```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Module 6: Classification\n",
    "\n",
    "The following tutorial contains Python examples for solving classification problems. You
should refer to the Chapters 3 and 4 of the \"Introduction to Data Mining\" book to understand
some of the concepts introduced in this tutorial. The notebook can be downloaded from
http://www.cse.msu.edu/~ptan/dmbook/tutorials/tutorial6/tutorial6.ipynb.\n",
    "\n",
    "Classification is the task of predicting a nominal-valued attribute (known as class label)
based on the values of other attributes (known as predictor variables). The goals for this
tutorial are as follows:\n",
    "1. To provide examples of using different classification techniques from the scikit-learn
library package.\n",
    "2. To demonstrate the problem of model overfitting.\n",
    "\n",
    "Read the step-by-step instructions below carefully. To execute the code, click on the
corresponding cell and press the SHIFT-ENTER keys simultaneously.\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## 6.1 Vertebrate Dataset\n",
    "\n",
    "We use a variation of the vertebrate data described in Example 3.1 of Chapter 3. Each
vertebrate is classified into one of 5 categories: mammals, reptiles, birds, fishes, and
amphibians, based on a set of explanatory attributes (predictor variables). Except for \"name\",
the rest of the attributes have been converted into a *one hot encoding* binary representation.
To illustrate this, we will first load the data into a Pandas DataFrame object and display its
content."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "import pandas as pd\n",
    "\n",
    "data = pd.read_csv('vertebrate.csv',header='infer')\n",
    "data"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Given the limited number of training examples, suppose we convert the problem into a binary
classification task (mammals versus non-mammals). We can do so by replacing the class labels of
the instances to *non-mammals* except for those that belong to the *mammals* class."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "data['Class'] = data['Class'].replace(['fishes','birds','amphibians','reptiles'],'non-
mammals')\n",
    "data"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "We can apply Pandas cross-tabulation to examine the relationship between the Warm-blooded
and Gives Birth attributes with respect to the class. "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "pd.crosstab([data['Warm-blooded'],data['Gives Birth']],data['Class'])"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The results above show that it is possible to distinguish mammals from non-mammals using
these two attributes alone since each combination of their attribute values would yield only
instances that belong to the same class. For example, mammals can be identified as warm-blooded
vertebrates that give birth to their young. Such a relationship can also be derived using a
decision tree classifier, as shown by the example given in the next subsection."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## 3.2 Decision Tree Classifier\n",
    "\n",
    "In this section, we apply a decision tree classifier to the vertebrate dataset described in
the previous subsection."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn import tree\n",
    "\n",
    "Y = data['Class']\n",
    "X = data.drop(['Name','Class'],axis=1)\n",
    "\n",
    "clf = tree.DecisionTreeClassifier(criterion='entropy',max_depth=3)\n",
    "clf = clf.fit(X, Y)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The preceding commands will extract the predictor (X) and target class (Y) attributes from
the vertebrate dataset and create a decision tree classifier object using entropy as its
impurity measure for splitting criterion. The decision tree class in Python sklearn library also
