Module 2

September 18, 2020

---

*You are currently looking at **version 1.1** 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 2 (Supervised Learning, Part I)

1.1 Preamble and Review

```python
In [1]: %matplotlib notebook
        import numpy as np
        import pandas as pd
        import seaborn as sn
        import matplotlib.pyplot as plt

        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier

        np.set_printoptions(precision=2)


        fruits = pd.read_table('readonly/fruit_data_with_colors.txt')

        feature_names_fruits = ['height', 'width', 'mass', 'color_score']
        X_fruits = fruits[feature_names_fruits]
        y_fruits = fruits['fruit_label']
        target_names_fruits = ['apple', 'mandarin', 'orange', 'lemon']

        X_fruits_2d = fruits[['height', 'width']]
        y_fruits_2d = fruits['fruit_label']

        X_train, X_test, y_train, y_test = train_test_split(X_fruits, y_fruits, ran

        from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()
        X_train_scaled = scaler.fit_transform(X_train)
```

```
          # we must apply the scaling to the test set that we computed for the train:
          X_test_scaled = scaler.transform(X_test)

          knn = KNeighborsClassifier(n_neighbors = 5)
          knn.fit(X_train_scaled, y_train)
          print('Accuracy of K-NN classifier on training set: {:.2f}'
                .format(knn.score(X_train_scaled, y_train)))
          print('Accuracy of K-NN classifier on test set: {:.2f}'
                .format(knn.score(X_test_scaled, y_test)))

          example_fruit = [[5.5, 2.2, 10, 0.70]]
          example_fruit_scaled = scaler.transform(example_fruit)
          print('Predicted fruit type for ', example_fruit, ' is ',
                target_names_fruits[knn.predict(example_fruit_scaled)[0]-1])

Accuracy of K-NN classifier on training set: 0.95
Accuracy of K-NN classifier on test set: 1.00
Predicted fruit type for  [[5.5, 2.2, 10, 0.7]]  is  mandarin
```

1.2 Datasets

```
In [2]: from sklearn.datasets import make_classification, make_blobs
        from matplotlib.colors import ListedColormap
        from sklearn.datasets import load_breast_cancer
        from adspy_shared_utilities import load_crime_dataset

        cmap_bold = ListedColormap(['#FFFF00', '#00FF00', '#0000FF','#000000'])


        # synthetic dataset for simple regression
        from sklearn.datasets import make_regression
        plt.figure()
        plt.title('Sample regression problem with one input variable')
        X_R1, y_R1 = make_regression(n_samples = 100, n_features=1,
                                     n_informative=1, bias = 150.0,
                                     noise = 30, random_state=0)
        plt.scatter(X_R1, y_R1, marker= 'o', s=50)
        plt.show()


        # synthetic dataset for more complex regression
        from sklearn.datasets import make_friedman1
        plt.figure()
        plt.title('Complex regression problem with one input variable')
        X_F1, y_F1 = make_friedman1(n_samples = 100,
                                    n_features = 7, random_state=0)
```

```python
        plt.scatter(X_F1[:, 2], y_F1, marker= 'o', s=50)
        plt.show()

        # synthetic dataset for classification (binary)
        plt.figure()
        plt.title('Sample binary classification problem with two informative featur
        X_C2, y_C2 = make_classification(n_samples = 100, n_features=2,
                                        n_redundant=0, n_informative=2,
                                        n_clusters_per_class=1, flip_y = 0.1,
                                        class_sep = 0.5, random_state=0)
        plt.scatter(X_C2[:, 0], X_C2[:, 1], c=y_C2,
                    marker= 'o', s=50, cmap=cmap_bold)
        plt.show()


        # more difficult synthetic dataset for classification (binary)
        # with classes that are not linearly separable
        X_D2, y_D2 = make_blobs(n_samples = 100, n_features = 2, centers = 8,
                                cluster_std = 1.3, random_state = 4)
        y_D2 = y_D2 % 2
        plt.figure()
        plt.title('Sample binary classification problem with non-linearly separable
        plt.scatter(X_D2[:,0], X_D2[:,1], c=y_D2,
                    marker= 'o', s=50, cmap=cmap_bold)
        plt.show()


        # Breast cancer dataset for classification
        cancer = load_breast_cancer()
        (X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)


        # Communities and Crime dataset
        (X_crime, y_crime) = load_crime_dataset()
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

<IPython.core.display.Javascript object>

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


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>



        ---------------------------------------------------------------------

        FileNotFoundError                         Traceback (most recent call last)

        <ipython-input-2-e3e70412cc56> in <module>()
         58
         59 # Communities and Crime dataset
    ---> 60 (X_crime, y_crime) = load_crime_dataset()


        /home/jovyan/work/adspy_shared_utilities.py in load_crime_dataset()
         17      # https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime+Unr
         18
    ---> 19      crime = pd.read_table('CommViolPredUnnormalizedData.txt', sep=',',
         20      # remove features with poor coverage or lower relevance, and keep \
         21      columns_to_keep = [5, 6] + list(range(11,26)) + list(range(32, 103)


        /opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in parser_f(fil
        644                      skip_blank_lines=skip_blank_lines)
        645
    --> 646          return _read(filepath_or_buffer, kwds)
        647
        648      parser_f.__name__ = name


        /opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in _read(filepa
        387
        388      # Create the parser.
    --> 389      parser = TextFileReader(filepath_or_buffer, **kwds)
        390
        391      if (nrows is not None) and (chunksize is not None):


        /opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in __init__(sel
        728              self.options['has_index_names'] = kwds['has_index_names']
        729
    --> 730          self._make_engine(self.engine)
```

