Project 1 - Support Vector Machine Classification

NAME(S): Morgan Rockett, Prithvi Shahani, Anthony Wong

DATE: 4/03/23

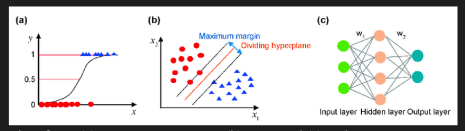
What will we do?

Using gradient descent, we will build a Support Vector Machine to find the optimal hyperplane that maximizes the margin between two toy data classes.

What are some use cases for SVMs?

-Classification, regression (time series prediction, etc.), outlier detection, clustering

How does an SVM compare to other ML algorithms?



Classifiers: (a) Logistic Regression, (b) SVM, and (c) Multi-Layer Perception (MLP)

- As a rule of thumb, SVMs are great for relatively small data sets with fewer outliers.
- Other algorithms (Random forests, deep neural networks, etc.) require more data but almost always develop robust models.
- The decision of which classifier to use depends on your dataset and the general complexity of the problem.
- "Premature optimization is the root of all evil (or at least most of it) in programming." - Donald Knuth, CS Professor (Turing award speech 1974)

Other Examples

- Learning to use Scikit-learn's SVM function to classify images https://github.com/ksopyla/svm_mnist_digit_classification
- Pulse classification, more useful dataset
  https://github.com/akasantony/pulse-classification-svm

```
## What is a Support Vector Machine?

It's a supervised machine learning algorithm that can be used for both classification and regression problems. But it's usually used for classification. Given two or more labeled data classes, it acts as a discriminative
 classifier, formally defined by an optimal hyperplane that separates all the classes. New examples mapped into that space can then be categorized based on which side of the gap they fall.
```

DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help 01:01:28:59 83% Thu Apr 13 11:02 PM

project1 Version control Add Configuration...

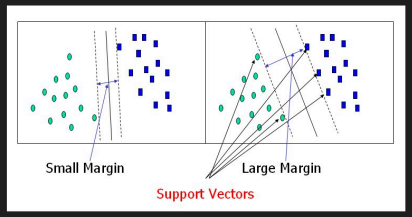
notebook.ipynb implementation.py

Markdown Managed: http://localhost:8890 Python 3 (ipykernel) Trusted

10 28

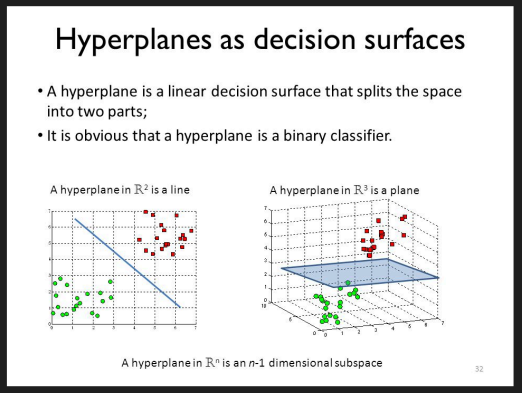
```
## What is a Support Vector Machine?

It's a supervised machine learning algorithm that can be used for both classification and regression problems. But it's usually used for classification. Given two or more labeled data classes, it acts as a discriminative
 classifier, formally defined by an optimal hyperplane that separates all the classes. New examples mapped into that space can then be categorized based on which side of the gap they fall.
```

What are Support Vectors?



Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set; they help us build our SVM.

What is a hyperplane?



Geometry tells us that a hyperplane is a subspace of one dimension less than its ambient space. For instance, a hyperplane of an n-dimensional space is a flat subset with size $n-1$. By its nature, it separates the space in half.

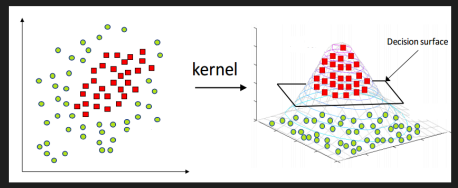
Linear vs nonlinear classification?

project1 > notebook.ipynb 30:1 LF UTF-8 4 spaces ml_0135

Linear vs nonlinear classification?

Sometimes our data is linearly separable. That means for N classes with M features. We can learn a mapping that is a linear combination. (like $y = mx + b$). Or even a multidimensional hyperplane ($y = x + z + b + q$). No matter how many dimensions/features a set of classes have, we can represent the mapping using a linear function.

But sometimes it is not. Like if there was a quadratic mapping. Luckily for us, SVMs can efficiently perform a non-linear classification using what is called the kernel trick.



More on this as a Bonus question comes at the end of notebook.

All right, let's get to the building!

Instructions

In this assignment, you will implement a support vector machine (SVM) from scratch, and you will use your implementation for multiclass classification on the MNIST dataset.

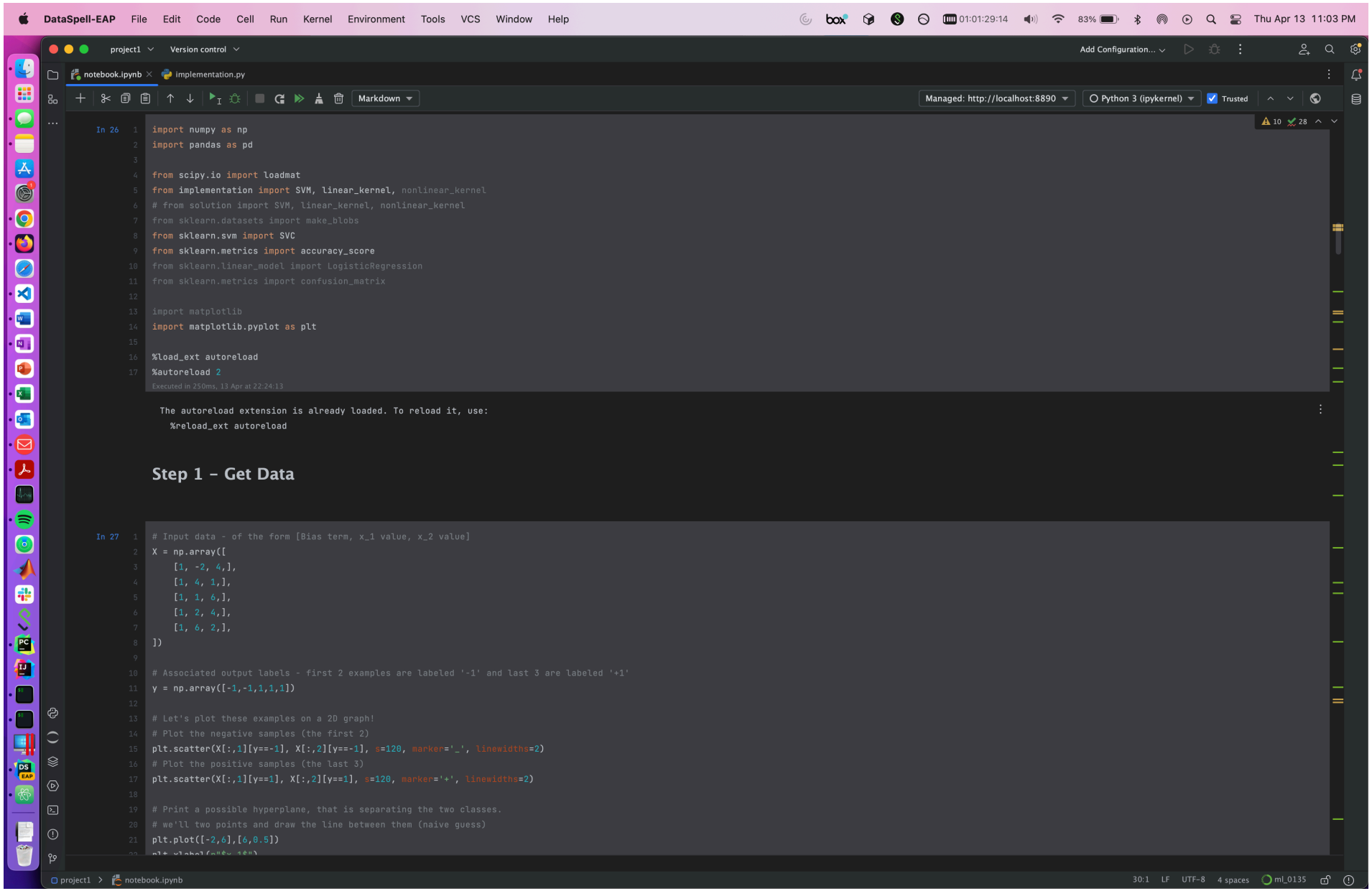
In `implementation.py` implement the SVM class. In the fit function, use `scipy.minimize` (see documentation) to solve the constrained optimization problem:

$$\begin{aligned} \underset{a}{\text{maximize}} \quad & \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j y_i y_j (x_i \cdot x_j) \\ \text{subject to} \quad & a_i \geq 0, i = 1, \ldots, n \\ & \sum_{i=1}^{n} a_i y_i = 0 \end{aligned}$$

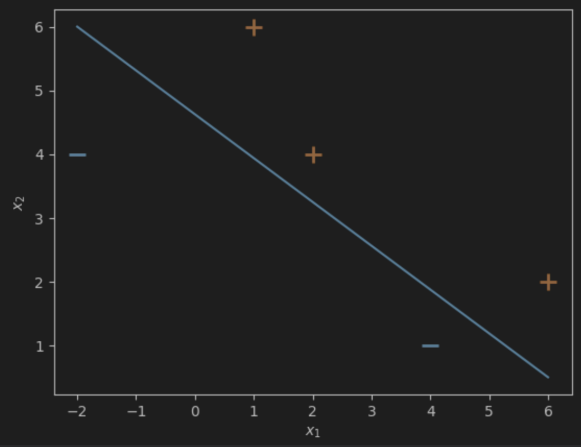
Note: An SVM is a convex optimization problem. Using to solve the equation above will be computationally expensive given larger datasets. CS 168 Convex Optimization is a course to take later if interested in optimization and the mathematics and intuition that drives it.

