evaluation_exercise

June 3, 2019

```
In [ ]:
```

0.0.1

```
In [ ]: from sklearn.model_selection import train_test_split
        from sklearn.datasets import load_digits

        digits = load_digits()
        y = digits.target == 9

        X_train, X_test, y_train, y_test = train_test_split(
            digits.data, y, random_state=0)
```

```
In [ ]: import numpy as np
        from sklearn.dummy import DummyClassifier
        dummy_majority = DummyClassifier(strategy='most_frequent').\
                                    fit(X_train, y_train)
        pred_most_frequent = dummy_majority.predict(X_test)
        print("  : {}".format(np.unique(pred_most_frequent)))
        print(" : {:.2f}".format(dummy_majority.score(X_test, y_test)))
```

```
In [ ]: from sklearn.tree import DecisionTreeClassifier
        tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
        pred_tree = tree.predict(X_test)
        print(" : {:.2f}".format(tree.score(X_test, y_test)))
```

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

        dummy = DummyClassifier().fit(X_train, y_train)
        pred_dummy = dummy.predict(X_test)
        print("dummy : {:.2f}".format(dummy.score(X_test, y_test)))

        logreg = LogisticRegression(C=0.1).fit(X_train, y_train)
        pred_logreg = logreg.predict(X_test)
        print("logreg : {:.2f}".format(logreg.score(X_test, y_test)))
```

(Confusion matrices)

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

        confusion = confusion_matrix(y_test, pred_logreg)
        print(" :\n{}".format(confusion))
```

```
In [ ]: mglearn.plots.plot_confusion_matrix_illustration()
```

```
In [ ]: mglearn.plots.plot_binary_confusion_matrix()
```

```
In [ ]: print("   :")
        print(confusion_matrix(y_test, pred_most_frequent))
        print("\n  :")
        print(confusion_matrix(y_test, pred_dummy))
        print("\n :")
        print(confusion_matrix(y_test, pred_tree))
        print("\n ")
        print(confusion_matrix(y_test, pred_logreg))
```

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

, , f-

$$= \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$= \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

$$\text{F} = 2 \cdot \frac{\cdot}{+} \tag{4}$$

```
In [ ]: from sklearn.metrics import f1_score
        print("    f1 score: {:.2f}".format(
            f1_score(y_test, pred_most_frequent)))
        print("   f1 score: {:.2f}".\
                        format(f1_score(y_test, pred_dummy)))
        print("  f1 score: {:.2f}".\
                        format(f1_score(y_test, pred_tree)))
        print("   f1 score: {:.2f}".format(
            f1_score(y_test, pred_logreg)))
```

```
In [ ]: from sklearn.metrics import classification_report
        print(classification_report(y_test, pred_most_frequent,
                                    target_names=["9 ", "9"]))
```

```
In [ ]: print(classification_report(y_test, pred_dummy,
                                    target_names=["9 ", "9"]))
```

```
In [ ]: print(classification_report(y_test, pred_tree,
                                    target_names=["9 ", "9"]))
```

```
In [ ]: print(classification_report(y_test, pred_logreg,
                                    target_names=["9 ", "9"]))
```

```
In [ ]: from sklearn.datasets import make_blobs
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import train_test_split

        from sklearn.model_selection import GridSearchCV
        from sklearn.svm import SVC

        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
```

```
In [ ]: from mglearn.datasets import make_blobs
        X, y = make_blobs(n_samples=(400, 50), centers=2, \
                          cluster_std=[7.0, 2], random_state=22)
        print(X, y)
```

```
In [ ]: X_train, X_test, y_train, y_test = \
                             train_test_split(X, y, random_state=0)
        svc = SVC(gamma=.05, probability=True)
        svc.fit(X_train, y_train)
```

```
In [ ]: print(classification_report(y_test, svc.predict(X_test)))
```

```
In [ ]: print(classification_report(svc.predict(X_test), y_test))
```

```
In [ ]: svc.decision_function(X_test)
```

```
In [ ]: y_pred_lower_threshold = svc.decision_function(X_test) > -.8
```

```
In [ ]: y_pred_lower_threshold
```

```
In [ ]: print(classification_report(y_test, y_pred_lower_threshold))
```

```
In [ ]: y_pred_lower_threshold = svc.decision_function(X_test) > 1
```

```
In [ ]: y_pred_lower_threshold
```

```
In [ ]: print(classification_report(y_test, y_pred_lower_threshold))
```

```
In [ ]: svc.predict_proba(X_test)
```

```
In [ ]: y_pred_threshold = svc.predict_proba(X_test)>0.8
```

```
In [ ]: import numpy as np
        y_pred_threshold = y_pred_threshold.argmax(axis=1)
```

```
In [ ]: print(classification_report(y_test, y_pred_threshold))
```