4

```
      731
      732     def close(self):


    /opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in _make_engine
      921     def _make_engine(self, engine='c'):
      922         if engine == 'c':
--> 923             self._engine = CParserWrapper(self.f, **self.options)
      924         else:
      925             if engine == 'python':


    /opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in __init__(sel
     1388         kwds['allow_leading_cols'] = self.index_col is not False
     1389
->  1390         self._reader = _parser.TextReader(src, **kwds)
     1391
     1392         # XXX


    pandas/parser.pyx in pandas.parser.TextReader.__cinit__ (pandas/parser.c:41


    pandas/parser.pyx in pandas.parser.TextReader._setup_parser_source (pandas/


    FileNotFoundError: File b'CommViolPredUnnormalizedData.txt' does not exist
```

1.3   K-Nearest Neighbors

1.3.1   Classification

```
In [ ]: from adspy_shared_utilities import plot_two_class_knn

        X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                            random_state=0)

        plot_two_class_knn(X_train, y_train, 1, 'uniform', X_test, y_test)
        plot_two_class_knn(X_train, y_train, 3, 'uniform', X_test, y_test)
        plot_two_class_knn(X_train, y_train, 11, 'uniform', X_test, y_test)
```

1.3.2   Regression

```
In [ ]: from sklearn.neighbors import KNeighborsRegressor

        X_train, X_test, y_train, y_test = train_test_split(X_R1, y_R1, random_stat
```

```
        knnreg = KNeighborsRegressor(n_neighbors = 5).fit(X_train, y_train)

        print(knnreg.predict(X_test))
        print('R-squared test score: {:.3f}'
              .format(knnreg.score(X_test, y_test)))

In [ ]: fig, subaxes = plt.subplots(1, 2, figsize=(8,4))
        X_predict_input = np.linspace(-3, 3, 50).reshape(-1,1)
        X_train, X_test, y_train, y_test = train_test_split(X_R1[0::5], y_R1[0::5],

        for thisaxis, K in zip(subaxes, [1, 3]):
            knnreg = KNeighborsRegressor(n_neighbors = K).fit(X_train, y_train)
            y_predict_output = knnreg.predict(X_predict_input)
            thisaxis.set_xlim([-2.5, 0.75])
            thisaxis.plot(X_predict_input, y_predict_output, '^', markersize = 10,
                          label='Predicted', alpha=0.8)
            thisaxis.plot(X_train, y_train, 'o', label='True Value', alpha=0.8)
            thisaxis.set_xlabel('Input feature')
            thisaxis.set_ylabel('Target value')
            thisaxis.set_title('KNN regression (K={})'.format(K))
            thisaxis.legend()
        plt.tight_layout()
```

