Convex Optimization

Prof. Amiri

Electrical Engineering Department

Naser Kazemi 99102059

Practical Homework 2

January 8, 2024

Convex Optimization

Practical Homework 2

Naser Kazemi 99102059

Problem 1: *Soft-Margin SVM and CVXPY*

1. *Primal Formulation for Soft-Margin SVM:*

The code for this problem implemented in CVXPY is:

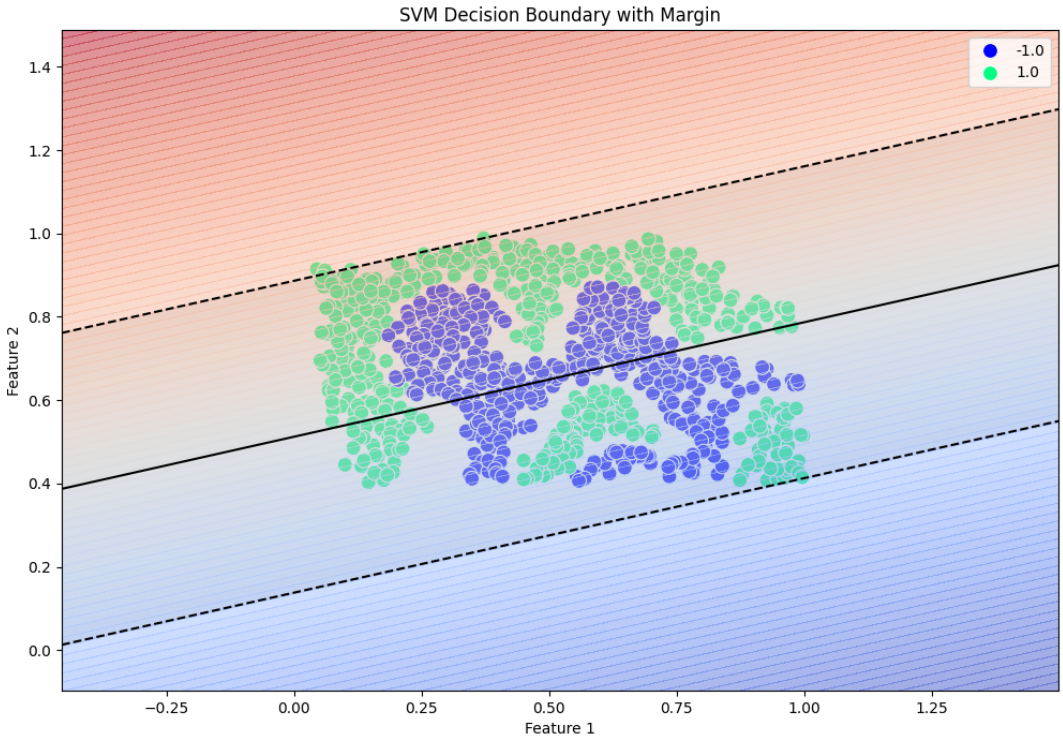
```
# read the data, the first row is the label
data_path = 'svm_train.txt'
data = np.loadtxt(data_path, delimiter=' ', dtype=np.float64, skiprows=1)
X = data[:, 0:2]
y = data[:, 2]

n = X.shape[0]  # number of samples
d = X.shape[1]  # number of features

w = cp.Variable(d)
b = cp.Variable()
ksi = cp.Variable(n)

C = cp.Parameter(nonneg=True)
C.value = 1

objective = 0.5 * cp.sum_squares(w) + C * cp.sum(ksi)

constraints = []
for i in range(n):
    constraints.append(y[i] * (X[i] @ w + b) >= 1 - ksi[i])
    constraints.append(ksi[i] >= 0)


# problem definition
prob = cp.Problem(cp.Minimize(objective), constraints)

# problem solution
prob.solve()

# print solution
print("status:", prob.status)
print("optimal value", prob.value)
print("optimal var w", w.value)
print("optimal var b", b.value)

#  plot the points and the hyperplane, color the area according to the
   predicted class
xx = np.linspace(X[:, 0].min() - 0.5, X[:, 0].max() + 0.5)
yy = - (w.value[0] * xx + b.value) / w.value[1]
margin = 1 / np.sqrt(np.sum(w.value ** 2))
yy_down = yy - np.sqrt(1 + w.value[0] ** 2) * margin
yy_up = yy + np.sqrt(1 + w.value[0] ** 2) * margin
```

```
# Plot the points, the hyperplane, and the margins
plt.figure(figsize=(12, 8))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y, palette='winter', s=100)
plt.plot(xx, yy, 'k-')
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
# Shade the margin area
plt.fill_between(xx, yy_down, yy_up, alpha=0.1, color='k')
# Color the plane according to the distance from the hyperplane
xx_dense, yy_dense = np.meshgrid(np.linspace(X[:, 0].min() - 0.5, X[:, 0].max() + 0.5, 200),
                                 np.linspace(X[:, 1].min() - 0.5, X[:, 1].max() + 0.5, 200))
Z = np.c_[xx_dense.ravel(), yy_dense.ravel()] @ w.value + b.value
Z = Z.reshape(xx_dense.shape)
plt.contourf(xx_dense, yy_dense, Z, levels=np.linspace(
    Z.min(), Z.max(), 100), cmap='coolwarm', alpha=0.5)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('SVM Decision Boundary with Margin')
plt.show()
```

