

Module 3

May 30, 2020

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the [Jupyter Notebook FAQ](#) course resource.*

1 Applied Machine Learning: Module 3 (Evaluation)

1.1 Evaluation for Classification

1.1.1 Preamble

```
In [1]: %matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits

dataset = load_digits()
X, y = dataset.data, dataset.target

for class_name, class_count in zip(dataset.target_names, np.bincount(dataset.target)):
    print(class_name, class_count)
```

0 178
1 182
2 177
3 183
4 181
5 182
6 181
7 179
8 174
9 180

```
print('Original labels:\t', y[1:30])
print('New binary labels:\t', y_binary_imbalanced[1:30])
```



```
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 [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]]
```

1.1.4 Evaluation metrics for binary classification

```
In [13]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
        # Accuracy = TP + TN / (TP + TN + FP + FN)
        # Precision = TP / (TP + FP)
        # Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Rate
        # F1 = 2 * Precision * Recall / (Precision + Recall)
        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]: # Combined report with all above metrics
         from sklearn.metrics import classification_report

         print(classification_report(y_test, tree_predicted, target_names=['not 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 [15]: print('Random class-proportional (dummy)\n',
              classification_report(y_test, y_classprop_predicted, target_names=['
        print('SVM\n',
              classification_report(y_test, svm_predicted, target_names = ['not 1'
        print('Logistic regression\n',
              classification_report(y_test, lr_predicted, target_names = ['not 1'
        print('Decision tree\n',
              classification_report(y_test, tree_predicted, target_names = ['not 1
```

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

not 1      0.90      0.87      0.88      407
 1         0.04      0.05      0.04       43

avg / total      0.81      0.79      0.80      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
avg / total	0.94	0.95	0.94	450

1.1.5 Decision functions

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

    # show the decision_function scores for first 20 instances
    y_score_list
```

```
Out[16]: [(0, -23.172292973469546),
(0, -13.542576515500063),
(0, -21.717588760007867),
(0, -18.903065133316439),
(0, -19.733169947138638),
(0, -9.7463217496747667),
(1, 5.2327155658831135),
(0, -19.308012306288916),
(0, -25.099330209728528),
(0, -21.824312362996),
(0, -24.14378275072049),
(0, -19.578811099762508),
(0, -22.568371393280199),
(0, -10.822590225240777),
(0, -11.907918741521932),
(0, -10.977026853802803),
(1, 11.206811164226373),
(0, -27.64415761980748),
(0, -12.857692102545409),
(0, -25.848149140240199)]
```

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

```
# show the probability of positive class for first 20 instances
y_proba_list
```

```
Out[17]: [(0, 8.6377579220606777e-11),
          (0, 1.3138118599563783e-06),
          (0, 3.6997386039099529e-10),
          (0, 6.1730972504865465e-09),
          (0, 2.6914925394345074e-09),
          (0, 5.8506057771143608e-05),
          (1, 0.99468934644404694),
          (0, 4.1175302368500096e-09),
          (0, 1.2574750894253029e-11),
          (0, 3.3252290754668869e-10),
          (0, 3.2695529799373086e-11),
          (0, 3.1407283576084884e-09),
          (0, 1.5800864117150149e-10),
          (0, 1.9943442430612578e-05),
          (0, 6.7368003023860014e-06),
          (0, 1.7089540581641637e-05),
          (1, 0.9999864188091131),
          (0, 9.8694940340195476e-13),
          (0, 2.6059983600823893e-06),
          (0, 5.9469113009063784e-12)]
```

1.1.6 Precision-recall curves

```
In [18]: from sklearn.metrics import precision_recall_curve
```

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

```
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

1.1.7 ROC curves, Area-Under-Curve (AUC)

```
In [19]: from sklearn.metrics import roc_curve, auc
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalancede

y_score_lr = lr.fit(X_train, y_train).decision_function(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_score_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:.2f})'.
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```
In [20]: from matplotlib import cm
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y_binary_imbalancede

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
for g in [0.01, 0.1, 0.20, 1]:
    svm = SVC(gamma=g).fit(X_train, y_train)
    y_score_svm = svm.decision_function(X_test)
    fpr_svm, tpr_svm, _ = roc_curve(y_test, y_score_svm)
    roc_auc_svm = auc(fpr_svm, tpr_svm)
    accuracy_svm = svm.score(X_test, y_test)
    print("gamma = {:.2f}  accuracy = {:.2f}  AUC = {:.2f}".format(g, acc
                                                                    roc_auc

plt.plot(fpr_svm, tpr_svm, lw=3, alpha=0.7,
        label='SVM (gamma = {:.2f}, area = {:.2f})'.format(g, roc_a

plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate (Recall)', fontsize=16)
plt.plot([0, 1], [0, 1], color='k', lw=0.5, linestyle='--')
```



```
plt.legend(loc="lower right", fontsize=11)
plt.title('ROC curve: (1-of-10 digits classifier)', fontsize=16)
plt.axes().set_aspect('equal')
```

```
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
gamma = 0.01    accuracy = 0.91    AUC = 1.00
gamma = 0.10    accuracy = 0.90    AUC = 0.98
gamma = 0.20    accuracy = 0.90    AUC = 0.66
gamma = 1.00    accuracy = 0.90    AUC = 0.50
```

