j887zeo78

February 23, 2024

1 Programming Problem: Model Comparison among Logistic Regression, Linear SVM and Gaussian Kernel SVM

In this problem, you are expected to implement and compare three models: (1) logistic regression, (2) a linear SVM, (3) a Gaussian kernel SVM. In this assignment, you are allowed to use the sklearn packages.

Data: we provide binary classification dataset camel_train.cvs and camel_test.cvs for training and testing respectively (these files are both in hw3_data_files.zip). The model is trained on training set, you are supposed to report accuracy both on the training set and on the test set. Each sample consists of input features $x^{(i)} \in \mathbb{R}^2$ and class lables $y^{(i)} \in \{-1, 1\}$. Note that here the labels are $\{-1, +1\}$ valued rather than $\{0, 1\}$ valued for convenience.

Part (a) [**5 points**] Visualize the training data in 2D space in Sec 1.1 and estimate or guess based on visualization which classifier you think will be most suitable for the dataset, and explain why. Write down this estimation {**before**} **you start your implementation and testing, and this is what you'll also turn in. Note that you will not be penalized for getting the wrong answer here, the main thing we would like to see is your prediction and your clear justification why.**

Part (b) [**10 points**] **Implement a function in Sec 1.2 that can learn the Logistic Regression, Linear SVM, Gaussian kernel SVM models by setting the appropriate model type.**

Part (c) [**5 points**] **Report the accuracy of each model and explain why one method is superior to the others (that is, fill in Sec 1.3).**

Part (d) [**5 points**] **Discuss how the decision boundary of each model looks like in 2D space and why. Please be very clear and precise in your discussion.**

Part (d) [**10 points**] **Fill in the code to visualize the decision boundary of the learned model in Sec 1.4. Do they follow your assumptions and guesses above? If so, explain why the experiment confirmed your hypothesis. If the models did not behave consistently with your expectations explain why, and most importantly, explain how your hypotheses were adjusted and critically what you learned from this experiment.**

[1]:
```
# Import Modules
import sklearn
import pandas as pd
import numpy as np
```

```
from sklearn import datasets
import matplotlib.pyplot as plt
```

[2]:
```
# Load dataset
# Read in the csv
df_train=pd.read_csv('camel_train.csv', encoding='utf-8')
df_test = pd.read_csv('camel_test.csv', encoding='utf-8')
# Difference between white rating and black rating - independent variable
df_train.head()
```

[2]:

	0.566992416097	0.84059391133	-1.0
0	0.297289	0.791696	-1.0
1	1.030462	0.429955	1.0
2	0.311769	0.867175	-1.0
3	0.883618	0.397090	1.0
4	0.362195	0.880017	1.0