1.3.3 Regression model complexity as a function of K

```
In [ ]: # plot k-NN regression on sample dataset for different values of K
        fig, subaxes = plt.subplots(5, 1, figsize=(5,20))
        X_predict_input = np.linspace(-3, 3, 500).reshape(-1,1)
        X_train, X_test, y_train, y_test = train_test_split(X_R1, y_R1,
                                                            random_state = 0)

        for thisaxis, K in zip(subaxes, [1, 3, 7, 15, 55]):
            knnreg = KNeighborsRegressor(n_neighbors = K).fit(X_train, y_train)
            y_predict_output = knnreg.predict(X_predict_input)
            train_score = knnreg.score(X_train, y_train)
            test_score = knnreg.score(X_test, y_test)
            thisaxis.plot(X_predict_input, y_predict_output)
            thisaxis.plot(X_train, y_train, 'o', alpha=0.9, label='Train')
            thisaxis.plot(X_test, y_test, '^', alpha=0.9, label='Test')
            thisaxis.set_xlabel('Input feature')
            thisaxis.set_ylabel('Target value')
            thisaxis.set_title('KNN Regression (K={})\n\
        Train $R^2 = {:.3f}$,  Test $R^2 = {:.3f}$'
                               .format(K, train_score, test_score))
            thisaxis.legend()
            plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```

1.4 Linear models for regression

1.4.1 Linear regression

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

        X_train, X_test, y_train, y_test = train_test_split(X_R1, y_R1,
                                                    random_state = 0)
        linreg = LinearRegression().fit(X_train, y_train)

        print('linear model coeff (w): {}'
            .format(linreg.coef_))
        print('linear model intercept (b): {:.3f}'
            .format(linreg.intercept_))
        print('R-squared score (training): {:.3f}'
            .format(linreg.score(X_train, y_train)))
        print('R-squared score (test): {:.3f}'
            .format(linreg.score(X_test, y_test)))
```

1.4.2 Linear regression: example plot

```
In [ ]: plt.figure(figsize=(5,4))
        plt.scatter(X_R1, y_R1, marker= 'o', s=50, alpha=0.8)
        plt.plot(X_R1, linreg.coef_ * X_R1 + linreg.intercept_, 'r-')
        plt.title('Least-squares linear regression')
        plt.xlabel('Feature value (x)')
        plt.ylabel('Target value (y)')
        plt.show()

In [ ]: X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                    random_state = 0)
        linreg = LinearRegression().fit(X_train, y_train)

        print('Crime dataset')
        print('linear model intercept: {}'
            .format(linreg.intercept_))
        print('linear model coeff:\n{}'
            .format(linreg.coef_))
        print('R-squared score (training): {:.3f}'
            .format(linreg.score(X_train, y_train)))
        print('R-squared score (test): {:.3f}'
            .format(linreg.score(X_test, y_test)))
```

1.4.3 Ridge regression

```
In [ ]: from sklearn.linear_model import Ridge
        X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                    random_state = 0)
```

```python
linridge = Ridge(alpha=20.0).fit(X_train, y_train)

print('Crime dataset')
print('ridge regression linear model intercept: {}'
      .format(linridge.intercept_))
print('ridge regression linear model coeff:\n{}'
      .format(linridge.coef_))
print('R-squared score (training): {:.3f}'
      .format(linridge.score(X_train, y_train)))
print('R-squared score (test): {:.3f}'
      .format(linridge.score(X_test, y_test)))
print('Number of non-zero features: {}'
      .format(np.sum(linridge.coef_ != 0)))
```

Ridge regression with feature normalization

```python
In [ ]: from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()

        from sklearn.linear_model import Ridge
        X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                    random_state = 0)

        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        linridge = Ridge(alpha=20.0).fit(X_train_scaled, y_train)

        print('Crime dataset')
        print('ridge regression linear model intercept: {}'
              .format(linridge.intercept_))
        print('ridge regression linear model coeff:\n{}'
              .format(linridge.coef_))
        print('R-squared score (training): {:.3f}'
              .format(linridge.score(X_train_scaled, y_train)))
        print('R-squared score (test): {:.3f}'
              .format(linridge.score(X_test_scaled, y_test)))
        print('Number of non-zero features: {}'
              .format(np.sum(linridge.coef_ != 0)))
```

