Classification

Elliott Ash

Text Data Course, Bocconi 2018

Regression vs. Classification

- In economics and other social sciences, most empirical analysis has an ordered, real-valued or binary (one-dimensional), outcome.
  - Regression gives us a linear prediction of $Y \in \mathbb{R}$ given the predictors X.
- But what if the outcome Y is a multi-dimensional classification, rather than a number?
  - For example, deciding the legal area of a court case, when there are dozens of areas.
    - There's no way to line legal areas up in one dimension.
  - This is a **classification task** rather than a regression task.

Document Classification

- Say you have a set of documents and a set of legal areas.
  - You only know the legal area for a subset of documents which have been hand-coded.
  - Can we use the existing labels, and the text of all cases, to machine-code the labels for the unlabelled cases?

Binary classifier

```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(max_iter=10)
sgd_clf.fit(train[features], train['any_cites'])
sgd_clf.score(test[features],test['any_cites'])

# compare to picking largest category
df1['any_cites'].mean()
```

cross_val_predict()

```
from sklearn.model_selection import cross_val_predi
df1['any_cites_sgd'] = cross_val_predict(sgd_clf,
                                         df1[features],
                                         df1['any_cites'],
                                         cv=3)
```

- Provides a "clean" prediction for each row, in the sense that it is trained on data outside that row's fold.

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix(df1['any_cites'],
                 df1['any_cites_sgd'])
```

- A confusion matrix is a nice way to visualize the performance of a classifier:

		Predicted Class	
		Negative	Positive
Actual Class	Negative	True Negatives	False Positives
	Positive	False Negatives	True Positives

- The values in the table give counts in the evaluation set.

Precision and Recall

		Predicted Class	
		Negative	Positive
Actual Class	Negative	True Negatives	False Positives
	Positive	False Negatives	True Positives

- Two alternative metrics used to understand classifers:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

```
from sklearn.metrics import precision_score, recall_score
precision_score(df1['any_cites'], df1['any_cites_sgd'])
recall_score(df1['any_cites'], df1['any_cites_sgd'])
```

F1 Score

- The F_1 score (also sometimes called F score) provides a single combined metric – it is the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} =$$

$$= \frac{\text{Total Positives}}{\text{Total Positives} + \frac{1}{2}(\text{False Negatives} + \text{False Positives})}$$

```
from sklearn.metrics import f1_score
f1_score(df1['any_cites'], df1['any_cites_sgd'])
```

The Precision/Recall Tradeoff

- In general, one can tweak a classifier to increase precision at the cost of reducing recall, and vice versa.
  - The F1 score values them symmetrically
  - But one can imagine contexts where they should be valued asymmetrically:
    - in the case of deciding "guilty" in court, you might prefer a model that let's many actual-guilty go free (high false negatives $\leftrightarrow$ low recall) but has very few actual-innocent put in jail (low false positives $\leftrightarrow$ high precision).
    - in the case of detecting bombs during flight screening, you might prefer a model that has many false alarms (low precision) to minimize the number of misses (high recall).

How sklearn predicts: decision_function()

- The predict() method works by calling a decision_function() method to produce a score for each row, and then predict a label based on a threshold rule.

```
y_scores = sgd_clf.decision_function(df1[features])
plt.hist(y_scores) # histogram of scores

# prediction using default threshold ...
threshold = 0
(y_scores > threshold).mean()

# ... gives default model prediction
ypred = sgd_clf.predict(df1[features])
ypred.mean()

# increasing threshold means more zeros are predicted
threshold = 1
(y_scores > threshold).mean()
```

- SGDClassifer uses a threshold of zero by default

Visualizing the Precision/Recall Tradeoff

```
# Visualizing the precision/recall tradeoff
y_scores = cross_val_predict(sgd_clf,
                             df1[features],
                             df1['any_cites'],
                             cv=3,
                             method='decision_function',
                             n_jobs=3)

from sklearn.metrics import precision_recall_curve
metrics = precision_recall_curve(df1['any_cites'],
                                 y_scores)
precisions, recalls, thresholds = metrics
plt.plot(thresholds, precisions[:-1], label="Precision")
plt.plot(thresholds, recalls[:-1], label="Recall")
plt.xlabel('Threshold')
plt.legend()

# Plot precisions vs recall
plt.step(recalls, precisions)
plt.xlabel('Recall')
plt.ylabel('Precision')
```

Assessing other thresholds

```
ypred_lower = y_scores > -1
ypred_higher = y_scores > 1

precision_score(y, ypred_lower)
precision_score(y, ypred_higher)
recall_score(y, ypred_lower)
recall_score(y, ypred_higher)
```

ROC Curve and AUC

- Plots recall (the true positive rate) against the false positive rate.

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y, y_scores)
plt.plot(fpr, tpr)
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

- The area under the ROC curve (AUC) is a popular metric ranging between 0.5 (random classification) and 1 (perfect classification)

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

Random Forests and `predict_proba()`

- Some classifiers, such as RandomForestClassifier, do not have a `decision_function()` method.
  - The `predict_proba()` method produces a predicted probability across classes for each row.
  - The `predict()` method chooses the class with the highest predicted probability.

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
y_probas_rfc = cross_val_predict(rfc,
                                 X,
                                 y,
                                 cv=3,
                                 method='predict_proba',
                                 n_jobs=3)
```

Using `predict_proba()` for scoring

```
y_scores_rfc = y_probas_rfc[:,1]
roc_metrics = roc_curve(y, y_scores_rfc)
fpr_rfc, tpr_rfc, thresholds_rfc = roc_metrics

# compare precision/recall tradeoff for SGD and RF
plt.plot(fpr, tpr, label="SGD")
plt.plot(fpr_rfc, tpr_rfc, label="RF")
plt.legend()

roc_auc_score(y, y_scores_rfc)
```

Multi-Class Models

- Some algorithms (such as random forests) are designed to handle multiple classes directly. But binary classifiers can be generalized to multiple classes:
  - *One-versus-All strategy:* train a different model for each class, and then choose the class whose model outputs the highest score
  - *One-versus-One strategy:* train a different model for each pair of classes, and then choose the class that wins the most pairwise comparisons.
  - OvO requires the training of $N \times (N-1)/2$ models, so OvA is usually preferred.

Multi-Class Models

Scikit-learn detects automatically when you want to do multi-class classification.

```
statepred = cross_val_predict(sgd_clf,
                              X,
                              state,
                              cv=3)
print((statepred == state).mean())
list(zip(state[:8], statepred[:8]))
```

Multi-Class Confusion Matrix

		Predicted Class		
		Class A	Class B	Class C
Actual Class	Class A	Correct A	A, classed as B	A, classed as C
	Class B	B, classed as A	Correct B	B, classed as C
	Class C	C, classed as A	C, classed as B	Correct C

```
conf_mx = confusion_matrix(state,statepred)
conf_mx
plt.matshow(conf_mx)

# normalize colors
conf_mx_norm = conf_mx / conf_mx.sum(axis=1, keepdims=True)
plt.matshow(conf_mx_norm)
```