[3]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99 entries, 0 to 98
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   0.566992416097  99 non-null     float64
 1   0.84059391133   99 non-null     float64
 2   -1.0            99 non-null     float64
dtypes: float64(3)
memory usage: 2.4 KB
```

[4]:
```
df_train.describe()
```

[4]:

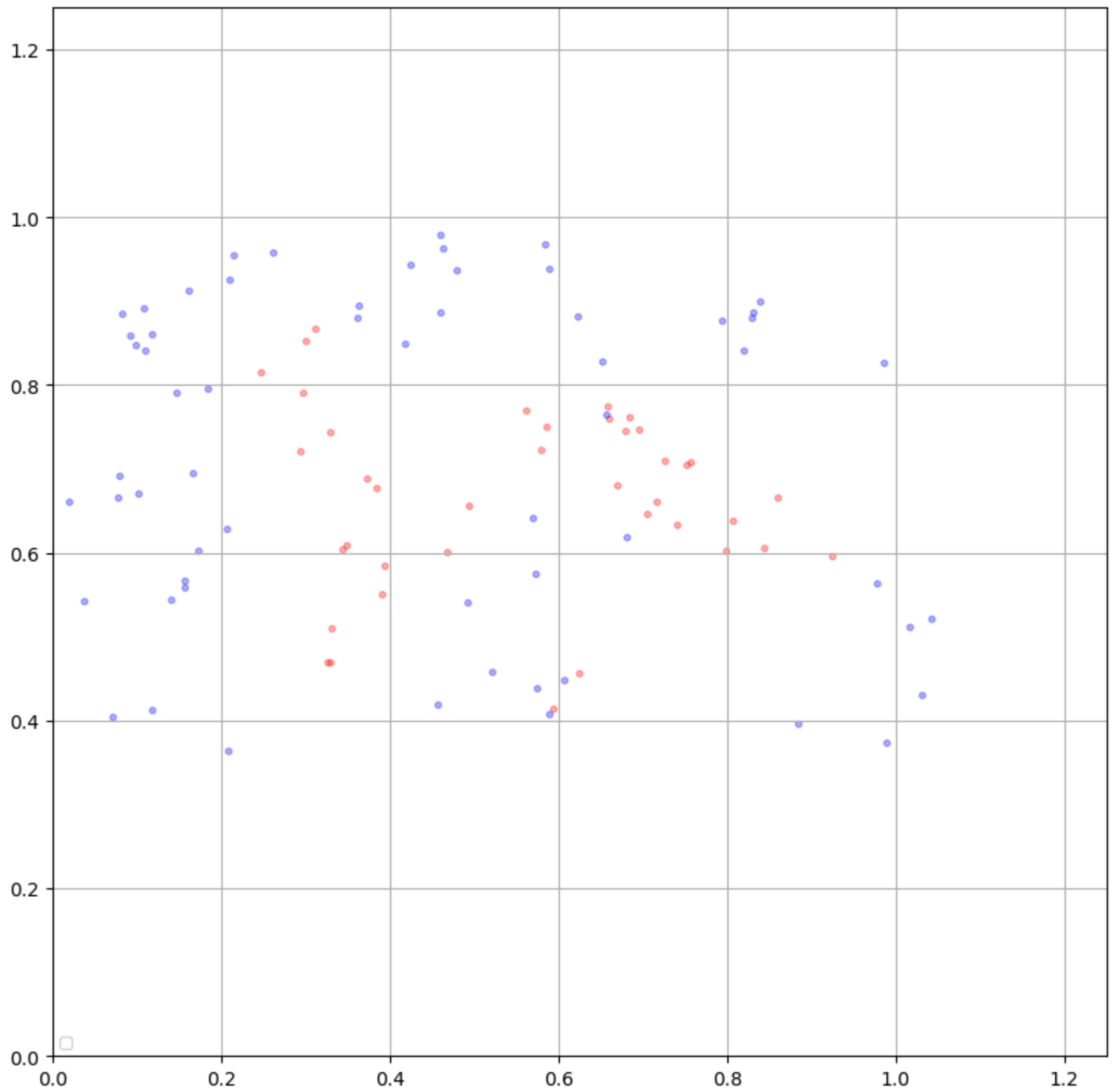
	0.566992416097	0.84059391133	-1.0
count	99.000000	99.000000	99.000000
mean	0.482068	0.694767	0.212121
std	0.277511	0.168909	0.982217
min	0.019830	0.363592	-1.000000
25%	0.230922	0.570406	-1.000000
50%	0.467246	0.691494	1.000000
75%	0.682227	0.848954	1.000000
max	1.041663	0.979701	1.000000

[5]:
```
# Select rating difference and turns as feature to predict the label
# training set
x_train = df_train.iloc[:, [0,1]].to_numpy()
y_train = df_train.iloc[:, 2].to_numpy()
```

```
# testing set
x_test = df_test.iloc[:, [0,1]].to_numpy()
y_test = df_test.iloc[:, 2].to_numpy()
```

[16]:
```
#@title Part (a)
# visualize training set
# TODO
fig, ax = plt.subplots(figsize=(10, 10))
num_samples, num_features = x_train.shape
for i in range(num_samples):
    if y_train[i] == -1 :
        ax.scatter(x_train[i][0], x_train[i][1], c='r', s=10.0, alpha=0.3)
    else :
        ax.scatter(x_train[i][0], x_train[i][1], c='b', s=10.0, alpha=0.3)
ax.grid()
ax.legend(('r', 'b'),
           ('y = -1', 'y = +1'),
           scatterpoints=1,
           loc='lower left',
           ncol=3,
           fontsize=8)
# Set x/y axis limits
ax.set_xlim([0, 1.25])
ax.set_ylim([0, 1.25])
fig.show()
```

```
/var/folders/c9/3q_3q36n4k9_t4gnwrhyd7s80000gn/T/ipykernel_16449/3689731530.py:1
2: UserWarning: Legend does not support handles for str instances.
A proxy artist may be used instead.
See:
https://matplotlib.org/stable/users/explain/axes/legend_guide.html#controlling-
the-legend-entries
  ax.legend(('r', 'b'),
/var/folders/c9/3q_3q36n4k9_t4gnwrhyd7s80000gn/T/ipykernel_16449/3689731530.py:2
1: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
  fig.show()
```



[27]:
```
#@title Part (b) implement
# Train logistic regression and print accuracy
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report