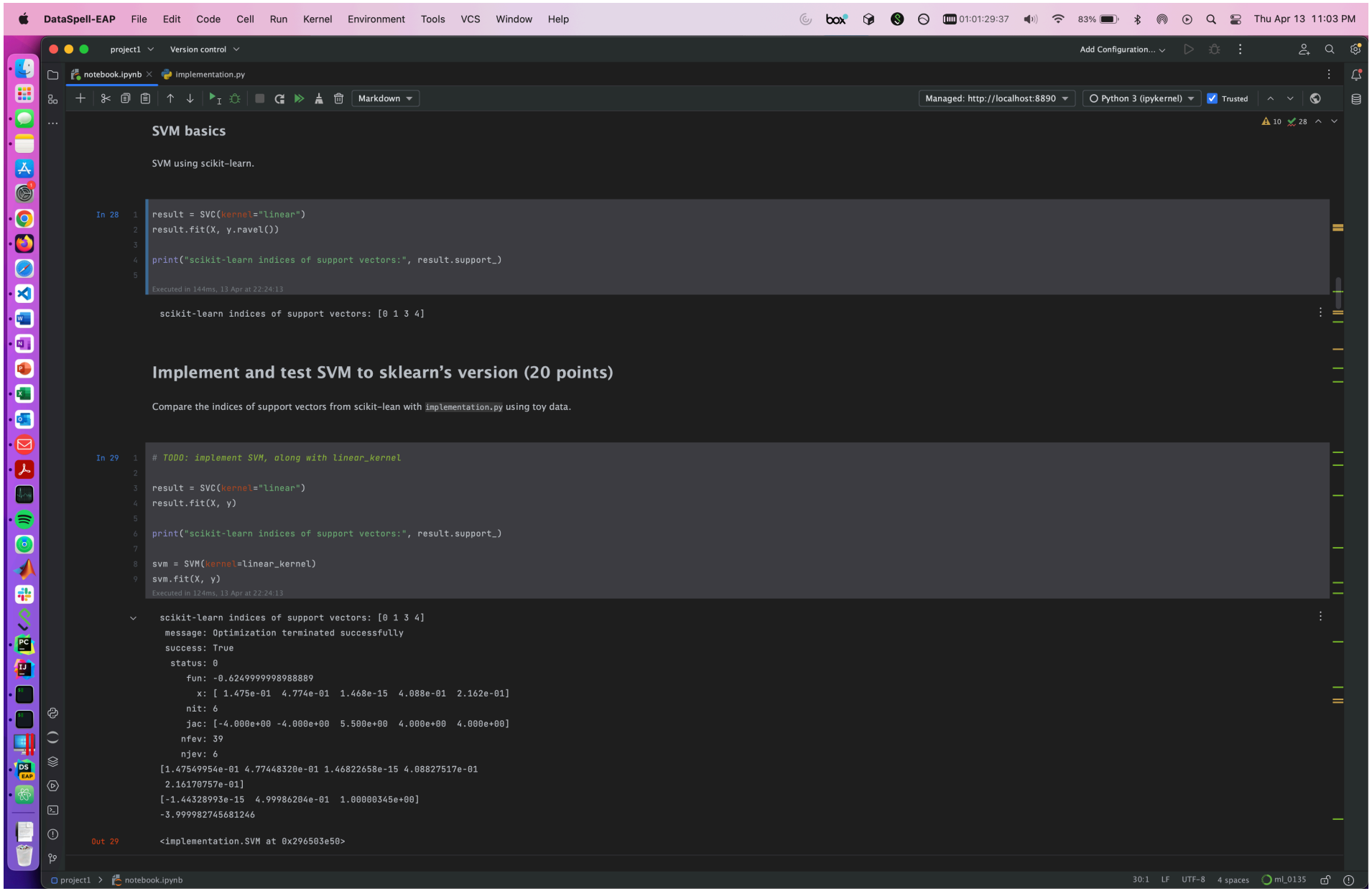
In 26

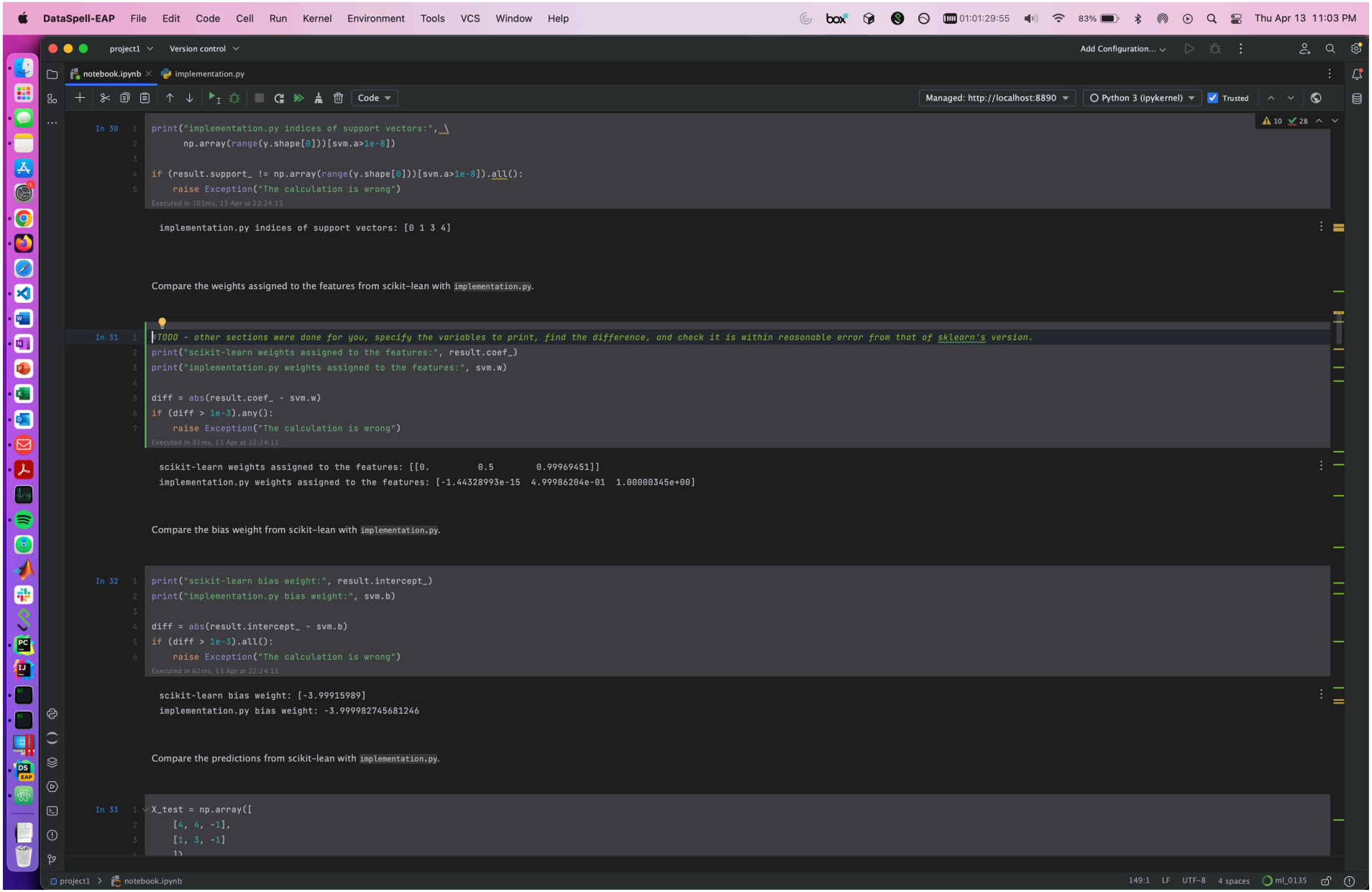
```python
import numpy as np
import pandas as pd

from scipy.io import loadmat
from implementation import SVM, linear_kernel, nonlinear_kernel
# from solution import SVM, linear_kernel, nonlinear_kernel
from sklearn.datasets import make_blobs
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

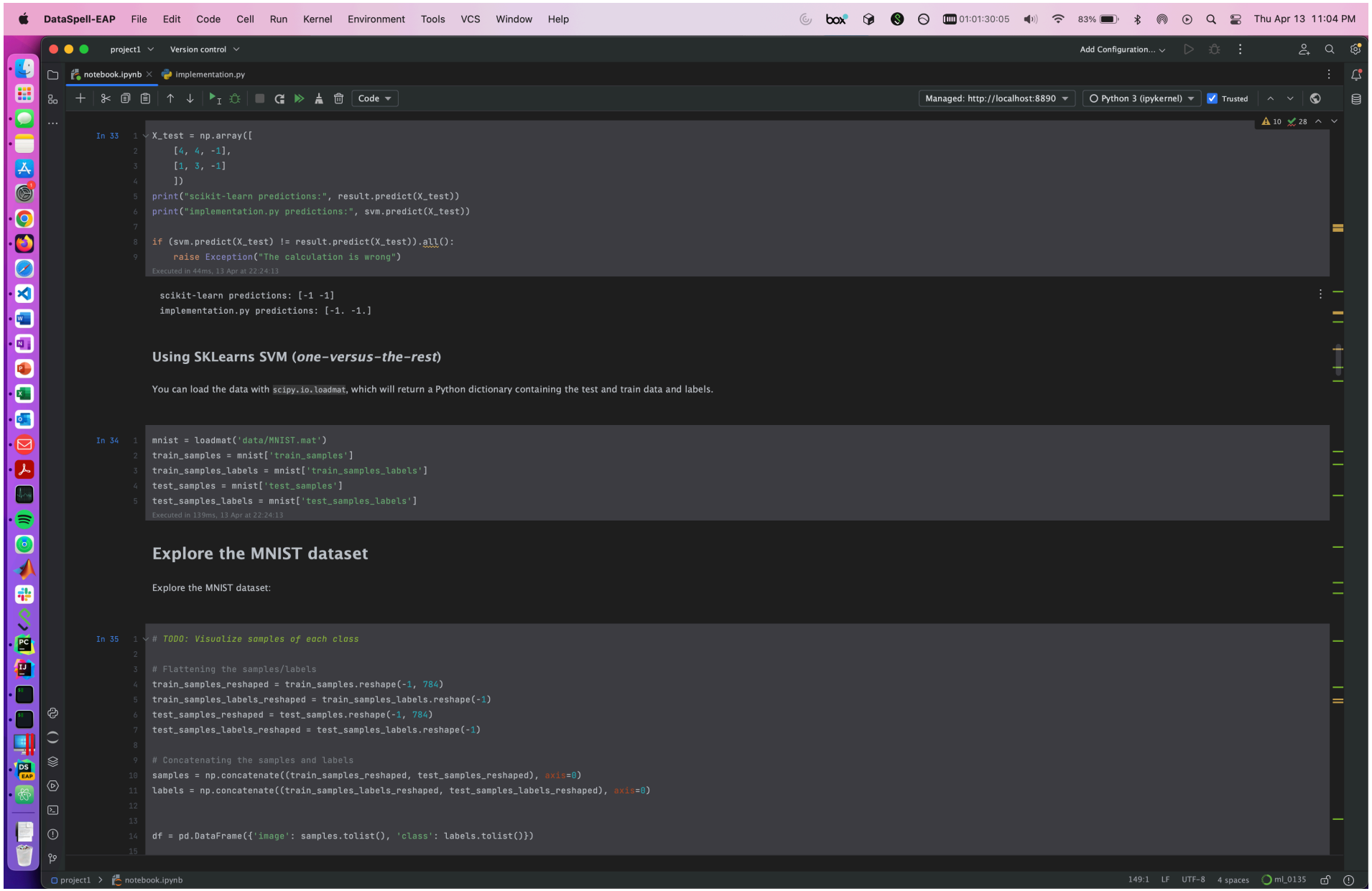

DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:03 PM
project1
Version control
Add Configuration...
notebook.ipynb
implementation.py
Markdown
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
In 26
import numpy as np
import pandas as pd
from scipy.io import loadmat
from implementation import SVM, linear_kernel, nonlinear_kernel
# from solution import SVM, linear_kernel, nonlinear_kernel
from sklearn.datasets import make_blobs
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import matplotlib
import matplotlib.pyplot as plt
%load_ext autoreload
%autoreload 2
Executed in 250ms, 13 Apr at 22:24:13
The autoreload extension is already loaded. To reload it, use:
%reload_ext autoreload
Step 1 - Get Data
In 27
# Input data - of the form [Bias term, x_1 value, x_2 value]
X = np.array([
[1, -2, 4,],
[1, 4, 1,],
[1, 1, 6,],
[1, 2, 4,],
[1, 6, 2,],
])
# Associated output labels - first 2 examples are labeled '-1' and last 3 are labeled '+1'
y = np.array([-1,-1,1,1,1])
# Let's plot these examples on a 2D graph!
# Plot the negative samples (the first 2)
plt.scatter(X[:,1][y==-1], X[:,2][y==-1], s=120, marker='_', linewidths=2)
# Plot the positive samples (the last 3)
plt.scatter(X[:,1][y==1], X[:,2][y==1], s=120, marker='+', linewidths=2)
# Print a possible hyperplane, that is separating the two classes.
# we'll two points and draw the line between them (naive guess)
plt.plot([-2,6],[6,0.5])
project1 > notebook.ipynb
30:1 LF UTF-8 4 spaces ml_0135

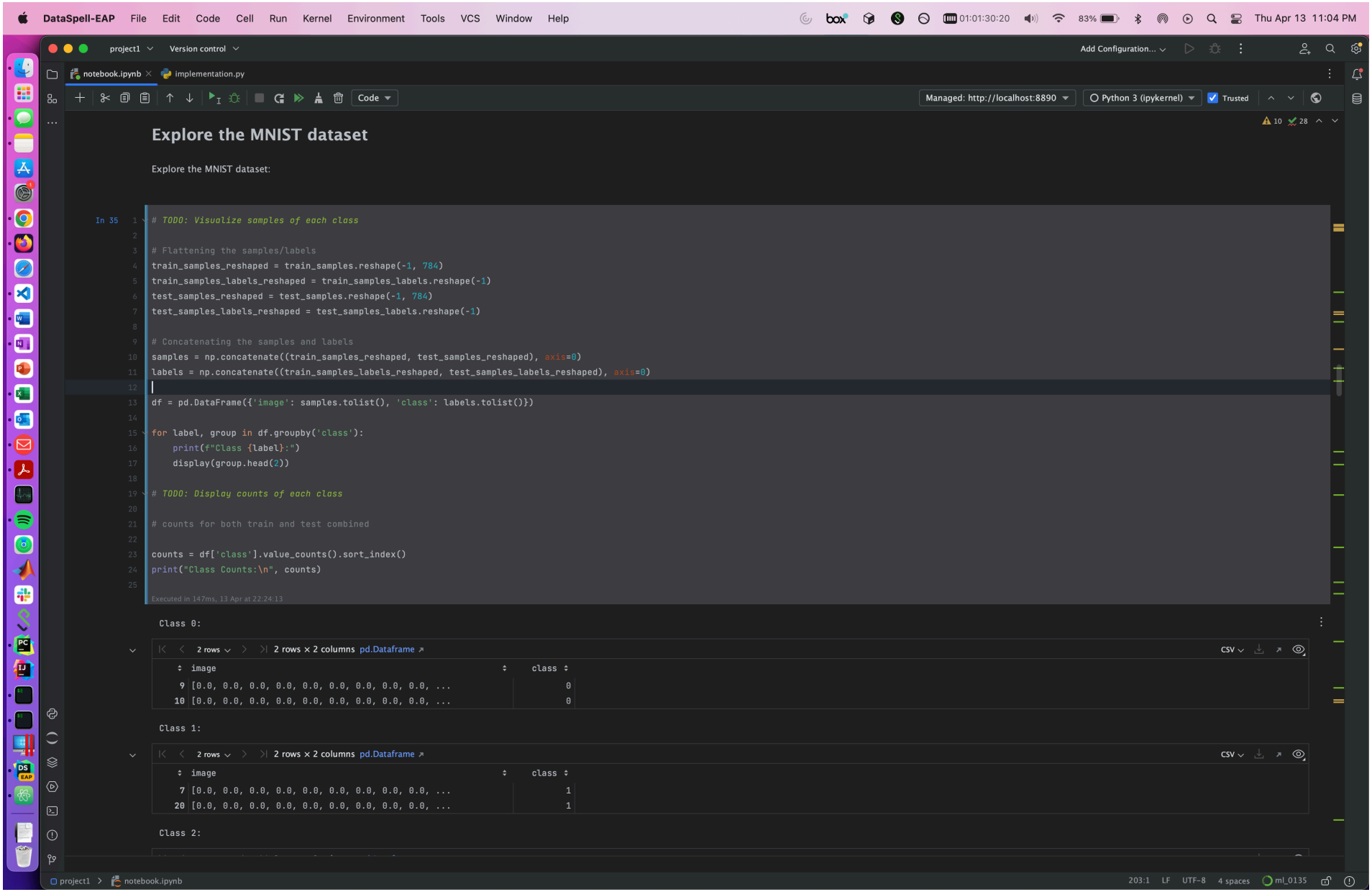
```
# Input data - of the form [Bias term, x_1 value, x_2 value]
X = np.array([
    [1, -2, 4,],
    [1, 4, 1,],
    [1, 1, 6,],
    [1, 2, 4,],
    [1, 6, 2,],
])

