CISC /CMPE 251 Assignment 4 Part 1: Theoretical

Lauren Steel – 20218337
November 25, 2024

Q1: Weighted Accuracy

For a classification problem, given the following confusion matrix, calculate the **weighted accuracy** of the model assuming class weights are based on the class distribution in the training set.

	Pred Class 1	Pred Class 2
Actual Class 1	50	10
Actual Class 2	5	35

Assume Class 1 had 200 samples in the training set, and Class 2 had 100 samples. Calculate the weighted accuracy.

The following is the general formula for weighted accuracy:

$$weighted\ accuracy = \frac{1}{N}\sum_{i=1}^{n} w_i \times \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$$

1. Definitions:
    a. TP: True Positives
    b. TN: True Negatives
    c. FP: False Positives
    d. FN: False Negatives
    e. W_i: Weight for class i
2. Therefore, we can update the confusion matrix with the following labels:

	Pred Class 1	Pred Class 2
Actual Class 1	$TP_1 = 50$	$FN_1 = 10$
Actual Class 2	$FP_1 = 5$	$TP_2 = 35$

And we get the following values for each class…

Class 1:

> $TP_1 = 50$
> $FN_1 = 10$
> $FP_1 = 5$ (Class 2 cases misclassified as class 1)
> $TN_1 = 35$ (Negatives that are correctly classified into class 2)

Class 2:

- $TP_2 = 35$
- $FN_2 = 5$
- $FP_2 = 10$ (Class 1 cases misclassified as class 2)
- $TN_2 = 50$ (Negatives that are correctly classified into class 1)

3. Now we calculate the class weights based on training distribution:

Weight for class #1:

$$w_1 = \frac{total\ samples\ class\ 1}{total\ samples\ in\ training} = \frac{200}{(200+100)} = \frac{2}{3}$$

Weight for class #2:

$$w_2 = \frac{total\ samples\ class\ 2}{total\ samples\ in\ training} = \frac{100}{(200+100)} = \frac{1}{3}$$

4. Now we calculate accuracy for each class

Using the following accuracy formula…

$$Accuracy = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$$

Accuracy for class #1:

$$Accuracy_1 = \frac{TP_1 + TN_1}{TP_1 + TN_1 + FP_1 + FN_1} = \frac{50+35}{50+35+5+10} = \frac{85}{100} = 0.85$$

Accuracy for class #2:

$$Accuracy_2 = \frac{TP_2 + TN_2}{TP_2 + TN_2 + FP_2 + FN_2} = \frac{35+50}{35+50+10+5} = \frac{85}{100} = 0.85$$

5. Finally, we can calculate the weighted accuracy…

$$weighted\ accuracy = \frac{1}{N}\sum_{i=1}^{n} w_i \times \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}$$

1/N term cancels since $\sum w_i = 1$ is already satisfied, the sum of the weights naturally includes normalization.

$$weighted\ accuracy = (w_1 \times Accuracy_1) + (w_2 \times Accuracy_2)$$
$$= \left(\frac{2}{3} \times \frac{85}{100}\right) + \left(\frac{1}{3} \times \frac{85}{100}\right) = \frac{85}{100} = 0.85$$

Therefore, the weighted accuracy of the model is approximately 0.85, alternately, 85%. Weighted accuracy reflects the accuracy adjusted for the class distribution in the training set. More weight is given to class #1 as it had majority of samples.

Q2: Hyperparameter Tuning

You are performing grid search with 5-fold cross-validation to tune hyperparameters for a neural network. You are evaluating 10 different combinations of hyperparameters. If the dataset contains 2000 samples, how many times will the model be trained and validated during the entire grid search process? What would be the impact on computational resources if you switch to 10-fold cross-validation?

In 5-fold cross-validation, the data set is divided into 5 equal parts. The model is trained with 4 of these folds and tested with the remaining folds. This process occurs 5 times, one training iteration for each fold. Given the 10 different combinations of hyperparameters we can calculate the number of runs as follows:

$$total\ runs = 5 \times 10 = 50$$

Using the same thought process as above if we switch to a 10-fold cross validation, the model must train 10 times per hyperparameter combination thus we have:

$$total\ runs = 10 \times 10 = 100$$

Therefore, if we increase to a 10-fold cross validation it will double the computational load and the training time of the model.