1.1.8 Evaluation measures for multi-class classification

Multi-class confusion matrix

```
In [21]: 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=42)

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,
                      index = [i for i in range(0,10)], columns = [i for i in range(0,10)])

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, index = [i for i in range(0,10)],
                      columns = [i for i in range(0,10)])

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, svm_pr

plt.ylabel('True label')
plt.xlabel('Predicted label');

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```

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

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)

1.1.9 Regression evaluation metrics

```
In [25]: %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 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)))

# Plot outputs
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.show()
```

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
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.1.10 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()
        # again, making this a binary problem with 'digit 1' as positive class
        # and 'not 1' as negative class
        X, y = dataset.data, dataset.target == 1
        clf = SVC(kernel='linear', C=1)

        # accuracy is the default scoring metric
        print('Cross-validation (accuracy)', cross_val_score(clf, X, y, cv=5))
        # use AUC as scoring metric
        print('Cross-validation (AUC)', cross_val_score(clf, X, y, cv=5, scoring =
        # use recall as scoring metric
        print('Cross-validation (recall)', cross_val_score(clf, X, y, cv=5, scorin

Cross-validation (accuracy) [ 0.91944444  0.98611111  0.97214485  0.97493036  0.969
Cross-validation (AUC) [ 0.9641871  0.9976571  0.99372205  0.99699002  0.98675611
Cross-validation (recall) [ 0.81081081  0.89189189  0.83333333  0.83333333  0.83333
```

Grid search example

```
In [27]: 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]}
```

```

# default metric to optimize over grid parameters: accuracy
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_)

# alternative metric to optimize over grid parameters: AUC
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

print(sorted(list(SCORERS.keys()))))

['accuracy', 'adjusted_rand_score', 'average_precision', 'f1', 'f1_macro', 'f1_micro']

```

1.1.11 Two-feature classification example using the digits dataset

Optimizing a classifier using different evaluation metrics

```

In [29]: from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from adspy_shared_utilities import plot_class_regions_for_classifier_subplot
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)

```

```

# Create a two-feature input vector matching the example plot above
# We jitter the points (add a small amount of random noise) in case there
# in feature space where many instances have the same features.
jitter_delta = 0.25
X_twovar_train = X_train[:, [20, 59]] + np.random.rand(X_train.shape[0], 2)
X_twovar_test = X_test[:, [20, 59]] + np.random.rand(X_test.shape[0], 2)

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}],
plt.figure(figsize=(9, 6))
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.subplots_adjust(wspace=0.3, hspace=0.3)
plot_class_regions_for_classifier_subplot(grid_clf_custom, X_twovar_test, y_test,
None, None, plt.subplot(2, 2, i+1))

plt.title(eval_metric+'-oriented SVC')
plt.tight_layout()
plt.show()

```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```

Grid best parameter (max. precision): {'class_weight': {1: 2}}
Grid best score (precision): 0.5553018160077229
Grid best parameter (max. recall): {'class_weight': {1: 50}}
Grid best score (recall): 0.935661321592438
Grid best parameter (max. f1): {'class_weight': {1: 4}}
Grid best score (f1): 0.5178121322213041
Grid best parameter (max. roc_auc): {'class_weight': {1: 10}}
Grid best score (roc_auc): 0.8931781954026993

```

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

```

In [30]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import precision_recall_curve
         from adspy_shared_utilities import plot_class_regions_for_classifier
         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)

# create a two-feature input vector matching the example plot above
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, 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]

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')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
print('At zero threshold, precision: {:.2f}, recall: {:.2f}'
      .format(closest_zero_p, closest_zero_r))

```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

At zero threshold, precision: 0.21, recall: 0.74

```
In [ ]:
```