# Associated output labels - first 2 examples are labeled '-1' and last 3 are labeled '+1'
y = np.array([-1,-1,1,1,1])

# Let's plot these examples on a 2D graph!
# Plot the negative samples (the first 2)
plt.scatter(X[:,1][y==-1], X[:,2][y==-1], s=120, marker='_', linewidths=2)
# Plot the positive samples (the last 3)
plt.scatter(X[:,1][y==1], X[:,2][y==1], s=120, marker='+', linewidths=2)

# Print a possible hyperplane, that is separating the two classes.
# we'll two points and draw the line between them (naive guess)
plt.plot([-2,6],[6,0.5])
plt.xlabel(r"$x_1$")
plt.ylabel(r"$x_2$")
plt.show()
```

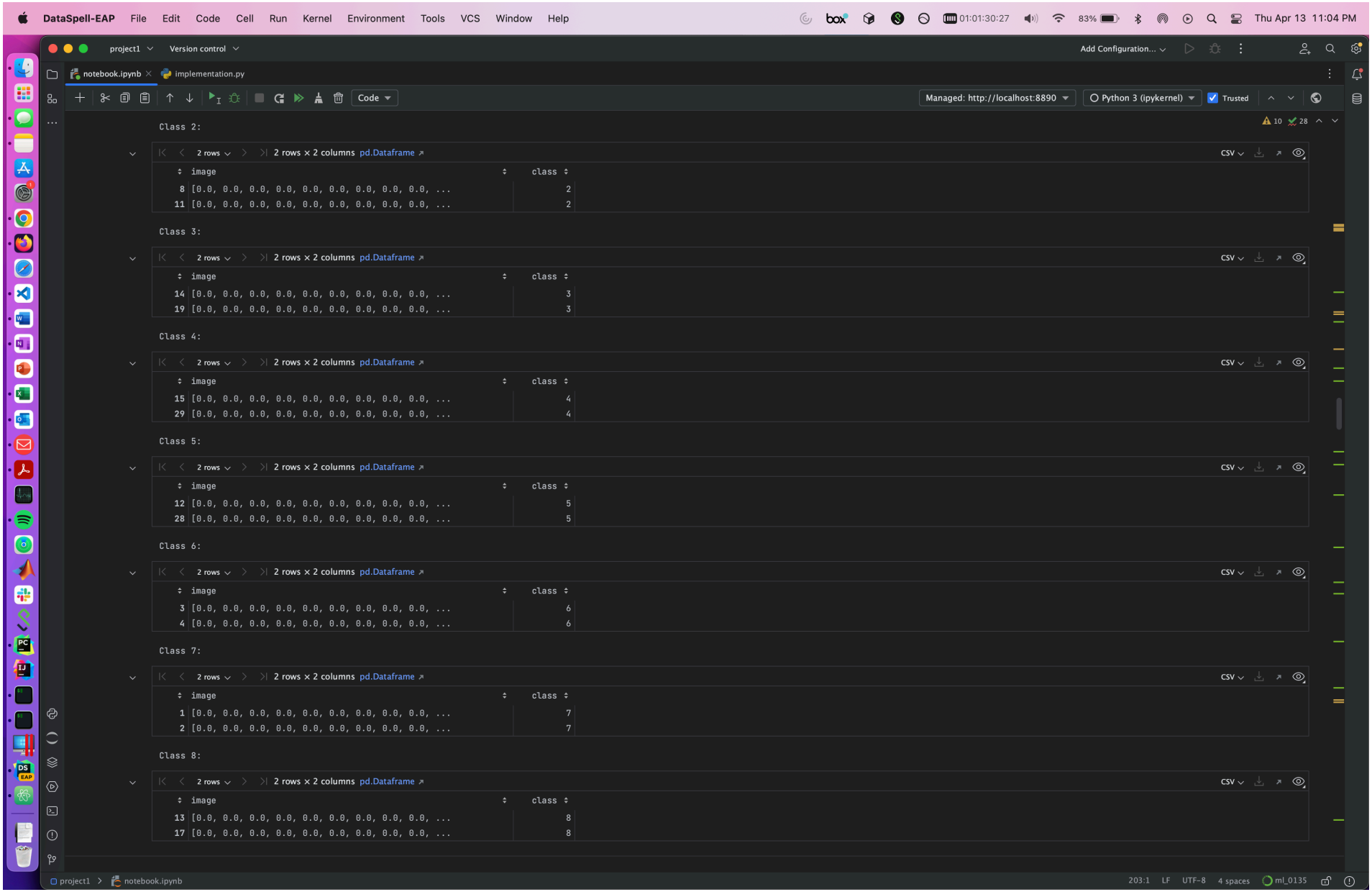


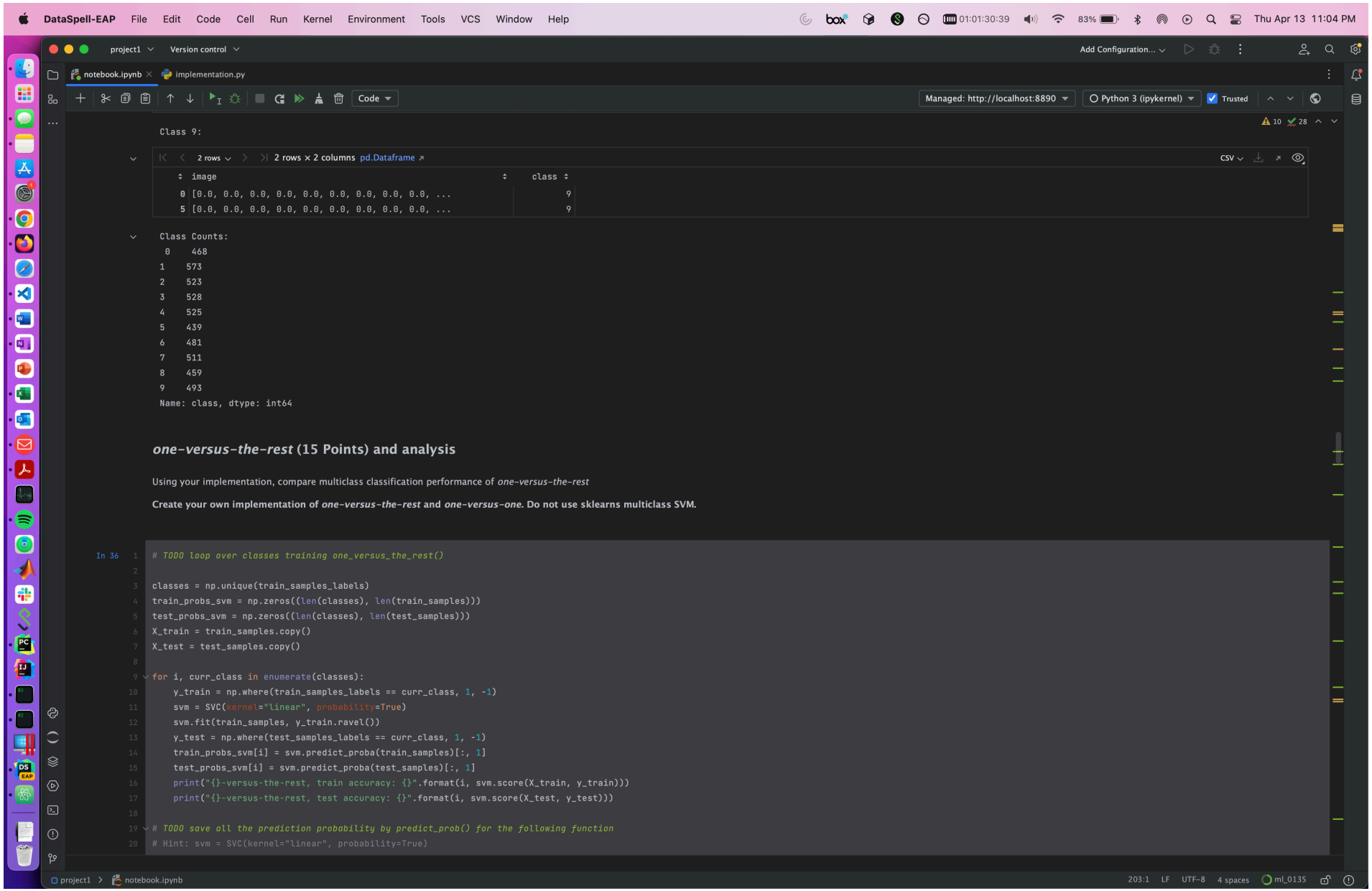

DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:03 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Markdown
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
SVM basics
SVM using scikit-learn.
In 28
result = SVC(kernel="linear")
result.fit(X, y.ravel())
print("scikit-learn indices of support vectors:", result.support_)
Executed in 144ms, 13 Apr at 22:24:13
scikit-learn indices of support vectors: [0 1 3 4]
Implement and test SVM to sklearn's version (20 points)
Compare the indices of support vectors from scikit-lean with implementation.py using toy data.
In 29
# TODO: implement SVM, along with linear_kernel
result = SVC(kernel="linear")
result.fit(X, y)
print("scikit-learn indices of support vectors:", result.support_)
svm = SVM(kernel=linear_kernel)
svm.fit(X, y)
Executed in 124ms, 13 Apr at 22:24:13
scikit-learn indices of support vectors: [0 1 3 4]
message: Optimization terminated successfully
success: True
status: 0
fun: -0.6249999998988889
x: [1.475e-01 4.774e-01 1.468e-15 4.088e-01 2.162e-01]
nit: 6
jac: [-4.000e+00 -4.000e+00 5.500e+00 4.000e+00 4.000e+00]
nfev: 39
njev: 6
[1.47549954e-01 4.77448320e-01 1.46822658e-15 4.08827517e-01
2.16170757e-01]
[-1.44328993e-15 4.99986204e-01 1.00000345e+00]
-3.999982745681246
Out 29
<implementation.SVM at 0x296503e50>
project1 notebook.ipynb
30:1 LF UTF-8 4 spaces ml_0135

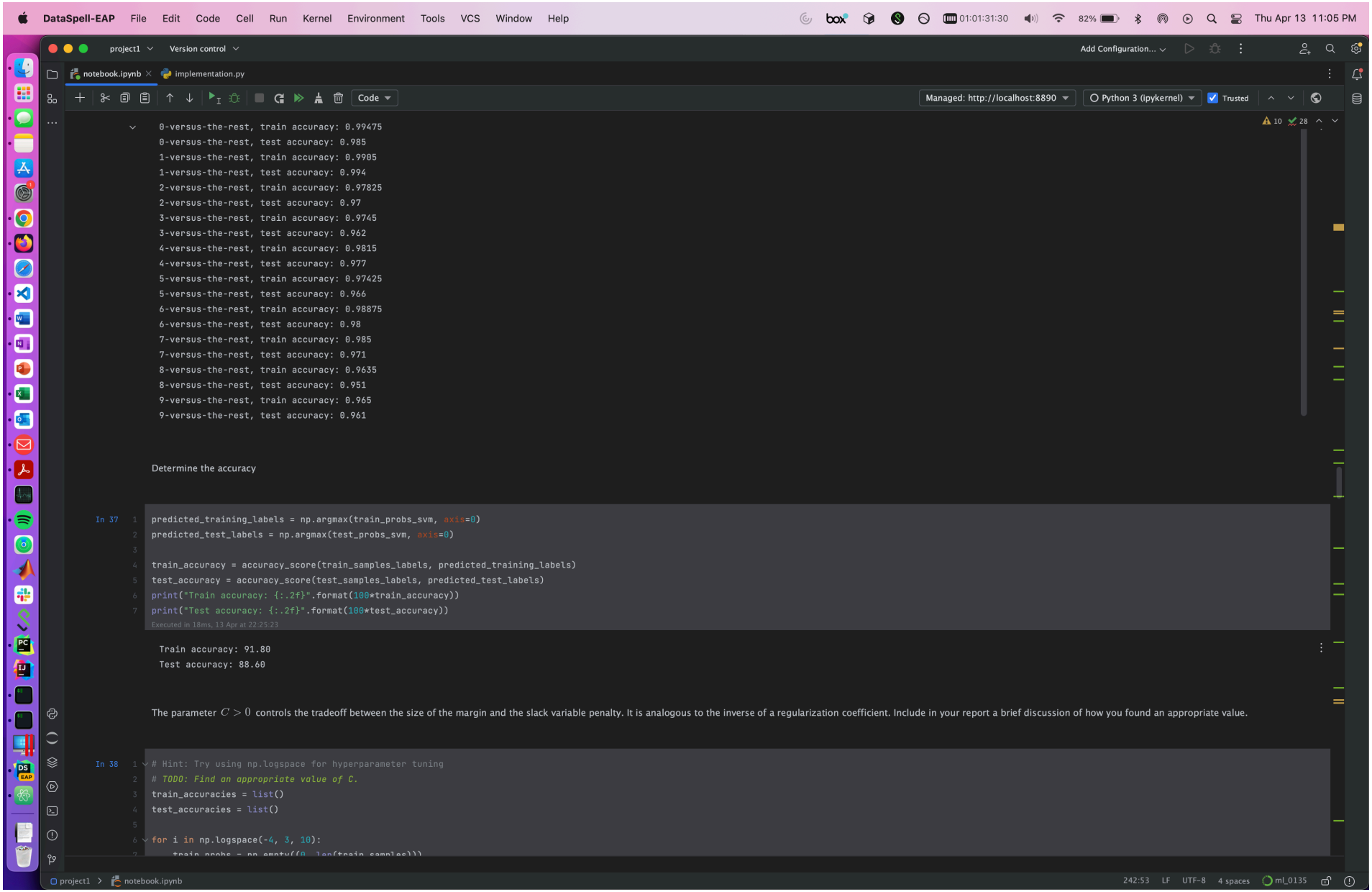

DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:03 PM
project1
Version control
Add Configuration...
notebook.ipynb
implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
In 30
print("implementation.py indices of support vectors:", \
np.array(range(y.shape[0]))[svm.a>1e-8])
if (result.support_ != np.array(range(y.shape[0]))[svm.a>1e-8]).all():
raise Exception("The calculation is wrong")
Executed in 101ms, 13 Apr at 22:24:13
implementation.py indices of support vectors: [0 1 3 4]
Compare the weights assigned to the features from scikit-lean with implementation.py.
In 31
#TODO - other sections were done for you, specify the variables to print, find the difference, and check it is within reasonable error from that of sklearn's version.
print("scikit-learn weights assigned to the features:", result.coef_)
print("implementation.py weights assigned to the features:", svm.w)
diff = abs(result.coef_ - svm.w)
if (diff > 1e-3).any():
raise Exception("The calculation is wrong")
Executed in 81ms, 13 Apr at 22:24:13
scikit-learn weights assigned to the features: [[0. 0.5 0.99969451]]
implementation.py weights assigned to the features: [-1.44328993e-15 4.99986204e-01 1.00000345e+00]
Compare the bias weight from scikit-lean with implementation.py.
In 32
print("scikit-learn bias weight:", result.intercept_)
print("implementation.py bias weight:", svm.b)
diff = abs(result.intercept_ - svm.b)
if (diff > 1e-3).all():
raise Exception("The calculation is wrong")
Executed in 62ms, 13 Apr at 22:24:13
scikit-learn bias weight: [-3.99915989]
implementation.py bias weight: -3.999982745681246
Compare the predictions from scikit-lean with implementation.py.
In 33
X_test = np.array([
[4, 4, -1],
[1, 3, -1]
])
project1 > notebook.ipynb
149:1 LF UTF-8 4 spaces ml_0135

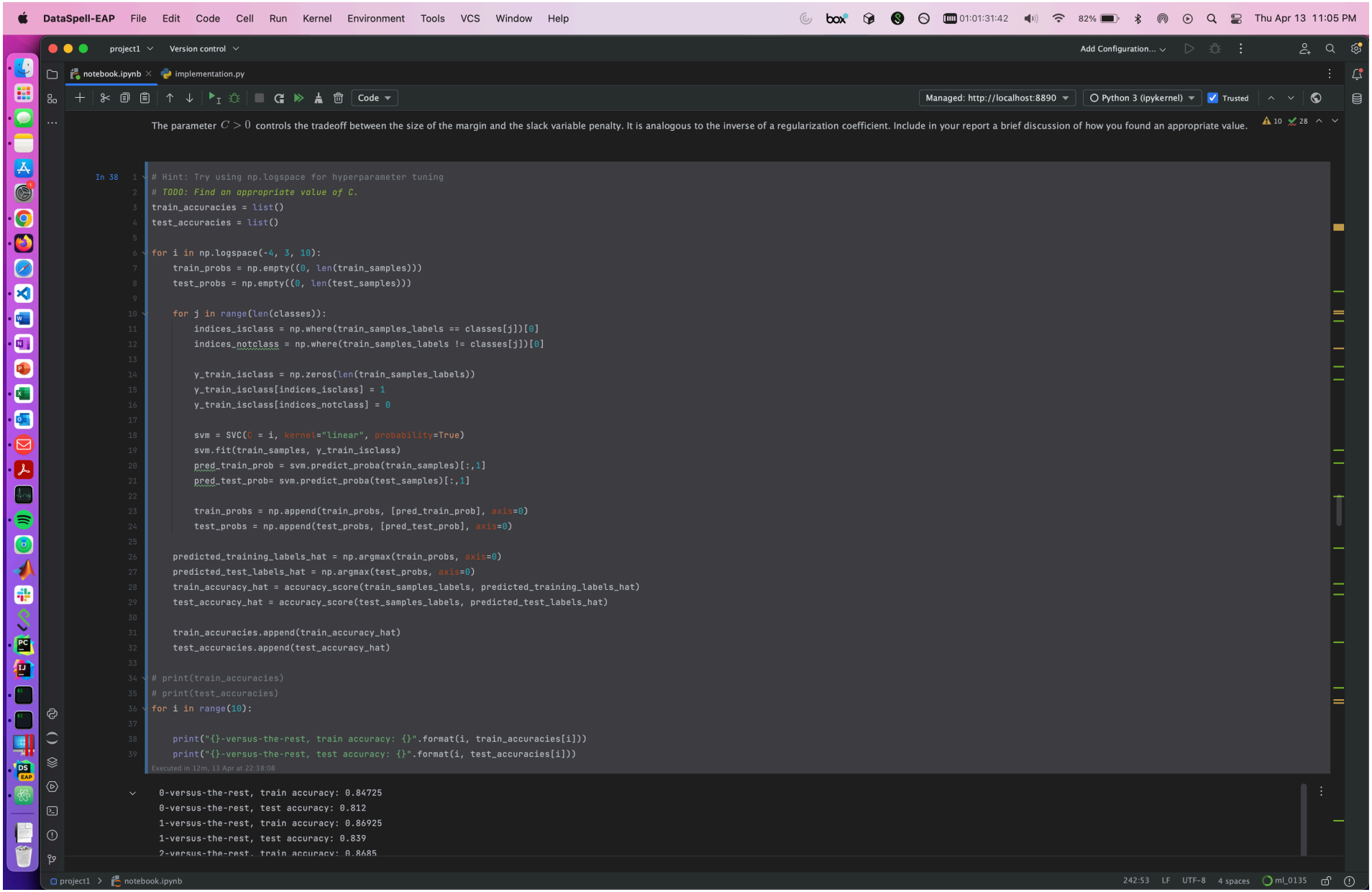

DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:04 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
In 33
X_test = np.array([
[4, 4, -1],
[1, 3, -1]
])
print("scikit-learn predictions:", result.predict(X_test))
print("implementation.py predictions:", svm.predict(X_test))
if (svm.predict(X_test) != result.predict(X_test)).all():
raise Exception("The calculation is wrong")
Executed in 44ms, 13 Apr at 22:24:13
scikit-learn predictions: [-1 -1]
implementation.py predictions: [-1. -1.]
Using SKLearns SVM (one-versus-the-rest)
You can load the data with scipy.io.loadmat, which will return a Python dictionary containing the test and train data and labels.
In 34
mnist = loadmat('data/MNIST.mat')
train_samples = mnist['train_samples']
train_samples_labels = mnist['train_samples_labels']
test_samples = mnist['test_samples']
test_samples_labels = mnist['test_samples_labels']
Executed in 139ms, 13 Apr at 22:24:13
Explore the MNIST dataset
Explore the MNIST dataset:
In 35
# TODO: Visualize samples of each class
# Flattening the samples/labels
train_samples_reshaped = train_samples.reshape(-1, 784)
train_samples_labels_reshaped = train_samples_labels.reshape(-1)
test_samples_reshaped = test_samples.reshape(-1, 784)
test_samples_labels_reshaped = test_samples_labels.reshape(-1)
# Concatenating the samples and labels
samples = np.concatenate((train_samples_reshaped, test_samples_reshaped), axis=0)
labels = np.concatenate((train_samples_labels_reshaped, test_samples_labels_reshaped), axis=0)
df = pd.DataFrame({'image': samples.tolist(), 'class': labels.tolist()})
project1 notebook.ipynb
149:1 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:04 PM
project1
Version control
Add Configuration...
notebook.ipynb
implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
Explore the MNIST dataset
Explore the MNIST dataset:
In 35
# TODO: Visualize samples of each class
# Flattening the samples/labels
train_samples_reshaped = train_samples.reshape(-1, 784)
train_samples_labels_reshaped = train_samples_labels.reshape(-1)
test_samples_reshaped = test_samples.reshape(-1, 784)
test_samples_labels_reshaped = test_samples_labels.reshape(-1)
# Concatenating the samples and labels
samples = np.concatenate((train_samples_reshaped, test_samples_reshaped), axis=0)
labels = np.concatenate((train_samples_labels_reshaped, test_samples_labels_reshaped), axis=0)
df = pd.DataFrame({'image': samples.tolist(), 'class': labels.tolist()})
for label, group in df.groupby('class'):
print(f"Class {label}:")
display(group.head(2))
# TODO: Display counts of each class
# counts for both train and test combined
counts = df['class'].value_counts().sort_index()
print("Class Counts:\n", counts)
Executed in 147ms, 13 Apr at 22:24:13
Class 0:
2 rows × 2 columns pd.Dataframe
CSV
image class
9 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 0
10 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 0
Class 1:
2 rows × 2 columns pd.Dataframe
CSV
image class
7 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 1
20 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 1
Class 2:
project1 notebook.ipynb
