filter out false positives
- **Precision-oriented machine learning tasks:**
  - Search engine ranking, query suggestion
  - Document classification
  - Many customer-facing tasks (users remember failures!)

F-score: generalizes F1-score for combining precision & recall into a single number

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta \cdot FN + FP}$$

β allows adjustment of the metric to control the emphasis on recall vs precision:

- **Precision-oriented users: $\beta = 0.5$ (false positives hurt performance more than false negatives)**
- **Recall-oriented users: $\beta = 2$ (false negatives hurt performance more than false positives)**

```
dt = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
tree_predicted = dt.predict(X_test)
confusion = confusion_matrix(y_test, tree_predicted)

print('Decision tree classifier (max_depth = 2)\n',
     confusion)
```

```
Decision tree classifier (max_depth = 2)
 [[400   7]
 [ 17  26]]
```

Evaluation metrics for binary classification

In [13]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('Accuracy: {:.2f}'.format(accuracy_score(y_test, tree_predicted)))
print('Precision: {:.2f}'.format(precision_score(y_test, tree_predicted)))
print('Recall: {:.2f}'.format(recall_score(y_test, tree_predicted)))
print('F1: {:.2f}'.format(f1_score(y_test, tree_predicted)))
```

```
Accuracy: 0.95
Precision: 0.79
Recall: 0.60
F1: 0.68
```

In [ ]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, tree_predicted, target_names=['n']))
```

In [13]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('Accuracy: {:.2f}'.format(accuracy_score(y_test, tree_predicted)))
print('Precision: {:.2f}'.format(precision_score(y_test, tree_predicted)))
print('Recall: {:.2f}'.format(recall_score(y_test, tree_predicted)))
print('F1: {:.2f}'.format(f1_score(y_test, tree_predicted)))
```

```
Accuracy: 0.95
Precision: 0.79
Recall: 0.60
F1: 0.68
```

In [14]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, tree_predicted, target_names=['not 1', '1']))
```

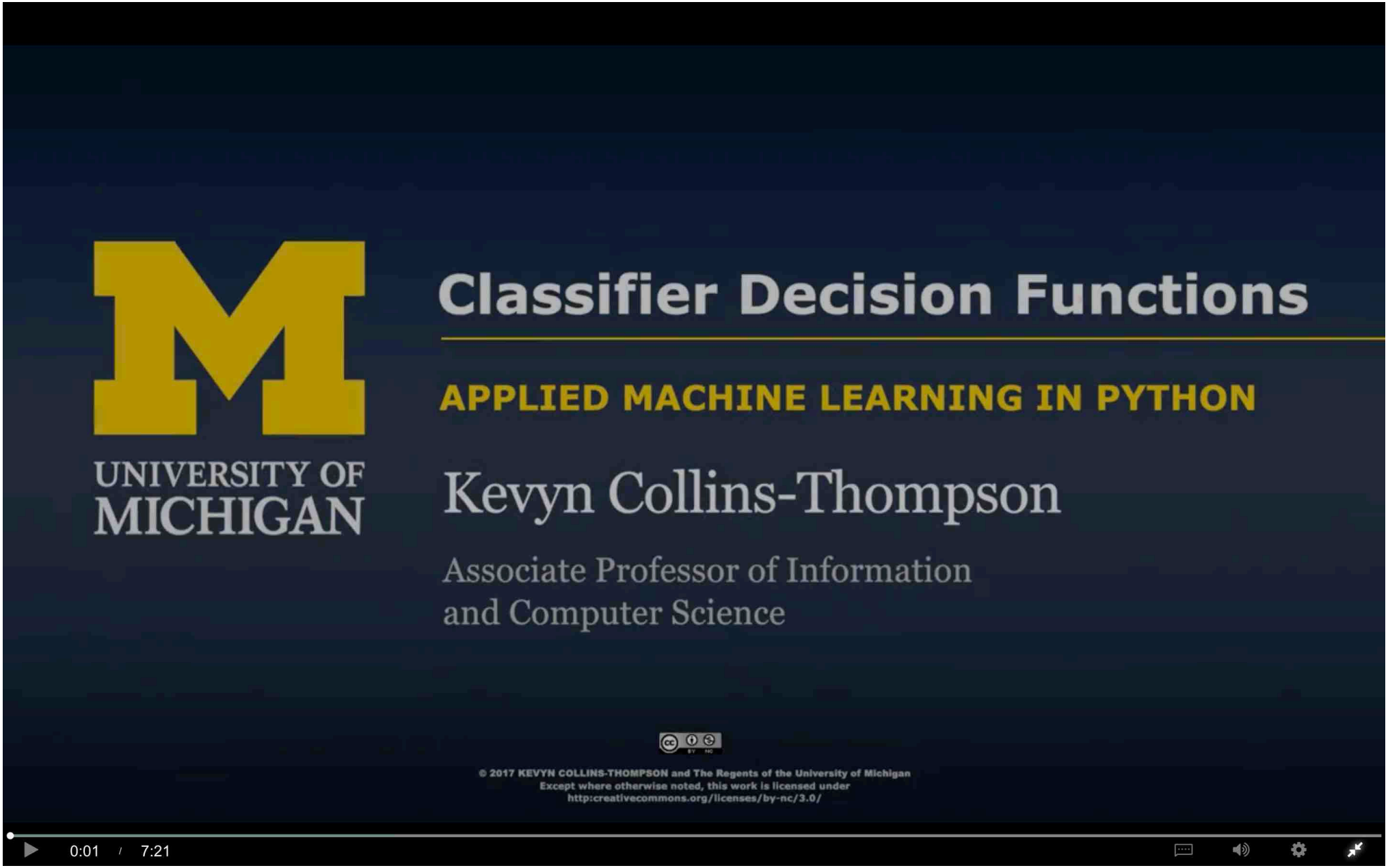
```
             precision    recall  f1-score   support

      not 1       0.96      0.98      0.97       407
          1       0.79      0.60      0.68        43

avg / total       0.94      0.95      0.94       450
```

In [ ]:

In [15]:
```
print('Random class-proportional (dummy)\n',
      classification_report(y_test, y_classprop_predicted, target_names=['not 1', '1']))

print('SVM\n',
      classification_report(y_test, svm_predicted, target_names = ['not 1', '1']))

print('Logistic regression\n',
      classification_report(y_test, lr_predicted, target_names = ['not 1', '1']))

print('Decision tree\n',
      classification_report(y_test, tree_predicted, target_names = ['not 1', '1']))
```

```
Random class-proportional (dummy)
             precision    recall  f1-score   support

      not 1       0.91      0.94      0.92       407
          1       0.19      0.14      0.16        43

avg / total       0.84      0.86      0.85       450

SVM
             precision    recall  f1-score   support

      not 1       0.99      0.99      0.99       407
          1       0.88      0.88      0.88        43

avg / total       0.98      0.98      0.98       450
```

```
                precision    recall  f1-score   support

      not 1       0.91      0.94      0.92       407
          1       0.19      0.14      0.16        43

avg / total       0.84      0.86      0.85       450

SVM
                precision    recall  f1-score   support

      not 1       0.99      0.99      0.99       407
          1       0.88      0.88      0.88        43

avg / total       0.98      0.98      0.98       450

Logistic regression
                precision    recall  f1-score   support

      not 1       0.99      0.99      0.99       407
          1       0.86      0.86      0.86        43

avg / total       0.97      0.97      0.97       450

Decision tree
                precision    recall  f1-score   support

      not 1       0.96      0.98      0.97       407
          1       0.79      0.60      0.68        43
```


UNIVERSITY OF
MICHIGAN
Classifier Decision Functions
APPLIED MACHINE LEARNING IN PYTHON
Kevyn Collins-Thompson
Associate Professor of Information
and Computer Science
© 2017 KEVYN COLLINS-THOMPSON and The Regents of the University of Michigan
Except where otherwise noted, this work is licensed under
http:creativecommons.org/licenses/by-nc/3.0/
0:01 / 7:21

Decision Functions (decision_function)

- Each classifier score value per test point indicates how confidently the classifier predicts the positive class (large-magnitude positive values) or the negative class (large-magnitude negative values).
- Choosing a fixed decision threshold gives a classification rule.
- By sweeping the decision threshold through the entire range of possible score values, we get a series of classification outcomes that form a curve.

```
avg / total       0.97      0.97      0.97       450

Decision tree
             precision    recall  f1-score   support

      not 1       0.96      0.98      0.97       407
          1       0.79      0.60      0.68        43

avg / total       0.94      0.95      0.94       450
```

Decision functions

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalanced,
                                                    random_state=0)

y_scores_lr = lr.fit(X_train, y_train).decision_function(X_test)
y_score_list = list(zip(y_test[0:20], y_scores_lr[0:20]))

y_score_list
```

Out[16]:
```
[(0, -23.177112362903241),
 (0, -13.541470729541413),
 (0, -21.722900989694729),
 (0, -18.907438437430027),
 (0, -19.735821729002289),
 (0, -9.7498078195600613),
```

Predicted Probability of Class Membership (predict_proba)

- Typical rule: choose most likely class
  - e.g class 1 if threshold > 0.50.
- Adjusting threshold affects predictions of classifier.
- Higher threshold results in a more conservative classifier
  - e.g. only predict Class 1 if estimated probability of class 1 is above 70%
  - This increases precision. Doesn't predict class 1 as often, but when it does, it gets high proportion of class 1 instances correct.
- Not all models provide realistic probability estimates

```
(1, 5.2349604859009276),
(0, -19.307551661127864),
(0, -25.101182889530396),
(0, -21.827362391350579),
(0, -24.151343401889438),
(0, -19.576969790071697),
(0, -22.574689400560423),
(0, -10.823324268750714),
(0, -11.912123406737392),
(0, -10.97922371337485),
(1, 11.206006114721543),
(0, -27.646002317931909),
(0, -12.859381428186682),
(0, -25.848764845244997)]
```

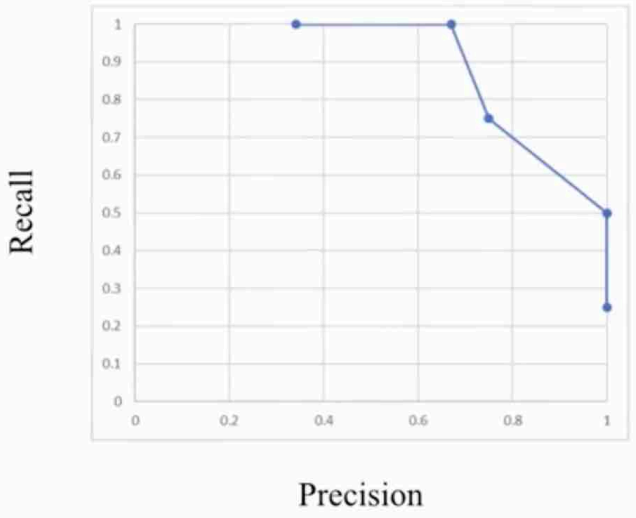
In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalanced,
                                                    random_state=0)

y_proba_lr = lr.fit(X_train, y_train).predict_proba(X_test)
y_proba_list = list(zip(y_test[0:20], y_proba_lr[0:20,1]))

y_proba_list
```

