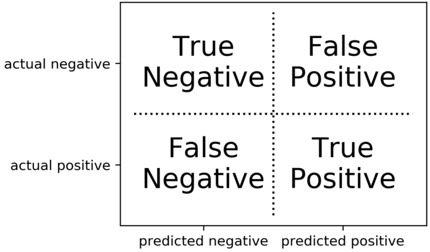
Module-4.7-Model Evaluation

Presented by Yasin Ceran

Table of Contents

1. Metrics for Binary Classification
2. Multi-class classification
3. Metrics for Regression Models
4. Imbalanced Data

Linear Models for Binary Classification

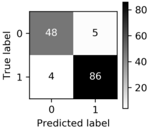


$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Example with ScikitLearn

```
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
data = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, stratify=data.target, random_state=0)

lr = LogisticRegression().fit(X_train, y_train)
y_pred = lr.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(lr.score(X_test, y_test))
plot_confusion_matrix(lr, X_test, y_test, cmap='gray_r')
```

```
[[48  5]
 [ 4 86]]
0.94
```

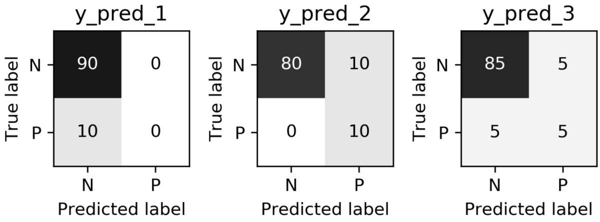


Problems with Accuracy

Data with 90% negatives:

```python
from sklearn.metrics import accuracy_score
for y_pred in [y_pred_1, y_pred_2, y_pred_3]:
    print(accuracy_score(y_true, y_pred))
```

```
0.9
0.9
0.9
```



Precision, Recall, and f-score

Precision, Recall, f-score

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

Positive Predicted Value (PPV)

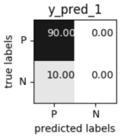
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Sensitivity, coverage, true positive rate.

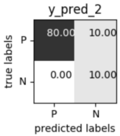
$$\text{F} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Harmonic mean of precision and recall

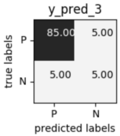
Precision, Recall, and f-score: Example



	precision	recall	f1-score	support
0	0.90	1.00	0.95	90
1	0.00	0.00	0.00	10
avg / total	0.81	0.90	0.85	100



	precision	recall	f1-score	support
0	1.00	0.89	0.94	90
1	0.50	1.00	0.67	10
avg / total	0.95	0.90	0.91	100



	precision	recall	f1-score	support
0	0.94	0.94	0.94	90
1	0.50	0.50	0.50	10
avg / total	0.90	0.90	0.90	100

Changing Threshold

```
data = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, stratify=data.target, random_state=0)

lr = LogisticRegression().fit(X_train, y_train)
y_pred = lr.predict(X_test)

print(classification_report(y_test, y_pred))
```

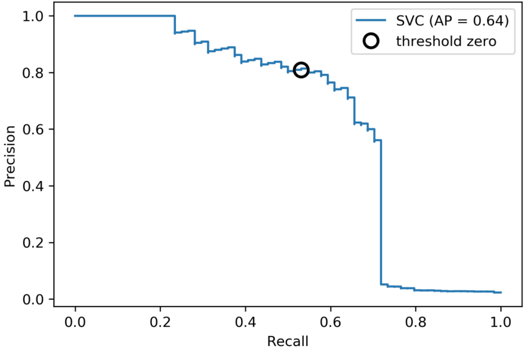
```
          precision    recall  f1-score   support
0              0.91      0.92      0.92        53
1              0.96      0.94      0.95        90
avg/total      0.94      0.94      0.94       143
```

```
y_pred = lr.predict_proba(X_test)[:, 1] > .85

print(classification_report(y_test, y_pred))
```

```
          precision    recall  f1-score   support
0              0.84      1.00      0.91        53
1              1.00      0.89      0.94        90
avg/total      0.94      0.93      0.93       143
```