Q3: Determining Decision Boundaries in SVC

A linear SVM is applied to the following 2D dataset:

Point	x_1	x_2	Label y
A	2	2	+1
B	1	1	+1
C	1	4	-1

D	3	1	-1

1. Determine the equation of the decision boundary $w^Tx + b = 0$ for this dataset.

To determine the decision boundary equation for this dataset a quick python script was run leveraging the SVC sci kit learn package and printing the output as displayed in Figure 1. We see that the decision boundary equation for the respective dataset is as follows:

$$decision\ boundary\ equation = -6X_1 - 4X_2 + 20.99$$

```python
import numpy as np
from sklearn.svm import SVC

# dataset
X = np.array([[2, 2], [1, 1], [1, 4], [3, 1]])
y = np.array([1, 1, -1, -1])

# train a linear SVM
svm = SVC(kernel='linear', C=1e10)
svm.fit(X, y)

# extract the weights and bias
w = svm.coef_[0]
b = svm.intercept_[0]

# print results
print("Weights (w):", w)
print("Bias (b):", b)
print(f"Decision boundary equation: {w[0]:.2f}x1 + {w[1]:.2f}x2 + ({b:.2f}) = 0")
```

✓ 10.1s

```
Weights (w): [-5.99800206 -3.99900103]
Bias (b): 20.99367319690593
Decision boundary equation: -6.00x1 + -4.00x2 + (20.99) = 0
```

Figure 1: Python script calculating the decision boundary in SVC

2. Verify if each point satisfies the margin condition $y_i(w^Tx_i + b) \geq 1$.

Below in Figure 2 is a python script that was run to verify whether or not each point meets the specified margin condition, $y_i(w^Tx_i + b) \geq 1$. As seen in the output, points B & C meet the condition while A & D do not since their margin values fall slightly below 1.0.

```python
import numpy as np
from sklearn.svm import SVC

# dataset
X = np.array([[2, 2], [1, 1], [1, 4], [3, 1]])
y = np.array([1, 1, -1, -1])

# train linear SVM
svm = SVC(kernel='linear', C=1e10)
svm.fit(X, y)

# extract weights and bias
w = svm.coef_[0]
b = svm.intercept_[0]

# verify the margin condition yi(w^T xi + b) >= 1 for all points
margin_values = y * (np.dot(X, w) + b)

tolerance = 1e-6
# print verification results
print("\nVerification Results with Tolerance:")
for i, (point, margin) in enumerate(zip(X, margin_values)):
    satisfies = margin >= (1 - tolerance)
    print(f"Point {i+1} ({point}): Margin value = {margin:.6f}, "
          f"{'Satisfies' if satisfies else 'Does not satisfy'} the margin condition.")
```

✓ 0.0s

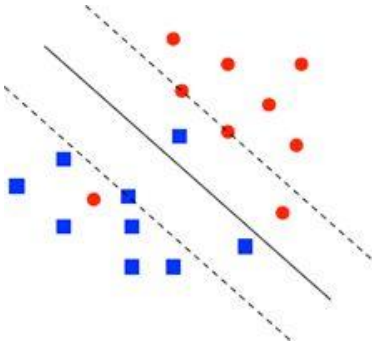
```
Verification Results with Tolerance:
Point 1 ([2 2]): Margin value = 0.999667, Does not satisfy the margin condition.
Point 2 ([1 1]): Margin value = 10.996670, Satisfies the margin condition.
Point 3 ([1 4]): Margin value = 1.000333, Satisfies the margin condition.
Point 4 ([3 1]): Margin value = 0.999334, Does not satisfy the margin condition.
```

Figure 2: Python script to verify whether or not each point meets the margin condition $y_i(w^T x_i + b) \geq 1$

The scripts displayed above in Figure 1and Figure 2 are provided in the Jupyter Notebook for the technical portion of this assignment, *a4.ipynb*.

Q4: Support Vectors in SVC

The figure below shows the decision boundary obtained upon running soft-margin SVM on a small data set of blue squares and red circles.



1. Mark the support vectors.

> There are a few support vectors to acknowledge given the decision boundary above
> Red circles either on or inside of the dashed margin on the right-hand side of the decision boundary are support vectors
    - This includes the two red dots on the margin line along with the red dot between the margin and the decision boundary which is still correctly classified
> Additionally, the blue squares either on or inside the dashed margin line on the left-hand side of the decision boundary are support vectors
    - This includes the one blue square on the decision boundary as well as the point which lays between the margin and decision boundary on the left-hand side
> Furthermore, we should also be considering the misclassified cases. In a soft margin SVM, points that violate a decision boundary still contribute to the optimization problem
> These points will directly influence the cost function (penalty term for misclassification) and therefore need to be considered as support vectors since they contribute to the way a model adjusts the boundary in attempts to minimize misclassification
    - With this in mind both the red dot and blue square that have crossed over the decision boundary and are misclassified contribute as support vectors as well
> All the points discussed are critical for defining both the decision boundary and the margin lines of the model

2. Suppose the factor C in the soft-margin SVM optimization problem were increased. Would you expect the margin to increase or decrease?