# define training procedure for logistic regression and svm
def training(x_train, y_train, x_test, y_test, model_type = 'logistic',␣
 ↪kernel=None):
  # specify the model used to learn by model_type
```

```
  # model_type == 'logistic' -> using logistic regression model
    if model_type == 'logistic':
        model = LogisticRegression(max_iter=200)
        model.fit(x_train, y_train)
        y_pred_test = model.predict(x_test)
        y_pred_train = model.predict(x_train)
        acc_test = accuracy_score(y_test, y_pred_test)
        acc_train = accuracy_score(y_train, y_pred_train)
  # model_type == 'svm' --> using SVM model
    elif model_type == 'svm':
  # kernel == None  --> using linear kenel for SVM
        if kernel == None:
            model = LinearSVC(max_iter=10000)
            model.fit(x_train, y_train)
            y_pred_test = model.predict(x_test)
            y_pred_train = model.predict(x_train)
            acc_test = accuracy_score(y_test, y_pred_test)
            acc_train = accuracy_score(y_train, y_pred_train)
  # kernel == 'rbf' --> using gaussian kernel for SVM
        elif kernel == 'rbf' :
            model = SVC(kernel='rbf', gamma='scale')
            model.fit(x_train, y_train)
            y_pred_test = model.predict(x_test)
            y_pred_train = model.predict(x_train)
            acc_test = accuracy_score(y_test, y_pred_test)
            acc_train = accuracy_score(y_train, y_pred_train)
  # define models
    return model, acc_train, acc_test
```

[28]:

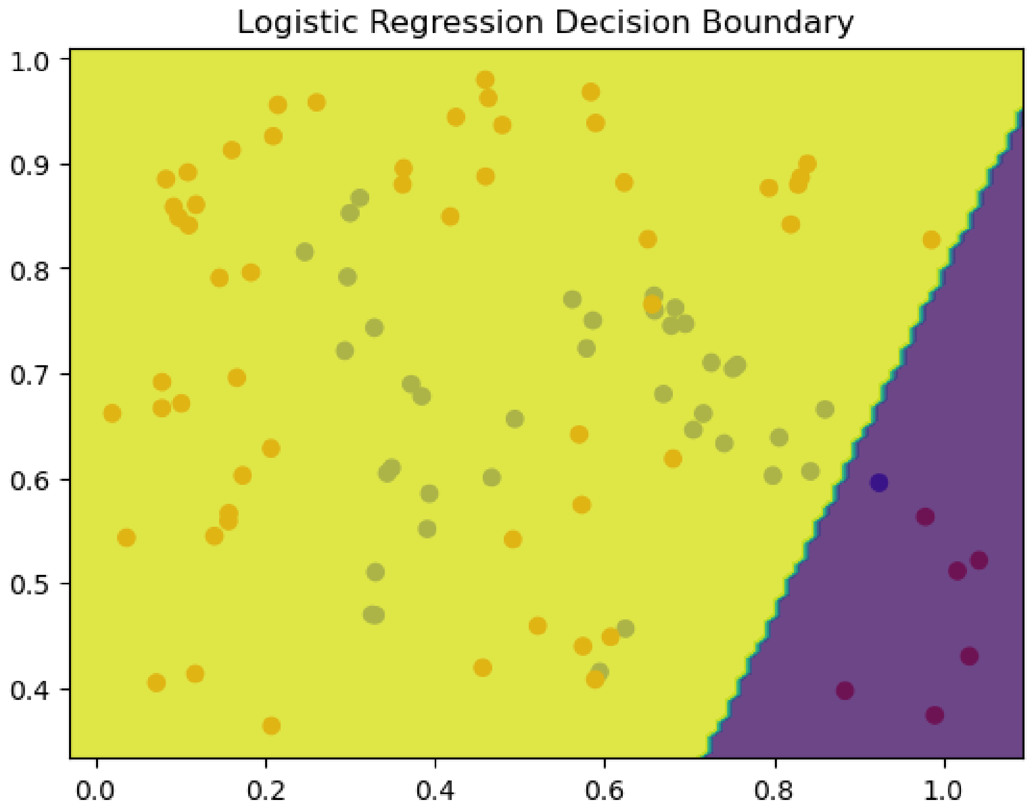
```
#@title Part (c)
# train logistic regression
# TODO
(model_log, acc_train_log, acc_test_log) = training(x_train, y_train, x_test,␣
 ↪y_test, model_type = 'logistic', kernel=None)
print("Logistic Training Accuracy : " + str(acc_train_log))
print("Logistic Testing Accuracy : " + str(acc_test_log))
# train linear SVM
# TODO
(model_lsvm, acc_train_lsvm, acc_test_lsvm) = training(x_train, y_train,␣
 ↪x_test, y_test, model_type = 'svm', kernel=None)
print("Linear SVM Training Accuracy : " + str(acc_train_lsvm))
print("Linear SVM Testing Accuracy : " + str(acc_test_lsvm))

