Lab 1a: Classification

By: Malachi Jones, PhD

Outline

- Objectives
- Lab 1a.1: Implementing Adaline SGD with OvR

Lab 1a.2: Tuning SVM Parameters

LAB 1A OBJECTIVES

Lab 1a Objectives

- After this lab, students should be able
  - Have a solid intuition for Machine Learning Algorithms
  - Use pandas, NumPy, and Matplotlib to read in, process, and visualize data
  - Implement linear classification algorithms in python
  - Understand the mechanics of ML algorithms such that they can design and implement custom (e.g. approximate) versions suitable for the problem they want to address

LAB 1A OVERVIEW

Lab 1A Overview

- This lab will consist of the following sections:
  - Lab 1.1: **Implementing Adaline Stochastic Gradient Descent with OvR (4 points)**
  - Lab 1.2: **Tuning parameters with SVM (1 point)**

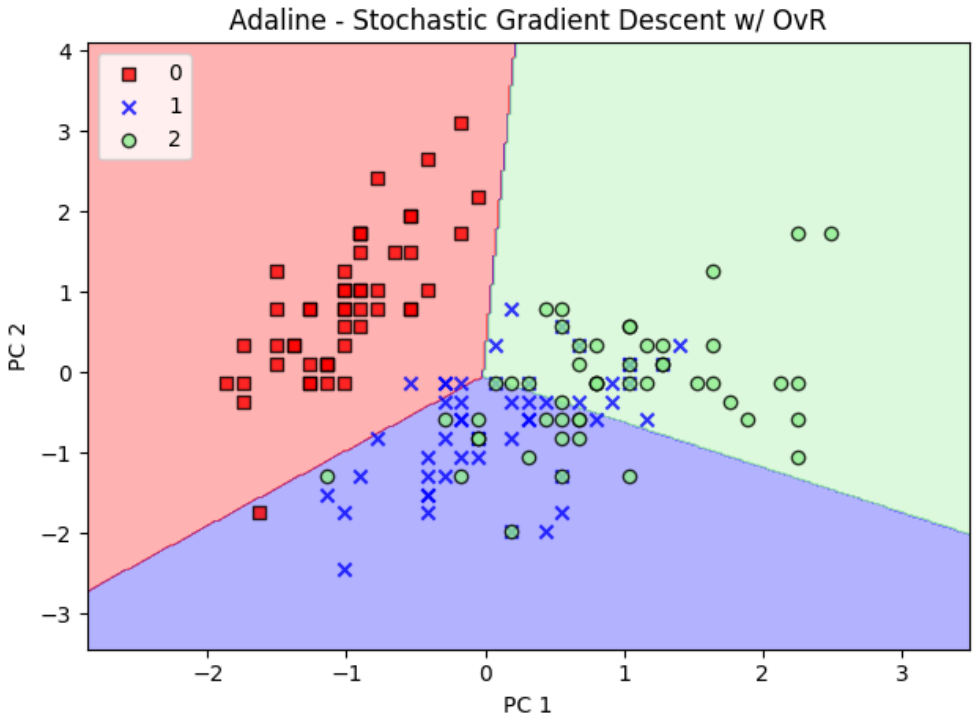
- **Reminder**: ***No additional imports shall be added to any of the python files. Any additions will result in an automatic 0 for the portion of the lab***

- **Note**: *The Iris dataset is a classic example in the field of machine learning and will be used for some of the supervised ML portion of this lab*

LAB 1A.1: IMPLEMENTING ADALINE SGD WITH OVR

Lab 1A.1 Implementing Adaline SGD w/ OVR

- **Lab 1 Objective**:
  - Implement Adaline SGD with one-versus-the-rest (OvR) for multinomial classification of the iris dataset
  - Once completed, the resulting plot should look like the following:



Lab 1A.1 Implementing Adaline SGD w/ OVR

- **Lab 1a.1 Steps:**
  1. Need to train K Adaline SGD classifiers, where K is the number of classes in the sample. In this instance, K=3

```
def fit(self, X, y):
    """
    Fit training data

    :param X: {array-like, shape = [n_samples, n_features]
              Training vectors, where n_samples is the number of samples and
                                    n_features  is the number of features
    :param y: array-like, shape = [n_samples]
              Target values
    :return:  self: object

    """
    # Additional hint about the incoming data types
    if not isinstance(X, np.ndarray) or not isinstance(y, np.ndarray):
        raise TypeError("Input must be an numpy.ndarray")

    logger.warning("@Todo: Need to follow OVR steps in lecture material to create K Adaline classifiers")

    """
        Hint:    Let fk denote a classifier that is trained to  classify class k
                 For each class k in K
                      Create a new label vector yk
                      Update yk such that all data points that are of class k are set to 1
                      Update yk such that all other data points not in class k to be -1
                      Train an AdalineSGD classifier with the yk labels
    """

    return self
```

