Classification Review

Review: What Is Classification?

- Is a supervised machine learning task
- Tries to predict one or more discrete classes as output
- Is trained on input features and a known label

Training and Evaluating a Logistic Regression Model

- Even though it is called regression, it is a linear model for classification
- Also known as:
  - Logit regression
  - Maximum-entropy classification
  - Log-linear classifier
- Let’s start by training a logistics regression model on a small dataset of labelled hand-written digits

Let’s Play With Handwritten Digits

- Scikit-learn has a toy data set to play with analysis of handwritten digits
- Here is how you access it
- It returns a dictionary-type object called a bunch

```
digits = datasets.load_digits()
print(type(digits))
```

Output: <class 'sklearn.utils.Bunch’>

Let’s Look At The Data Structure

- To see the keys in a bunch run the following code:

```
print(digits.keys())
```

```
Output: dict_keys(['data', 'target', 'target_names', 'images', 'DESCR’])
```

- You can see there are five keys, including:
  - Data: an array of one row per instance and one column per feature
  - Target: an array containing the labels
  - DESCR: a description of the data set

Let’s Look at the Data in a Little More Detail

print(digits['DESCR’])

```
Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.
```

How Is the Data Structured?

print(type(digits['data']))

print(len(digits['data’]))

print(type(digits['target']))

```
<class 'numpy.ndarray'>
1797
<class 'numpy.ndarray'>
```

Question

- How can you get a count of each digit from the target set?
- Hint: look at the collections library

Classification Review

The End

What Does the Data Look Like?

Answer

- How can you get a count of each digit from the target set?

```
import collections
print(collections.Counter(digits['target’]))
```

```
Counter({3: 183, 1: 182, 5: 182, 4: 181, 6: 181, 9: 180, 7: 179, 0: 178, 2: 177, 8: 174})
```

What Does the Data Look Like?

```
some_digit = digits.data[33]
some_image = digits.images[33]
some_label = digits.target[33]
print(some_digit)
print(some_digit.shape)
print(some_image)
print(some_image.shape)
print(some_label)
```

```
[ 0.   6. 13.   5.   8.   8.   1.   0.   0.   8. 16. 16. 16. 16.   6.   0.   0.   6.
 16.   9.   6.   4.   0.   0.   0.   6. 16. 16. 15.   5.   0.   0.   0.   0.   4.   5.
 15. 12.   0.   0.   0.   0.   0.   3. 16.   9.   0.   0.   0.   1.   8. 13. 15.   3.
  0.   0.   0.   4. 16. 15.   3.   0.   0.   0.]
(64,)
[[ 0.   6. 13.   5.   8.   8.   1.   0.]
 [ 0.   8. 16. 16. 16. 16.   6.   0.]
 [ 0.   6. 16.   9.   6.   4.   0.   0.]
 [ 0.   6. 16. 16. 15.   5.   0.   0.]
 [ 0.   0.   4.   5. 15. 12.   0.   0.]
 [ 0.   0.   0.   3. 16.   9.   0.   0.]
 [ 0.   1.   8. 13. 15.   3.   0.   0.]
 [ 0.   4. 16. 15.   3.   0.   0.   0.]]
(8, 8)
5
```

Question

- How can you take this Numpy array and graph it?
- Hint: look at imshow from matplotlib

What Does the Data Look Like?

The End

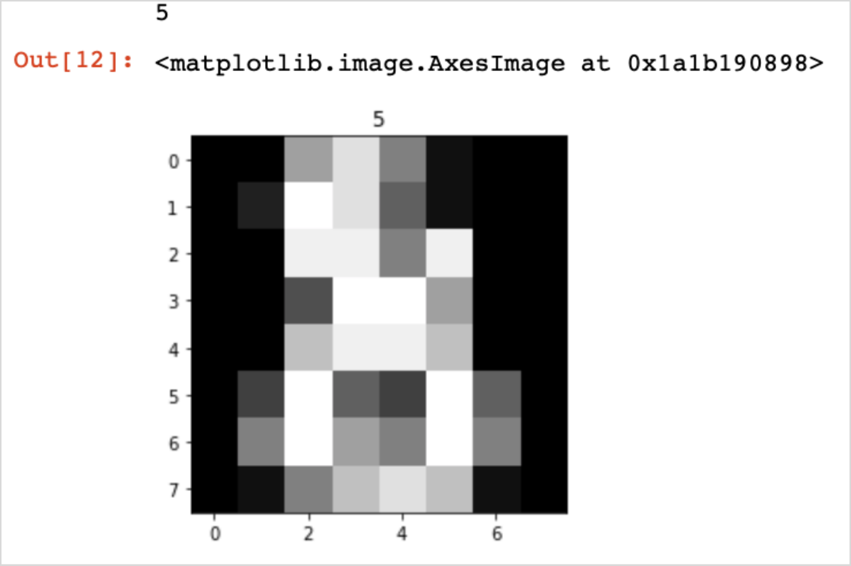
Plotting a Number

Answer: Graphing a Number

- How can you take this Numpy array and graph it?