# train Gaussian SVM
# TODO
(model_gsvm, acc_train_gsvm, acc_test_gsvm) = training(x_train, y_train,␣
 ↪x_test, y_test, model_type = 'svm', kernel='rbf')
```

```
print("Gaussian SVM Training Accuracy : " + str(acc_train_gsvm))
print("Gaussian SVM Testing Accuracy : " + str(acc_test_gsvm))
# Report accuracies of the above three
```

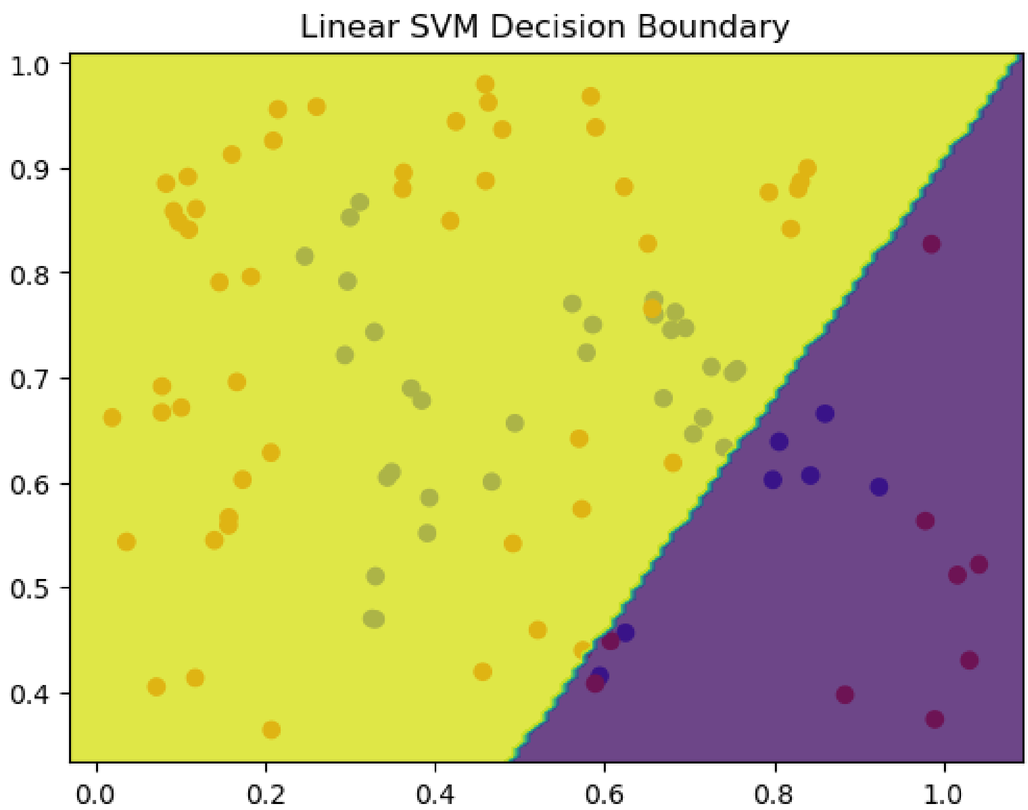
```
Logistic Training Accuracy : 0.5555555555555556
Logistic Testing Accuracy : 0.531328320802005
Linear SVM Training Accuracy : 0.5858585858585859
Linear SVM Testing Accuracy : 0.581453634085213
Gaussian SVM Training Accuracy : 0.8585858585858586
Gaussian SVM Testing Accuracy : 0.7192982456140351
```

[39]:
```
# Part (d) plot decision boundary for learned models, add discussion in your␣
↪writeup.
class_colors = {-1: 'b', 1: 'r'}
def plot_decision_boundary(model, x, y, title='', output_path=None,
        file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'svm' ):
    colors = y > 0 if class_colors is None else [class_colors[c] for c in y]
    fig = plt.figure()
    plt.scatter(x[:, 0], x[:, 1], c=colors)