The `fit()` method that will need to be updated is in the file `AdalineSGDOVR.py`

Lab 1A.1 Implementing Adaline SGD w/ OVR

- **Lab 1a.1 Steps:**
  2. Implement the OvR Classification algorithm as outlined in the corresponding lecture

```
def predict(self, X):
    """Return class label

    :param X: numpy nd-array

    """

    if not isinstance(X, np.ndarray):
        raise TypeError("Input must be an numpy.ndarray")


    y_ovr = np.zeros(len(X))

    logger.warning("@Todo: Implement the OvR Classification Algorithm as outlined in lecture")

    """
        Hint:    1. Apply all classifiers fk(x) to an unseen sample x
                 2. Each classifier fk(x) will produce a confidence score
                 3. Select the fk(x) with the highest confidence score
                 4. The sample x will inherit the label associated with the fk(x)

    """

    return y_ovr
```

The `predict()` method that will need to be updated is in the file `AdalineSGDOVR.py`

Lab 1A.1 Implementing Adaline SGD w/ OVR

- **Lab 1a.1 Additional Observations**

```
# Load the iris data into a "data frame"
df = pd.read_csv("iris.data", header=None)

# Map the string labels to integers that we will then pass into the classifier
y = df.iloc[:, 4].values
y[y == 'Iris-setosa'] = 0
y[y == 'Iris-versicolor'] = 1
y[y == 'Iris-virginica'] = 2

# extract the four iris features for each of the samples to build the sample set X
X = df.iloc[:, [0, 1, 2, 3]].values


# We need to normalize the the dataset
standard_scaler = StandardScaler()
X_std = standard_scaler.fit_transform(X)

# Since we are using 4 features and would like to project into a 2-d space, we'll use
# PCA to accomplish this.
X_pca = PCA(n_components=2).fit_transform(X_std)

# Create our model
ada = AdalineSGDOVR(n_iter=15, eta=0.01, random_state=1).fit(X_pca, y)

# Plot the decision boundaries decided by our model
plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Stochastic Gradient Descent w/ OvR')
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()

plt.close()
```

(The above code is in the file AdalineSGDOVRTraining.py)

Lab 1A.1 Implementing Adaline w/ SGD

- The relevant files for Lab 1a.1 (included in folder Lab1) ***that will be updated*** are the following:
  - AdalineSGDOVR.py

- Files you may need to reference but ***should NOT be updated***
  - AdalineSGDOVRTraining.py
  - AdalineSGD.py
  - AdalineSGDTraining.py
  - iris.data

Lab 1A.1 Implementing Adaline w/ SGD

- **Submission**
  - You will submit the following file(s) into a folder called lab_1a :
    i. Your implementation of AdalineSGDOVR.py

LAB 1.2: TUNING SVM

Lab 1a.2 Tuning SVM

- **Lab 1a.2 Objective**: Tune the SVM parameters in svn_kernel.py such that we obtain a zero one loss error of approximately .02

Lab 1a.2 Tuning SVM

- **Lab 1a.2 Steps**:
  1. Adjust the following parameters of the SVM model to achieve the error rate of approximately .02:
     i. Kernel,
     ii. Gamma
     iii. C

```
svm = SVC(kernel='linear', random_state=1, gamma=1, C=1)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor, classifier=svm)
plt.legend(loc='upper left')

pred_y = svm.predict(X_xor)

error = zero_one_loss(y_xor, pred_y)

print("Zero one loss error:{}".format(error))
```

The file that that will need to updated with the parameters is named `svn_kernel.py`

Lab 1.2 Tuning SVM

- **Hint #1**
  - For additional details about the SVM parameters, please see the following link: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Lab 1a.2 Tuning SVM

- The relevant files **to be modified** for Lab 1a.2 (included in folder Lab1) consists of the following:
  - svn_kernel.py

Lab 1.2 Implementing Adaline w/ SGD

- **Submission**
  - You will submit the following file(s) into a folder called lab_1a :
    i. An svn_kernel.py file with the appropriate parameters tuned to achieve the Lab 1.2 objective