Reinelle Jan Bugnot
G2304329L
Nanyang Technological University, Singapore

March 1, 2024

ASSIGNMENT 1

1 Question 1

Consider a multi-class classification problem of C classes. Based on the parametric forms of the conditional probabilities of each class introduced on the 39th Page ("Extension to Multiple Classes") of the lecture notes of L4, derive the learning procedure of regularized logistic regression for multi-class classification problems.

Hint: define a loss function by borrowing an idea from binary classification, and derive the gradient descent rules to update $w^{(c)}$'s.

Solution

From the referenced page in Lecture 4, we have the following equations that represent the Probability of y falling to a specific class c, given an input data $\mathbf{x}$.

$$P(y=c|\mathbf{x}) = \frac{e^{-\mathbf{w}^{(c)T}\mathbf{x}}}{1+\sum_{j=1}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} \quad \forall \quad c>0 \tag{1}$$

$$P(y=0|\mathbf{x}) = \frac{1}{1+\sum_{j=1}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} \quad for \quad c=0 \tag{2}$$

Given these probabilities are mutually exclusive and collectively exhaustive, Eq. 1 and 2 add up to a total probability of 1.

$$\sum_{c=0}^{C-1} P(y=c|\mathbf{x}) = 1 \tag{3}$$

Expanding Eq. 3 using Eq. 1 and 2, we get:

$$\frac{1}{1+\sum_{j=1}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} + \frac{\sum_{c=1}^{C-1} e^{-\mathbf{w}^{(c)T}\mathbf{x}}}{1+\sum_{j=1}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} = 1 \tag{3a}$$

We assume that there exists a learned weight vector $\mathbf{w}^{(0)}$ for when $c=0$ such that

$$e^{-\mathbf{w}^{(0)T}\mathbf{x}} = 1 \tag{4}$$

Substituting Eq. 4 back to Eq. 3a,

$$\frac{e^{-\mathbf{w}^{(0)T}\mathbf{x}}}{e^{-\mathbf{w}^{(0)T}\mathbf{x}}+\sum_{j=1}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} + \frac{\sum_{c=1}^{C-1} e^{-\mathbf{w}^{(c)T}\mathbf{x}}}{e^{-\mathbf{w}^{(0)T}\mathbf{x}}+\sum_{j=1}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} = 1 \tag{5}$$

Simplifying the denominator and expanding the numerator of Eq. 5 gives,

$$\frac{e^{-\mathbf{w}^{(0)T}\mathbf{x}} + e^{-\mathbf{w}^{(1)T}\mathbf{x}} + \dots + e^{-\mathbf{w}^{(C-1)T}\mathbf{x}}}{\sum_{j=0}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} = 1 \tag{6}$$

examining each individual components in the summation shown in Eq. 6, we can conclude that

$$P(y=c|\mathbf{x}) = \frac{e^{-\mathbf{w}^{(c)T}\mathbf{x}}}{\sum_{j=0}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} \quad \forall \quad c \in \{0,1,...,C-1\} \tag{7}$$

Eq. 7, in fact, is otherwise referred to as the *Softmax* function. The softmax function gives us a vector that represents the probability distributions of a list of potential outcomes. It's a way of normalizing the outputs of our model to get a distribution of probabilities.

We can then define the cross-entropy loss function as:

$$E(\mathbf{w}) = -\sum_{k=0}^{C-1} y_k log(\hat{y_k}) \tag{8}$$

where,

$$\hat{y_c} = P(y=c|\mathbf{x}) = \frac{e^{-\mathbf{w}^{(c)T}\mathbf{x}}}{\sum_{j=0}^{C-1} e^{-\mathbf{w}^{(j)T}\mathbf{x}}} \quad \forall \quad c \in \{0,1,...,C-1\} \tag{9}$$

With N training examples with input features $x_i \in \mathbb{R}^M$, the total loss function is:

$$J(\mathbf{w}) = -\sum_{i=1}^{N}\left[-\sum_{j=0}^{C-1} y_i log(\hat{y_j})\right] + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{10}$$