supports using 'gini' as impurity measure. The classifier above is also constrained to generate
trees with a maximum depth equals to 3. Next, the classifier is trained on the labeled data
using the fit() function. \n",
    "\n",
    "We can plot the resulting decision tree obtained after training the classifier. To do this,
you must first install both graphviz (http://www.graphviz.org) and its Python interface called
pydotplus (http://pydotplus.readthedocs.io/)."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "import pydotplus \n",
    "from IPython.display import Image\n",
    "\n",
    "dot_data = tree.export_graphviz(clf, feature_names=X.columns, class_names=['mammals','non-
mammals'], filled=True, \n",
    "                                out_file=None) \n",
    "graph = pydotplus.graph_from_dot_data(dot_data) \n",
    "Image(graph.create_png())"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Next, suppose we apply the decision tree to classify the following test examples."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "testData = [['gila monster',0,0,0,0,1,1,'non-mammals'],\n",
    "           ['platypus',1,0,0,0,1,1,'mammals'],\n",
    "           ['owl',1,0,0,1,1,0,'non-mammals'],\n",
    "           ['dolphin',1,1,1,0,0,0,'mammals']]\n",
    "testData = pd.DataFrame(testData, columns=data.columns)\n",
    "testData"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "We first extract the predictor and target class attributes from the test data and then
apply the decision tree classifier to predict their classes."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "testY = testData['Class']\n",
    "testX = testData.drop(['Name','Class'],axis=1)\n",
    "\n",
    "predY = clf.predict(testX)\n",
    "predictions = pd.concat([testData['Name'],pd.Series(predY,name='Predicted Class')],
axis=1)\n",
    "predictions"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Except for platypus, which is an egg-laying mammal, the classifier correctly predicts the
class label of the test examples. We can calculate the accuracy of the classifier on the test
data as shown by the example given below."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn.metrics import accuracy_score\n",
    "\n",
    "print('Accuracy on test data is %.2f' % (accuracy_score(testY, predY)))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## 3.3 Model Overfitting\n",
    "\n",
    "To illustrate the problem of model overfitting, we consider a two-dimensional dataset
containing 1500 labeled instances, each of which is assigned to one of two classes, 0 or 1.
Instances from each class are generated as follows:\n",
    "1. Instances from class 1 are generated from a mixture of 3 Gaussian distributions,
centered at [6,14], [10,6], and [14 14], respectively. \n",
    "2. Instances from class 0 are generated from a uniform distribution in a square region,
whose sides have a length equals to 20.\n",
    "\n",
    "For simplicity, both classes have equal number of labeled instances. The code for
generating and plotting the data is shown below. All instances from class 1 are shown in red
while those from class 0 are shown in black."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "import numpy as np\n",
    "import matplotlib.pyplot as plt\n",
    "from numpy.random import random\n",
    "\n",
    "%matplotlib inline\n",
    "\n",
    "N = 1500\n",
    "\n",
    "mean1 = [6, 14]\n",
    "mean2 = [10, 6]\n",
    "mean3 = [14, 14]\n",
    "cov = [[3.5, 0], [0, 3.5]]  # diagonal covariance\n",
    "\n",
    "np.random.seed(50)\n",
    "X = np.random.multivariate_normal(mean1, cov, int(N/6))\n",
    "X = np.concatenate((X, np.random.multivariate_normal(mean2, cov, int(N/6))))\n",
    "X = np.concatenate((X, np.random.multivariate_normal(mean3, cov, int(N/6))))\n",
    "X = np.concatenate((X, 20*np.random.rand(int(N/2),2)))\n",