Out[17]:
```
[(0, 8.59622935435435e-11),
 (0, 1.3152654562877047e-06),
 (0, 3.6801368532691122e-10),
 (0, 6.1461593657683724e-09),
 (0, 2.6843647432086779e-09),
 (0, 5.8302468525539246e-05),
```

Varying the Decision Threshold

True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597

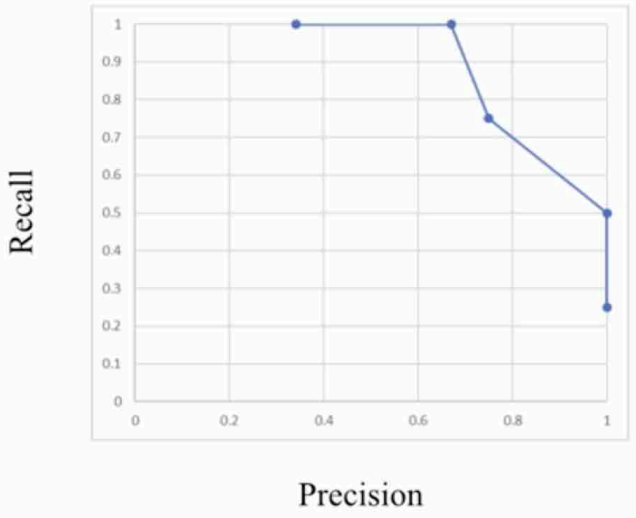
Classifier score	Precision	Recall
-20	4/12=0.34	4/4=1.00
-10	4/6=0.67	4/4=1.00
0	3/4=0.75	3/4=0.75
10	2/2=1.0	2/4=0.50
20	1/1=1.0	1/4 = 0.25



Varying the Decision Threshold

True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597

Classifier score	Precision	Recall
-20	4/12=0.34	4/4=1.00
-10	4/6=0.67	4/4=1.00
0	3/4=0.75	3/4=0.75
10	2/2=1.0	2/4=0.50
20	1/1=1.0	1/4 = 0.25



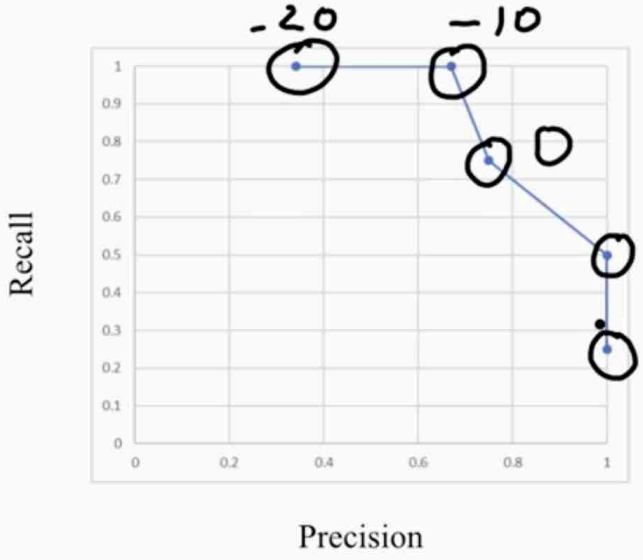
Varying the Decision Threshold

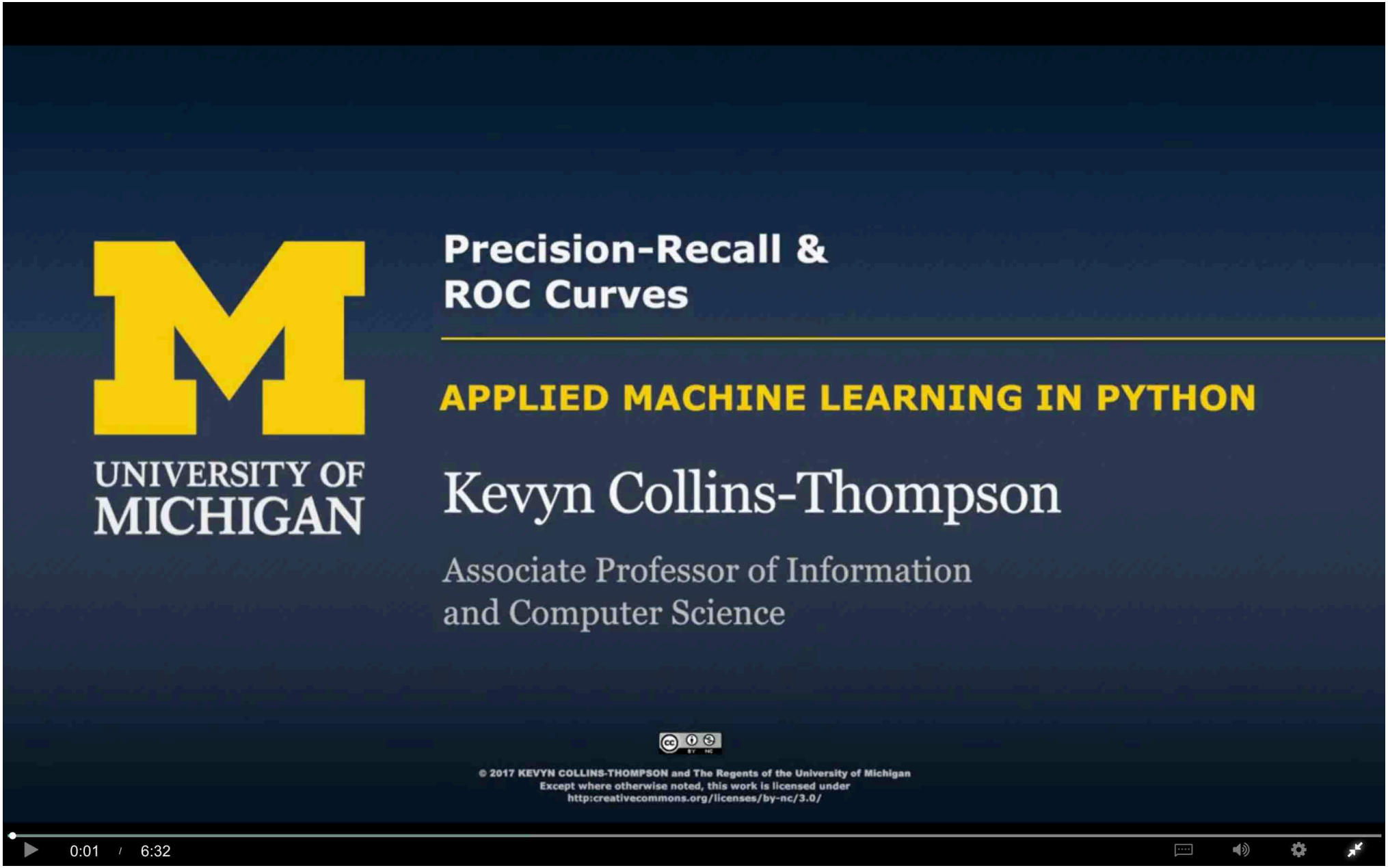
True Label	Classifier score
0	-27.6457
0	-25.8486
0	-25.1011
0	-24.1511
0	-23.1765
0	-22.575
0	-21.8271
0	-21.7226
0	-19.7361
0	-19.5768
0	-19.3071
0	-18.9077
0	-13.5411
0	-12.8594
1	-3.9128
0	-1.9798
1	1.824
0	4.74931
1	15.234624
1	21.20597

4/6

4/4

Classifier score	Precision	Recall
-20	4/12=0.34	4/4=1.00
-10	4/6=0.67	4/4=1.00
0	3/4=0.75	3/4=0.75
10	2/2=1.0	2/4=0.50
20	1/1=1.0	1/4 = 0.25



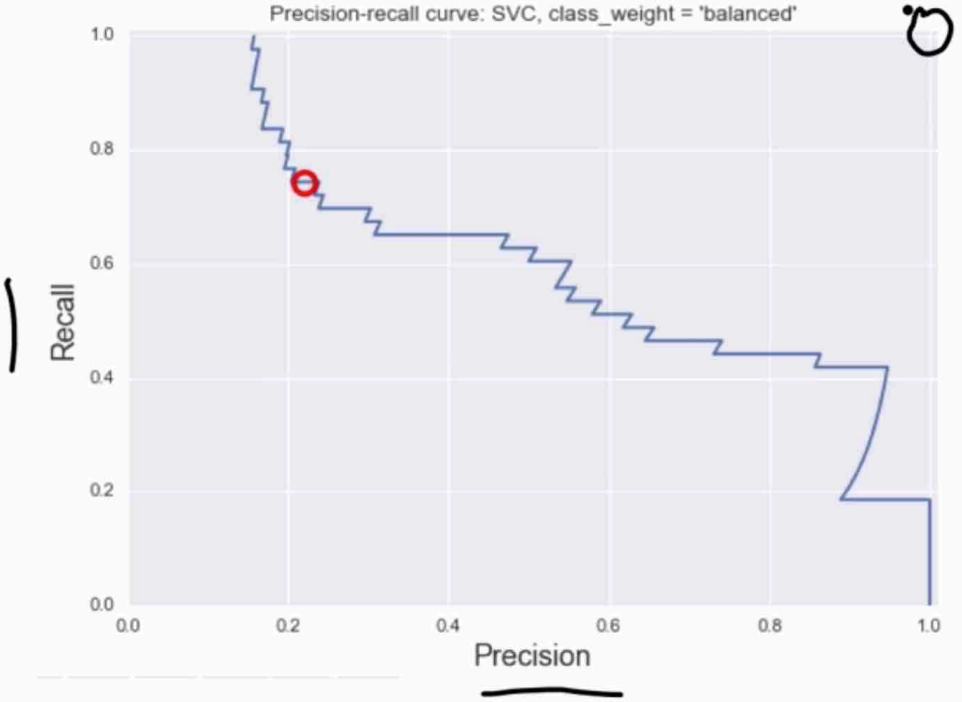

Precision-Recall &
ROC Curves
APPLIED MACHINE LEARNING IN PYTHON
UNIVERSITY OF
MICHIGAN
Kevyn Collins-Thompson
Associate Professor of Information
and Computer Science
© 2017 KEVYN COLLINS-THOMPSON and The Regents of the University of Michigan
Except where otherwise noted, this work is licensed under
http:creativecommons.org/licenses/by-nc/3.0/
0:01 / 6:32

Precision-Recall Curves

X-axis: Precision
Y-axis: Recall

Top right corner:
- The "ideal" point
- Precision = 1.0
- Recall = 1.0

"Steepness" of P-R curves is important:
- Maximize precision
- while maximizing recall