Using the Gradient Descent update rule, we need to compute the 1st order derivative of the loss function $J(w)$ with respect to the weights w, as follows:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \alpha\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \\ &= \mathbf{w}_t - \alpha\left[-\sum_{i=1}^{N}\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} + \frac{\partial \frac{\lambda}{2}||\mathbf{w}||_2^2}{\partial \mathbf{w}}\right]\end{aligned} \tag{11}$$

where,

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial E(\mathbf{w})}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial \mathbf{w}} \tag{12}$$

and where,

$$z^{(c)} = \mathbf{w}^{(c)^T}\mathbf{x}_i \tag{13}$$

Eq. 12 shows that by applying the Chain Rule, we can solve for the derivative of E($\mathbf{w}$) with respect to $\mathbf{w}$ by solving for the individual derivatives of the composite function $E(\hat{y}(z(\mathbf{w})))$.

We start by taking the derivative of the Softmax function. If $i = j$, the derivative of the softmax function with respect to its own input is [1]:

$$\frac{\partial \hat{y_i}}{\partial z} = \hat{y_i}(1-\hat{y_i}) \tag{14}$$

If $i \neq j$, the derivative of the softmax function with respect to the other inputs is [1]:

$$\frac{\partial \hat{y_i}}{\partial z} = -\hat{y_i}(\hat{y_j}) \tag{15}$$

Using these results, we can compute the derivative of the cross-entropy loss $E(\mathbf{w})$ with respect to the inputs to the Softmax function z [1],

$$\frac{\partial E(\mathbf{w})}{\partial z} = \sum_i \begin{cases} y_i(1-\hat{y}_i) & i = j \\ -y_i(\hat{y}_j) & i \neq j \end{cases} \tag{16}$$

Given that exactly one value from the true labels y_i is 1 and the rest are zeros, we can further simplify the expression to,

$$\frac{\partial E(\mathbf{w})}{\partial z} = \hat{y}_i - y_i \tag{17}$$

Next, from Eq. 13, the derivative of $z^{(c)}$ w.r.t to the weight vector $\mathbf{w}^{(c)}$ is given by:

$$\begin{aligned} \frac{\partial z^{(c)}}{\partial \mathbf{w}^{(c)}} &= \frac{\partial (\mathbf{w}^{(c)^T}\mathbf{x}_i)}{\mathbf{w}^{(c)}} \\ &= \mathbf{x}_i \end{aligned} \tag{18}$$

Therefore,

$$\begin{aligned} \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial E(\mathbf{w})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \mathbf{w}} \\ &= (\hat{y}_i - y_i)\mathbf{x}_i \end{aligned} \tag{19}$$

Also, we find the derivative of the regularization term, $\frac{\lambda}{2}||\mathbf{w}||_2^2$,

$$\frac{\partial \frac{\lambda}{2}||\mathbf{w}||_2^2}{\partial \mathbf{w}} = \lambda \mathbf{w} \tag{20}$$

Finally, substituting Eq. 19 and Eq. 20 to the derivative of the gradient update rule in Eq. 11, we get:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(\sum_{i=1}^{N} (\hat{y}_i - y_i)\mathbf{x}_i - \lambda \mathbf{w} \right) \tag{21}$$

2 Question 2

This is a hands-on exercise to use the SVC API of scikit-learn to train a SVM with the linear kernel and the rbf kernel, respectively, on a binary classification dataset. The details of instructions are described as follows:

1. Download the a9a dataset from the LIBSVM Dataset page
2. Regarding the linear kernel, show 3-fold cross-validation results in terms of classification accuracy on the training set with different values of the parameter C.
3. Regarding the rbf kernel, show 3-fold cross-validation results in terms of classification accuracy on the training set with different values of the parameter gamma and different values of the parameter C.
4. Based on the results, determine the best kernel and the best parameter setting.

Import Data

First, I imported the relevant libraries that will be used for this exercise; primarily, sklearn and numpy.

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC, LinearSVC
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import cross_val_score, GridSearchCV
```

I then defined the following constants and variables as provided:

```python
DIRPATH = './data/'
TRAIN_FILE = 'a9a.txt'
TEST_FILE = 'a9a.t'

C_vals = [0.01, 0.05, 0.1, 0.5, 1]
gamma_vals = [0.01, 0.05, 0.1, 0.5, 1]
k_fold = 3
```

Next, the data is loaded using the `load_svmlight_file()` from `sklearn.datasets`, which is specifically tailored to data in text-based format, with one sample per line and does not store zero valued features; hence, a sparse dataset. I set the `n_features` argument to 123, which is the total number of features as defined in the a9a dataset documentation.