    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xx = np.linspace(xlim[0], xlim[1], 100)
    yy = np.linspace(ylim[0], ylim[1], 100)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = model.predict(xy)
    Z = Z.reshape(XX.shape)
    # TODO
    plt.contourf(XX, YY, Z, alpha=0.8)
    plt.title(title)
    plt.show()
    if output_path is not None and file_name is not None:
        fig.savefig(os.path.join(output_path, file_name))
```

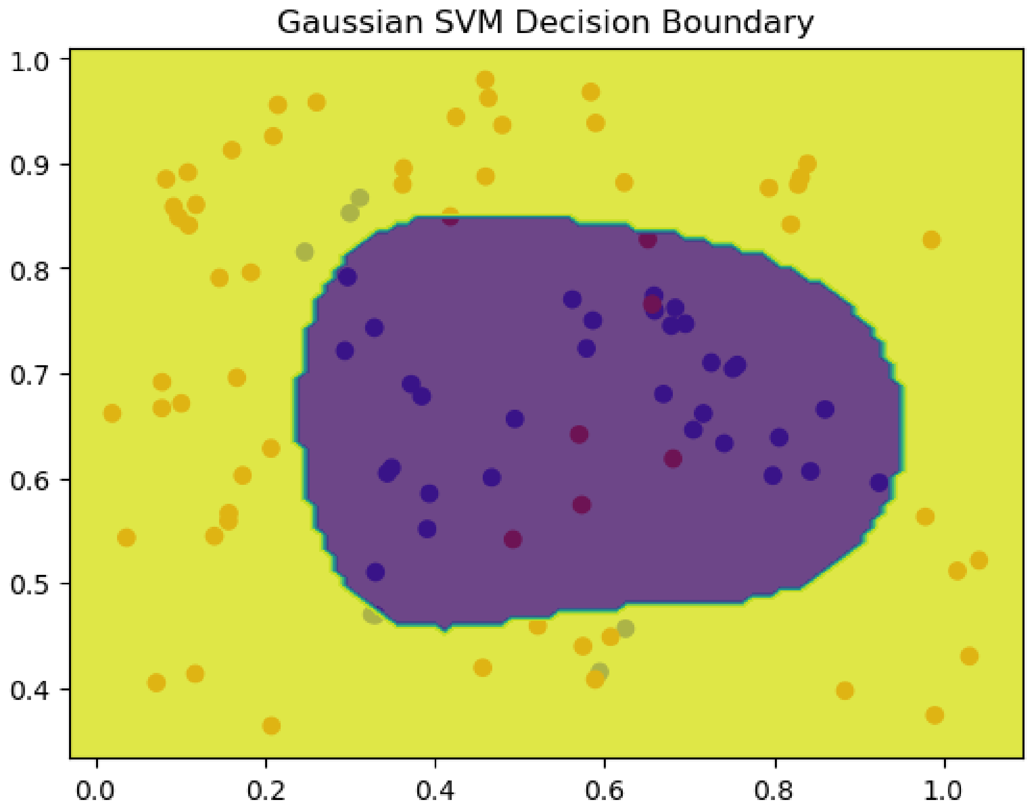
[40]:
```
# Part (e)
# plot decision boundary for logistic regression
# TODO
plot_decision_boundary(model_log, x_train, y_train, title='Logistic Regression␣
↪Decision Boundary', output_path=None,
        file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'logistic' )
```



[41]:
```
# plot decision boundary for linear svm
# TODO
plot_decision_boundary(model_lsvm, x_train, y_train, title='Linear SVM Decision␣
↪Boundary', output_path=None,
        file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'svm' )
```



[42]:
```
# plot decision boundary for linear svm
# TODO
plot_decision_boundary(model_gsvm, x_train, y_train, title='Gaussian SVM␣
↪Decision Boundary', output_path=None,
        file_name=None, class_colors={-1: 'b', 1: 'r'}, model_type= 'svm' )
```


Gaussian SVM Decision Boundary
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0

```
[ ]:
```