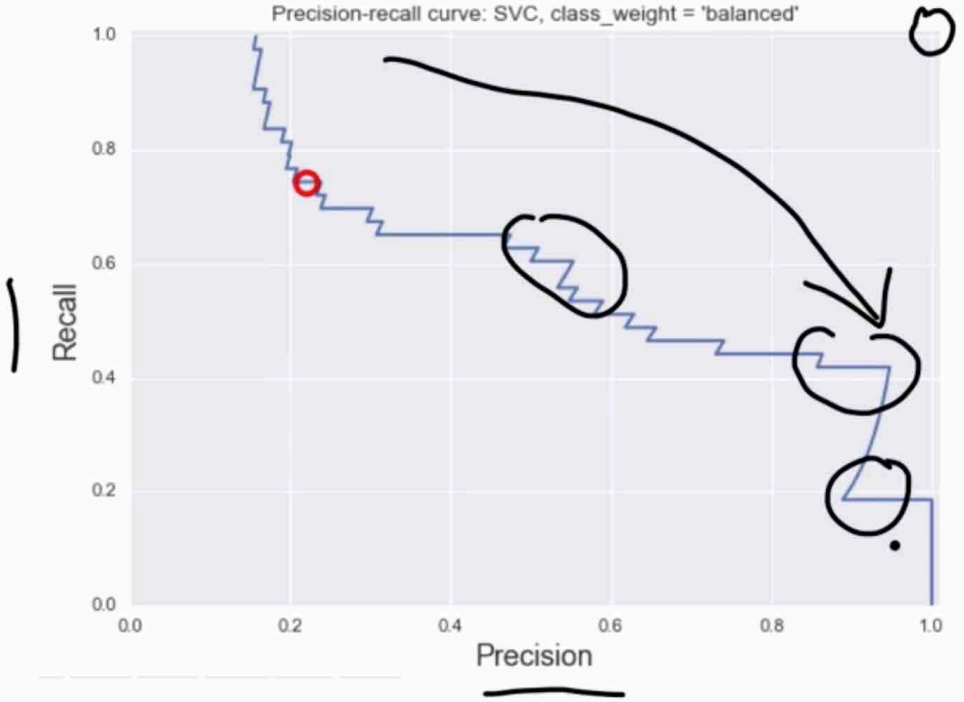


Precision-Recall Curves

X-axis: Precision
Y-axis: Recall

Top right corner:
- The "ideal" point
- Precision = 1.0
- Recall = 1.0

"Steepness" of P-R curves is important:
- Maximize precision
- while maximizing recall

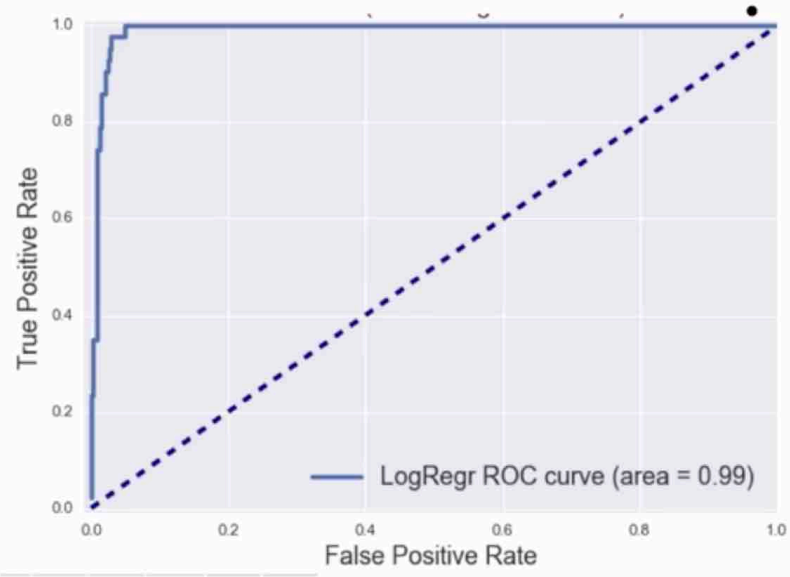


ROC Curves

X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:
- The "ideal" point
- False positive rate of zero
- True positive rate of one

"Steepness" of ROC curves is important:
- Maximize the true positive rate
- while minimizing the false positive rate

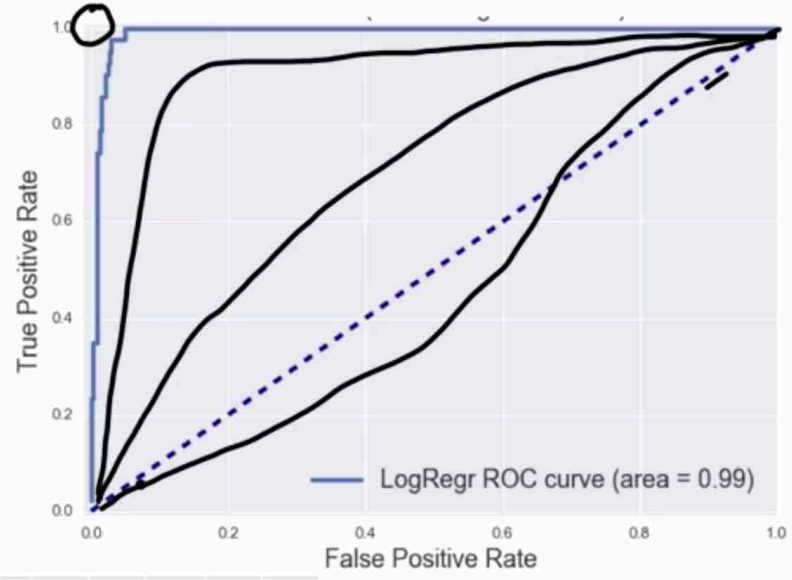


ROC Curves

X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:
- The "ideal" point
- False positive rate of zero
- True positive rate of one

"Steepness" of ROC curves is important:
- Maximize the true positive rate
- while minimizing the false positive rate



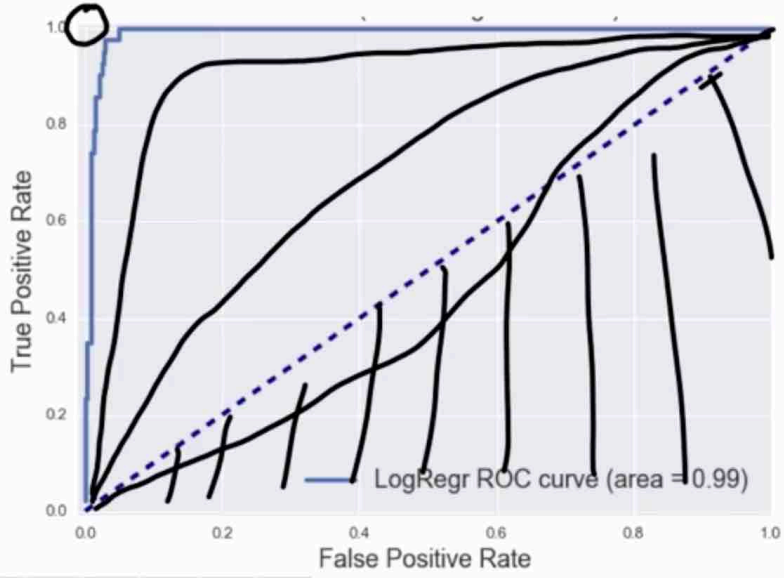
ROC Curves

AUC

X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:
- The "ideal" point
- False positive rate of zero
- True positive rate of one

"Steepness" of ROC curves is important:
- Maximize the true positive rate
- while minimizing the false positive rate



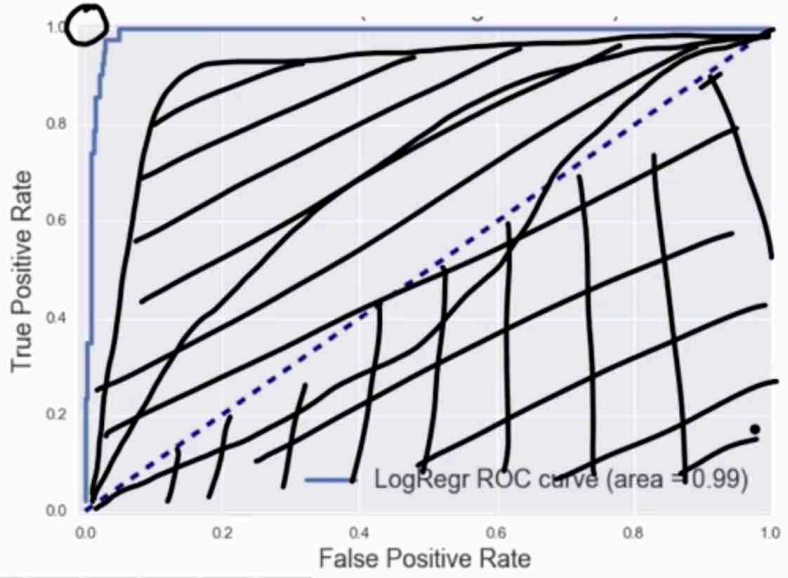
ROC Curves

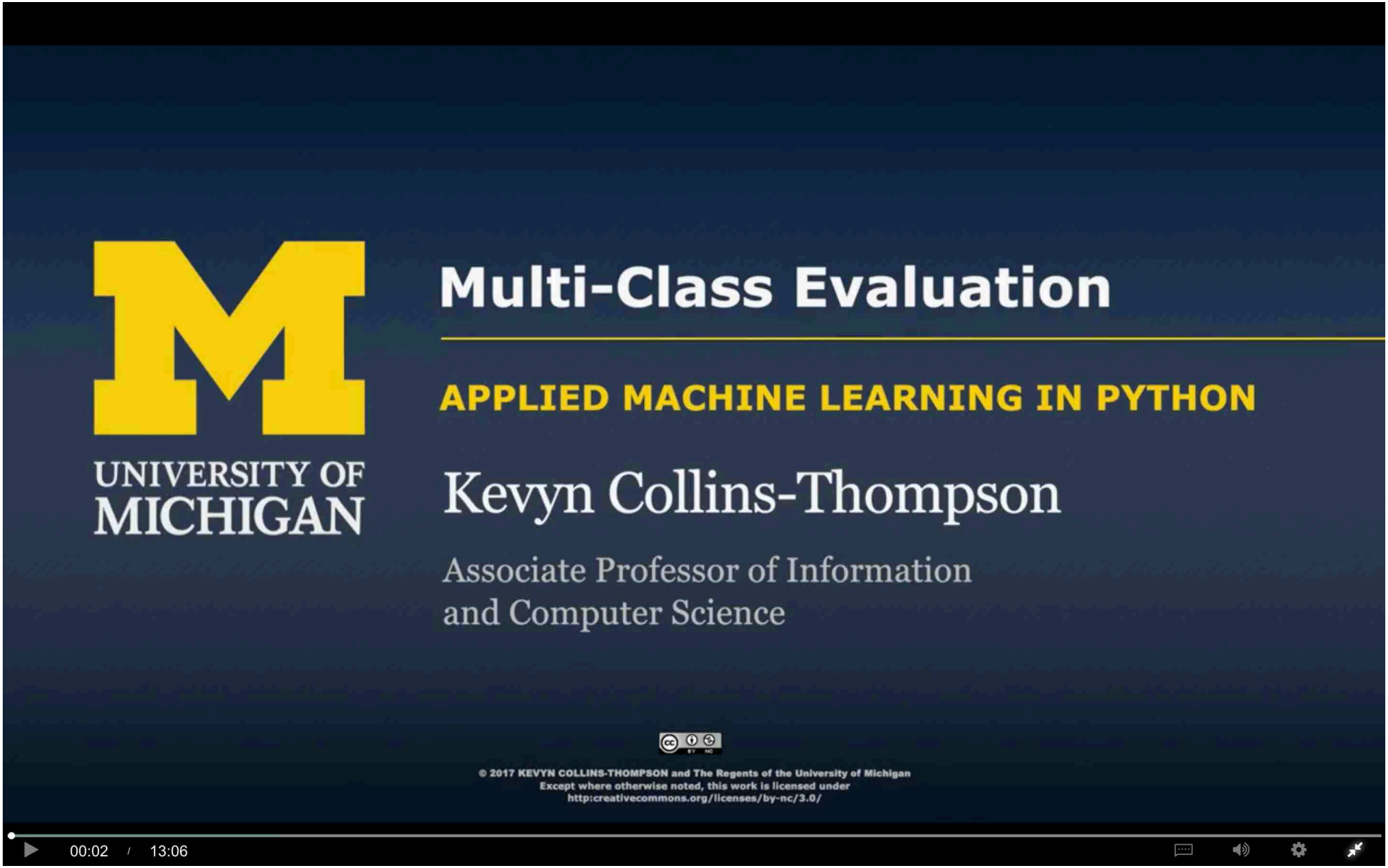
AUC

X-axis: False Positive Rate
Y-axis: True Positive Rate

Top left corner:
- The "ideal" point
- False positive rate of zero
- True positive rate of one