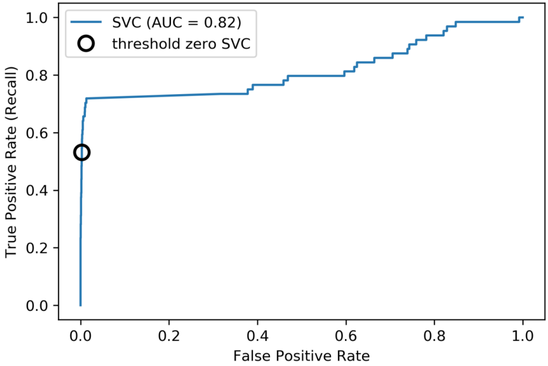
Precision-Recall Curve

```
svc = make_pipeline(StandardScaler(), SVC(C=100, gamma=0.1))
svc.fit(X_train, y_train)
plot_precision_recall_curve(svc, X_test, y_test, name='SVC')
```

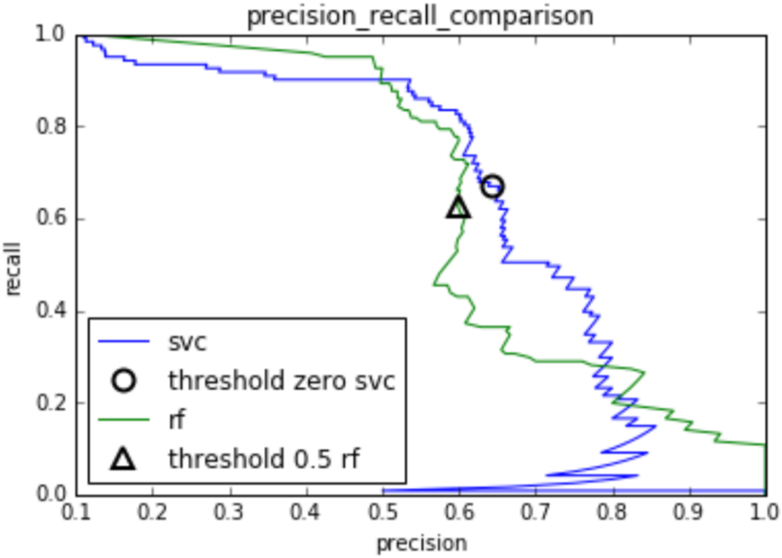


ROC (Receiver Operating Characteristics) Curve

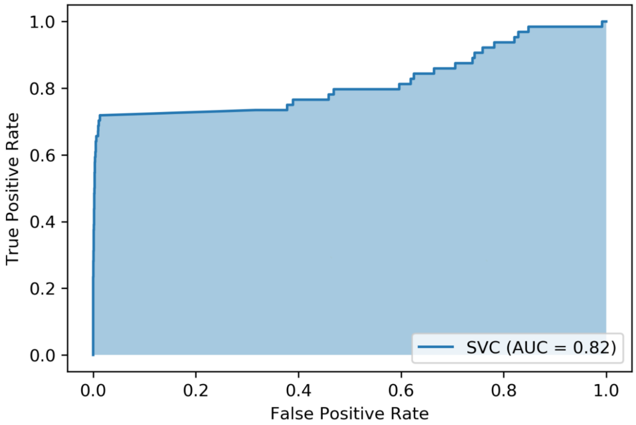
```
plot_roc_curve(svc, X_test, y_test, name='SVC')
```



Comparing RF and SVC



Area Under ROC Curve (AUC)