Ridge regression with regularization parameter: alpha

```python
In [ ]: print('Ridge regression: effect of alpha regularization parameter\n')
        for this_alpha in [0, 1, 10, 20, 50, 100, 1000]:
            linridge = Ridge(alpha = this_alpha).fit(X_train_scaled, y_train)
            r2_train = linridge.score(X_train_scaled, y_train)
            r2_test = linridge.score(X_test_scaled, y_test)
            num_coeff_bigger = np.sum(abs(linridge.coef_) > 1.0)
```

```
        print('Alpha = {:.2f}\nnum abs(coeff) > 1.0: {}, \
r-squared training: {:.2f}, r-squared test: {:.2f}\n'
            .format(this_alpha, num_coeff_bigger, r2_train, r2_test))
```

1.4.4 Lasso regression

```
In [ ]: from sklearn.linear_model import Lasso
        from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()

        X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                            random_state = 0)

        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        linlasso = Lasso(alpha=2.0, max_iter = 10000).fit(X_train_scaled, y_train)

        print('Crime dataset')
        print('lasso regression linear model intercept: {}'
            .format(linlasso.intercept_))
        print('lasso regression linear model coeff:\n{}'
            .format(linlasso.coef_))
        print('Non-zero features: {}'
            .format(np.sum(linlasso.coef_ != 0)))
        print('R-squared score (training): {:.3f}'
            .format(linlasso.score(X_train_scaled, y_train)))
        print('R-squared score (test): {:.3f}\n'
            .format(linlasso.score(X_test_scaled, y_test)))
        print('Features with non-zero weight (sorted by absolute magnitude):')

        for e in sorted (list(zip(list(X_crime), linlasso.coef_)),
                        key = lambda e: -abs(e[1])):
            if e[1] != 0:
                print('\t{}, {:.3f}'.format(e[0], e[1]))
```

Lasso regression with regularization parameter: alpha

```
In [ ]: print('Lasso regression: effect of alpha regularization\n\
        parameter on number of features kept in final model\n')

        for alpha in [0.5, 1, 2, 3, 5, 10, 20, 50]:
            linlasso = Lasso(alpha, max_iter = 10000).fit(X_train_scaled, y_train)
            r2_train = linlasso.score(X_train_scaled, y_train)
            r2_test = linlasso.score(X_test_scaled, y_test)

            print('Alpha = {:.2f}\nFeatures kept: {}, r-squared training: {:.2f}, \
```

```
              r-squared test: {:.2f}\n'
                   .format(alpha, np.sum(linlasso.coef_ != 0), r2_train, r2_test))
```

1.4.5 Polynomial regression

```
In [ ]: from sklearn.linear_model import LinearRegression
        from sklearn.linear_model import Ridge
        from sklearn.preprocessing import PolynomialFeatures


        X_train, X_test, y_train, y_test = train_test_split(X_F1, y_F1,
                                                            random_state = 0)
        linreg = LinearRegression().fit(X_train, y_train)

        print('linear model coeff (w): {}'
              .format(linreg.coef_))
        print('linear model intercept (b): {:.3f}'
              .format(linreg.intercept_))
        print('R-squared score (training): {:.3f}'
              .format(linreg.score(X_train, y_train)))
        print('R-squared score (test): {:.3f}'
              .format(linreg.score(X_test, y_test)))

        print('\nNow we transform the original input data to add\n\
        polynomial features up to degree 2 (quadratic)\n')
        poly = PolynomialFeatures(degree=2)
        X_F1_poly = poly.fit_transform(X_F1)

        X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, y_F1,
                                                            random_state = 0)
        linreg = LinearRegression().fit(X_train, y_train)

        print('(poly deg 2) linear model coeff (w):\n{}'
              .format(linreg.coef_))
        print('(poly deg 2) linear model intercept (b): {:.3f}'
              .format(linreg.intercept_))
        print('(poly deg 2) R-squared score (training): {:.3f}'
              .format(linreg.score(X_train, y_train)))
        print('(poly deg 2) R-squared score (test): {:.3f}\n'
              .format(linreg.score(X_test, y_test)))

        print('\nAddition of many polynomial features often leads to\n\
        overfitting, so we often use polynomial features in combination\n\
        with regression that has a regularization penalty, like ridge\n\
        regression.\n')

        X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, y_F1,
                                                            random_state = 0)
```

```
linreg = Ridge().fit(X_train, y_train)

print('(poly deg 2 + ridge) linear model coeff (w):\n{}'
      .format(linreg.coef_))
print('(poly deg 2 + ridge) linear model intercept (b): {:.3f}'
      .format(linreg.intercept_))
print('(poly deg 2 + ridge) R-squared score (training): {:.3f}'
      .format(linreg.score(X_train, y_train)))
print('(poly deg 2 + ridge) R-squared score (test): {:.3f}'
      .format(linreg.score(X_test, y_test)))
```