"Steepness" of ROC curves is important:
- Maximize the true positive rate
- while minimizing the false positive rate

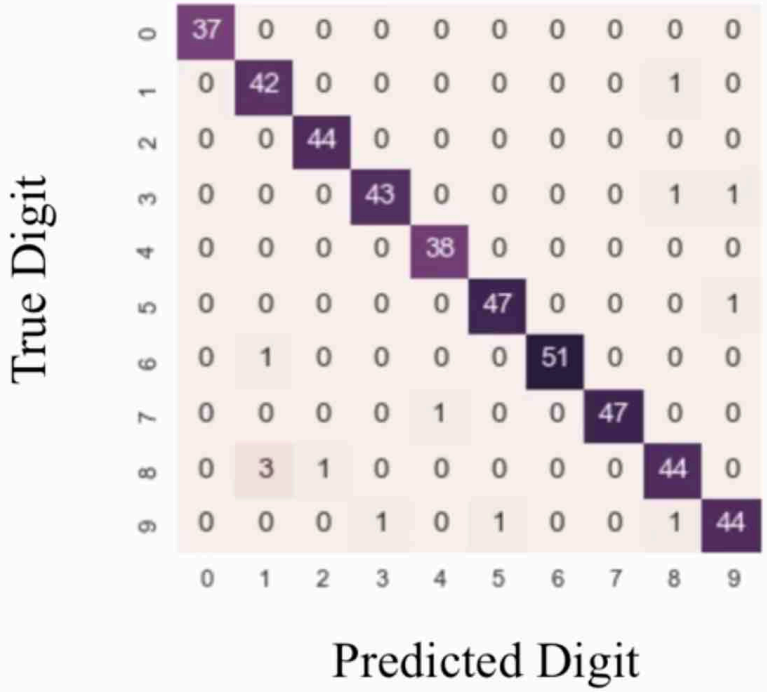



UNIVERSITY OF MICHIGAN
Multi-Class Evaluation
APPLIED MACHINE LEARNING IN PYTHON
Kevyn Collins-Thompson
Associate Professor of Information and Computer Science
© 2017 KEVYN COLLINS-THOMPSON and The Regents of the University of Michigan
Except where otherwise noted, this work is licensed under
http:creativecommons.org/licenses/by-nc/3.0/
00:02 / 13:06

Multi-Class Evaluation

- **Multi-class evaluation is an extension of the binary case.**
  - A collection of true vs predicted binary outcomes, one per class
  - Confusion matrices are especially useful
  - Classification report
- **Overall evaluation metrics are averages across classes**
  - But there are different ways to average multi-class results: we will cover these shortly.
  - The support (number of instances) for each class is important to consider, e.g. in case of imbalanced classes
- **Multi-label classification: each instance can have multiple labels (not covered here)**

Multi-Class Confusion Matrix



In [ ]:

```
dataset = load_digits()

X, y = dataset.data, dataset.target
X_train_mc, X_test_mc, y_train_mc, y_test_mc = train_test_split(X, y, random_state=0)


svm = SVC(kernel = 'linear').fit(X_train_mc, y_train_mc)
svm_predicted_mc = svm.predict(X_test_mc)
confusion_mc = confusion_matrix(y_test_mc, svm_predicted_mc)
df_cm = pd.DataFrame(confusion_mc)

plt.figure(figsize=(5.5,4))
sns.heatmap(df_cm, annot=True)
plt.title('SVM Linear Kernel \nAccuracy: {0:.3f}'
          .format(accuracy_score(y_test_mc, svm_predicted_mc)))
plt.ylabel('True label')
plt.xlabel('Predicted label')


svm = SVC(kernel = 'rbf').fit(X_train_mc, y_train_mc)
svm_predicted_mc = svm.predict(X_test_mc)
confusion_mc = confusion_matrix(y_test_mc, svm_predicted_mc)
df_cm = pd.DataFrame(confusion_mc)

plt.figure(figsize = (5.5,4))
sns.heatmap(df_cm, annot=True)
plt.title('SVM RBF Kernel \nAccuracy: {0:.3f}'
          .format(accuracy_score(y_tes)))
```

```
svm = SVC(kernel = 'rbf').fit(X_train_mc, y_train_mc)
svm_predicted_mc = svm.predict(X_test_mc)
confusion_mc = confusion_matrix(y_test_mc, svm_predicted_mc)
df_cm = pd.DataFrame(confusion_mc)

plt.figure(figsize = (5.5,4))
sns.heatmap(df_cm, annot=True)
plt.title('SVM RBF Kernel \nAccuracy: {0:.3f}'
          .format(accuracy_score(y_test_mc, svm_predicted_mc)))
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

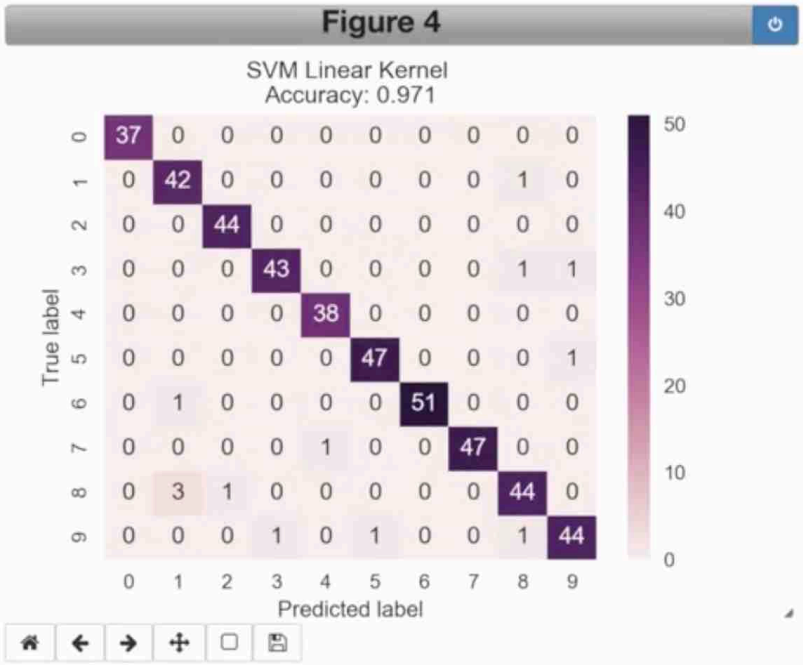


Figure 4
SVM Linear Kernel
Accuracy: 0.971
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
50
40
30
20
10
0

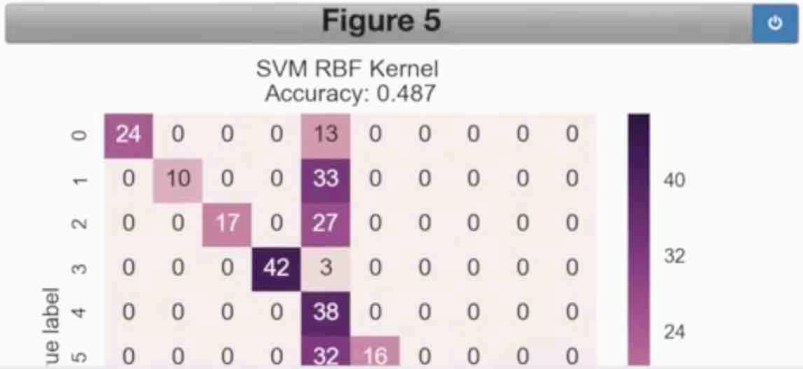
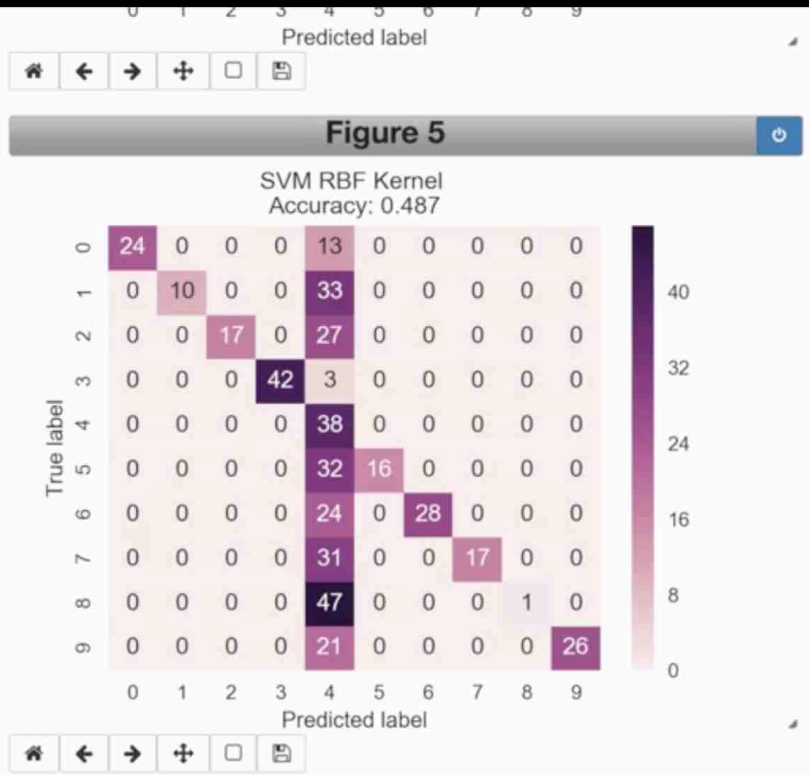


Figure 5
SVM RBF Kernel
Accuracy: 0.487
40
32
24



Out[21]: <matplotlib.text.Text at 0x119c6c780>

In [ ]:

Out[21]: <matplotlib.text.Text at 0x119c6c780>

Multi-class classification report

In [22]:
```
print(classification_report(y_test_mc, svm_predicted_mc))
```