- Always .5 for random/constant prediction

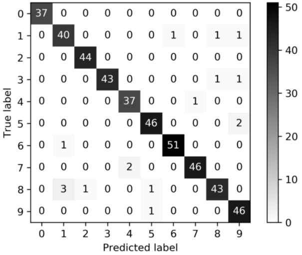
Table of Contents

1. Metrics for Binary Classification
2. Multi-class classification
3. Metrics for Regression Models
4. Imbalanced Data

Confusion Matrix

```python
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score

digits = load_digits()
 # data is between 0 and 16
X_train, X_test, y_train, y_test = train_test_split(
    digits.data / 16., digits.target, random_state=0)
lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
plot_confusion_matrix(lr, X_test, y_test, cmap='gray_r')
```

```
Accuracy: 0.964
```




```python
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        37
           1       0.91      0.93      0.92        43
           2       0.98      1.00      0.99        44
           3       1.00      0.96      0.98        45
           4       0.95      0.97      0.96        38
           5       0.96      0.96      0.96        48
           6       0.98      0.98      0.98        52
           7       0.98      0.96      0.97        48
           8       0.96      0.90      0.92        48
           9       0.92      0.98      0.95        47

    accuracy                           0.96       450
   macro avg       0.96      0.96      0.96       450
weighted avg       0.96      0.96      0.96       450
```

Multi-class ROC AUC

- Hand & Till, 2001, one vs one

$$\frac{1}{c(c-1)} \sum_{j=1}^{c} \sum_{k \neq j}^{c} AUC(j,k)$$

- Provost & Domingo, 2000, one vs rest

$$\frac{1}{c} \sum_{j=1}^{c} p(j) AUC(j, \mathrm{rest}_j)$$

Table of Contents

1. Metrics for Binary Classification

2. Multi-class classification

3. Metrics for Regression Models

4. Imbalanced Data

Standard Metrics

- R^2 : easy to understand scale
- MSE : easy to relate to input
- Mean absolute error, median absolute error: more robust

Absolute vs Relative: MAPE

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y - \hat{y}}{y} \right|$$

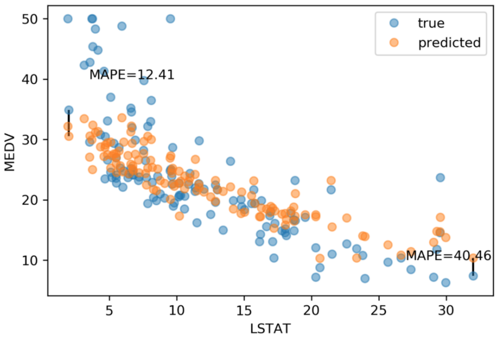


Table of Contents

1. Metrics for Binary Classification

2. Multi-class classification

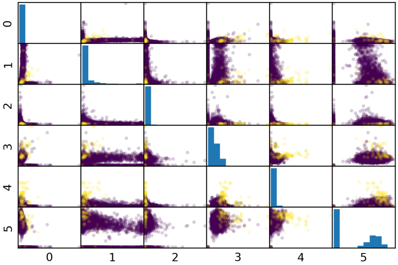
3. Metrics for Regression Models

4. Imbalanced Data

Imbalanced Data

- Two sources of imbalance: Asymmetric Cost and Asymmetric Data

```
from sklearn.datasets import fetch_openml
# mammography https://www.openml.org/d/310
data = fetch_openml('mammography', as_frame=True)
X, y = data.data, data.target
X.shape

(11183, 6)

y.value_counts()

-1    10923
1       260

# make y boolean
# this allows sklearn to determine the positive class more easily
X_train, X_test, y_train, y_test = train_test_split(X, y == '1', random_state=0)
```



Mammography Data

```
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LogisticRegression

scores = cross_validate(LogisticRegression(),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))


scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()

0.920, 0.630




from sklearn.ensemble import RandomForestClassifier
scores = cross_validate(RandomForestClassifier(),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))


scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()

0.939, 0.722
```

Random Under-Sampling

pip install -U imbalanced-learn Extends sklearn API

```
from imblearn.under_sampling import RandomUnderSampler


rus = RandomUnderSampler(replacement=False)


X_train_subsample, y_train_subsample = rus.fit_sample(
    X_train, y_train)


print(X_train.shape)
print(X_train_subsample.shape)
print(np.bincount(y_train_subsample))

(8387, 6)
(390, 6)
[195 195]
```