The result of the above code on the `svm_train.txt` data is

```
status: optimal
optimal value 746.2681355070262
optimal var w [-1.00209063  3.64672864]
optimal var b -1.8672371701191284
```

And the plot for the found solution is:



2. *Dual formation of Soft-Margin SVM:*
The primal problem of the soft-margin SVM is to minimize the objective function given by

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, N \end{aligned}$$

The Lagrangian is constructed as

$$L(w, b, \xi, \alpha, \mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\left(y_i(w^T x_i + b) - 1 + \xi_i\right) - \sum_{i=1}^{N}\mu_i\xi_i$$

By finding the saddle point of the Lagrangian and setting the derivatives with respect to the primal variables to zero, we obtain

$$\begin{aligned} \frac{\partial L}{\partial w} &= w - \sum_{i=1}^{N}\alpha_i y_i x_i = 0, \\ \frac{\partial L}{\partial b} &= -\sum_{i=1}^{N}\alpha_i y_i = 0, \\ \frac{\partial L}{\partial \xi_i} &= C - \alpha_i - \mu_i = 0. \end{aligned}$$

We solve for w and ξ:

$$w = \sum_{i=1}^{N}\alpha_i y_i x_i, \quad \mu_i = C - \alpha_i$$

Substituting the solutions back into the Lagrangian, we simplify to get

$$L = \frac{1}{2}\left(\sum_{i=1}^{N}\alpha_i y_i x_i\right)^T\left(\sum_{j=1}^{N}\alpha_j y_j x_j\right) + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\xi_i - \sum_{i=1}^{N}(C - \alpha_i)\xi_i$$

After simplification, we are left with the dual function:

$$W(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

This function is maximized with respect to α under the constraints

$$0 \leq \alpha_i \leq C$$

and

$$\sum_{i=1}^{N}\alpha_i y_i = 0$$

.

By setting the Y as $diag(y)$ and X as the matrix with x_is as its rows, we get the dual problem

$$\begin{aligned} \text{maximize} \quad & W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \alpha^T Y (XX)^T Y \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \cdots, N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

The solution for the dual problem is

```
status: optimal
optimal value 746.2681355070262
optimal var w [-1.00209063  3.64672864]
optimal var b -1.8672371701191284
```

The implementation is available in `svm_dual.py`.

The solution to the primal problems can be recovered from the dual problem solution by the KKT conditions. The KKT conditions are

$$\begin{aligned} y_i(w^T x_i + b) &\geq 1 - \xi_i, \\ \xi_i &\geq 0, \\ \alpha_i &\geq 0, \\ \mu_i &\geq 0, \\ \alpha_i \left(y_i(w^T x_i + b) - 1 + \xi_i \right) &= 0, \\ \mu_i \xi_i &= 0, \\ w - \sum_{i=1}^{N} \alpha_i y_i x_i &= 0 \\ \sum_{i=1}^{N} \alpha_i y_i &= 0 \\ C - \alpha_i - \mu_i &= 0 \end{aligned}$$

Assume that the optimal dual solution α^*, μ^* has been found. We can recover w^* from the KKT conditions. To recover the optimal b, the paper suggests taking any i such that $0 < \alpha_i^* < C$. Then, by complementary slackness, $y_i(b + (w^*)^T x_i) = 1 - \xi_i$. Also, since $\alpha_i^* + \mu_i^* = C$, it follows that $\mu_i^* > 0$. Complementary slackness again implies that $\xi_i = 0$. Now, b can be recovered by $y_i(b + (w^*)^T x_i) = 1$. Instead of taking such a i, we can just average over all the is.