```
             precision    recall  f1-score   support

          0       1.00      0.65      0.79        37
          1       1.00      0.23      0.38        43
          2       1.00      0.39      0.56        44
          3       1.00      0.93      0.97        45
          4       0.14      1.00      0.25        38
          5       1.00      0.33      0.50        48
          6       1.00      0.54      0.70        52
          7       1.00      0.35      0.52        48
          8       1.00      0.02      0.04        48
          9       1.00      0.55      0.71        47

avg / total       0.93      0.49      0.54       450
```

In [ ]:

Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

Macro-average:

- Each class has equal weight.

1. Compute metric within each class
2. Average resulting metrics across classes

Class	Precision
orange	1/5 = 0.20
lemon	1/2 = 0.50
apple	2/2 = 1.00

Macro-average precision:
(0.20 + 0.50 + 1.00) / 3 = **0.57**

Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

1/3 1/3 1/3 0.20 0.50 1.00

Macro-average:

- Each class has equal weight.

1. Compute metric within each class
2. Average resulting metrics across classes

Class	Precision
orange	1/5 = 0.20
lemon	1/2 = 0.50
apple	2/2 = 1.00

Macro-average precision:

(0.20 + 0.50 + 1.00) / 3 = **0.57**

Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

4/9

Micro-average:

- Each instance has equal weight.
- Largest classes have most influence

1. Aggregate outcomes across all classes
2. Compute metric with aggregate outcomes

Micro-average precision:
4 / 9 = **0.44**

Macro-Average vs Micro-Average

- If the classes have about the same number of instances, macro- and micro-average will be about the same.
- If some classes are much larger (more instances) than others, and you want to:
  - Weight your metric toward the largest ones, use micro-averaging.
  - Weight your metric toward the smallest ones, use macro-averaging.
- If the micro-average is much lower than the macro-average then examine the larger classes for poor metric performance.
- If the macro-average is much lower than the micro-average then examine the smaller classes for poor metric performance.

6	1.00	0.54	0.70	52
7	1.00	0.35	0.52	48
8	1.00	0.02	0.04	48
9	1.00	0.55	0.71	47
avg / total	0.93	0.49	0.54	450

Micro- vs. macro-averaged metrics

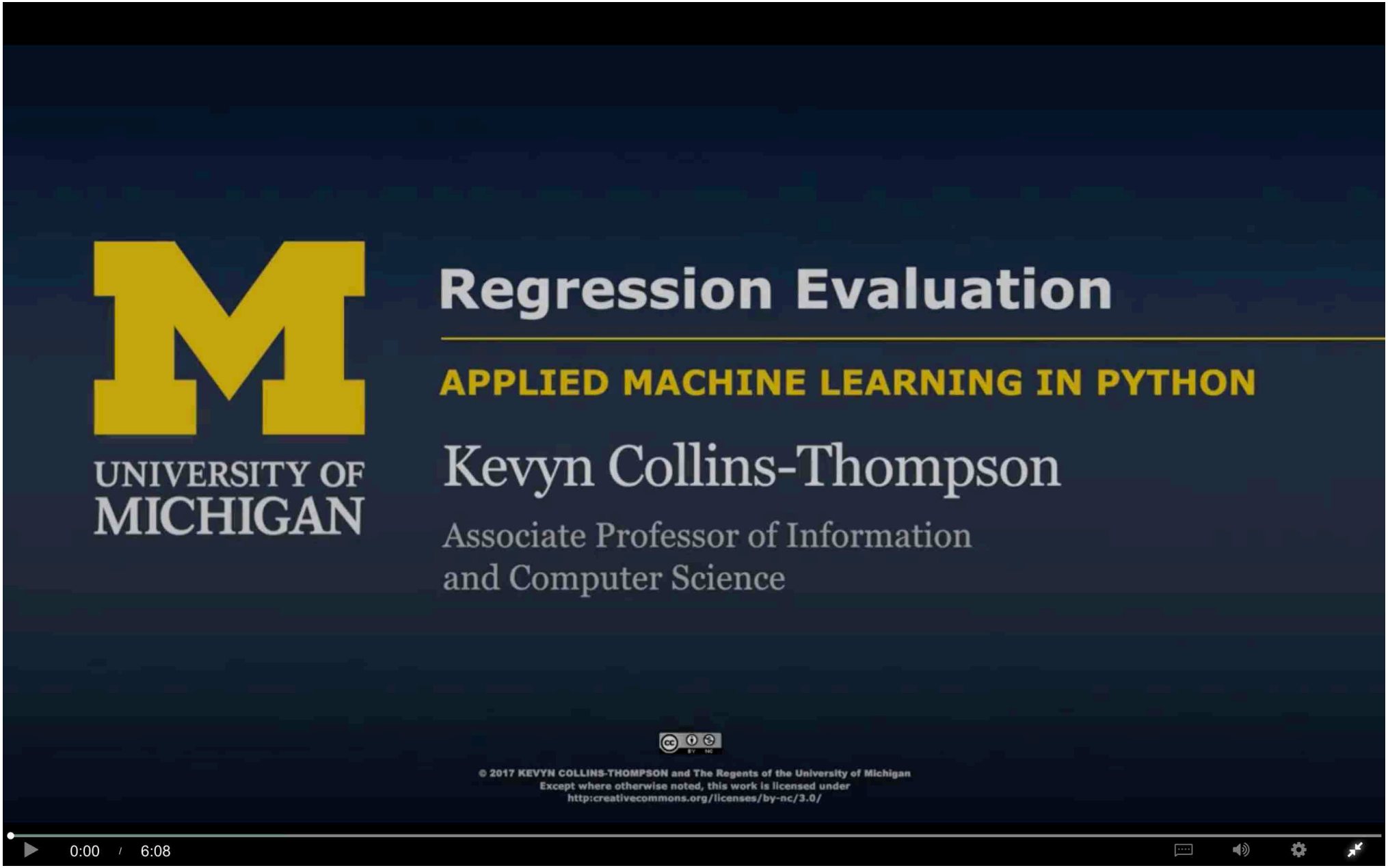
In [23]:
```
print('Micro-averaged precision = {:.2f} (treat instances equally)'
      .format(precision_score(y_test_mc, svm_predicted_mc, average = 'micro')))
print('Macro-averaged precision = {:.2f} (treat classes equally)'
      .format(precision_score(y_test_mc, svm_predicted_mc, average = 'macro')))
```

```
Micro-averaged precision = 0.49 (treat instances equally)
Macro-averaged precision = 0.91 (treat classes equally)
```

In [24]:
```
print('Micro-averaged f1 = {:.2f} (treat instances equally)'
      .format(f1_score(y_test_mc, svm_predicted_mc, average = 'micro')))
print('Macro-averaged f1 = {:.2f} (treat classes equally)'
      .format(f1_score(y_test_mc, svm_predicted_mc, average = 'macro')))
```

```
Micro-averaged f1 = 0.49 (treat instances equally)
Macro-averaged f1 = 0.54 (treat classes equally)
```

In [ ]:


UNIVERSITY OF MICHIGAN
Regression Evaluation
APPLIED MACHINE LEARNING IN PYTHON
Kevyn Collins-Thompson
Associate Professor of Information and Computer Science
© 2017 KEVYN COLLINS-THOMPSON and The Regents of the University of Michigan
Except where otherwise noted, this work is licensed under
http:creativecommons.org/licenses/by-nc/3.0/
0:00 / 6:08

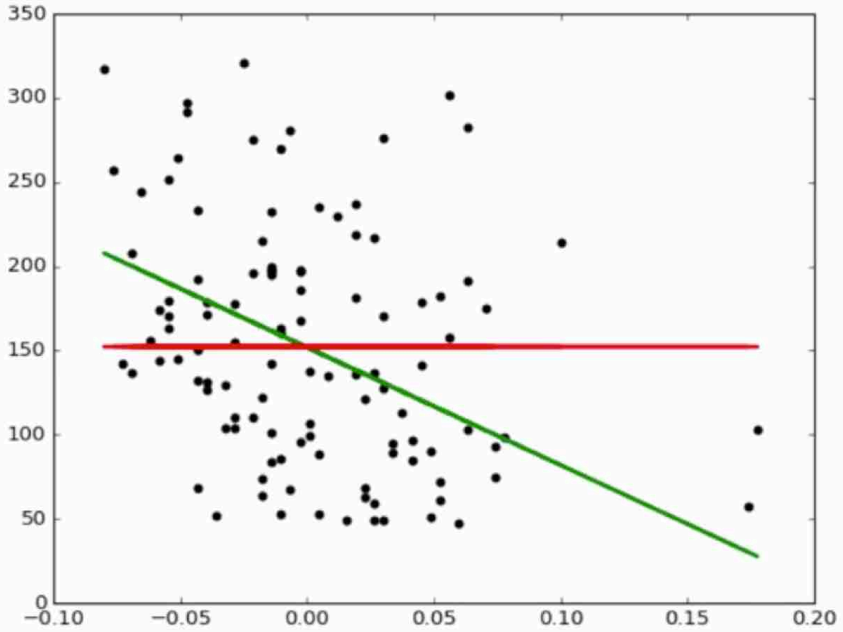
Regression Metrics

- **Typically r2_score is enough**
  - *Reminder: computes how well future instances will be predicted*
  - *Best possible score is 1.0*
  - *Constant prediction score is 0.0*
- **Alternative metrics include:**
  - *mean_absolute_error (absolute difference of target & predicted values)*
  - *mean_squared_error (squared difference of target & predicted values)*
  - *median_absolute_error (robust to outliers)*

Dummy Regressors

As in classification, comparison to a 'dummy' prediction model that uses a fixed rule can be useful.

For this, scikit.learn provides dummy regressors.

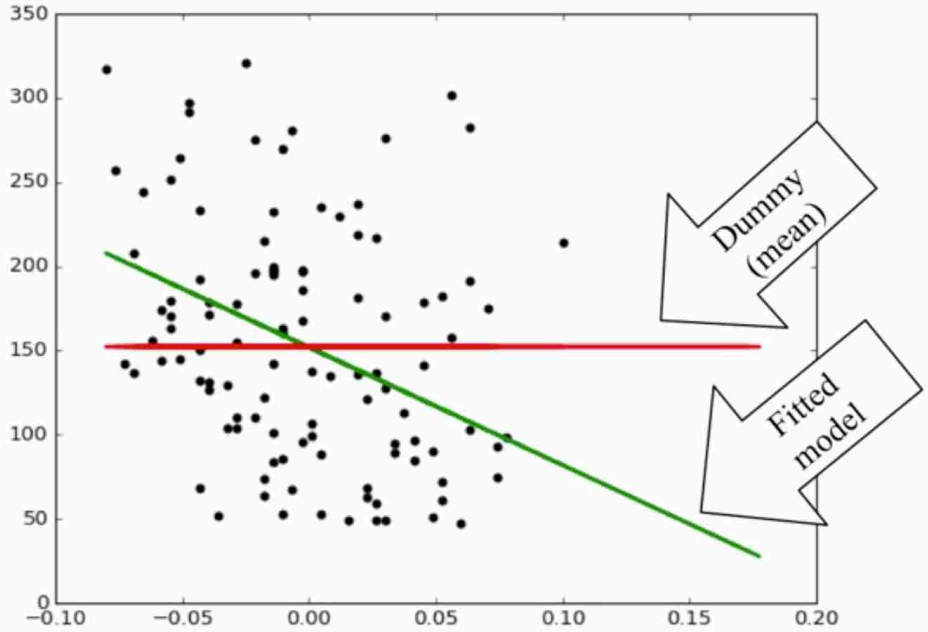


Dummy Regressors

As in classification, comparison to a 'dummy' prediction model that uses a fixed rule can be useful.

For this, scikit.learn provides dummy regressors