1.5 Linear models for classification

1.5.1 Logistic regression

Logistic regression for binary classification on fruits dataset using height, width features (positive class: apple, negative class: others)

```
In [ ]: from sklearn.linear_model import LogisticRegression
        from adspy_shared_utilities import (
        plot_class_regions_for_classifier_subplot)

        fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))
        y_fruits_apple = y_fruits_2d == 1   # make into a binary problem: apples vs
        X_train, X_test, y_train, y_test = (
        train_test_split(X_fruits_2d.as_matrix(),
                         y_fruits_apple.as_matrix(),
                         random_state = 0))

        clf = LogisticRegression(C=100).fit(X_train, y_train)
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train, None,
                                                  None, 'Logistic regression \
        for binary classification\nFruit dataset: Apple vs others',
                                                  subaxes)

        h = 6
        w = 8
        print('A fruit with height {} and width {} is predicted to be: {}'
              .format(h,w, ['not an apple', 'an apple'][clf.predict([[h,w]])[0]]))

        h = 10
        w = 7
        print('A fruit with height {} and width {} is predicted to be: {}'
              .format(h,w, ['not an apple', 'an apple'][clf.predict([[h,w]])[0]]))
        subaxes.set_xlabel('height')
        subaxes.set_ylabel('width')

        print('Accuracy of Logistic regression classifier on training set: {:.2f}'
              .format(clf.score(X_train, y_train)))
```

11

```
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
       .format(clf.score(X_test, y_test)))
```

Logistic regression on simple synthetic dataset

```
In [ ]: from sklearn.linear_model import LogisticRegression
        from adspy_shared_utilities import (
        plot_class_regions_for_classifier_subplot)


        X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                    random_state = 0)

        fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))
        clf = LogisticRegression().fit(X_train, y_train)
        title = 'Logistic regression, simple synthetic dataset C = {:.3f}'.format(1
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                            None, None, title, subaxes)

        print('Accuracy of Logistic regression classifier on training set: {:.2f}'
               .format(clf.score(X_train, y_train)))
        print('Accuracy of Logistic regression classifier on test set: {:.2f}'
               .format(clf.score(X_test, y_test)))
```

Logistic regression regularization: C parameter

```
In [ ]: X_train, X_test, y_train, y_test = (
        train_test_split(X_fruits_2d.as_matrix(),
                        y_fruits_apple.as_matrix(),
                        random_state=0))

        fig, subaxes = plt.subplots(3, 1, figsize=(4, 10))

        for this_C, subplot in zip([0.1, 1, 100], subaxes):
            clf = LogisticRegression(C=this_C).fit(X_train, y_train)
            title ='Logistic regression (apple vs rest), C = {:.3f}'.format(this_C)

            plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                X_test, y_test, title,
                                                subplot)
        plt.tight_layout()
```

Application to real dataset

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

        X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ran
```

```
clf = LogisticRegression().fit(X_train, y_train)
print('Breast cancer dataset')
print('Accuracy of Logistic regression classifier on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))
```

1.5.2 Support Vector Machines

Linear Support Vector Machine

```
In [ ]: from sklearn.svm import SVC
        from adspy_shared_utilities import plot_class_regions_for_classifier_subplo