203:1 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:04 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
Class 2:
2 rows × 2 columns pd.Dataframe
CSV
image class
8 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 2
11 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 2
Class 3:
2 rows × 2 columns pd.Dataframe
CSV
image class
14 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 3
19 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 3
Class 4:
2 rows × 2 columns pd.Dataframe
CSV
image class
15 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 4
29 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 4
Class 5:
2 rows × 2 columns pd.Dataframe
CSV
image class
12 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 5
28 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 5
Class 6:
2 rows × 2 columns pd.Dataframe
CSV
image class
3 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 6
4 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 6
Class 7:
2 rows × 2 columns pd.Dataframe
CSV
image class
1 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 7
2 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 7
Class 8:
2 rows × 2 columns pd.Dataframe
CSV
image class
13 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 8
17 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ... 8
project1 notebook.ipynb
203:1 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:04 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
Class 9:
2 rows × 2 columns pd.Dataframe
CSV
image
class
0 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
9
5 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
9
Class Counts:
0 468
1 573
2 523
3 528
4 525
5 439
6 481
7 511
8 459
9 493
Name: class, dtype: int64
one-versus-the-rest (15 Points) and analysis
Using your implementation, compare multiclass classification performance of one-versus-the-rest
Create your own implementation of one-versus-the-rest and one-versus-one. Do not use sklearns multiclass SVM.
In 36
# TODO loop over classes training one_versus_the_rest()
classes = np.unique(train_samples_labels)
train_probs_svm = np.zeros((len(classes), len(train_samples)))
test_probs_svm = np.zeros((len(classes), len(test_samples)))
X_train = train_samples.copy()
X_test = test_samples.copy()
for i, curr_class in enumerate(classes):
y_train = np.where(train_samples_labels == curr_class, 1, -1)
svm = SVC(kernel="linear", probability=True)
svm.fit(train_samples, y_train.ravel())
y_test = np.where(test_samples_labels == curr_class, 1, -1)
train_probs_svm[i] = svm.predict_proba(train_samples)[:, 1]
test_probs_svm[i] = svm.predict_proba(test_samples)[:, 1]
print("{}-versus-the-rest, train accuracy: {}".format(i, svm.score(X_train, y_train)))
print("{}-versus-the-rest, test accuracy: {}".format(i, svm.score(X_test, y_test)))
# TODO save all the prediction probability by predict_prob() for the following function
# Hint: svm = SVC(kernel="linear", probability=True)
project1 notebook.ipynb
203:1 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:05 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
0-versus-the-rest, train accuracy: 0.99475
0-versus-the-rest, test accuracy: 0.985
1-versus-the-rest, train accuracy: 0.9905
1-versus-the-rest, test accuracy: 0.994
2-versus-the-rest, train accuracy: 0.97825
2-versus-the-rest, test accuracy: 0.97
3-versus-the-rest, train accuracy: 0.9745
3-versus-the-rest, test accuracy: 0.962
4-versus-the-rest, train accuracy: 0.9815
4-versus-the-rest, test accuracy: 0.977
5-versus-the-rest, train accuracy: 0.97425
5-versus-the-rest, test accuracy: 0.966
6-versus-the-rest, train accuracy: 0.98875
6-versus-the-rest, test accuracy: 0.98
7-versus-the-rest, train accuracy: 0.985
7-versus-the-rest, test accuracy: 0.971
8-versus-the-rest, train accuracy: 0.9635
8-versus-the-rest, test accuracy: 0.951
9-versus-the-rest, train accuracy: 0.965
9-versus-the-rest, test accuracy: 0.961
Determine the accuracy
In 37
predicted_training_labels = np.argmax(train_probs_svm, axis=0)
predicted_test_labels = np.argmax(test_probs_svm, axis=0)
train_accuracy = accuracy_score(train_samples_labels, predicted_training_labels)
test_accuracy = accuracy_score(test_samples_labels, predicted_test_labels)
print("Train accuracy: {:.2f}".format(100*train_accuracy))
print("Test accuracy: {:.2f}".format(100*test_accuracy))
Executed in 18ms, 13 Apr at 22:25:23
Train accuracy: 91.80
Test accuracy: 88.60
The parameter $C > 0$ controls the tradeoff between the size of the margin and the slack variable penalty. It is analogous to the inverse of a regularization coefficient. Include in your report a brief discussion of how you found an appropriate value.
In 38
# Hint: Try using np.logspace for hyperparameter tuning
# TODO: Find an appropriate value of C.
train_accuracies = list()
test_accuracies = list()
for i in np.logspace(-4, 3, 10):
project1 notebook.ipynb
242:53 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:05 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
The parameter $C > 0$ controls the tradeoff between the size of the margin and the slack variable penalty. It is analogous to the inverse of a regularization coefficient. Include in your report a brief discussion of how you found an appropriate value.
In 38
# Hint: Try using np.logspace for hyperparameter tuning
# TODO: Find an appropriate value of C.
train_accuracies = list()
test_accuracies = list()
for i in np.logspace(-4, 3, 10):
train_probs = np.empty((0, len(train_samples)))
test_probs = np.empty((0, len(test_samples)))
for j in range(len(classes)):
indices_isclass = np.where(train_samples_labels == classes[j])[0]
indices_notclass = np.where(train_samples_labels != classes[j])[0]
y_train_isclass = np.zeros(len(train_samples_labels))
y_train_isclass[indices_isclass] = 1
y_train_isclass[indices_notclass] = 0
svm = SVC(C = i, kernel="linear", probability=True)
svm.fit(train_samples, y_train_isclass)
pred_train_prob = svm.predict_proba(train_samples)[:,1]
pred_test_prob= svm.predict_proba(test_samples)[:,1]
train_probs = np.append(train_probs, [pred_train_prob], axis=0)
test_probs = np.append(test_probs, [pred_test_prob], axis=0)
predicted_training_labels_hat = np.argmax(train_probs, axis=0)
predicted_test_labels_hat = np.argmax(test_probs, axis=0)
train_accuracy_hat = accuracy_score(train_samples_labels, predicted_training_labels_hat)
test_accuracy_hat = accuracy_score(test_samples_labels, predicted_test_labels_hat)
train_accuracies.append(train_accuracy_hat)
test_accuracies.append(test_accuracy_hat)
# print(train_accuracies)
# print(test_accuracies)
for i in range(10):
print("{}-versus-the-rest, train accuracy: {}".format(i, train_accuracies[i]))
print("{}-versus-the-rest, test accuracy: {}".format(i, test_accuracies[i]))
Executed in 12m, 13 Apr at 22:38:08
0-versus-the-rest, train accuracy: 0.84725
0-versus-the-rest, test accuracy: 0.812
1-versus-the-rest, train accuracy: 0.86925
1-versus-the-rest, test accuracy: 0.839
2-versus-the-rest, train accuracy: 0.8685
project1 notebook.ipynb
242:53 LF UTF-8 4 spaces ml_0135