```
Linear model, coefficients: [-698.80206267]
Mean squared error (dummy): 4965.13
Mean squared error (linear model): 4646.74
r2_score (dummy): -0.00
r2_score (linear model): 0.06
```



Regression evaluation metrics

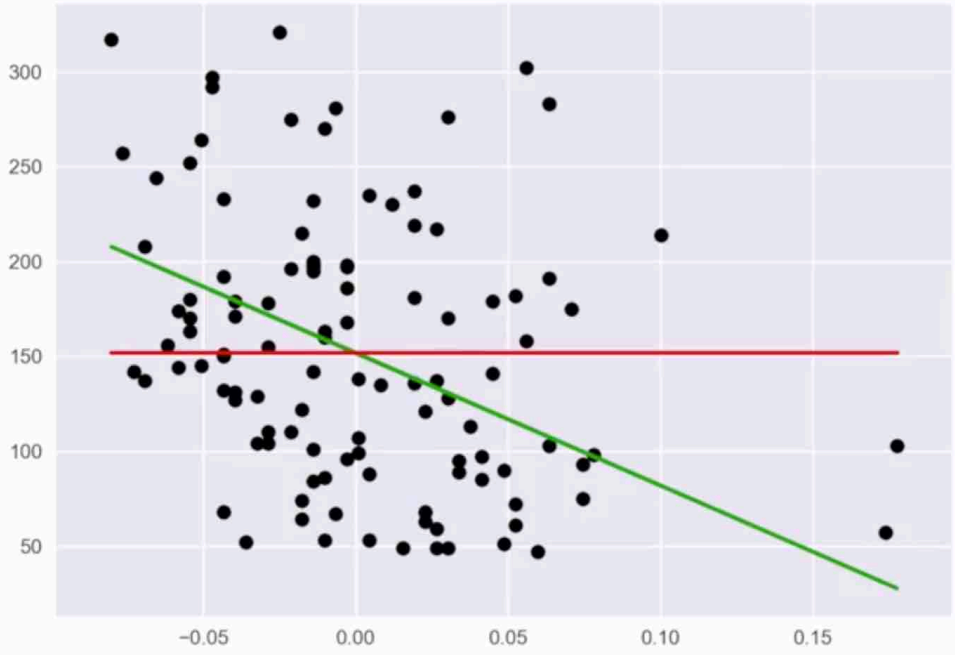
In [ ]:

```
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.dummy import DummyRegressor

diabetes = datasets.load_diabetes()

X = diabetes.data[:, None, 6]
y = diabetes.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

lm = LinearRegression().fit(X_train, y_train)
lm_dummy_mean = DummyRegressor(strategy = 'mean').fit(X_train, y_train)

y_predict = lm.predict(X_test)
y_predict_dummy_mean = lm_dummy_mean.predict(X_test)

print('Linear model, coefficients: ', lm.coef_)
print('Mean squared error (dummy): {:.2f}'
      .format(mean_squared_error(y_test, y_predict_dummy_mean)))
print('Mean squar')
```

```
diabetes = datasets.load_diabetes()

X = diabetes.data[:, None, 6]
y = diabetes.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

lm = LinearRegression().fit(X_train, y_train)
lm_dummy_mean = DummyRegressor(strategy = 'mean').fit(X_train, y_train)

y_predict = lm.predict(X_test)
y_predict_dummy_mean = lm_dummy_mean.predict(X_test)

print('Linear model, coefficients: ', lm.coef_)
print('Mean squared error (dummy): {:.2f}'
      .format(mean_squared_error(y_test, y_predict_dummy_mean)))
print('Mean squared error (linear model): {:.2f}'
      .format(mean_squared_error(y_test, y_predict)))
print('r2_score (dummy): {:.2f}'
      .format(r2_score(y_test, y_predict_dummy_mean)))
print('r2_score (linear model): {:.2f}'
      .format(r2_score(y_test, y_predict)))

plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_predict, color='green', linewidth=2)
plt.plot(X_test, y_predict_dummy_mean, color='red', linestyle = 'dashed',
         linewidth=2, label = 'dummy')

plt.
```

```
Linear model, coefficients:  [-698.80206267]
Mean squared error (dummy): 4965.13
Mean squared error (linear model): 4646.74
r2_score (dummy): -0.00
r2_score (linear model): 0.06
```

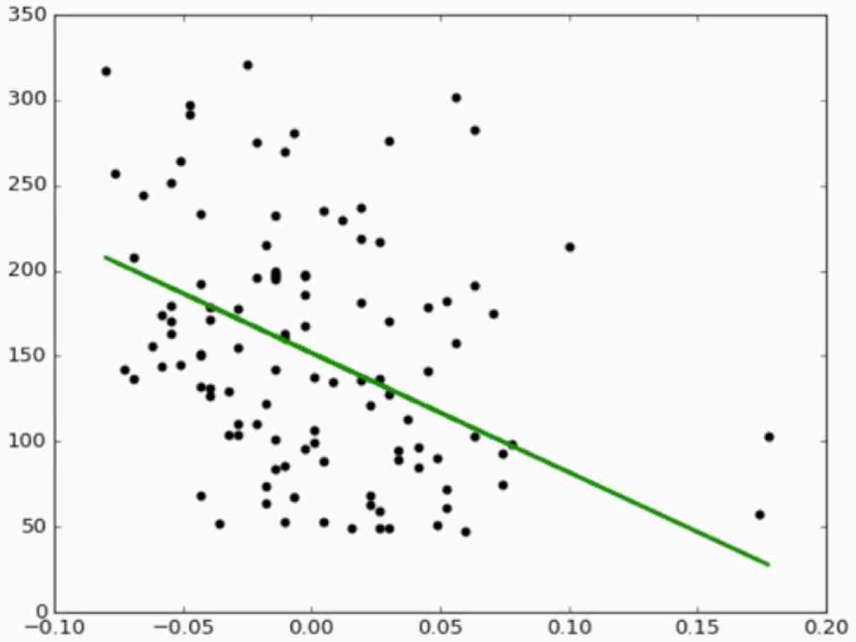
Figure 1

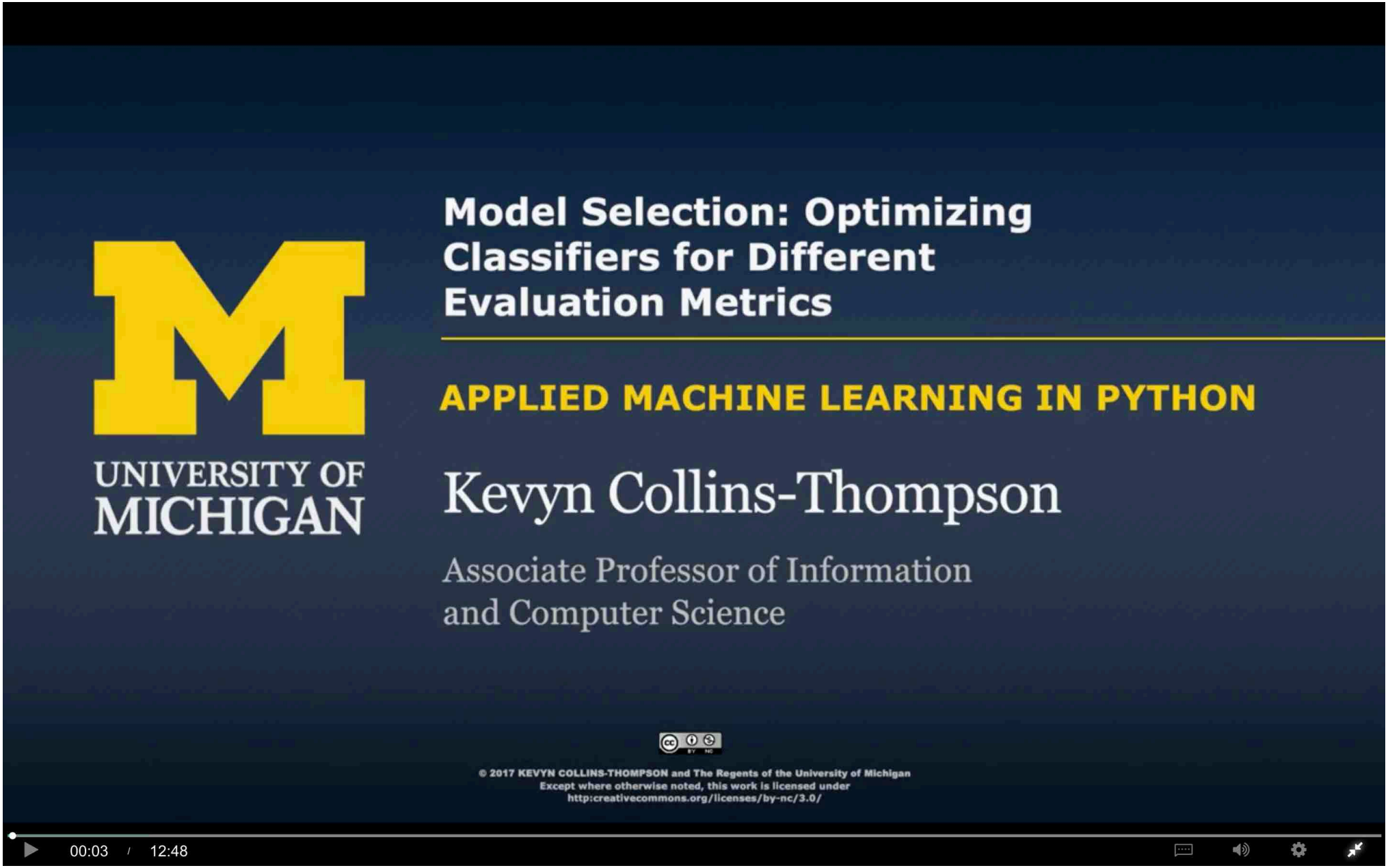


Dummy Regressors

The DummyRegressor class implements four simple baseline rules for regression, using the `strategy` parameter:

- `mean` predicts the mean of the training target values.
- `median` predicts the median of the training target values.
- `quantile` predicts a user-provided quantile of the training target values (e.g. value at the 75th percentile)
- `constant` predicts a custom constant value provided by the user.




Model Selection: Optimizing Classifiers for Different Evaluation Metrics
APPLIED MACHINE LEARNING IN PYTHON
UNIVERSITY OF MICHIGAN
Kevyn Collins-Thompson
Associate Professor of Information and Computer Science
© 2017 KEVYN COLLINS-THOMPSON and The Regents of the University of Michigan
Except where otherwise noted, this work is licensed under
http:creativecommons.org/licenses/by-nc/3.0/
00:03 / 12:48

Model Selection Using Evaluation Metrics

- **Train/test on same data**
  - Single metric.
  - Typically overfits and likely won't generalize well to new data.
  - But can serve as a sanity check: low accuracy on the training set may indicate an implementation problem.
- **Single train/test split**
  - Single metric.
  - Speed and simplicity.
  - Lack of variance information
- **K-fold cross-validation**
  - K train-test splits.
  - Average metric over all splits.
  - Can be combined with parameter grid search: GridSearchCV (def. cv = 3)

Model selection using evaluation metrics

Cross-validation example

In [26]:
```
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

dataset = load_digits()