Random Under-Sampling in Action

```python
from imblearn.pipeline import make_pipeline as make_imb_pipeline

undersample_pipe = make_imb_pipeline(RandomUnderSampler(), LogisticRegressionCV())
scores = cross_validate(undersample_pipe,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630

0.927, 0.527

############################################################
############################################################

undersample_pipe_rf = make_imb_pipeline(RandomUnderSampler(),
                                        RandomForestClassifier())
scores = cross_validate(undersample_pipe_rf,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722


0.951, 0.629
```

Random Over-Sampling

```

from imblearn.over_sampling import RandomOverSampler


ros = RandomOverSampler()


X_train_oversample, y_train_oversample = ros.fit_sample(
    X_train, y_train)


print(X_train.shape)
print(X_train_oversample.shape)
print(np.bincount(y_train_oversample))



(8387, 6)
(16384, 6)
[8192 8192]
```

Random Over-Sampling in Action

```

oversample_pipe = make_imb_pipeline(RandomOverSampler(), LogisticRegression())

scores = cross_validate(oversample_pipe,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))

scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630

0.917, 0.585

########################################################################
########################################################################

oversample_pipe_rf = make_imb_pipeline(RandomOverSampler(),
                                       RandomForestClassifier())
scores = cross_validate(oversample_pipe_rf,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722

0.926, 0.715
```

Class Weights

- Instead of repeating samples, re-weight the loss function. Same effect as over-sampling (though not random), but not as expensive (dataset size the same)
- Class weights in Linear Models

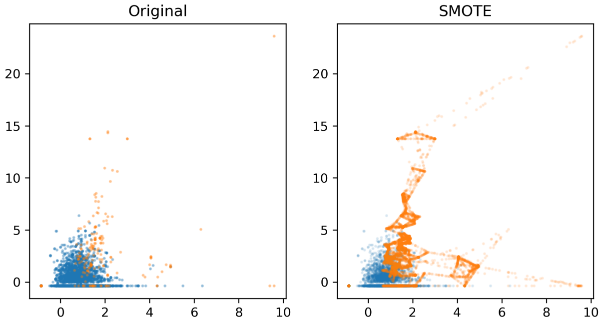
$$\min_{w\in^p, b\in\mathbb{R}} -C\sum_{i=1}^{n}\log(\exp(-y_i(w^T\mathbf{x}_i + b)) + 1) + ||w||_2^2$$

$$\min_{w\in^p, b\in\mathbb{R}} -C\sum_{i=1}^{n} c_{y_i}\log(\exp(-y_i(w^T\mathbf{x}_i + b)) + 1) + ||w||_2^2$$

```
scores = cross_validate(LogisticRegression(class_weight='balanced'),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))

scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630

0.918, 0.587

########################################################################
########################################################################

scores = cross_validate(RandomForestClassifier(n_estimators=100,
                                               class_weight='balanced'),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722

0.917, 0.701
```

Synthetic Minority Oversampling Technique (SMOTE)

- Adds synthetic interpolated data to smaller class
- For each sample in minority class:
  - Pick random neighbor from k neighbors
  - Pick point on line connecting the two uniformly (or within rectangle)
  - Repeat



SMOTE in Action

```python
smote_pipe = make_imb_pipeline(SMOTE(), LogisticRegression())


scores = cross_validate(smote_pipe, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))


pd.DataFrame(scores)[['test_roc_auc', 'test_average_precision']].mean()
# baseline was 0.920, 0.630

0.919, 0.585


#########################################################################
#########################################################################

smote_pipe_rf = make_imb_pipeline(SMOTE(),
                                  RandomForestClassifier())


scores = cross_validate(smote_pipe_rf, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))


pd.DataFrame(scores)[['test_roc_auc', 'test_average_precision']].mean()
# baseline was 0.939, 0.722

0.946, 0.688
```

Summary

- Accuracy rarely what you want
- Problems are rarely balanced
- Find the right criterion for the task
- Emphasis on recall or precision?
- Which classes are the important ones?
- Always check roc_auc and $average_precision$ look at curves
- Undersampling is very fast and can help!
- Undersampling + Ensembles is very powerful!
- Can add synthetic samples with SMOTE