    "Y = np.concatenate((np.ones(int(N/2)),np.zeros(int(N/2))))\n",
    "\n",
    "plt.plot(X[:int(N/2),0],X[:int(N/2),1],'r+',X[int(N/2):,0],X[int(N/2):,1],'k.',ms=4)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "In this example, we reserve 80% of the labeled data for training and the remaining 20% for
testing. We then fit decision trees of different maximum depths (from 2 to 50) to the training
set and plot their respective accuracies when applied to the training and test sets. "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "#########################################\n",
    "# Training and Test set creation\n",
    "#########################################\n",
    "\n",
    "from sklearn.model_selection import train_test_split\n",
    "X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.8,
random_state=1)\n",
    "\n",
    "from sklearn import tree\n",
    "from sklearn.metrics import accuracy_score\n",
    "\n",
    "#########################################\n",
    "# Model fitting and evaluation\n",
    "#########################################\n",
    "\n",
    "maxdepths = [2,3,4,5,6,7,8,9,10,15,20,25,30,35,40,45,50]\n",
    "\n",
    "trainAcc = np.zeros(len(maxdepths))\n",
    "testAcc = np.zeros(len(maxdepths))\n",
    "\n",
    "index = 0\n",
    "for depth in maxdepths:\n",
    "    clf = tree.DecisionTreeClassifier(max_depth=depth)\n",
    "    clf = clf.fit(X_train, Y_train)\n",
    "    Y_predTrain = clf.predict(X_train)\n",
    "    Y_predTest = clf.predict(X_test)\n",
    "    trainAcc[index] = accuracy_score(Y_train, Y_predTrain)\n",
    "    testAcc[index] = accuracy_score(Y_test, Y_predTest)\n",
    "    index += 1\n",
    "    \n",
    "#########################################\n",
    "# Plot of training and test accuracies\n",
    "#########################################\n",
    "    \n",
    "plt.plot(maxdepths,trainAcc,'ro-',maxdepths,testAcc,'bv--')\n",
    "plt.legend(['Training Accuracy','Test Accuracy'])\n",
    "plt.xlabel('Max depth')\n",
    "plt.ylabel('Accuracy')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The plot above shows that training accuracy will continue to improve as the maximum depth
of the tree increases (i.e., as the model becomes more complex). However, the test accuracy
initially improves up to a maximum depth of 5, before it gradually decreases due to model
overfitting."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## 3.4 Alternative Classification Techniques\n",
    "\n",
    "Besides decision tree classifier, the Python sklearn library also supports other
classification techniques. In this section, we provide examples to illustrate how to apply the
k-nearest neighbor classifier, linear classifiers (logistic regression and support vector
machine), as well as ensemble methods (boosting, bagging, and random forest) to the 2-
dimensional data given in the previous section."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### 3.4.1 K-Nearest neighbor classifier\n",
    "\n",
    "In this approach, the class label of a test instance is predicted based on the majority
class of its *k* closest training instances. The number of nearest neighbors, *k*, is a
hyperparameter that must be provided by the user, along with the distance metric. By default, we
can use Euclidean distance (which is equivalent to Minkowski distance with an exponent factor
equals to p=2):\n",
    "\n",
    "\\begin{equation*}\n",
    "\\textrm{Minkowski distance}(x,y) = \\bigg[\\sum_{i=1}^N |x_i-y_i|^p \\bigg]^{\\frac{1}
{p}}\n",
    "\\end{equation*}"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn.neighbors import KNeighborsClassifier\n",
    "import matplotlib.pyplot as plt\n",
    "%matplotlib inline\n",
    "\n",
    "numNeighbors = [1, 5, 10, 15, 20, 25, 30]\n",
    "trainAcc = []\n",
    "testAcc = []\n",
    "\n",
    "for k in numNeighbors:\n",