X, y = dataset.data, dataset.target == 1
clf = SVC(kernel='linear', C=1)

print('Cross-validation (accuracy)',
     cross_val_score(clf, X, y, cv=5))
print('Cross-validation (AUC)',
     cross_val_score(clf, X, y, cv=5, scoring = 'roc_auc'))
print('Cross-validation (recall)',
     cross_val_score(clf, X, y, cv=5, scoring = 'recall'))
```

```
Cross-validation (accuracy) [ 0.91944444  0.98611111  0.97214485  0.97493036  0.96935933]
Cross-validation (AUC) [ 0.9641871   0.9976571   0.99372205  0.99699002  0.98675611]
Cross-validation (recall) [ 0.81081081  0.89189189  0.83333333  0.83333333  0.83333333]
```

In [ ]:

Grid search example

In [ ]:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

dataset = load_digits()

X, y = dataset.data, dataset.target == 1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = SVC(kernel='rbf')
grid_values = {'gamma': [0.001, 0.01, 0.05, 0.1, 1, 10, 100]}


grid_clf_acc = GridSearchCV(clf, param_grid = grid_values)
grid_clf_acc.fit(X_train, y_train)
y_decision_fn_scores_acc = grid_clf_acc.decision_function(X_test)

print('Grid best parameter (max. accuracy): ', grid_clf_acc.best_params_)
print('Grid best score (accuracy): ', grid_clf_acc.best_score_)


grid_clf_auc = GridSearchCV(clf, param_grid = grid_values,
                            scoring = 'roc_auc')
grid_clf_auc.fit(X_train, y_train)
y_decision_fn_scores_auc = grid_clf_auc.decision_function(X_test)

print('Tes')
```

```
X, y = dataset.data, dataset.target == 1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = SVC(kernel='rbf')
grid_values = {'gamma': [0.001, 0.01, 0.05, 0.1, 1, 10, 100]}


grid_clf_acc = GridSearchCV(clf, param_grid = grid_values)
grid_clf_acc.fit(X_train, y_train)
y_decision_fn_scores_acc = grid_clf_acc.decision_function(X_test)

print('Grid best parameter (max. accuracy): ', grid_clf_acc.best_params_)
print('Grid best score (accuracy): ', grid_clf_acc.best_score_)


grid_clf_auc = GridSearchCV(clf, param_grid = grid_values,
                            scoring = 'roc_auc')
grid_clf_auc.fit(X_train, y_train)
y_decision_fn_scores_auc = grid_clf_auc.decision_function(X_test)

print('Test set AUC: ', roc_auc_score(y_test, y_decision_fn_scores_auc))
print('Grid best parameter (max. AUC): ', grid_clf_auc.best_params_)
print('Grid best score (AUC): ', grid_clf_auc.best_score_)
```

```
Grid best parameter (max. accuracy):  {'gamma': 0.001}
Grid best score (accuracy):  0.996288047513
Test set AUC:  0.999828581224
Grid best parameter (max. AUC):  {'gamma': 0.001}
Grid best score (AUC):  0.99987412783
```


```
grid_clf_auc = GridSearchCV(clf, param_grid = grid_values,
                            scoring = 'roc_auc')
grid_clf_auc.fit(X_train, y_train)
y_decision_fn_scores_auc = grid_clf_auc.decision_function(X_test)

print('Test set AUC: ', roc_auc_score(y_test, y_decision_fn_scores_auc))
print('Grid best parameter (max. AUC): ', grid_clf_auc.best_params_)
print('Grid best score (AUC): ', grid_clf_auc.best_score_)
```

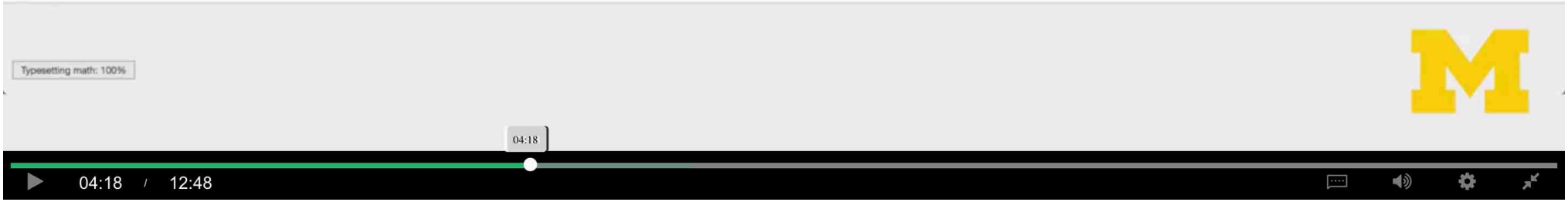
```
Grid best parameter (max. accuracy):  {'gamma': 0.001}
Grid best score (accuracy):  0.996288047513
Test set AUC:  0.999828581224
Grid best parameter (max. AUC):  {'gamma': 0.001}
Grid best score (AUC):  0.99987412783
```

Evaluation metrics supported for model selection

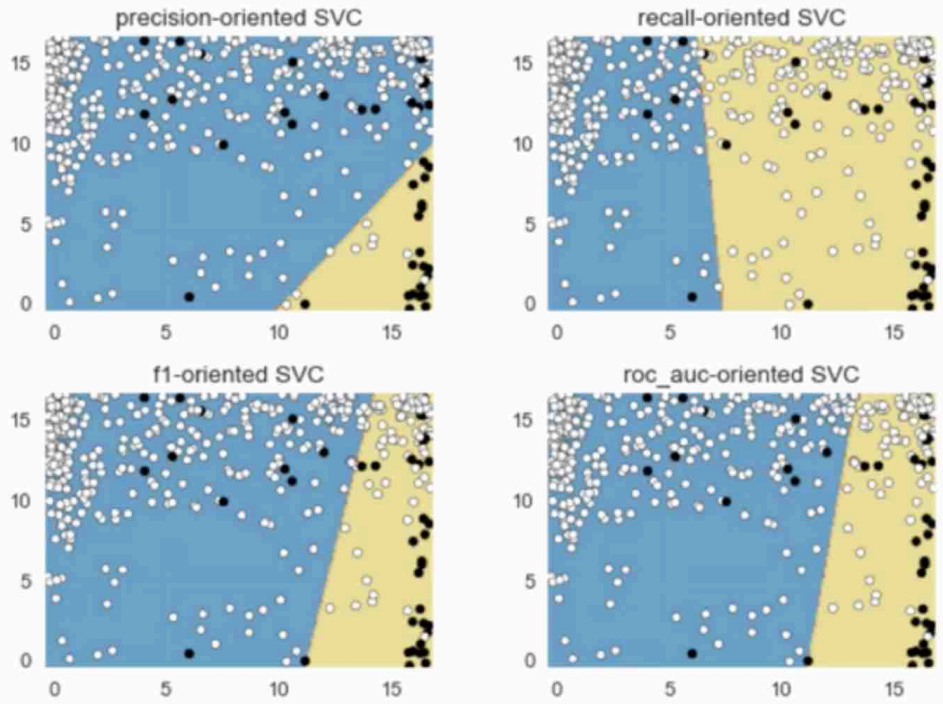
In [28]:
```
from sklearn.metrics.scorer import SCORERS

sorted(list(SCORERS.keys()))
```

Out[28]:
```
['accuracy',
 'adjusted_rand_score',
 'average_precision',
 'f1',
 'f1_macro',
 'f1_micro',
```

```
'f1_samples',
'f1_weighted',
'log_loss',
'mean_absolute_error',
'mean_squared_error',
'median_absolute_error',
'neg_log_loss',
'neg_mean_absolute_error',
'neg_mean_squared_error',
'neg_median_absolute_error',
'precision',
'precision_macro',
'precision_micro',
'precision_samples',
'precision_weighted',
'r2',
'recall',
'recall_macro',
'recall_micro',
'recall_samples',
'recall_weighted',
'roc_auc']
```



Example: Optimizing a Classifier Using Different Evaluation Metrics



Two-feature classification example using the digits dataset

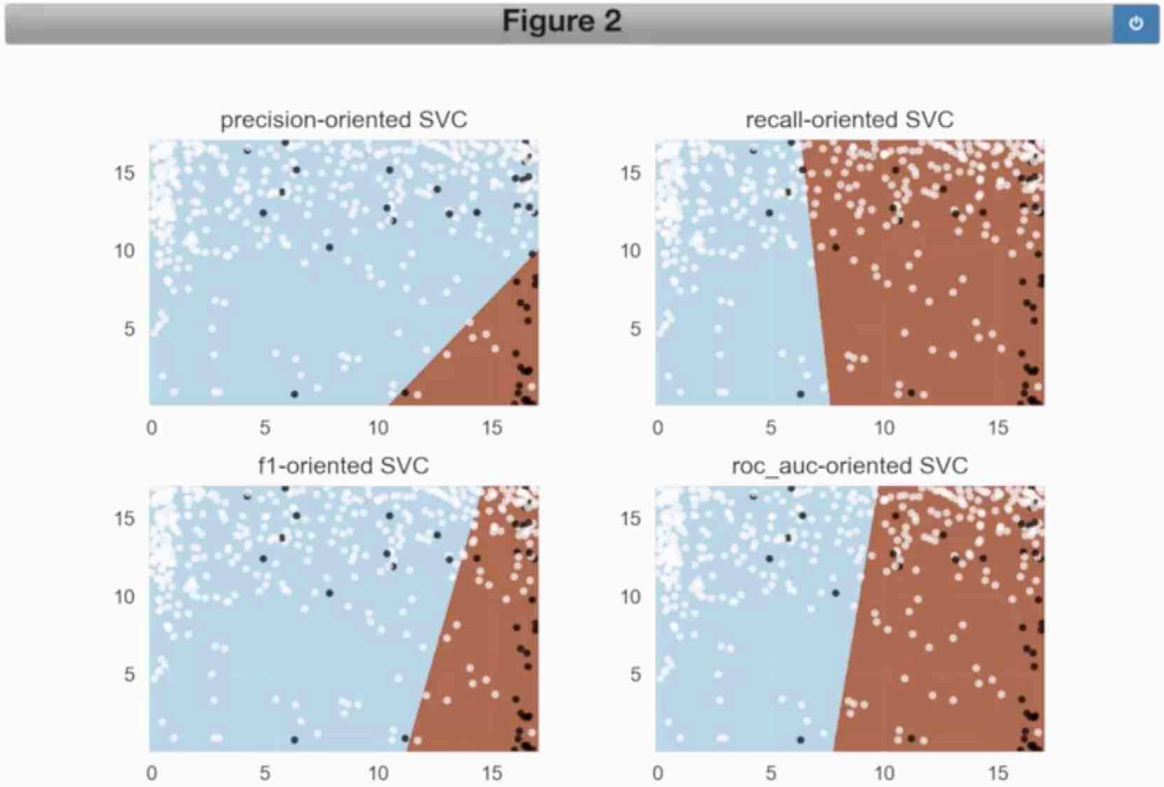
Optimizing a classifier using different evaluation metrics