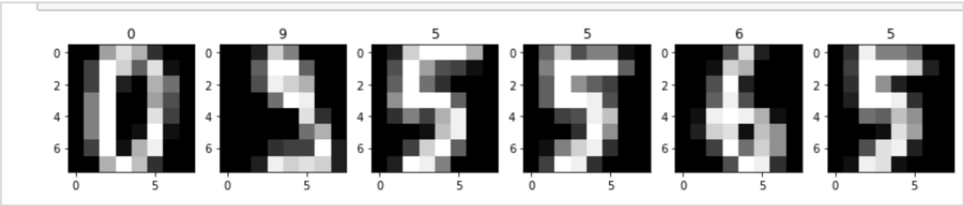
plt.title(some_label)

plt.imshow(digits.images[-1], cmap=plt.cm.gray)



Here Is Some Code to Plot a Range of Them

```
plt.figure(figsize = (24,24))
for element, (image, label) in enumerate(zip(digits.data[30:40],digits.target[30:40])):
    plt.subplot(1,10,element +1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('%i' % label)
```



Exercise: Splitting your data

- Split your data into a training and test set with 20% going towards your test set
- Hint: the constructor you need is in scikit-learn.model_selection

Plotting a Number

The End

Looking at Your Training Set

Answer: Splitting Your Data

```
#Let's use train_test_split to split our data
d_train, d_test, l_train, l_test =
train_test_split(digits.data, digits.target,
test_size=0.20, random_state=0)
```

Let’s Look at Our Shuffled Labels

- Run this code to look at the shape and shuffling of the training set labels

    print(l_train.shape)

    print(l_train[:5])

```
(1437,)
[6 5 3 4 4]
```

Let’s Create a Binary Classifier

- Pick a digit and let’s train a model on just that digit or not
- Question: how do we come up with a Boolean array (an array that only has true/false values in it?
- Hint: Look at Numpy’s logic operators (==)

Looking at Your Training Set

The End

Training a Binary Classifier

Answer: Boolean Array

- Here is how we create a Boolean array of trues for just one number:

```
l_train_5 = (l_train == 5)
l_test_5 = (l_test == 5)
print(l_train_5[:5])
```

```
[False  True False False False]
```

- Comparing this against the labels of our training set:

```
[6 5 3 4 4]
```

- You can see 5 is evaluated as true:

Instantiate Your Model

- Here is the code we use to implement a logistic regression classifier in scikit-learn

```
from sklearn.linear_model import LogisticRegression
lr_class = LogisticRegression()
print(type(lr_class))
```

```
<class 'sklearn.linear_model._logistic.LogisticRegression'>
```

Setting Your Model Parameters

- Most models in scikit-learn have paramaters that are used to set their architecture
- To see the paramaters available for a particular model you can run .get_params()
- To set them you use .set_params()

```
lr_class.get_params()
```

```
<class 'sklearn.linear_model._logistic.LogisticRegression'>

{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

```
lr_class.set_params(solver='lbfgs',max_iter=300)
```

Fit Your Model

- Here is the code to fit your model on our binary dataset:

```
lr_class.fit(d_train, l_train_5)
```

- And here is how you predict:

```
print(lr_class.predict([some_digit]))
```

```
[ True]
```

Let’s Compare Truth Against Predicted

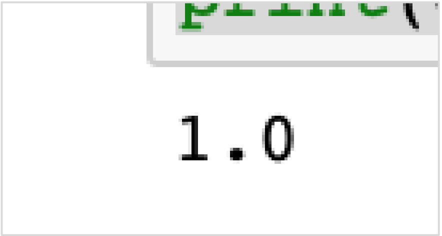
```
In [30]: #predicted class
         lr_class.predict(d_train[0:10])

Out[30]: array([False,  True, False, False, False, False, False, False, False,
                False])

In [31]: #true class
         l_train_5[0:10]

Out[31]: array([False,  True, False, False, False, False, False, False, False,
                False])
```

How Accurate Are We?

- Let's see how accurate we are:

```
from sklearn.metrics import accuracy_score
digit_predictions = lr_class.predict(d_train)
digit_accuracy = accuracy_score(l_train_5,digit_predictions)
print(digit_accuracy)
```

- Wow! 100% accuracy



Cross Validation/Better Accuracy

- Something seems fishy
- Let's do cross-validation like we did last time

```
from sklearn.model_selection import cross_val_score
cross_val_score(lr_class,d_train,l_train_5, cv=3, scoring="accuracy")
```

```
array([0.98956159, 0.9874739 , 0.99791232])
```

- Still 98%, not bad!

What Is Going On Here?

- Remember, we only have roughly 180 5s, and around 1200 not 5s
- Accuracy is not a good measure when your data is highly skewed like this
- So what other options do we have?
- The answer is to categorize predictions as compared to actual labels
- A common way to do this is a confusion matrix

Training a Binary Classifier

The End

What Is a Confusion Matrix?

Confusion Matrix Defined

- A common way to evaluate the performance of a classifier
- Each row in a confusion matrix represents an actual class
- Each column represents a predicted class
- Let’s look at our current example:

	Predicted Negative	Predicted Positive
Actual Negative	**1200**	**3**
Actual Positive	8	126

Cross Validation Prediction

- Instead of using cross_val_score, like last time, this time we will use cross_val_predict:

```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(lr_class, d_train, l_train_5, cv=3)
```

- This takes our classifier, our training data, and our binary label as input
- It outputs predictions based on the trained model for the training set

Confusion Matrix Constructor

- Next, we import a confusion matrix with our training label and predicited label:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(l_train_5, y_train_pred)
```

```
Out[44]: array([[1291,    4],
                [   8,  134]])
```

Reading a Confusion Matrix

- First row, first column (1200) is called true negative (TN)
- First row second column (3) is called false positive (FP)

	Predicted Negative	Predicted Positive
Actual Negative	1200 TN	3 FP
Actual Positive	8	126

Reading a Confusion Matrix (cont.)

- Second row, first column (8) were predicted negative but actually positive so are false negatives (FN)
- Second row, second column (126) are true positives (TP)

	Predicted Negative	Predicted Positive
Actual Negative	**1200**	**3**
Actual Positive	8 **FN**	126 **TP**

What Is a Confusion Matrix?

The End

Performance Measures

How Do We Calculate Accuracy?

- Accuracy is ratio of correct predictions

$$(TP + TN)/(TN + FP + FN + TP)$$

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Other Metrics: Precision

- The accuracy when the model thinks it has it right

$$(TP)/(TP + FP)$$

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Precision is calculated down the second column

When to Use Precision

- Use precision when your data is highly skewed (when some classes happen more often than others)
- Use precision when you want to minimize false positives
- Example: spam detection
- If you have a lot of false positives you won’t see valid emails that you need to act on

Other Metrics: Recall

- Also called sensitivity or true positive rate

$$(TP)/(TP + FN)$$

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Recall is calculated across the second row

When to Use Recall

- Use Recall when you want to minimize the impact of false negatives
- Example: medical diagnosis
- If you are sick but the model say you aren’t
  - They will send you home
  - They will not start treatment
  - You might not get better

F1 Score

- Harmonic mean of precision and recall

$$2 \times \frac{precision \times recall}{precision + recall}$$

$$= \frac{TP}{TP + \frac{FN + FP}{2}}$$

- Harmonic mean gives weight to low values
- A high F1 score requires precision and recall to both be high

Performance Measures

The End

Precision and Recall Tradeoff

Precision/Recall Tradeoff

- Increasing precision reduces recall and visa versa
- A classifier chooses a score based on a decision function
- If a score is greater than a threshold, it assigns the instance a positive class, otherwise it will be assigned to the negative class
- Increasing the threshold will increase the precision but lower the recall
- Decreasing the threshold increases recall and reduces precision

Playing Around With the Decision Function Threshold

- Scikit-learn doesn’t let you set the decision function threshold outright
- You can output the threshold value of a specific prediction
- Then set an arbitrary threshold and see how it affects the prediction

Changing Decision Function Threshold

```
d_scores = lr_class.decision_function([some_digit])
print(d_scores)
threshold = 0
y_some_digit_pred = (d_scores > threshold)
print(y_some_digit_pred)
```

```
[13.65105929]
[ True]
```

```
threshold = 20
y_some_digit_pred = (d_scores > threshold)
print(y_some_digit_pred)
```

```
[False]
```