The primal solution therefore is:

```
optimal var w [-1.00209063  3.64672864]
optimal var b -1.9150744111653262
```

Problem 2: *SVM in Machine Learning*

The full implementation is available in `svm.py` file in the folder for this problem. Here we briefly explain the code and answer the questions.

1. Data Preparation

The dataset is loaded, and categorical features are encoded numerically. The 'Age' and 'EstimatedSalary' features are used for model training, and the dataset is split into training and testing sets with the data scaled appropriately.

```
df = pd.read_csv("age_salary.csv")
df = df.drop('User ID', axis=1)
gender_encoder = LabelEncoder()
df['Gender'] = gender_encoder.fit_transform(df['Gender'])

X = df[['Age', 'EstimatedSalary']]
y = 2 * df['Purchased'] - 1  # Convert {0, 1} to {-1, 1}

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

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

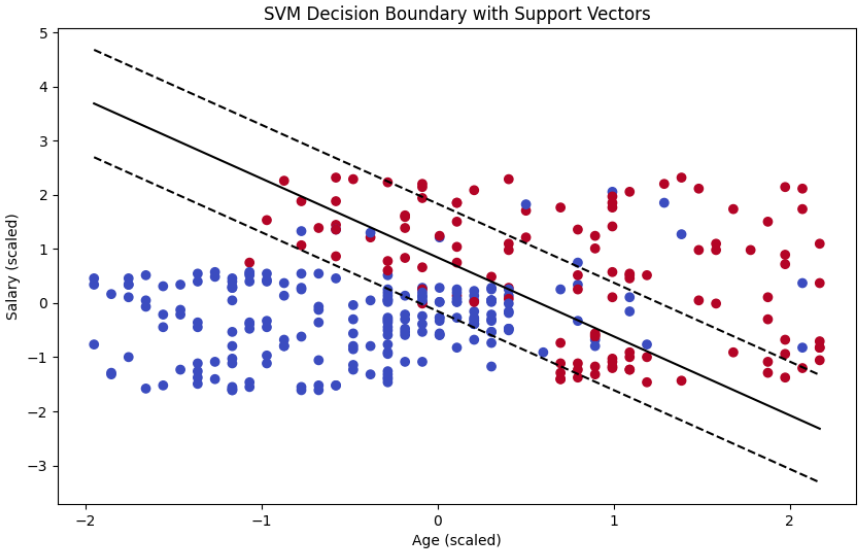
2. SVM Training

A Support Vector Machine with a linear kernel is trained on the prepared dataset.

```
clf = svm.SVC(kernel='linear')
clf.fit(X_train_scaled, y_train)
```

3. Visualization

The decision boundary of the trained SVM model is plotted. The support vectors and margins are clearly indicated.



4. Model Evaluation

The trained SVM model is evaluated on the testing set, and the classification report is provided.

```
y_pred = clf.predict(X_test_scaled)
print("Classification report for SVM:")
print(classification_report(y_test, y_pred))
```

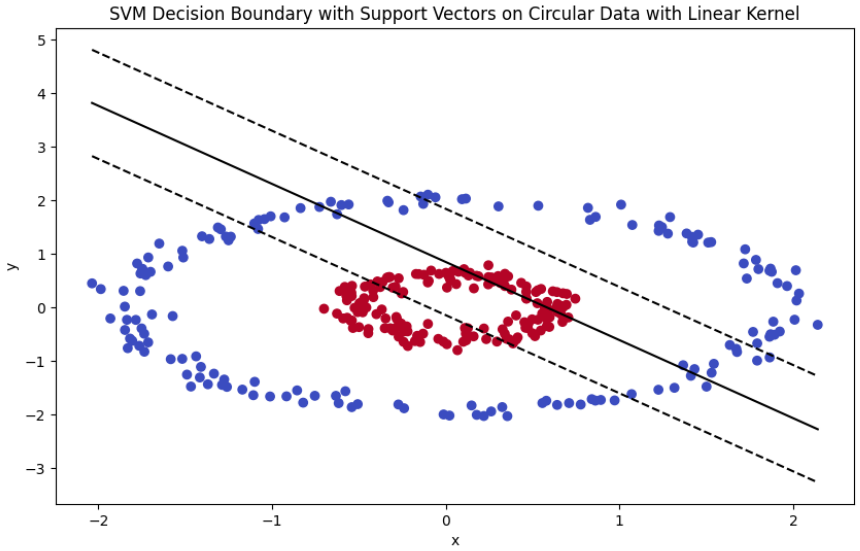
```
Classification report for SVM:
              precision    recall  f1-score   support