        X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2, random_stat

        fig, subaxes = plt.subplots(1, 1, figsize=(7, 5))
        this_C = 1.0
        clf = SVC(kernel = 'linear', C=this_C).fit(X_train, y_train)
        title = 'Linear SVC, C = {:.3f}'.format(this_C)
        plot_class_regions_for_classifier_subplot(clf, X_train, y_train, None, None
```

Linear Support Vector Machine: C parameter

```
In [ ]: from sklearn.svm import LinearSVC
        from adspy_shared_utilities import plot_class_regions_for_classifier

        X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2, random_stat
        fig, subaxes = plt.subplots(1, 2, figsize=(8, 4))

        for this_C, subplot in zip([0.00001, 100], subaxes):
            clf = LinearSVC(C=this_C).fit(X_train, y_train)
            title = 'Linear SVC, C = {:.5f}'.format(this_C)
            plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                          None, None, title, subplot)
        plt.tight_layout()
```

Application to real dataset

```
In [ ]: from sklearn.svm import LinearSVC
        X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ran

        clf = LinearSVC().fit(X_train, y_train)
        print('Breast cancer dataset')
        print('Accuracy of Linear SVC classifier on training set: {:.2f}'
              .format(clf.score(X_train, y_train)))
        print('Accuracy of Linear SVC classifier on test set: {:.2f}'
              .format(clf.score(X_test, y_test)))
```

1.5.3 Multi-class classification with linear models

LinearSVC with M classes generates M one vs rest classifiers.

```
In [ ]: from sklearn.svm import LinearSVC

        X_train, X_test, y_train, y_test = train_test_split(X_fruits_2d, y_fruits_2

        clf = LinearSVC(C=5, random_state = 67).fit(X_train, y_train)
        print('Coefficients:\n', clf.coef_)
        print('Intercepts:\n', clf.intercept_)
```

Multi-class results on the fruit dataset

```
In [ ]: plt.figure(figsize=(6,6))
        colors = ['r', 'g', 'b', 'y']
        cmap_fruits = ListedColormap(['#FF0000', '#00FF00', '#0000FF','#FFFF00'])

        plt.scatter(X_fruits_2d[['height']], X_fruits_2d[['width']],
                    c=y_fruits_2d, cmap=cmap_fruits, edgecolor = 'black', alpha=.7)

        x_0_range = np.linspace(-10, 15)

        for w, b, color in zip(clf.coef_, clf.intercept_, ['r', 'g', 'b', 'y']):
            # Since class prediction with a linear model uses the formula y = w_0 x
            # and the decision boundary is defined as being all points with y = 0,
            # function of x_0 we just solve w_0 x_0 + w_1 x_1 + b = 0 for x_1:
            plt.plot(x_0_range, -(x_0_range * w[0] + b) / w[1], c=color, alpha=.8)

        plt.legend(target_names_fruits)
        plt.xlabel('height')
        plt.ylabel('width')
        plt.xlim(-2, 12)
        plt.ylim(-2, 15)
        plt.show()
```

1.6 Kernelized Support Vector Machines

1.6.1 Classification

```
In [ ]: from sklearn.svm import SVC
        from adspy_shared_utilities import plot_class_regions_for_classifier

        X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_stat

        # The default SVC kernel is radial basis function (RBF)
        plot_class_regions_for_classifier(SVC().fit(X_train, y_train),
                                          X_train, y_train, None, None,
                                          'Support Vector Classifier: RBF kernel')
```

```python
# Compare decision boundries with polynomial kernel, degree = 3
plot_class_regions_for_classifier(SVC(kernel = 'poly', degree = 3)
                                  .fit(X_train, y_train), X_train,
                                  y_train, None, None,
                                  'Support Vector Classifier: Polynomial ker
```

Support Vector Machine with RBF kernel: gamma parameter

```python
In [ ]: from adspy_shared_utilities import plot_class_regions_for_classifier

        X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_stat
        fig, subaxes = plt.subplots(3, 1, figsize=(4, 11))

        for this_gamma, subplot in zip([0.01, 1.0, 10.0], subaxes):
            clf = SVC(kernel = 'rbf', gamma=this_gamma).fit(X_train, y_train)
            title = 'Support Vector Classifier: \nRBF kernel, gamma = {:.2f}'.forma
            plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                      None, None, title, subplot)
        plt.tight_layout()
```

Support Vector Machine with RBF kernel: using both C and gamma parameter

