# Chapters 8 and 9: Python Workbook

Ahmad Tahmid

## Contents

# 1 Chapter 8 Solutions

## 1.1 Exercise 8.4.1

### Goal

Fit a regression tree (or classification tree, depending on dataset) and examine the results (splits, tree depth, etc.).

### Solution

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor, plot_tree

# Example: Carseats data might be used for a regression tree, or
   Boston dataset.
# Let's use Boston for demonstration, predicting 'medv' from the
   other features.

Boston = pd.read_csv("Boston.csv")  # or from the ISLP package
X = Boston.drop(columns=["medv"])
y = Boston["medv"]

# Fit a regression tree
tree_reg = DecisionTreeRegressor(max_depth=5, random_state=1)
tree_reg.fit(X, y)

# Visualize the tree structure
plt.figure(figsize=(12, 8))
plot_tree(tree_reg, feature_names=X.columns, filled=True, max_depth
   =3)
plt.title("Regression Tree (Depth <= 3 Displayed)")
plt.show()

# Evaluate MSE
preds = tree_reg.predict(X)
mse = np.mean((y - preds)**2)
print("MSE (training set):", mse)
```

## 1.2 Exercise 8.4.2

### Goal

Use bagging or random forests on the same dataset, compare test error with a train-test split or cross-validation.

### Solution

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split,
    mean_squared_error

Boston = pd.read_csv("Boston.csv")
X = Boston.drop(columns=["medv"])
y = Boston["medv"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3, random_state=42)

# Bagging is essentially a RandomForest with max_features = number
    of features
bag_reg = RandomForestRegressor(n_estimators=500, max_features=X.
    shape[1], random_state=42)
bag_reg.fit(X_train, y_train)
y_pred_bag = bag_reg.predict(X_test)
mse_bag = mean_squared_error(y_test, y_pred_bag)
print("Bagging Test MSE:", mse_bag)

# Random Forest with default max_features = sqrt(p) for regression
rf = RandomForestRegressor(n_estimators=500, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
print("Random Forest Test MSE:", mse_rf)

# Feature importances
importances = rf.feature_importances_
feature_importance = sorted(zip(X.columns, importances), key=lambda
    x: x[1], reverse=True)
print("Feature importances (RF):")
for feat, imp in feature_importance:
    print(f"{feat}: {imp:.4f}")
```

## 1.3 Exercise 8.4.3

### Goal

Implement boosting (e.g., `GradientBoostingRegressor`) or a Bayesian Additive Regression Trees (BART) approach. Compare performance with bagging/RF.

### Solution

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
Boston = pd.read_csv("Boston.csv")
X = Boston.drop(columns=["medv"])
y = Boston["medv"]

# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
   =0.3, random_state=3)

# Gradient Boosting
gbm = GradientBoostingRegressor(n_estimators=500, learning_rate
   =0.01, max_depth=3, random_state=3)
gbm.fit(X_train, y_train)
y_pred_gbm = gbm.predict(X_test)

from sklearn.metrics import mean_squared_error
mse_gbm = mean_squared_error(y_test, y_pred_gbm)
print("Gradient Boosting Test MSE:", mse_gbm)

# BART isn't in sklearn by default, but we can do an example if a
   library is installed:
# e.g., from bartpy.sklearnmodel import SklearnModel
# ...
```

# 2 Chapter 9 Solutions

## 2.1 Exercise 9.7.1

### Goal

Fit a support vector classifier (linear SVM) to a dataset (e.g., `OJ`, `Smarket`, or a synthetic set), compare training and test error.

### Solution

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# Example: Smarket classification with a linear SVM
Smarket = pd.read_csv("Smarket.csv", index_col=0)
Smarket['Direction_binary'] = (Smarket['Direction'] == 'Up').astype(
   int)
```

```
train = Smarket['Year'] < 2005
test  = ~train

X_train = Smarket.loc[train, ['Lag1','Lag2','Lag3','Lag4','Lag5','
   Volume']]
y_train = Smarket.loc[train, 'Direction_binary']
X_test  = Smarket.loc[test, ['Lag1','Lag2','Lag3','Lag4','Lag5','
   Volume']]
y_test  = Smarket.loc[test, 'Direction_binary']

# Linear SVM (C is cost parameter)
svc_linear = SVC(kernel='linear', C=1.0)
svc_linear.fit(X_train, y_train)
train_preds = svc_linear.predict(X_train)
test_preds = svc_linear.predict(X_test)

print("Training Confusion Matrix:\n", confusion_matrix(y_train,
   train_preds))
print("Test Confusion Matrix:\n", confusion_matrix(y_test,
   test_preds))

train_acc = np.mean(train_preds == y_train)
test_acc = np.mean(test_preds == y_test)
print("Training Accuracy:", train_acc)
print("Test Accuracy:", test_acc)
```

## 2.2 Exercise 9.7.2

### Goal

Use a non-linear SVM (e.g., radial kernel). Tune parameters (cost $C$, gamma $\gamma$) via cross-validation.

### Solution

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

X = Smarket.loc[train, ['Lag1','Lag2','Lag3','Lag4','Lag5','Volume'
   ]]
y = Smarket.loc[train, 'Direction_binary']

param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'gamma': [0.01, 0.1, 1, 10]
}
```

```
svm_rbf = SVC(kernel='rbf')

grid_cv = GridSearchCV(svm_rbf, param_grid, scoring='accuracy', cv
    =5)
grid_cv.fit(X, y)

print("Best parameters:", grid_cv.best_params_)
print("Best CV accuracy:", grid_cv.best_score_)

# Evaluate on test set
best_svm = grid_cv.best_estimator_
X_test = Smarket.loc[test, ['Lag1','Lag2','Lag3','Lag4','Lag5','
    Volume']]
y_test = Smarket.loc[test, 'Direction_binary']

test_preds = best_svm.predict(X_test)
test_accuracy = np.mean(test_preds == y_test)
print("Test Accuracy with best rbf SVM:", test_accuracy)
```

## 2.3 Exercise 9.7.3

### Goal

Examine ROC curves, compare sensitivity, specificity. Possibly do multi-class SVM with `one-vs-one` or `one-vs-all`.

### Solution

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

y_scores = best_svm.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, color='blue', label=f"ROC curve (area = {roc_auc
    :.2f})")
plt.plot([0,1], [0,1], color='red', linestyle='--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("SVM RBF ROC Curve")
plt.legend(loc="lower right")
plt.show()
```

# 3 My Learnings

- saw how decision trees offer a simple yet interpretable model for regression or classification
- used ensemble methods (bagging, random forests, boosting) to improve predictive performance, at the cost of interpretability
- explored how to tune tree-based methods using parameters like depth, number of estimators, learning rate
- learned that SVMs handle both linear and non-linear boundaries through different kernel choices
- recognized the need for tuning parameters like cost (C), gamma, etc., via cross-validation to achieve best performance
- observed how ROC curves provide a useful way to compare classifiers in terms of trade-off between TPR and FPR