CS 559: Machine Learning – Fundamentals & Applications

Assignment 2 Due: 2/18/2025 Tuesday 11:59 p.m.

- The assignment must be individual work and must not be copied or shared. Any tendency to cheat/copy evidence will lead to a 0 mark for the assignment.
- Students must only use Pandas, NumPy, Matplotlib, and Scipy if the problem does not specify libraries/packages.
- All codes will be tested in grading. Any codes with an error will be marked 0. Make sure to restart the kernel and run it all before submission. Delete any codes that should not be graded.
- Results must be displayed.
- All problems must be submitted in a single notebook file. Do not use a text editor to write codes.

1 Naive Bayes Classification [40 pts]

Use the following code to generate the train data set. The code will generate a random data set with 5 features and 4 classes.

```
from sklearn import datasets
X, y = datasets.make_blobs(n_samples=400, n_features=5,
    centers=4, cluster_std=2, random_state=100)
```

a. [5 pts] Compute the prior probability of each class, $p(C_k)\ \forall k = 1, \ldots, 4$.

b. [10 pts] Compute the likelihood of each data point, $p(\mathbf{X}|C_k)\ \forall k = 1, \ldots, 4$.

c. [15 pts] Compute the posterior probability of each data point, $p(C_k|\mathbf{X})\ \forall k = 1, \ldots, 4$. Assign the class ID to each point.

d. [5 pts] Construct the confusion matrix to show the classification rate using `sklearn.metrics.confusion_matrix`. The confusion matrix visualizes and summarizes the performance of a classification algorithm, as shown below.

Total Prediction = 400	$y = 0$	$y = 1$	$y = 2$	$y = 3$
$\hat{h} = 0$				
$\hat{h} = 1$				
$\hat{h} = 2$				
$\hat{h} = 3$				

e. [5 pts] Classify the target using `sklearn.naive_bayes.GaussianNB`. Report the accuracy of the model.

2 Perceptron [30 pts]

In the lecture, the implementation of the perceptron algorithm was discussed with pseudo-code. In this problem, students will implement the algorithm and train the model using the following data set.

```
X, y = datasets.make_blobs(n_samples=400, n_features=5,
    centers=2, cluster_std=2, random_state=100)
```

a. [15 pts] In the lecture slide, the methods needed for the perceptron algorithm are provided (`step(X)` and `perceptron_predict(w, X, i)`). Write a method `perceptron_fit(w, X, y, iteration)` that fits the data and returns $\mathbf{w}$.

b. [15 pts] Fit the sample data and find $\mathbf{w}$ when the iteration is 1. Report the accuracy. Perform more iterations as necessary to get as close to perfect accuracy as you can.

3 Logistic Regression with Regularization [30 pts]

A common problem with statistical models is **overfitting**, which occurs when the model fits the training data very well, but has poor accuracy on the test data. For linear models like logistic regression, overfitting can occur when the weight parameters become too large in absolute value. One solution is **regularization**, which adds a penalty term to the error function that increases based on the size of the weights. Let $\mathcal{E}(\mathbf{w})$ be the error function for logistic regression presented in the lecture. Then the regularized error function has the form

$$\tilde{\mathcal{E}}(\mathbf{w}) = \mathcal{E}(\mathbf{w}) + \lambda \mathcal{R}(\mathbf{w}) \tag{1}$$

where $\mathcal{R}(\mathbf{w})$ is the regularization penalty, and the coefficient λ governs the relative importance of the regularization term. In this problem, we will observe how the model gets regularized using the following data set.

```
X, y = datasets.make_blobs(n_samples=400, n_features=5,
    centers=4, cluster_std=2, random_state=100)
```

We will use **Lasso** regularization, which attempts to make the weights **sparse**, meaning that few of them are nonzero. Note that if a weight is zero, or very close to zero, then the corresponding feature can be regarded as unimportant for the model. Thus, Lasso regularization can be used as a **feature selection** technique.

a. [10 pts] Train a `sklearn.linear_model.LogisticRegression` model with a parameter **penalty**='l1' (Lasso regularization). The parameter **C**= λ^{-1}, where λ is defined in Equation 1, has a default value of 1, which represents a strong penalty. The parameter **solver='saga'**, which is one of the fastest convergence solvers for any regularization in `LogisticRegression()`, must be used when the 'l1' penalty parameter is used.

```
from sklearn.linear_model import LogisiticRegression
clf = LogisiticRegression(penalty = 'l1', solver = 'saga', C = 1.0)
...
w, w0 = clf.coef_, clf.intercept_
```

Make a plot of $\mathbf{w}$ versus C to show the convergence of $\mathbf{w}$ as C varies from 100 (weak penalty) down to 1 (strong penalty).

b. [10 pts] Explain which features, if any, are **unimportant** for each class.

c. [10 pts] Repeat part **a**. using Ridge regularization (penalty='l2') to show that it does not provide a sparse solution.