Comparing Threshold, Precision and Recall

- Scikit-learn has a module called the precision_recall_curve that produces a Numpy array of precision and recall for all the values of threshold
- To use it you need to figure out the decision function threshold for every predicted point:

```
y_scores = cross_val_predict(lr_class, d_train, l_train_5, cv=3,
        method="decision_function")
```

Precision Recall Curve

- You provide the precision_recall_curve values from your dataset labels and the threshold results from the decision function cross-val

```
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(l_train_5, y_scores)
print(type(precisions))
print(precisions[:5])
```

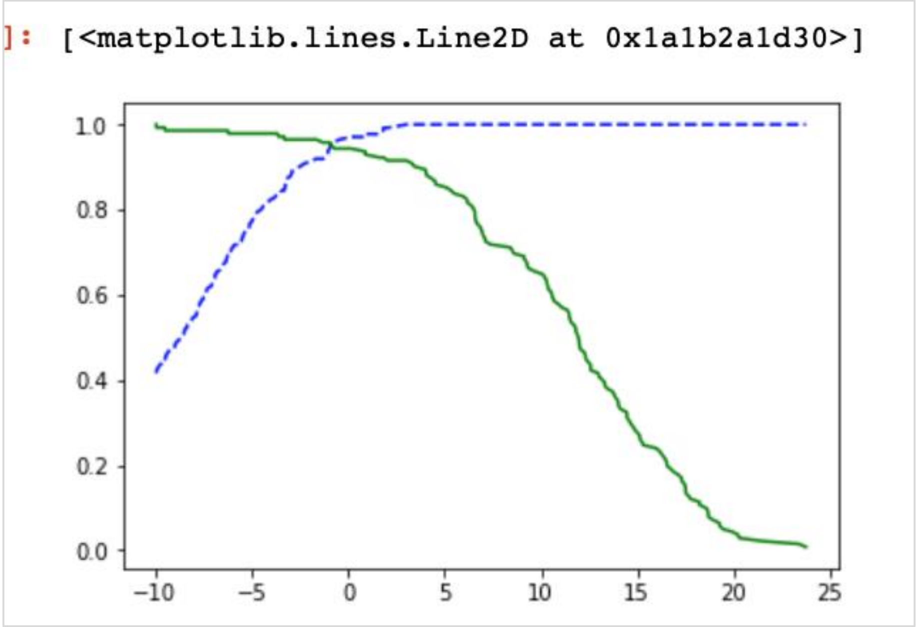
```
<class 'numpy.ndarray'>
[0.42011834 0.41839763 0.41964286 0.42089552 0.42215569]
```

Graphing a Precision Recall Curve

- To graph a precision recall curve use matplotlib
- Threshold is your x-axis
- Make precision blue
- Make recall green

```
plt.plot(thresholds, precisions[:-1], "b--",label="Precision")
plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
```

- You can see as precision increases, recall decreases



Question: How Would You Plot Precision Against Recall?

- Hint: Let recall be the x axis and precision be the y axis

Question: Specifying Precision

- How do you return results with only a certain precision?
- Hint: Look for threshold values where precision is greater than what you want (look at np.argmax)