```
0-versus-the-rest, train accuracy: 0.84725
0-versus-the-rest, test accuracy: 0.812
1-versus-the-rest, train accuracy: 0.86925
1-versus-the-rest, test accuracy: 0.839
2-versus-the-rest, train accuracy: 0.8685
2-versus-the-rest, test accuracy: 0.842
3-versus-the-rest, train accuracy: 0.86875
3-versus-the-rest, test accuracy: 0.841
4-versus-the-rest, train accuracy: 0.88125
4-versus-the-rest, test accuracy: 0.853
5-versus-the-rest, train accuracy: 0.9115
5-versus-the-rest, test accuracy: 0.881
6-versus-the-rest, train accuracy: 0.938
6-versus-the-rest, test accuracy: 0.894
7-versus-the-rest, train accuracy: 0.965
7-versus-the-rest, test accuracy: 0.881
8-versus-the-rest, train accuracy: 0.98525
8-versus-the-rest, test accuracy: 0.841
9-versus-the-rest, train accuracy: 0.99725
9-versus-the-rest, test accuracy: 0.824
```

Provide details on how you found an appropriate value.

We are examining the C value from a range between $10^{-4 \text{ to } 10}3$. This iterates through 10 classes to predict the highest probability by using the argmax. Then, we see the accuracies of these methods from the accuracy_score function. These metrics are stored in train and test accuracies and eventually the code shows a comparison of the accuracies.

Plot accuracies for train and test using logspace for x-axis (i.e., C values)

In 39