          -1       0.85      0.96      0.90        52
           1       0.90      0.68      0.78        28

    accuracy                           0.86        80
   macro avg       0.88      0.82      0.84        80
weighted avg       0.87      0.86      0.86        80
```

5. Reset and Repeat with make_circles

A new circular dataset is generated, and the SVM is trained with a linear kernel. The model's performance is evaluated, indicating that the linear kernel is not the best option for circular data.

```
X_circle, y_circle = datasets.make_circles(n_samples=400, factor=.3, noise
    =.05)
y_circle = 2 * y_circle - 1  # Convert {0, 1} to {-1, 1}
X_train_circle, X_test_circle, y_train_circle, y_test_circle =
    train_test_split(
     X_circle, y_circle, test_size=0.2, random_state=42)
scaler_circle = StandardScaler()
X_train_circle_scaled = scaler_circle.fit_transform(X_train_circle)
X_test_circle_scaled = scaler_circle.transform(X_test_circle)
clf_circle = svm.SVC(kernel='linear')
clf_circle.fit(X_train_circle_scaled, y_train_circle)
y_pred_circle = clf_circle.predict(X_test_circle_scaled)
print("Classification report for SVM with linear kernel on circular data:")
print(classification_report(y_test_circle, y_pred_circle))
```

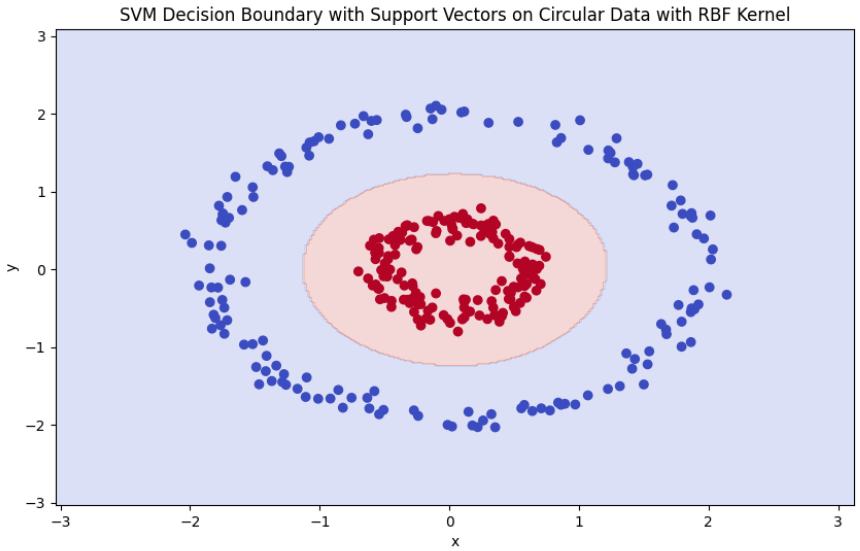
```
Classification report for SVM with linear kernel on circular data:
              precision    recall  f1-score   support

          -1       0.93      0.30      0.46        43
           1       0.55      0.97      0.70        37

    accuracy                           0.61        80
   macro avg       0.74      0.64      0.58        80
weighted avg       0.75      0.61      0.57        80
```



An SVM with an RBF kernel is trained, and its performance is evaluated. The RBF kernel shows significant improvement over the linear kernel for this dataset. The reason is that the data is far from linearly separable, and therefor a SVM with linear kernel would perform poorly for a classification task on such a dataset.

Classification report for SVM with RBF kernel on circular data:

	precision	recall	f1-score	support
-1	1.00	1.00	1.00	43
1	1.00	1.00	1.00	37
accuracy			1.00	80
macro avg	1.00	1.00	1.00	80
weighted avg	1.00	1.00	1.00	80



Problem 3: *Optimization Methods*

Solution in Jupyter Notebook.

Problem 4: *Data Classification with One Hidden Layer (Optional)*

Solution in Jupyter Notebook.

End of Practical Homework 2