```python
X_train, y_train = load_svmlight_file(DIRPATH + TRAIN_FILE, n_features=123)
X_test, y_test = load_svmlight_file(DIRPATH + TEST_FILE, n_features=123)

print(f'Shape of Training Dataset: {X_train.shape}')
print(f'Shape of Testing Dataset:{ X_test.shape}')

# Shape of Training Dataset: (32561, 123)
# Shape of Testing Dataset:(16281, 123)
```

3-fold CV on SVC with Linear Kernel

I defined the model using the `LinearSVC` function of the `sklearn.svm` module. To perform 3-fold cross validation, I used the `cross_val_score()` function from the `model_selection` module of sklearn with the argument `cv` set to 3. The code snippet below loops through all the provided values of C. The average accuracy for each loop is stored and presented in Table I.

```python
RESULTS = {}
for C in C_vals:
    clf = LinearSVC(C=C, max_iter=10000)
    scores = cross_val_score(clf, X_train, y_train, cv=k_fold, scoring='accuracy')
    print(f"3-fold CV Mean Accuracy score for C = {C}: {scores.mean()}")
    RESULTS[C] = scores.mean()

# 3-fold CV Mean Accuracy score for C = 0.01: 0.8468413733053096
# ...
```

TABLE I. The 3-fold cross-validation results of varying values of C in SVC with linear kernel on the a9a training set (in accuracy). Max value is highlighted in yellow.

C	0.01	0.05	0.1	0.5	1
Mean Accuracy Score	0.84688	0.84746	0.84749	0.84733	0.84727

3-fold CV on SVC with RBF Kernel

Like the approach done in the previous section, I defined the SVC model using the `SVC` function of the `sklearn.svm` module. The `cross_val_score()` function is also used to implement a 3-fold cross validation strategy by setting the `cv` parameter to 3. Unlike the Linear SVC model presented in the previous section, I now define a parameter grid of C and $gamma$ values defined in the snippets above as `C_vals` and `gamma_Vals`.

The code below loops across all the combinations of values of C and $gamma$ and calculates the average accuracy across all the folds for each iteration. The result of this process is shown in Table II.

```python
RESULTS_2 = {}
for C in C_vals:
    for gamma in gamma_vals:

        clf = SVC(kernel='rbf', C=C, gamma=gamma, max_iter=10000)
        scores = cross_val_score(clf,
                                 X_train,
                                 y_train,
                                 cv=k_fold,
                                 scoring='accuracy')

        print(f"3-fold CV Mean Accuracy score for C = {C} and \
        gamma = {gamma}: {scores.mean()}")

        RESULTS_2[(C, gamma)] = scores.mean()

# 3-fold CV Mean Accuracy score for C = 0.01 and gamma = 0.01: 0.7591904439861589
# ...
```

TABLE II. The 3-fold cross-validation results of varying values of gamma and C in SVC with rbf kernel on the a9a training set (in accuracy). Max value is highlighted in yellow.

	gamma=0.01	**gamma=0.05**	**gamma=0.1**	**gamma=0.5**	**gamma=1**
C=0.01	0.75919	0.81991	0.81985	0.75919	0.75919
C=0.05	0.83121	0.83575	0.83425	0.78916	0.75919
C=0.1	0.83772	0.83965	0.83876	0.80612	0.76199
C=0.5	0.84297	0.84577	0.84681	0.83241	0.79021
C=1	0.84442	0.84675	0.84742	0.83686	0.80007

Train Model using Best Parameters

From the results presented in Table I and Table II, we can see that the best performance was obtained when using a Linear SVC model with `C=0.1`. A new model is then trained using these parameters and tested on the a9a testing dataset. The result of which is shown in Table III.