- ROC

```
In [ ]: from sklearn.metrics import precision_recall_curve
        precision, recall, thresholds = precision_recall_curve(
            y_test, svc.decision_function(X_test))
        print(thresholds)
```

```
In [ ]: import matplotlib.pyplot as plt
        #
        X, y = make_blobs(n_samples=(4000, 500), centers=2, \
                          cluster_std=[7.0, 2], random_state=22)
        X_train, X_test, y_train, y_test = \
                          train_test_split(X, y, random_state=0)

        svc = SVC(gamma=.05).fit(X_train, y_train)

        precision, recall, thresholds = precision_recall_curve(
            y_test, svc.decision_function(X_test))
        # 0    - predict
        close_zero = np.argmin(np.abs(thresholds))
        plt.plot(precision[close_zero], recall[close_zero], 'o', \
            markersize=10, label=" 0", fillstyle="none", c='k', mew=2)

        plt.plot(precision, recall, label="- ")
        plt.xlabel("")
        plt.ylabel("")
        plt.legend(loc="best")
```

```
In [ ]: from sklearn.ensemble import RandomForestClassifier

        rf = RandomForestClassifier(n_estimators=100, \
                                    random_state=0, max_features=2)
        rf.fit(X_train, y_train)

        precision_rf, recall_rf, thresholds_rf = precision_recall_curve(
            y_test, rf.predict_proba(X_test)[:, 1])

        svc = SVC(gamma=.05, probability=True).fit(X_train, y_train)
        precision, recall, thresholds = precision_recall_curve(
            y_test, svc.predict_proba(X_test)[:, 1])
        plt.plot(precision, recall, label="svc")

        close_default_svc = np.argmin(np.abs(thresholds - 0.5))
        plt.plot(precision[close_default_svc], \
                 recall[close_default_svc], 'o', markersize=10,
                 label="svc:  0", fillstyle="none", c='k', mew=2)

        plt.plot(precision_rf, recall_rf, label="rf")
```

```
          close_default_rf = np.argmin(np.abs(thresholds_rf - 0.5))
          plt.plot(precision_rf[close_default_rf], recall_rf[close_default_rf], \
           '^', c='k', markersize=10, label="rf:  0.5", fillstyle="none", mew=2)
          plt.xlabel("precision")
          plt.ylabel("recall")
          plt.legend(loc="best")
```

```
In [ ]: print("  f1_score: {:.3f}".format(
              f1_score(y_test, rf.predict(X_test))))
        print("svc f1_score: {:.3f}".\
              format(f1_score(y_test, svc.predict(X_test))))
```

```
In [ ]: from sklearn.metrics import average_precision_score
        ap_rf = average_precision_score(y_test, \
                                    rf.predict_proba(X_test)[:, 1])
        ap_svc = average_precision_score(y_test, \
                                    svc.decision_function(X_test))
        print("    : {:.3f}".format(ap_rf))
        print("svc  : {:.3f}".format(ap_svc))
```