```python
In [ ]: from sklearn.svm import SVC
        from adspy_shared_utilities import plot_class_regions_for_classifier_subplo

        from sklearn.model_selection import train_test_split


        X_train, X_test, y_train, y_test = train_test_split(X_D2, y_D2, random_stat
        fig, subaxes = plt.subplots(3, 4, figsize=(15, 10), dpi=50)

        for this_gamma, this_axis in zip([0.01, 1, 5], subaxes):

            for this_C, subplot in zip([0.1, 1, 15, 250], this_axis):
                title = 'gamma = {:.2f}, C = {:.2f}'.format(this_gamma, this_C)
                clf = SVC(kernel = 'rbf', gamma = this_gamma,
                          C = this_C).fit(X_train, y_train)
                plot_class_regions_for_classifier_subplot(clf, X_train, y_train,
                                                          X_test, y_test, title,
                                                          subplot)
                plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```

1.6.2 Application of SVMs to a real dataset: unnormalized data

```python
In [ ]: from sklearn.svm import SVC
        X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer,
                                                            random_state = 0)
```

```
clf = SVC(C=10).fit(X_train, y_train)
print('Breast cancer dataset (unnormalized features)')
print('Accuracy of RBF-kernel SVC on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of RBF-kernel SVC on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))
```

1.6.3 Application of SVMs to a real dataset: normalized data with feature preprocessing using minmax scaling

```
In [ ]: from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        clf = SVC(C=10).fit(X_train_scaled, y_train)
        print('Breast cancer dataset (normalized with MinMax scaling)')
        print('RBF-kernel SVC (with MinMax scaling) training set accuracy: {:.2f}'
              .format(clf.score(X_train_scaled, y_train)))
        print('RBF-kernel SVC (with MinMax scaling) test set accuracy: {:.2f}'
              .format(clf.score(X_test_scaled, y_test)))
```

1.7 Cross-validation

1.7.1 Example based on k-NN classifier with fruit dataset (2 features)

```
In [ ]: from sklearn.model_selection import cross_val_score

        clf = KNeighborsClassifier(n_neighbors = 5)
        X = X_fruits_2d.as_matrix()
        y = y_fruits_2d.as_matrix()
        cv_scores = cross_val_score(clf, X, y)

        print('Cross-validation scores (3-fold):', cv_scores)
        print('Mean cross-validation score (3-fold): {:.3f}'
              .format(np.mean(cv_scores)))
```

1.7.2 A note on performing cross-validation for more advanced scenarios.

In some cases (e.g. when feature values have very different ranges), we've seen the need to scale or normalize the training and test sets before use with a classifier. The proper way to do cross-validation when you need to scale the data is *not* to scale the entire dataset with a single transform, since this will indirectly leak information into the training data about the whole dataset, including the test data (see the lecture on data leakage later in the course). Instead, scaling/normalizing must be computed and applied for each cross-validation fold separately. To do this, the easiest way in scikit-learn is to use *pipelines*. While these are beyond the scope of this course, further information is available in the scikit-learn documentation here:

http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

or the Pipeline section in the recommended textbook: Introduction to Machine Learning with Python by Andreas C. Müller and Sarah Guido (O'Reilly Media).

1.8 Validation curve example

```python
In [ ]: from sklearn.svm import SVC
        from sklearn.model_selection import validation_curve

        param_range = np.logspace(-3, 3, 4)
        train_scores, test_scores = validation_curve(SVC(), X, y,
                                                     param_name='gamma',
                                                     param_range=param_range, cv=3)
```

```python
In [ ]: print(train_scores)
```

```python
In [ ]: print(test_scores)
```

```python
In [ ]: # This code based on scikit-learn validation_plot example
        #  See:  http://scikit-learn.org/stable/auto_examples/model_selection/plot_
        plt.figure()

        train_scores_mean = np.mean(train_scores, axis=1)
        train_scores_std = np.std(train_scores, axis=1)
        test_scores_mean = np.mean(test_scores, axis=1)
        test_scores_std = np.std(test_scores, axis=1)

        plt.title('Validation Curve with SVM')
        plt.xlabel('$\gamma$ (gamma)')
        plt.ylabel('Score')
        plt.ylim(0.0, 1.1)
        lw = 2

        plt.semilogx(param_range, train_scores_mean, label='Training score',
                     color='darkorange', lw=lw)

        plt.fill_between(param_range, train_scores_mean - train_scores_std,
                         train_scores_mean + train_scores_std, alpha=0.2,
                         color='darkorange', lw=lw)

        plt.semilogx(param_range, test_scores_mean, label='Cross-validation score',
                     color='navy', lw=lw)

        plt.fill_between(param_range, test_scores_mean - test_scores_std,
                         test_scores_mean + test_scores_std, alpha=0.2,
                         color='navy', lw=lw)

        plt.legend(loc='best')
        plt.show()
```