```python
# TODO: Plot the result.

import matplotlib.pyplot as plt

C_values = np.logspace(-4, 3, 10)

plt.plot(C_values, train_accuracies, label="Accuracy on train")
plt.plot(C_values, test_accuracies, label="Accuracy on test")

plt.gca().axes.set_xscale("log")

plt.axvline(x=1, color='green', linestyle='--')
```

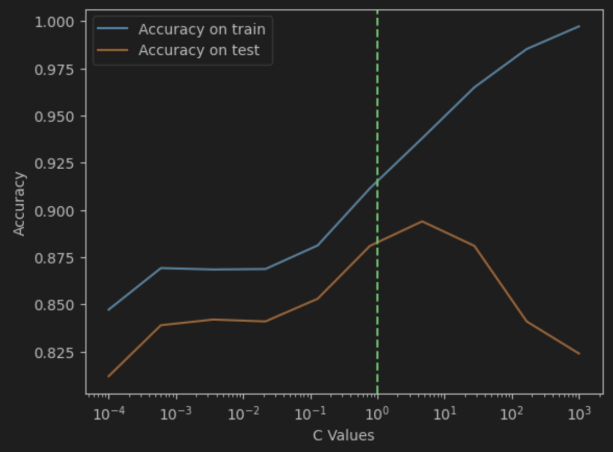
for i in range(10)

Plot accuracies for train and test using logspace for x-axis (i.e., C values)

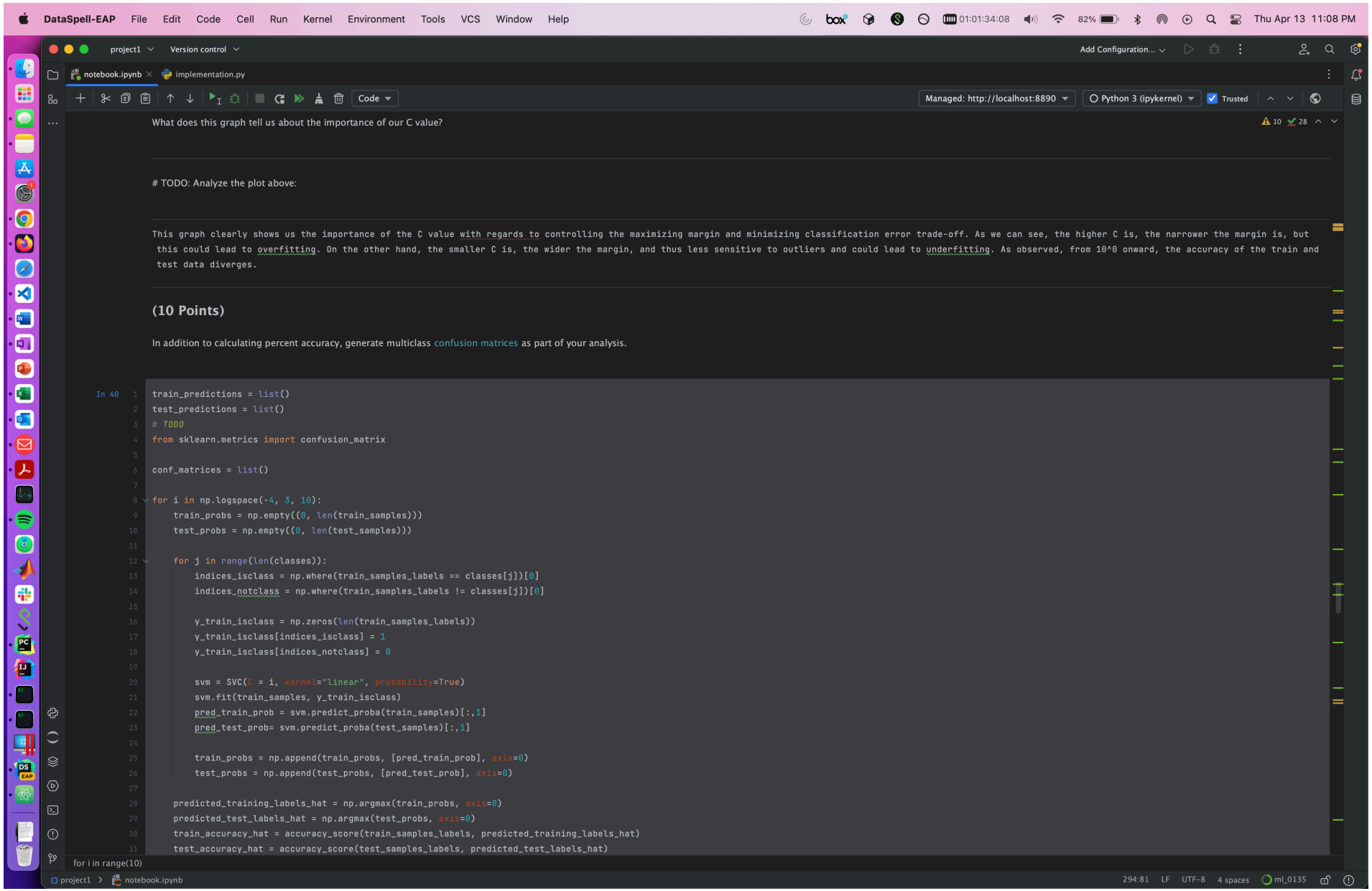
In 39

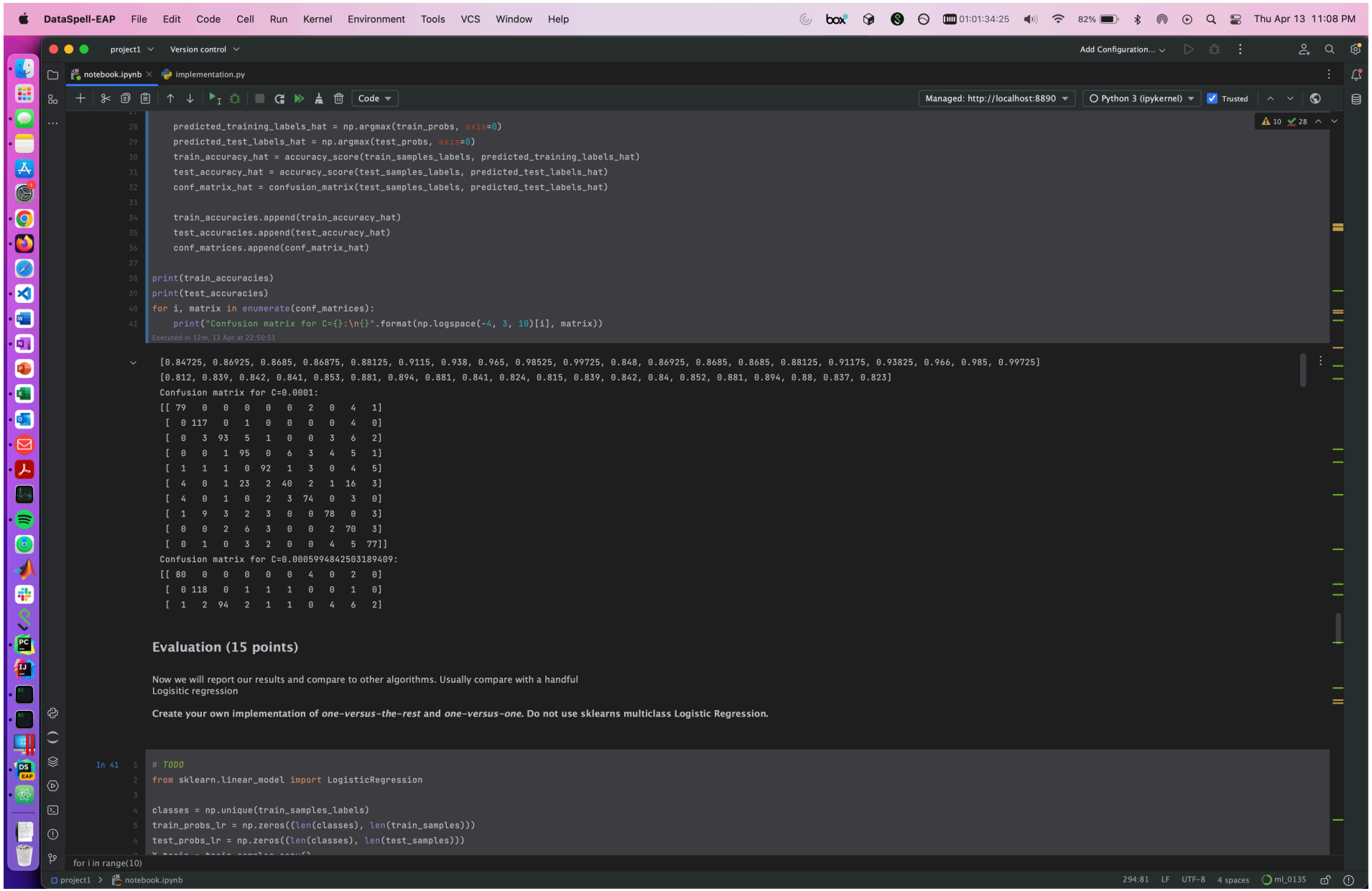
```python
# TODO: Plot the result.