```python
best_C = max(RESULTS, key= lambda x: RESULTS[x])
clf = LinearSVC(C=best_C, max_iter=10000).fit(X_train, y_train)
result = clf.score(X_test, y_test)
print(f'Linear SVC Test Accuracy at C = {best_C} is {result}')

# Linear SVC Test Accuracy at C = 0.1 is 0.8498249493274369
```

TABLE III. Test results of SVC on the a9a test set (in accuracy).

	kernel* = "Linear", C* = 0.1
Test Score	0.84982

3 Question 3

The optimization problem of linear soft-margin SVMs can be re-formulated as an instance of empirical risk minimization (refer to Page 37 on L5 notes). Show how to reformulate it. Hint: search reference about the hinge loss.

Solution

When training a Support Vector Machine, our goal is to find the hyperplane that maximizes the margin that separates data points that belong to separate classes. Unlike in common linear separators, this hyperplane is fully defined by the points that lie closest to the margin, called *support vectors*.

However in most real-world scenarios, the data we have cannot be linearly separated. To deal with this, we will use a variant of SVM called *soft-margin SVM*. Soft-margin SVM permits some violations in the margin requirement; i.e., not all data points need to be perfectly classified. This is implemented using a so-called *slack variable* ζ such that:

1. If $\zeta = 0$, the data point x_i is correctly classified and not within the margin
2. If $0 < \zeta < 1$, the data point x_i is correctly classified but within the margin
3. If $\zeta = 1$, the data point x_i is on the decision boundary
4. If $\zeta > 1$, the data point x_i is beyond the decision boundary and is incorrectly classified.

From here, we can extend the objective function of (hard-margin) SVM to include the slack variable ζ_n:

$$\min_{\mathbf{w},b,\zeta} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{N}\zeta_i \quad \textbf{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \zeta_i \quad \textbf{where} \quad \zeta_i \geq 0 \quad \textbf{for} \quad i = 1, ..., N \tag{1}$$

The parameters C here ($C > 0$) is a trade-off parameter that balances the impact of margin maximization and tolerable errors. Next, we introduce the *hinge loss function*, which is a loss function used for training classifiers, most notably for support vector machines. The hinge loss function is defined as [3]:

$$\ell_{hinge}(y, f(x)) = \max(0, 1 - yf(x)) \tag{2}$$

where:

- y is the true class label (-1 or 1)
- $f(x)$ is the output of the decision function for data point x

The hinge loss function is used for "maximum-margin" classification. When $yf(x) \geq 1$, this means that the instance is correctly classified and the loss is 0. When $yf(x) < 1$, it means that the instance is misclassified and the loss is proportional to how far the decision function is from correctly classifying the instance [2].

We can rewrite the constraints of the SVM objective function in Eq. 1 using the hinge loss function, as follows:

$$\begin{aligned} y_i(\mathbf{w}^T\mathbf{x}_i + b) &\geq 1 - \zeta_i \\ \zeta_i &\geq 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \end{aligned} \tag{3}$$

Since it is always true that $\zeta_i \geq 0$ and $\ell_{hinge}(y, f(x)) \geq 0$, it follows that:

$$\zeta_i \geq \ell_{hinge}(y_i, (\mathbf{w}^T\mathbf{x}_i + b)) \tag{3a}$$

From Eq. 3a, we can see that the hinge loss function ℓ_{hinge} provides a lower bound for the slack variable ζ_i. Since the objective function defined in Eq. 1 is a minimization problem, then the minimum value of ζ_i must be equivalent to,

$$\zeta_i^* = \ell_{hinge}(y_i, (\mathbf{w}^T\mathbf{x}_i + b)) = \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) \tag{4}$$

Substituting Eq. 4 to the objective function in Eq. 1, we get:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{N} \ell_{hinge}(y_i, (\mathbf{w}^T\mathbf{x}_i + b)) \tag{5}$$

By re-arranging the terms inside the minimization operator, we can show that the objective function in Eq. 4 is in the form of an Empirical Risk Minimization expression, where the loss function corresponds to the hinge loss and the regularization term Ω corresponds to $||\mathbf{w}||_2^2$.