ROC AUC

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{5}$$

```
In [ ]: from sklearn.metrics import roc_curve
        fpr, tpr, thresholds = roc_curve(y_test, \
                                    svc.decision_function(X_test))

        plt.plot(fpr, tpr, label="ROC ")
        plt.xlabel("FPR")
        plt.ylabel("TPR ()")
        # 0
        close_zero = np.argmin(np.abs(thresholds))
        plt.plot(fpr[close_zero], tpr[close_zero], 'o', \
            markersize=10, label=" 0", \
                fillstyle="none", c='k', mew=2)
        plt.legend(loc=4)
```

```
In [ ]: from sklearn.metrics import roc_curve
        fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, \
                                        rf.predict_proba(X_test)[:, 1])

        plt.plot(fpr, tpr, label="SVC ROC ")
        plt.plot(fpr_rf, tpr_rf, label="RF ROC ")

        plt.xlabel("FPR")
        plt.ylabel("TPR ()")
        plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
                 label="SVC  0", fillstyle="none", c='k', mew=2)
        close_default_rf = np.argmin(np.abs(thresholds_rf - 0.5))
        plt.plot(fpr_rf[close_default_rf], tpr[close_default_rf], '^', markersize=10,
                 label="RF  0.5", fillstyle="none", c='k', mew=2)

        plt.legend(loc=4)
```

```
In [ ]: from sklearn.metrics import roc_auc_score
        rf_auc = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
        svc_auc = roc_auc_score(y_test, svc.decision_function(X_test))
        print("  AUC: {:.3f}".format(rf_auc))
        print("SVC AUC: {:.3f}".format(svc_auc))
        # ROC  (AUC) - roc_auc_socre
```

```
In [ ]: from sklearn.metrics import accuracy_score
        X_train, X_test, y_train, y_test = train_test_split(
            digits.data, digits.target, random_state=0)
        lr = LogisticRegression().fit(X_train, y_train)
        pred = lr.predict(X_test)
        print(": {:.3f}".format(accuracy_score(y_test, pred)))
        print(" :\n{}".format(confusion_matrix(y_test, pred)))
```

```
In [ ]: scores_image = mglearn.tools.heatmap(
            confusion_matrix(y_test, pred), xlabel='predict',
            ylabel='real', xticklabels=digits.target_names,
            yticklabels=digits.target_names, cmap=plt.cm.gray_r, fmt="%d")
        plt.title("confusion matrix")
        plt.gca().invert_yaxis()
```

```
In [ ]: print(classification_report(y_test, pred))
```

```
In [ ]: print("accuracy : {:.3f}".format(accuracy_score(y_test, pred)))
        print("micro  f1 : {:.3f}".\
              format(f1_score(y_test, pred, average="micro")))
        print("macro  f1 : {:.3f}".\
              format(f1_score(y_test, pred, average="macro")))
        print("macro  f1 : {:.3f}".\
              format(f1_score(y_test, pred, average="weighted")))
```

```
In [ ]: #
        print("  : {}".format(
              cross_val_score(SVC(), digits.data, digits.target == 9)))
        # scoring="accuracy"  .
        explicit_accuracy =  cross_val_score(SVC(), digits.data, \
                            digits.target == 9, scoring="accuracy")
        print(" : {}".format(explicit_accuracy))
        roc_auc =  cross_val_score(SVC(), digits.data, \
                        digits.target == 9, scoring="roc_auc")

In [ ]: X_train, X_test, y_train, y_test = train_test_split(
            digits.data, digits.target == 9, random_state=0)

        #
        param_grid = {'gamma': [0.0001, 0.01, 0.1, 1, 10]}
        #     .   accuracy
        grid = GridSearchCV(SVC(), param_grid=param_grid)
        grid.fit(X_train, y_train)
        print("    ")
        print(" :", grid.best_params_)
        print("    ()): {:.3f}".\
              format(grid.best_score_))
        print("  accuracy: {:.3f}".\
              format(grid.score(X_test, y_test)))
        print("  AUC: {:.3f}".format(
                roc_auc_score(y_test, \
                        grid.decision_function(X_test))))
        print("  accuracy: {:.3f}".\
              format(grid.score(X_test, y_test)))

In [ ]: # AUC
        grid = GridSearchCV(SVC(), param_grid=param_grid, \
                            scoring="roc_auc")
        grid.fit(X_train, y_train)
        print("AUC    ")
        print(" :", grid.best_params_)
        print("    (AUC): {:.3f}".\
              format(grid.best_score_))
        print("  AUC: {:.3f}".\
              format(grid.score(X_test, y_test)))
```

0.0.2