1.9 Decision Trees

```
In [ ]: from sklearn.datasets import load_iris
        from sklearn.tree import DecisionTreeClassifier
        from adspy_shared_utilities import plot_decision_tree
        from sklearn.model_selection import train_test_split


        iris = load_iris()

        X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
        clf = DecisionTreeClassifier().fit(X_train, y_train)

        print('Accuracy of Decision Tree classifier on training set: {:.2f}'
              .format(clf.score(X_train, y_train)))
        print('Accuracy of Decision Tree classifier on test set: {:.2f}'
              .format(clf.score(X_test, y_test)))
```

Setting max decision tree depth to help avoid overfitting

```
In [ ]: clf2 = DecisionTreeClassifier(max_depth = 3).fit(X_train, y_train)

        print('Accuracy of Decision Tree classifier on training set: {:.2f}'
              .format(clf2.score(X_train, y_train)))
        print('Accuracy of Decision Tree classifier on test set: {:.2f}'
              .format(clf2.score(X_test, y_test)))
```

Visualizing decision trees

```
In [ ]: plot_decision_tree(clf, iris.feature_names, iris.target_names)
```

Pre-pruned version (max_depth = 3)

```
In [ ]: plot_decision_tree(clf2, iris.feature_names, iris.target_names)
```

Feature importance

```
In [ ]: from adspy_shared_utilities import plot_feature_importances

        plt.figure(figsize=(10,4), dpi=80)
        plot_feature_importances(clf, iris.feature_names)
        plt.show()

        print('Feature importances: {}'.format(clf.feature_importances_))

In [ ]: from sklearn.tree import DecisionTreeClassifier
        from adspy_shared_utilities import plot_class_regions_for_classifier_subplo

        X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
```

```python
        fig, subaxes = plt.subplots(6, 1, figsize=(6, 32))

        pair_list = [[0,1], [0,2], [0,3], [1,2], [1,3], [2,3]]
        tree_max_depth = 4

        for pair, axis in zip(pair_list, subaxes):
            X = X_train[:, pair]
            y = y_train

            clf = DecisionTreeClassifier(max_depth=tree_max_depth).fit(X, y)
            title = 'Decision Tree, max_depth = {:d}'.format(tree_max_depth)
            plot_class_regions_for_classifier_subplot(clf, X, y, None,
                                                      None, title, axis,
                                                      iris.target_names)

            axis.set_xlabel(iris.feature_names[pair[0]])
            axis.set_ylabel(iris.feature_names[pair[1]])

        plt.tight_layout()
        plt.show()
```

Decision Trees on a real-world dataset

```python
In [ ]: from sklearn.tree import DecisionTreeClassifier
        from adspy_shared_utilities import plot_decision_tree
        from adspy_shared_utilities import plot_feature_importances

        X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, ran

        clf = DecisionTreeClassifier(max_depth = 4, min_samples_leaf = 8,
                                     random_state = 0).fit(X_train, y_train)

        plot_decision_tree(clf, cancer.feature_names, cancer.target_names)

In [ ]: print('Breast cancer dataset: decision tree')
        print('Accuracy of DT classifier on training set: {:.2f}'
              .format(clf.score(X_train, y_train)))
        print('Accuracy of DT classifier on test set: {:.2f}'
              .format(clf.score(X_test, y_test)))

        plt.figure(figsize=(10,6),dpi=80)
        plot_feature_importances(clf, cancer.feature_names)
        plt.tight_layout()

        plt.show()
```