    "    clf = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)\n",
    "    clf.fit(X_train, Y_train)\n",
    "    Y_predTrain = clf.predict(X_train)\n",
    "    Y_predTest = clf.predict(X_test)\n",
    "    trainAcc.append(accuracy_score(Y_train, Y_predTrain))\n",
    "    testAcc.append(accuracy_score(Y_test, Y_predTest))\n",
    "\n",
    "plt.plot(numNeighbors, trainAcc, 'ro-', numNeighbors, testAcc,'bv--')\n",
    "plt.legend(['Training Accuracy','Test Accuracy'])\n",
    "plt.xlabel('Number of neighbors')\n",
    "plt.ylabel('Accuracy')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### 3.4.2 Linear Classifiers\n",
    "\n",
    "Linear classifiers such as logistic regression and support vector machine (SVM) constructs
a linear separating hyperplane to distinguish instances from different classes. \n",
    "\n",
    "For logistic regression, the model can be described by the following equation:\n",
    "\\begin{equation*}\n",
    "P(y=1|x) = \\frac{1}{1 + \\exp^{-w^Tx - b}} = \\sigma(w^Tx + b)\n",
    "\\end{equation*}\n",
    "The model parameters (w,b) are estimated by optimizing the following regularized negative
log-likelihood function:\n",
    "\\begin{equation*}\n",
    "(w^*,b^*) = \\arg\\min_{w,b} - \\sum_{i=1}^N y_i \\log\\bigg[\\sigma(w^Tx_i + b)\\bigg] +
(1-y_i) \\log\\bigg[\\sigma(-w^Tx_i - b)\\bigg] + \\frac{1}{C} \\Omega([w,b])\n",
    "\\end{equation*}\n",
    "where $C$ is a hyperparameter that controls the inverse of model complexity (smaller values
imply stronger regularization) while $\\Omega(\\cdot)$ is the regularization term, which by
default, is assumed to be an $l_2$-norm in sklearn.\n",
    "\n",
    "For support vector machine, the model parameters $(w^*,b^*)$ are estimated by solving the
following constrained optimization problem:\n",
    "\\begin{eqnarray*}\n",
    "&&\\min_{w^*,b^*,\\{\\xi_i\\}} \\frac{\\|w\\|^2}{2} + \\frac{1}{C} \\sum_i \\xi_i \\\\\n",
    "\\textrm{s.t.} && \\forall i: y_i\\bigg[w^T \\phi(x_i) + b\\bigg] \\ge 1 - \\xi_i, \\ \\
\\xi_i \\ge 0  \n",
    "\\end{eqnarray*}"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn import linear_model\n",
    "from sklearn.svm import SVC\n",
    "\n",
    "C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]\n",
    "LRtrainAcc = []\n",
    "LRtestAcc = []\n",
    "SVMtrainAcc = []\n",
    "SVMtestAcc = []\n",
    "\n",
    "for param in C:\n",
    "    clf = linear_model.LogisticRegression(C=param)\n",
    "    clf.fit(X_train, Y_train)\n",
    "    Y_predTrain = clf.predict(X_train)\n",
    "    Y_predTest = clf.predict(X_test)\n",
    "    LRtrainAcc.append(accuracy_score(Y_train, Y_predTrain))\n",
    "    LRtestAcc.append(accuracy_score(Y_test, Y_predTest))\n",
    "\n",
    "    clf = SVC(C=param,kernel='linear')\n",
    "    clf.fit(X_train, Y_train)\n",
    "    Y_predTrain = clf.predict(X_train)\n",
    "    Y_predTest = clf.predict(X_test)\n",
    "    SVMtrainAcc.append(accuracy_score(Y_train, Y_predTrain))\n",
    "    SVMtestAcc.append(accuracy_score(Y_test, Y_predTest))\n",
    "\n",
    "fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))\n",
    "ax1.plot(C, LRtrainAcc, 'ro-', C, LRtestAcc,'bv--')\n",
    "ax1.legend(['Training Accuracy','Test Accuracy'])\n",
    "ax1.set_xlabel('C')\n",
    "ax1.set_xscale('log')\n",
    "ax1.set_ylabel('Accuracy')\n",
    "\n",
    "ax2.plot(C, SVMtrainAcc, 'ro-', C, SVMtestAcc,'bv--')\n",
    "ax2.legend(['Training Accuracy','Test Accuracy'])\n",
    "ax2.set_xlabel('C')\n",
    "ax2.set_xscale('log')\n",
    "ax2.set_ylabel('Accuracy')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Note that linear classifiers perform poorly on the data since the true decision boundaries between classes are nonlinear for the given 2-dimensional dataset."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### 3.4.3 Nonlinear Support Vector Machine\n",