> The factor *C* in soft margin SVM optimization controls the tradeoff between a larger margin and minimizing classification errors
> When C gets increased there is more emphasis placed on minimizing misclassification errors
> This means margins (the dashed lines) will be pushed closer together and the SVM will have a higher focus on correctly classifying all the training data

- Increasing C will decrease the margin size which makes the model more sensitive to the training data
- A balance should be struck such that the model does not overfit to training data and is still able to generalize, correctly classifying new data from outside the training set

Q5: Constructing a Decision Tree

Consider the following dataset with 3 features (X1, X2, X3) and one target variable Y:

Instance	X_1	X_2	X_3	Y
1	Yes	No	High	Yes
2	No	Yes	Medium	No
3	Yes	Yes	Low	Yes
4	No	No	Medium	No
5	Yes	No	Low	Yes
6	No	Yes	High	No

1. Calculate the entropy of the target variable Y.

The entropy of the target variable Y:

$$Entropy(Y) = -\sum p_i \log_2(p_i)$$

Where p_i is the proportion of instances in each class (y/n)

The target variable Y has the following values:

$$Y = [yes, no, yes, no, yes, no]$$

Class counts: yes – 3, no – 3, total – 6

$$P(yes) = \frac{3}{6} \qquad P(no) = \frac{3}{6}$$

$$Entropy(Y) = -(0.5 \times \log_2(0.5) + (0.5 \times \log_2(0.5))$$

$$Entropy(Y) = -(0.5 \times -1 + 0.5 \times -1)$$

$$Entropy(Y) = 1.0$$

2. Calculate the information gain for each feature (X1, X2, X3) and identify the best feature to split on.

The information gain for a feature is calculated with the following formula:

$$info\ gain(X) = Entropy(Y) - \sum(\frac{instances\ in\ split}{total\ instances} \times\ Entropy\ of\ split)$$

Feature X_1:

$$X_1 = [yes, no, yes, no, yes, no]$$

X_1 *Yes* subset of Y = [yes, yes, yes]

$$p(yes) = 1.0 \qquad p(no) = 0.0 \qquad Entropy = \ 0.0$$

X_1 *No* subset of Y = [no, no, no]

$$p(yes) = 0.0 \qquad p(no) = 1.0 \qquad Entropy = 0.0$$

Weighted entropy for X_1:

$$weighted\ entropy(X_{1)} = \frac{3}{6} \times 0 + \frac{3}{6} \times 0 = 0$$

Information gain for X_1:

$$info\ gain(X_1) = Entropy(Y) - \ 0$$

$$info\ gain(X_1) = 1.0 - \ 0.0 = 1.0$$

Feature X_2:

$$X_2 = [no, yes, yes, no, no, yes]$$

X_2 *Yes* subset of Y = [no, yes, no]

$$p(yes) = \frac{1}{3} \qquad p(no) = \frac{2}{3}$$

$$Entropy = \ -(\frac{1}{3} \times \log_2\left(\frac{1}{3}\right) + (\frac{2}{3} \times \log_2(\frac{2}{3}))$$

$$Entropy = \ -(\frac{1}{3} \times (-1.58496) + \frac{2}{3} \times (-0.58496))$$

$$Entropy \approx 0.918$$

X_2 *No* subset of Y = [yes, no, yes]

$$p(yes) = \frac{2}{3} \qquad p(no) = \frac{1}{3}$$

$$Entropy = -(\frac{2}{3} \times \log_2\left(\frac{2}{3}\right) + (\frac{1}{3} \times \log_2(\frac{1}{3}))$$

$$Entropy = -(\frac{2}{3} \times (-0.58496) + \frac{1}{3} \times (-1.58496))$$

$$Entropy \approx 0.918$$

Weighted entropy for X_2:

$$weighted\ entropy(X_2) = \frac{3}{6} \times 0.918 + \frac{3}{6} \times 0.918 = 0.918$$

Information gain for X_2:

$$info\ gain(X_2) = Entropy(Y) - 0.918$$

$$info\ gain(X_2) = 1.0 - 0.918 \approx 0.0817$$

Feature X_3:

$$X_3 = [high, medium, low, medium, low, high]$$

X_3 *high* subset of Y = [yes, no]

$$p(yes) = \frac{1}{2} \qquad p(no) = \frac{1}{2} \qquad Entropy = 1.0$$

X_3 *medium* subset of Y = [no, no]

$$p(yes) = 0.0 \qquad p(no) = 1.0 \qquad Emtropy = 0.0$$

X_3 *low* subset of Y = [yes, yes]

$$p(yes) = 1.0 \qquad p(no) = 0.0 \qquad Emtropy = 0.0$$

Weighted entropy for X_3:

$$weighted\ entropy(X_3) = \frac{2}{6} \times 1.0 + \frac{2}{6} \times 0.0 + \frac{2}{6} \times 0.0 \approx 0.333$$

Information gain for X_3:

$$info\ gain(X_3) = Entropy(Y) - 0.333$$

$$info\ gain(X_3) = 1.0 - 0.333 \approx 0.6667$$

3. Construct the first split of the decision tree.

To determine the best feature on which to implement the first split of a decision tree we must compare information gain values for all respective features:

$$info\ gain(X_1) = 1.0$$

$$info\ gain(X_2) \approx 0.0817$$

$$info\ gain(X_3) \approx 0.6667$$

As shown above we observe that X_1 is the feature with the highest information gain and therefore this is where the first split should be implemented.