import matplotlib.pyplot as plt

C_values = np.logspace(-4, 3, 10)

plt.plot(C_values, train_accuracies, label="Accuracy on train")
plt.plot(C_values, test_accuracies, label="Accuracy on test")

plt.gca().axes.set_xscale("log")

plt.axvline(x=1, color='green', linestyle='--')

plt.legend()

plt.xlabel("C Values")
plt.ylabel("Accuracy")

plt.show()
```

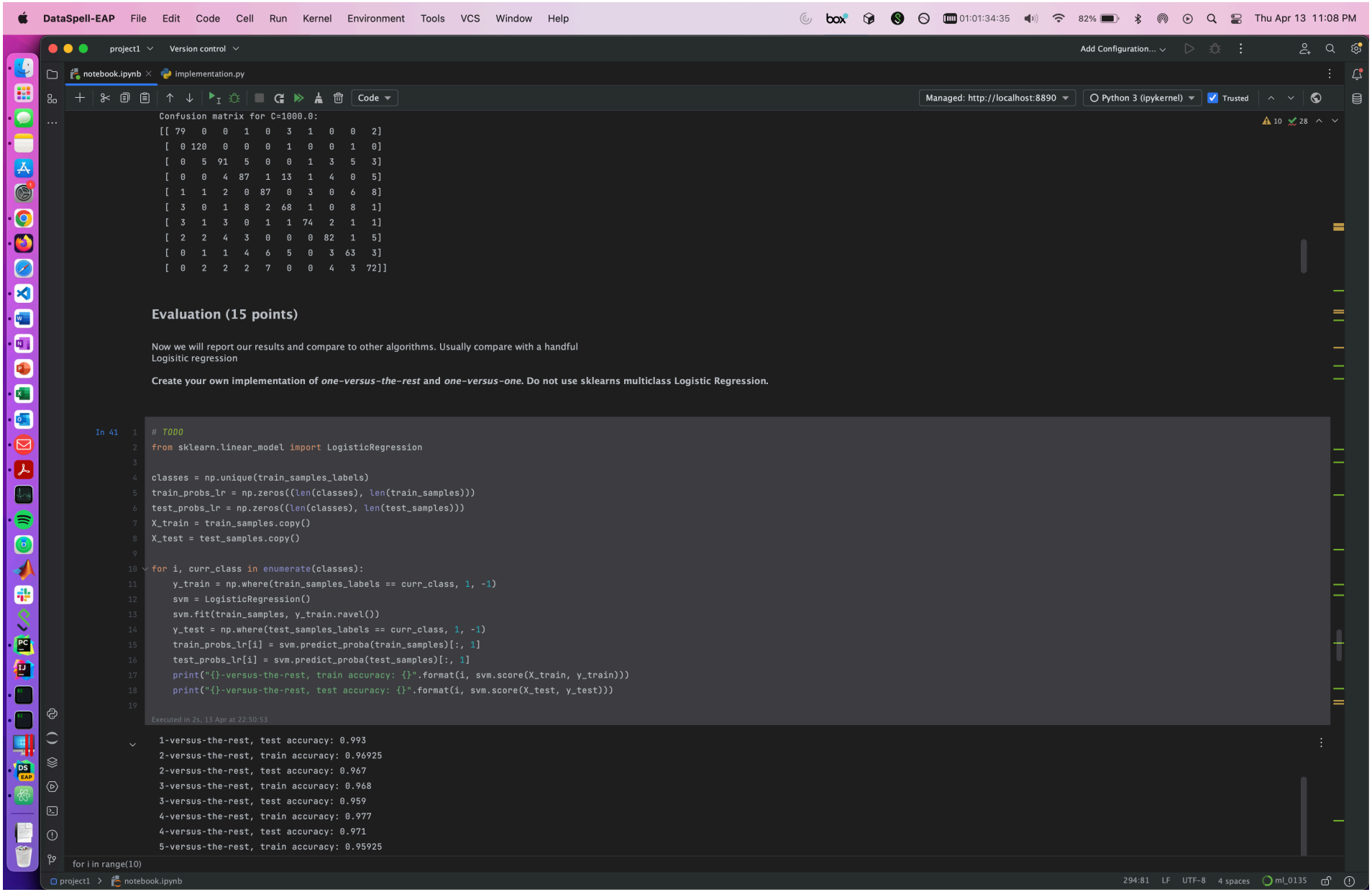
Executed in 252ms, 13 Apr at 22:38:09



for i in range(10)


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:08 PM
project1
Version control
Add Configuration...
notebook.ipynb
implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
What does this graph tell us about the importance of our C value?
TODO: Analyze the plot above:
This graph clearly shows us the importance of the C value with regards to controlling the maximizing margin and minimizing classification error trade-off. As we can see, the higher C is, the narrower the margin is, but this could lead to overfitting. On the other hand, the smaller C is, the wider the margin, and thus less sensitive to outliers and could lead to underfitting. As observed, from 10^0 onward, the accuracy of the train and test data diverges.
(10 Points)
In addition to calculating percent accuracy, generate multiclass confusion matrices as part of your analysis.
In 40
train_predictions = list()
test_predictions = list()
# TODO
from sklearn.metrics import confusion_matrix
conf_matrices = list()
for i in np.logspace(-4, 3, 10):
train_probs = np.empty((0, len(train_samples)))
test_probs = np.empty((0, len(test_samples)))
for j in range(len(classes)):
indices_isclass = np.where(train_samples_labels == classes[j])[0]
indices_notclass = np.where(train_samples_labels != classes[j])[0]
y_train_isclass = np.zeros(len(train_samples_labels))
y_train_isclass[indices_isclass] = 1
y_train_isclass[indices_notclass] = 0
svm = SVC(C = i, kernel="linear", probability=True)
svm.fit(train_samples, y_train_isclass)
pred_train_prob = svm.predict_proba(train_samples)[:,1]
pred_test_prob= svm.predict_proba(test_samples)[:,1]
train_probs = np.append(train_probs, [pred_train_prob], axis=0)
test_probs = np.append(test_probs, [pred_test_prob], axis=0)
predicted_training_labels_hat = np.argmax(train_probs, axis=0)
predicted_test_labels_hat = np.argmax(test_probs, axis=0)
train_accuracy_hat = accuracy_score(train_samples_labels, predicted_training_labels_hat)
test_accuracy_hat = accuracy_score(test_samples_labels, predicted_test_labels_hat)
for i in range(10)
project1 > notebook.ipynb
294:81 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:08 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
predicted_training_labels_hat = np.argmax(train_probs, axis=0)
predicted_test_labels_hat = np.argmax(test_probs, axis=0)
train_accuracy_hat = accuracy_score(train_samples_labels, predicted_training_labels_hat)
test_accuracy_hat = accuracy_score(test_samples_labels, predicted_test_labels_hat)
conf_matrix_hat = confusion_matrix(test_samples_labels, predicted_test_labels_hat)
train_accuracies.append(train_accuracy_hat)
test_accuracies.append(test_accuracy_hat)
conf_matrices.append(conf_matrix_hat)
print(train_accuracies)
print(test_accuracies)
for i, matrix in enumerate(conf_matrices):
print("Confusion matrix for C={}:\n{}".format(np.logspace(-4, 3, 10)[i], matrix))
Executed in 12m, 13 Apr at 22:50:51
[0.84725, 0.86925, 0.8685, 0.86875, 0.88125, 0.9115, 0.938, 0.965, 0.98525, 0.99725, 0.848, 0.86925, 0.8685, 0.8685, 0.88125, 0.91175, 0.93825, 0.966, 0.985, 0.99725]
[0.812, 0.839, 0.842, 0.841, 0.853, 0.881, 0.894, 0.881, 0.841, 0.824, 0.815, 0.839, 0.842, 0.84, 0.852, 0.881, 0.894, 0.88, 0.837, 0.823]
Confusion matrix for C=0.0001:
[[79 0 0 0 0 0 2 0 4 1]
[0 117 0 1 0 0 0 0 4 0]
[0 3 93 5 1 0 0 3 6 2]
[0 0 1 95 0 6 3 4 5 1]
[1 1 1 0 92 1 3 0 4 5]
[4 0 1 23 2 40 2 1 16 3]
[4 0 1 0 2 3 74 0 3 0]
[1 9 3 2 3 0 0 78 0 3]
[0 0 2 6 3 0 0 2 70 3]
[0 1 0 3 2 0 0 4 5 77]]
Confusion matrix for C=0.0005994842503189409:
[[80 0 0 0 0 0 4 0 2 0]
[0 118 0 1 1 1 0 0 1 0]
[1 2 94 2 1 1 0 4 6 2]
Evaluation (15 points)
Now we will report our results and compare to other algorithms. Usually compare with a handful
Logisitic regression
Create your own implementation of one-versus-the-rest and one-versus-one. Do not use sklearns multiclass Logistic Regression.
In 41
# TODO
from sklearn.linear_model import LogisticRegression
classes = np.unique(train_samples_labels)
train_probs_lr = np.zeros((len(classes), len(train_samples)))
test_probs_lr = np.zeros((len(classes), len(test_samples)))
for i in range(10)
project1 notebook.ipynb
294:81 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:08 PM
project1
Version control
Add Configuration...
notebook.ipynb
implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
Confusion matrix for C=1000.0:
[[79 0 0 1 0 3 1 0 0 2]
[0 120 0 0 0 1 0 0 1 0]
[0 5 91 5 0 0 1 3 5 3]
[0 0 4 87 1 13 1 4 0 5]
[1 1 2 0 87 0 3 0 6 8]
[3 0 1 8 2 68 1 0 8 1]
[3 1 3 0 1 1 74 2 1 1]
[2 2 4 3 0 0 0 82 1 5]
[0 1 1 4 6 5 0 3 63 3]
[0 2 2 2 7 0 0 4 3 72]]
Evaluation (15 points)
Now we will report our results and compare to other algorithms. Usually compare with a handful
Logisitic regression
Create your own implementation of one-versus-the-rest and one-versus-one. Do not use sklearns multiclass Logistic Regression.
In 41
# TODO
from sklearn.linear_model import LogisticRegression
classes = np.unique(train_samples_labels)
train_probs_lr = np.zeros((len(classes), len(train_samples)))
test_probs_lr = np.zeros((len(classes), len(test_samples)))
X_train = train_samples.copy()
X_test = test_samples.copy()
for i, curr_class in enumerate(classes):
y_train = np.where(train_samples_labels == curr_class, 1, -1)
svm = LogisticRegression()
svm.fit(train_samples, y_train.ravel())
y_test = np.where(test_samples_labels == curr_class, 1, -1)
train_probs_lr[i] = svm.predict_proba(train_samples)[:, 1]
test_probs_lr[i] = svm.predict_proba(test_samples)[:, 1]
print("{}-versus-the-rest, train accuracy: {}".format(i, svm.score(X_train, y_train)))
print("{}-versus-the-rest, test accuracy: {}".format(i, svm.score(X_test, y_test)))
Executed in 2s, 13 Apr at 22:50:53
1-versus-the-rest, test accuracy: 0.993
2-versus-the-rest, train accuracy: 0.96925
2-versus-the-rest, test accuracy: 0.967
3-versus-the-rest, train accuracy: 0.968
3-versus-the-rest, test accuracy: 0.959
4-versus-the-rest, train accuracy: 0.977
4-versus-the-rest, test accuracy: 0.971
5-versus-the-rest, train accuracy: 0.95925
for i in range(10)
project1 > notebook.ipynb
294:81 LF UTF-8 4 spaces ml_0135