In [ ]:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from adspy_shared_utilities import plot_class_regions_for_classifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV


dataset = load_digits()

X, y = dataset.data, dataset.target == 1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)


jitter_delta = 0.25
X_twovar_train = X_train[:,[20,59]] + np.random.rand(X_train.shape[0], 2)
- jitter_delta
X_twovar_test = X_test[:,[20,59]] + np.random.rand(X_test.shape[0], 2)
- jitter_delta

clf = SVC(kernel = 'linear').fit(X_twovar_train, y_train)
grid_values = {'class_weight':
               ['balanced', {1:2},{1:3},{1:4},{1:5},{1:10}]}
```

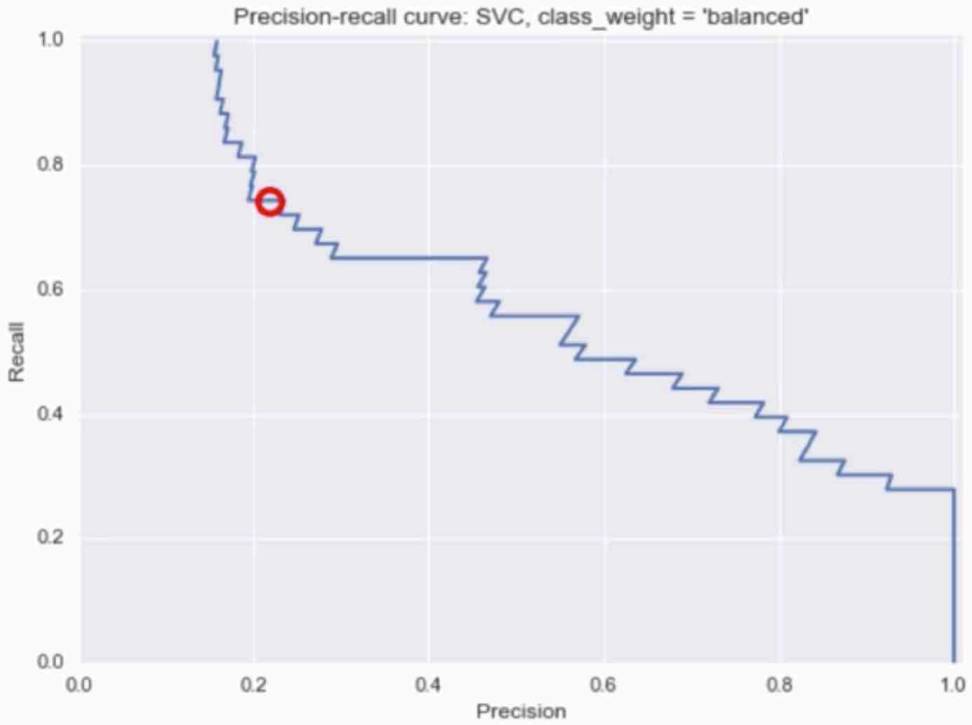
```
X_twovar_train = X_train[:,[20,59]] + np.random.rand(X_train.shape[0], 2)
- jitter_delta
X_twovar_test = X_test[:,[20,59]] + np.random.rand(X_test.shape[0], 2)
- jitter_delta

clf = SVC(kernel = 'linear').fit(X_twovar_train, y_train)
grid_values = {'class_weight':
               ['balanced', {1:2},{1:3},{1:4},{1:5},{1:10},{1:20},{1:50}]}
plt.figure()

for i, eval_metric in enumerate(('precision','recall', 'f1','roc_auc')):
    grid_clf_custom = GridSearchCV(clf, param_grid=grid_values, scoring=eval_metric)
    grid_clf_custom.fit(X_twovar_train, y_train)
    print('Grid best parameter (max. {0}): {1}'
          .format(eval_metric, grid_clf_custom.best_params_))
    print('Grid best score ({0}): {1}'
          .format(eval_metric, grid_clf_custom.best_score_))
    plt.subplot(2, 2, i+1)
    plt.subplots_adjust(wspace=0.3, hspace=0.3)
    plot_class_regions_for_classifier(grid_clf_custom, X_twovar_test, y_test)
    plt.title(eval_metric + '-oriented SVC')

plt.show()
```

In [ ]:




```
Grid best parameter (max. precision): {'class_weight': {1: 2}}
Grid best score (precision): 0.5437429682418546
Grid best parameter (max. recall): {'class_weight': {1: 50}}
Grid best score (recall): 0.935661321592438
```

Example: Precision-Recall Curve of Default Support Vector Classifier



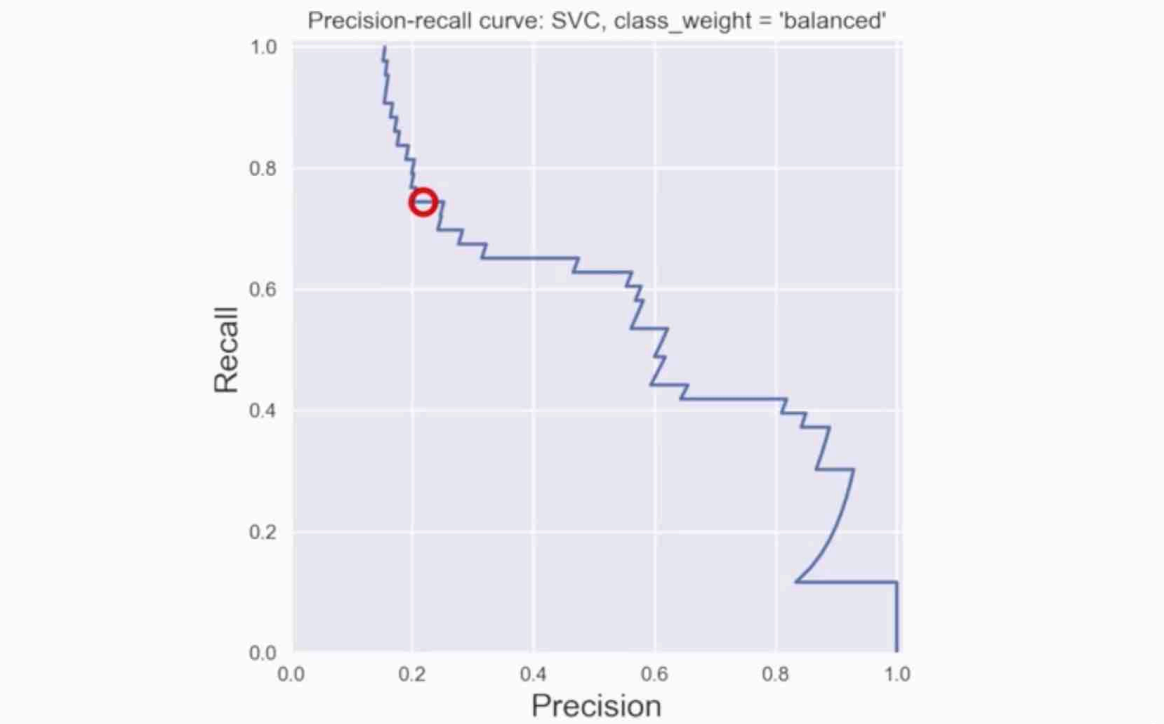
```
Grid best parameter (max. precision): {'class_weight': {1: 2}}
Grid best score (precision): 0.5340299805778648
Grid best parameter (max. recall): {'class_weight': {1: 50}}
Grid best score (recall): 0.9284310837047003
Grid best parameter (max. f1): {'class_weight': {1: 4}}
Grid best score (f1): 0.5003565812625315
Grid best parameter (max. roc_auc): {'class_weight': {1: 20}}
Grid best score (roc_auc): 0.887832176719427
```

Precision-recall curve for the default SVC classifier (with balanced class weights)

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.svm import SVC

dataset = load_digits()

X, y = dataset.data, dataset.target == 1
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)


jitter_delta = 0.25
X_twovar_train = X_train[:,[20,59]]+ np.random.rand(X_train.shape[0], 2)
- jitter_delta
X_twovar_test = X_test[:,[20,59]] + np.random.rand(X_test.shape[0], 2)
- jitter_delta

clf = SVC(kernel='linear', class_weight='balanced').fit(X_twovar_train)
```

```
- jitter_delta

clf = SVC(kernel='linear', class_weight='balanced').fit(X_twovar_train, y_train)

y_scores = clf.decision_function(X_twovar_test)

precision, recall, thresholds = precision_recall_curve(y_test, y_scores)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]

plt.figure()
plot_class_regions_for_classifier(clf, X_twovar_test, y_test)
plt.title("SVC, class_weight = 'balanced', optimized for accuracy")
plt.show()

plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.title ("Precision-recall curve: SVC, class_weight = 'balanced'")
plt.plot(precision, recall, label = 'Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize=12,
         fillstyle='none', c='r', mew=3)
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()

print('At zero threshold, precision: {:}')
```




```
At zero threshold, precision: 0.22, recall: 0.74
```

```
print('At zero threshold, precision: {:.2f}, recall: {:.2f}'
      .format(closest_zero_p, closest_zero_r))
```

Figure 3

Figure 4

Training, Validation, and Test Framework for Model Selection and Evaluation

- Using only cross-validation or a test set to do model selection may lead to more subtle overfitting / optimistic generalization estimates
- Instead, use three data splits:
  1. Training set (model building)
  2. Validation set (model selection)
  3. Test set (final evaluation)
- In practice:
  - Create an initial training/test split
  - Do cross-validation on the training data for model/parameter selection
  - Save the held-out test set for final model evaluation

Concluding Notes

- **Accuracy is often not the right evaluation metric for many real-world machine learning tasks**
  - False positives and false negatives may need to be treated very differently
  - Make sure you understand the needs of your application and choose an evaluation metric that matches your application, user, or business goals.
- **Examples of additional evaluation methods include:**
  - Learning curve: How much does accuracy (or other metric) change as a function of the amount of training data?
  - Sensitivity analysis: How much does accuracy (or other metric) change as a function of key learning parameter values?