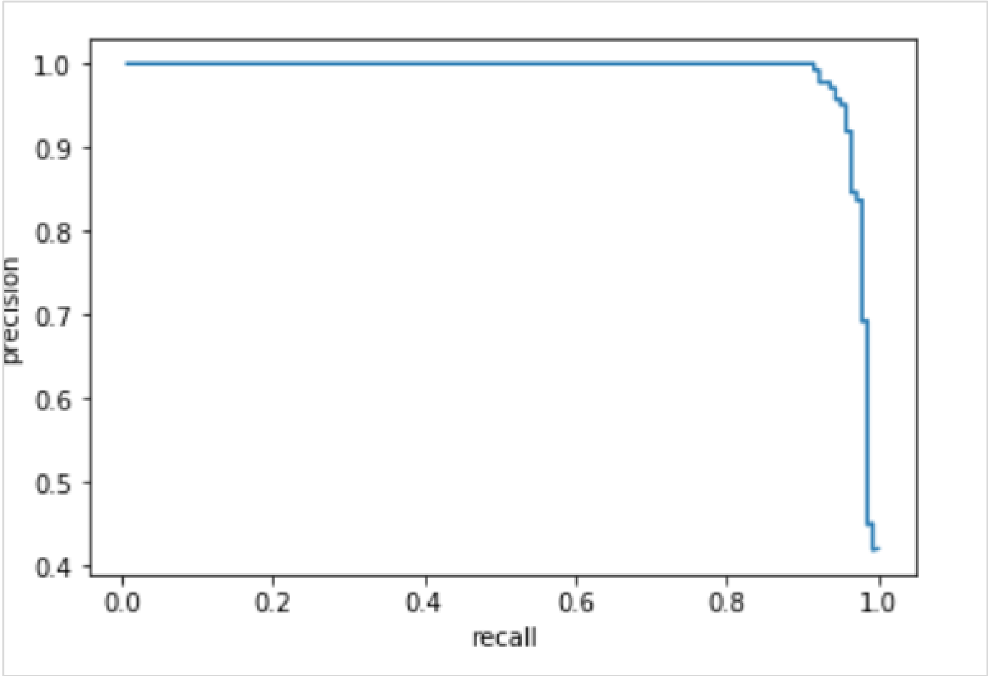
Precision and Recall Tradeoff

The End

Receiver Operating Characteristic

Answer: Plotting Precision Against Recall

- plt.plot(recalls[:-1],precisions[:-1])
- plt.xlabel("recall")
- plt.ylabel("precision")
- plt.show()



Answer: How Do You Guarantee a Certain Precision?

- Take the argmax of the Boolean array with the precisions greater than or equal to 0.90
- The argmax returns the index of the first value which is true
- Then create another Boolean array where your threshold is greater than that value
- Boom! You have a classifier with a guaranteed precision

```
#what is the threshold value that reaches 90% precision?
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
print(threshold_90_precision)
print(y_scores[:5])
#now look for scores higher than that
y_train_pred_90 = (y_scores >= threshold_90_precision)
print(type(y_train_pred_90))
print(y_train_pred_90.dtype)
print(y_train_pred_90[:5])
```

```
-2.5157767701023896
[-20.29182243   4.55489446  -9.61036663 -18.41095695 -22.0786528 ]
<class 'numpy.ndarray'>
bool
[False  True False False False]
```

Receiver Operating Characteristic

- The receiver operating characteristic is another evaluation tool used for binary classifiers
- It plots true positive rate (another name for recall) against the false positive rate
- False positive rate is ratio of instances labelled negative that the model thought was positive
- Scikit-learn has a method for calculating these metrics against various threshold values: the roc_curve

Scikit-learn roc_curve() Function

- Here is the code to compute these metrics:

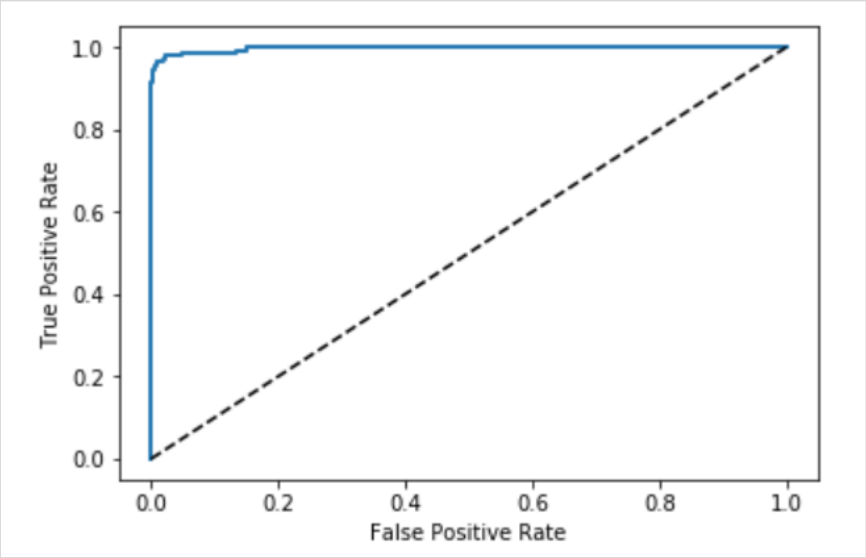
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(l_train_5, y_scores)
```

- And to graph it:

```
plt.plot(fpr, tpr, linewidth=2, label=label)
plt.plot([0,],[0,1],‘k--)
```

Reading a ROC Curve

- The diagonal line represents a random classifier
- You want the curve to be up near the top left corner, so you minimize the false positives even when recall goes up
- If you need to compare multiple models you can use scikit-learn's area under the curve score (AUC) to see how close to the upper left corner they are



ROC Area Under the Curve

- Here is the code to implement a ROC AUC score in scikit-learn:

```
from sklearn.metrics import roc_auc_score
roc_auc_score(l_train_5, y_scores)
```

```
: 0.9971830985915493
```

- A score closer to 1 is better than a score of .5 (a random classifier)

Receiver Operating Characteristic

The End