```
0-versus-the-rest, train accuracy: 0.99125
0-versus-the-rest, test accuracy: 0.985
1-versus-the-rest, train accuracy: 0.98725
1-versus-the-rest, test accuracy: 0.993
2-versus-the-rest, train accuracy: 0.96925
2-versus-the-rest, test accuracy: 0.967
3-versus-the-rest, train accuracy: 0.968
3-versus-the-rest, test accuracy: 0.959
4-versus-the-rest, train accuracy: 0.977
4-versus-the-rest, test accuracy: 0.971
5-versus-the-rest, train accuracy: 0.95925
5-versus-the-rest, test accuracy: 0.945
6-versus-the-rest, train accuracy: 0.9815
6-versus-the-rest, test accuracy: 0.976
7-versus-the-rest, train accuracy: 0.98825
7-versus-the-rest, test accuracy: 0.969
8-versus-the-rest, train accuracy: 0.9525
8-versus-the-rest, test accuracy: 0.946
9-versus-the-rest, train accuracy: 0.95475
9-versus-the-rest, test accuracy: 0.961
```

Create a table comparing model accuracy on train and test data.

Method	Train Acc. (%)	Test Acc. (%)
0	99.125	98.5
1	98.725	99.3
2	96.925	96.7
3	96.8	95.9
4	97.7	97.1
5	95.925	94.5
6	98.15	97.6
7	98.025	96.9
8	95.25	94.6
9	95.475	96.1

Create 9 graphs (one for each label) with two ROC curves (one for each model).

DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help

Thu Apr 13 11:08 PM

project1 Version control

Add Configuration...

notebook.ipynb implementation.py

Code

Managed: http://localhost:8890 Python 3 (ipykernel) Trusted

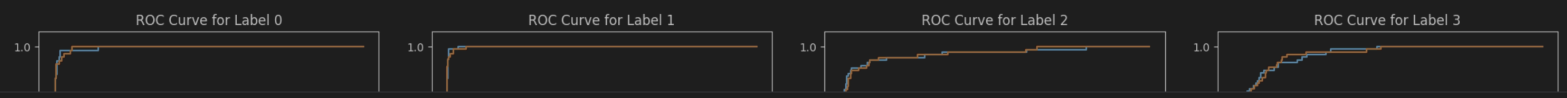
10 28

Create 9 graphs (one for each label) with two ROC curves (one for each model).

In 42

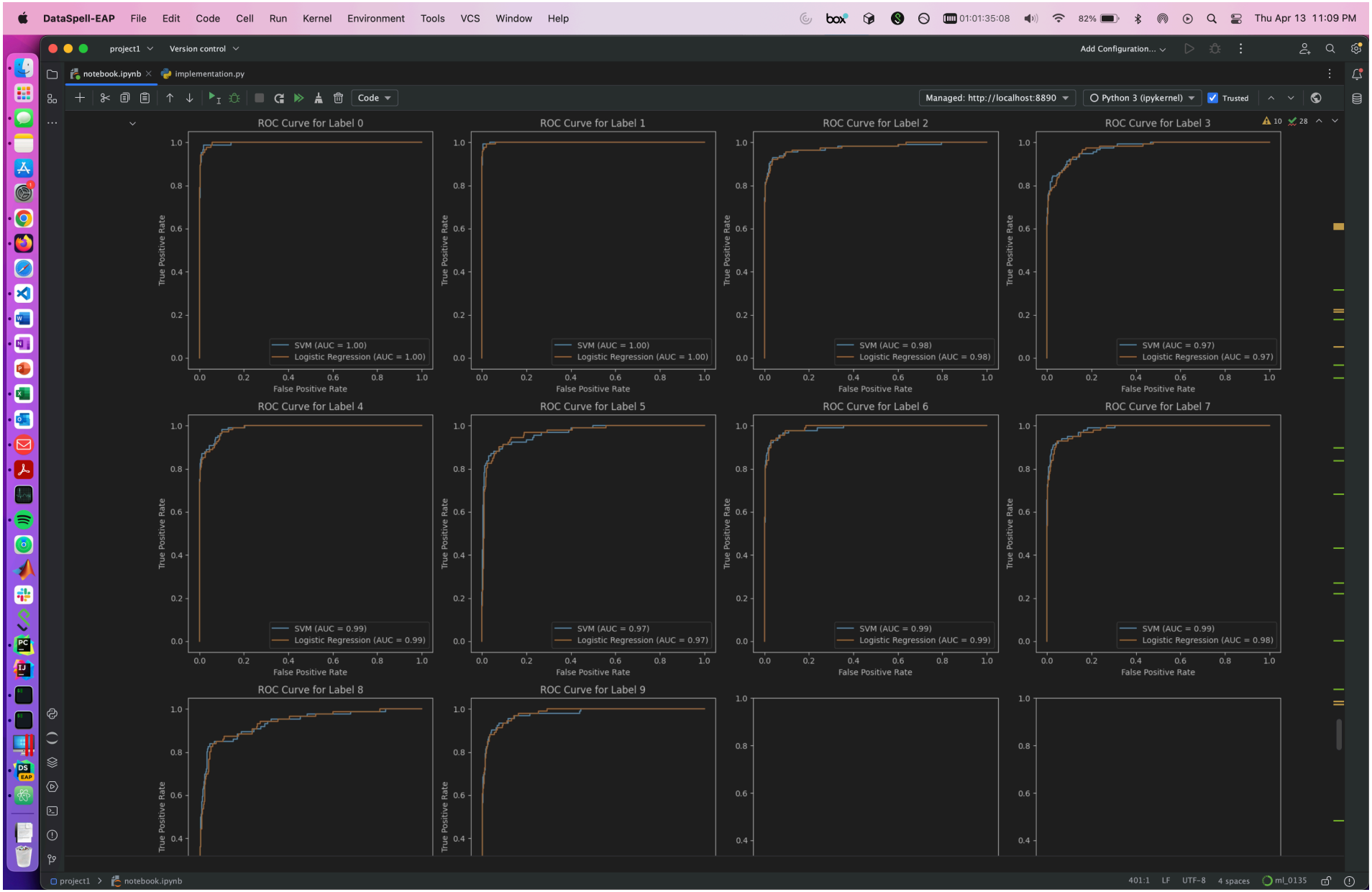
```python
# TODO
from sklearn.metrics import roc_curve, auc

# Define the classes
classes = np.unique(train_samples_labels)

# Create subplots for each label
fig, axs = plt.subplots(3, 4, figsize=(20, 15))
axs = axs.ravel()
threshold = 0.0
# Iterate over each class
for i, curr_class in enumerate(classes):

    # Extract the true labels for the current class
    y_true = np.where(test_samples_labels == curr_class, 1, 0)

    # Calculate the ROC curve and AUC for SVM model
    fpr_svm, tpr_svm, threshold = roc_curve(y_true, test_probs_svm[i])
    auc_svm = auc(fpr_svm, tpr_svm)

    # Calculate the ROC curve and AUC for logistic regression model
    fpr_lr, tpr_lr, threshold = roc_curve(y_true, test_probs_lr[i])
    auc_lr = auc(fpr_lr, tpr_lr)

    # Plot the ROC curves
    axs[i].plot(fpr_svm, tpr_svm, label='SVM (AUC = {:.2f})'.format(auc_svm))
    axs[i].plot(fpr_lr, tpr_lr, label='Logistic Regression (AUC = {:.2f})'.format(auc_lr))

    # Set the axis labels and title
    axs[i].set_xlabel('False Positive Rate')
    axs[i].set_ylabel('True Positive Rate')
    axs[i].set_title('ROC Curve for Label {}'.format(curr_class))

    # Add the legend
    axs[i].legend()

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

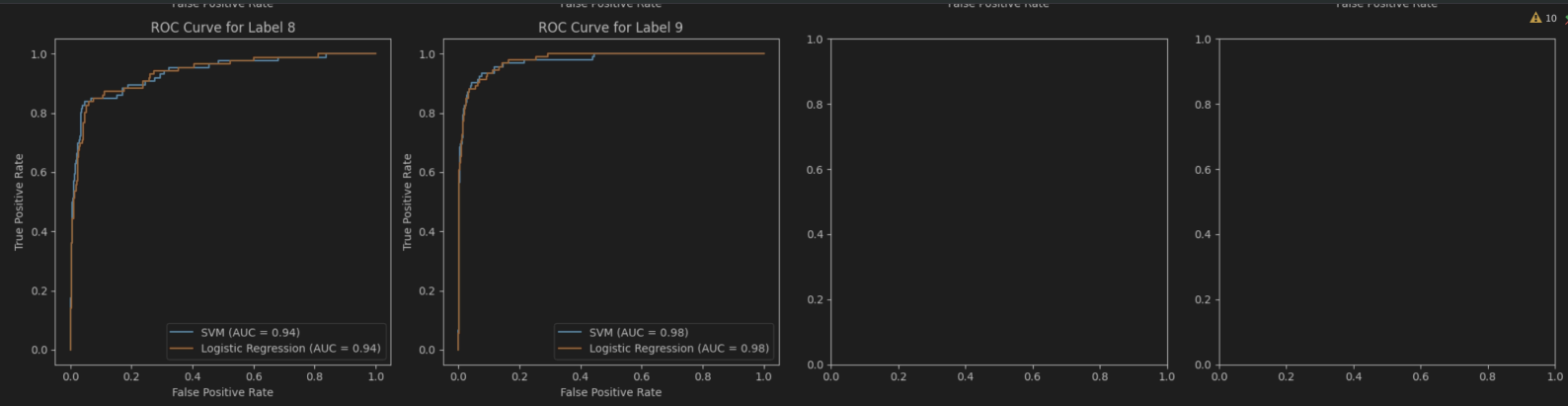
Executed in 1s, 13 Apr at 22:50:54



project1 > notebook.ipynb

401:1 LF UTF-8 4 spaces ml_0135


DataSpell-EAP File Edit Code Cell Run Kernel Environment Tools VCS Window Help
Thu Apr 13 11:09 PM
project1 Version control
Add Configuration...
notebook.ipynb implementation.py
Code
Managed: http://localhost:8890
Python 3 (ipykernel)
Trusted
ROC Curve for Label 0
SVM (AUC = 1.00)
Logistic Regression (AUC = 1.00)
ROC Curve for Label 1
SVM (AUC = 1.00)
Logistic Regression (AUC = 1.00)
ROC Curve for Label 2
SVM (AUC = 0.98)
Logistic Regression (AUC = 0.98)
ROC Curve for Label 3
SVM (AUC = 0.97)
Logistic Regression (AUC = 0.97)
ROC Curve for Label 4
SVM (AUC = 0.99)
Logistic Regression (AUC = 0.99)
ROC Curve for Label 5
SVM (AUC = 0.97)
Logistic Regression (AUC = 0.97)
ROC Curve for Label 6
SVM (AUC = 0.99)
Logistic Regression (AUC = 0.99)
ROC Curve for Label 7
SVM (AUC = 0.99)
Logistic Regression (AUC = 0.98)
ROC Curve for Label 8
ROC Curve for Label 9
True Positive Rate
False Positive Rate
project1 notebook.ipynb
401:1 LF UTF-8 4 spaces ml_0135



BONUS (+5 points): Non-linear kernel

Intuition Behind Kernels

The SVM classifier obtained by solving the convex Lagrange dual of the primal max-margin SVM formulation is as follows:

$$ f(x) = \sum_{i=1}^N\alpha_i\cdot y_i\cdot\mathrm{K}(x,x_i)+b,$$

where N is the number of support vectors.

If you know the intuition behind a linear discriminant function, the non-parametric SVM classifier above is very easy to understand. Instead of imagining the original features of each data point, consider a transformation to a new feature space where the data point has N features, one for each support vector. The value of the i^{th} feature is equal to the value of the kernel between the i^{th} support vector and the data point is classified. The original (possibly non-linear) SVM classifier is like any other linear discriminant in this space.

Note that after the transformation, the original features of the data point are irrelevant. Its dot products with support vectors (special data points chosen by the SVM optimization algorithm) represent it only. One of my professors used a loose analogy while explaining this idea: A person has seen lakes, rivers, streams, fords, etc., but has never seen the sea. How would you explain to this person what a sea is? By relating the amount of water in an ocean to that found in a water body, the person already knows, etc.

In some instances, like the RBF kernel, defining the transformed features in terms of the original features of a data point leads to an infinite-dimensional representation. Unfortunately, though this an awe-inspiring fact often mentioned while explaining how powerful SVMs are, it drops in only after repeated encounters with the idea ranging from introductory machine learning to statistical learning theory.

Intuition Behind Gaussian Kernels

The Gaussian/RBF kernel is as follows:

$$ \mathrm{K}(x,y)=exp(-\frac{||x-y||^2}{2\sigma^2})$$

Like any other kernel, we can understand the RBF kernel regarding feature transformation via the dot products given above. However, the intuition that helps best when analyzing the RBF kernel is that of the Gaussian