$$\begin{aligned} \text{soft-margin SVM expressed using hinge loss:} \quad & \min_{\mathbf{w},b} C\sum_{i=1}^{N} \ell_{hinge}(y_i, (\mathbf{w}^T\mathbf{x}_i + b)) + \frac{1}{2}||\mathbf{w}||_2^2 \\ \text{empirical risk minimization expression:} \quad & \min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell(y_i, f(\mathbf{x}_i|\boldsymbol{\theta})) + \lambda\Omega(\boldsymbol{\theta}) \end{aligned} \tag{6}$$

4 Question 4

Using the kernel trick introduced in L5 to extend the regularized linear regression model (L3) to solve nonlinear regression problems. Derive a closed-form solution (i.e., to derive a kernelized version of the closed-form solution on Page 50 of L3).

Solution

The regularized linear regression formula from slide 47 of lecture note 3 is shown below:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2 \tag{1}$$

A closed-form solution to this unconstrained optimization problem can be achieved by setting the derivative of $\hat{\mathbf{w}}$ w.r.t to $\mathbf{w}$ to 0, as shown.

$$\frac{\partial\left[\frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T - y_i)^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2\right]}{\partial\mathbf{w}} = \mathbf{0} \tag{2}$$

Further simplifying the lefthand side of Eq. 2,

$$\begin{aligned} \frac{\partial\left[\frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T - y_i)^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2\right]}{\partial\mathbf{w}} &= \frac{\partial\frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2}{\partial\mathbf{w}} + \frac{\partial\lambda||\mathbf{w}||_2^2}{\partial\mathbf{w}} \\ &= \left[\sum_{i=1}^{N}(\mathbf{x}_i\mathbf{x}_i^T)\right]\mathbf{w} - \sum_{i=1}^{N} y_i\mathbf{x}_i + \lambda\mathbf{w} \\ &= \mathbf{X}\mathbf{X}^T\mathbf{w} - \mathbf{X}\mathbf{y} + \lambda\mathbf{I}\mathbf{w} \\ &= (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\mathbf{w} - \mathbf{X}\mathbf{y} \end{aligned} \tag{3}$$

Substituting Eq. 3 to Eq. 2,

$$\begin{aligned} (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\mathbf{w} - \mathbf{X}\mathbf{y} &= 0 \\ (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\mathbf{w} &= \mathbf{X}\mathbf{y} \\ \mathbf{w} &= (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{y} \quad \forall \quad \lambda > 0 \end{aligned} \tag{4}$$

The term $\mathbf{XX}^T$ represents the inner product in an arbitrary feature space ϕ, such that the dataset $\mathbf{X}$ in our data domain is equivalent to $\phi(\mathbf{X})$ in this arbitrary feature space. When we apply the kernel trick, we can replace the $\mathbf{XX}^T$ term with a kernel matrix $\mathbf{K}$, defined as:

$$\mathbf{K} = \mathbf{XX}^T \tag{5}$$

where $K_{ij} = k(x_i, x_j) =< \phi(x_i), \phi(x_j) >$

Eq. 4 then becomes,

$$\mathbf{w} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\phi(\mathbf{X})^T\mathbf{y} \tag{6}$$

Here in Eq. 6, $\mathbf{w}$ is our solution (weights) vector in the feature space and $\phi(\mathbf{X})^T\mathbf{y}$ is the projection of the target vector $\mathbf{y}$ onto the same feature space. Since the model is operating in the data space and not the feature space, the kernel trick allows us to implicitly map the data into a higher dimensional feature space and compute the inner products in that space without having to explicitly compute the individual mapping of our data points x_i into the feature space $\phi(x_i)$.

Given this, we are interested in a vector of coefficients α that gives us the weights for a linear combination of our data points, rather than a vector $\mathbf{w}$ of weights in the feature space. Hence, we define α as,

$$\alpha = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} \tag{7}$$

By using the kernel trick on the regularized linear regression model to derive α, we can now solve for nonlinear regression problems.

References

[1] David Bieber. *Derivative of Softmax and the softmax cross entropy loss.* Dec. 2020. URL: `https://davidbieber.com/snippets/2020-12-12-derivative-of-softmax-and-the-softmax-cross-entropy-loss/`.

[2] Kunal Chowdhury. *Understanding loss functions : Hinge loss.* Medium, Analytics Vidhya. URL: `https://medium.com/analytics-vidhya/understanding-loss-functions-hinge-loss-a0ff112b40a1` (visited on 03/01/2024).

[3] *Hinge loss.* Wikipedia. URL: `https://en.wikipedia.org/wiki/Hinge_loss` (visited on 03/01/2024).

Submitted by Reinelle Jan Bugnot on March 1, 2024.