    "\n",
    "The code below shows an example of using nonlinear support vector machine with a Gaussian radial basis function kernel to fit the 2-dimensional dataset."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn.svm import SVC\n",
    "\n",
    "C = [0.01, 0.1, 0.2, 0.5, 0.8, 1, 5, 10, 20, 50]\n",
    "SVMtrainAcc = []\n",
    "SVMtestAcc = []\n",
    "\n",
    "for param in C:\n",
    "    clf = SVC(C=param,kernel='rbf',gamma='auto')\n",
    "    clf.fit(X_train, Y_train)\n",
    "    Y_predTrain = clf.predict(X_train)\n",
    "    Y_predTest = clf.predict(X_test)\n",
    "    SVMtrainAcc.append(accuracy_score(Y_train, Y_predTrain))\n",
    "    SVMtestAcc.append(accuracy_score(Y_test, Y_predTest))\n",
    "\n",
    "plt.plot(C, SVMtrainAcc, 'ro-', C, SVMtestAcc,'bv--')\n",
    "plt.legend(['Training Accuracy','Test Accuracy'])\n",
    "plt.xlabel('C')\n",
    "plt.xscale('log')\n",
    "plt.ylabel('Accuracy')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Observe that the nonlinear SVM can achieve a higher test accuracy compared to linear SVM."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### 3.4.4 Ensemble Methods\n",
    "\n",
    "An ensemble classifier constructs a set of base classifiers from the training data and
performs classification by taking a vote on the predictions made by each base classifier. We
consider 3 types of ensemble classifiers in this example: bagging, boosting, and random forest.
Detailed explanation about these classifiers can be found in Section 4.10 of the book.\n",
    "\n",
    "In the example below, we fit 500 base classifiers to the 2-dimensional dataset using each
ensemble method. The base classifier corresponds to a decision tree with maximum depth equals to
10."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn import ensemble\n",
    "from sklearn.tree import DecisionTreeClassifier\n",
    "\n",
    "numBaseClassifiers = 500\n",
    "maxdepth = 10\n",
    "trainAcc = []\n",
    "testAcc = []\n",
    "\n",
    "clf = ensemble.RandomForestClassifier(n_estimators=numBaseClassifiers)\n",
    "clf.fit(X_train, Y_train)\n",
    "Y_predTrain = clf.predict(X_train)\n",
    "Y_predTest = clf.predict(X_test)\n",
    "trainAcc.append(accuracy_score(Y_train, Y_predTrain))\n",
    "testAcc.append(accuracy_score(Y_test, Y_predTest))\n",
    "\n",
    "clf =
ensemble.BaggingClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClassi
fiers)\n",
    "clf.fit(X_train, Y_train)\n",
    "Y_predTrain = clf.predict(X_train)\n",
    "Y_predTest = clf.predict(X_test)\n",
    "trainAcc.append(accuracy_score(Y_train, Y_predTrain))\n",
    "testAcc.append(accuracy_score(Y_test, Y_predTest))\n",
    "\n",
    "clf =
ensemble.AdaBoostClassifier(DecisionTreeClassifier(max_depth=maxdepth),n_estimators=numBaseClass
ifiers)\n",
    "clf.fit(X_train, Y_train)\n",
    "Y_predTrain = clf.predict(X_train)\n",
    "Y_predTest = clf.predict(X_test)\n",
    "trainAcc.append(accuracy_score(Y_train, Y_predTrain))\n",
    "testAcc.append(accuracy_score(Y_test, Y_predTest))\n",
    "\n",
    "methods = ['Random Forest', 'Bagging', 'AdaBoost']\n",
    "fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,6))\n",
    "ax1.bar([1.5,2.5,3.5], trainAcc)\n",
    "ax1.set_xticks([1.5,2.5,3.5])\n",
    "ax1.set_xticklabels(methods)\n",
    "ax2.bar([1.5,2.5,3.5], testAcc)\n",
    "ax2.set_xticks([1.5,2.5,3.5])\n",
    "ax2.set_xticklabels(methods)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## 3.5 Summary\n",
    "\n",
    "This section provides several examples of using Python sklearn library to build
classification models from a given input data. We also illustrate the problem of model
overfitting and show how to apply different classification methods to the given dataset."
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.3"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```