Predicting Mortality in Heart Failure Patients Using Logistic Regression and SVC

1 INTRODUCTION

In recent times, predicting healthcare related diseases and issues has become ever increasingly more important, including the threats which COVID-19 and its variants pose. By analysing medical data, it is possible to recognize occurring patterns and help identify the key sources of diseases and their symptoms, which can help diagnosis, prevention, and treatment, improving survival rates and advancing healthcare providing. In this study, I will be evaluating different learning models to investigate which can predict heart diseases in the most accurate manner; the application domain being medical diagnosis.

The following report will go over problem formulation, the dataset introduction and features, labels used in Section 2. Methods, models, data distribution and feature engineering in Section 3, end results in Section 4, conclusions of the report in Section 5, and finally, references in Section 6. At the end of the document, an appendix for the code is attached.

2 PROBLEM FORMULATION

2.1 PROBLEM

How accurately can we predict the chances of a patient getting a heart disease?

2.2 DATASET

The dataset of different patient's information contains 13 features, along with 299 datapoints (without missing values). Each datapoint represents a patient's medical data.

The properties – or features of the dataset are the following:

- presence of anaemia, presence of diabetes, if the patient has high blood pressure, gender, if they smoke, and if the patient has passed away are all **binary** data.
- age, platelets amount in blood and serum sodium amount in blood are **continuous** data.
- CPK enzyme amount in blood, ejection fraction of heart (percentage), serum creatine amount in blood, serum sodium amount in blood, time follow-up period (of diagnosis, in days) are **numerical** data.

My project aims to prove and explore that heart failure can be predicted from a selected number of features alone, investigating risk factors of heart disease, while also attempting to forecast a patient's chances of dying from heart failure. The dataset can be found on Kaggle [1] and is additionally used in another related research paper [2].

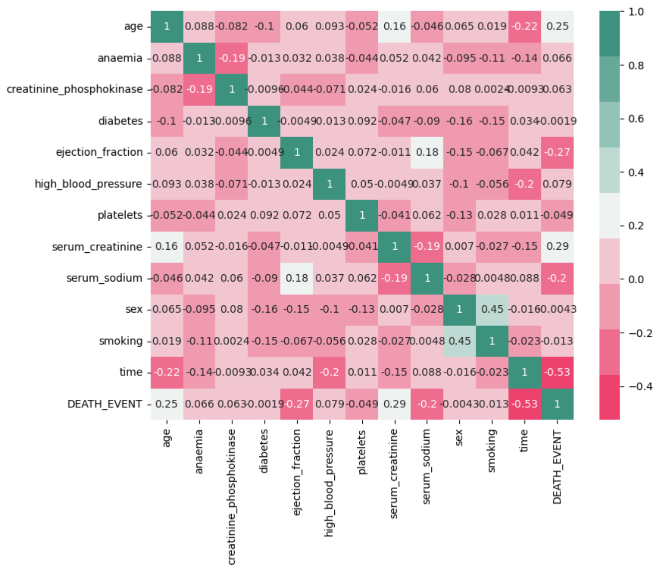
2.3 Features and Labels

The target variable or label is the feature called 'DEATH_EVENT', which indicates if the patient has passed away or not. Therefore, I will be aiming to use a supervised model.

3 Methods

3.1 Feature Engineering – Dataset Splitting

Having looked at the related research paper [2], I've selected features that would correlate the best with my model, having dropped redundant ones onward. The research indicated: "[...] **serum creatinine** and **ejection fraction** are sufficient to predict survival of heart failure patients from medical records [...]". To examine this claim, I have used a correlation matrix further and analysed feature importance to see which features should be retained. Notably, the research paper's findings and the correlation matrix' results are mostly equivalent, as shown in Figure 1 below.



1. Figure The correlation of features, shown in a matrix. In the 'DEATH_EVENT' column, we can observe that time, serum_sodium and ejection_fraction play an important part in whether death incurs.

The dataset will be split according to the 70-20-10 ratio for training-validation-testing, as I aspire the model to be more generalized and robust, unlike the case with the 80-10-10 ratio. Notably, the 70-20-10 is also popular to use on smaller datasets [3], such as the current one, although the chosen ratio can vary a lot depending on context and model. Given the simplicity of the models, I will not be dividing the dataset according to cross validation. The data splitting can be done with the function 'train_test_split()' from the scikit-learn library.

The test set, as stated previously, will be 10% of the original dataset. It will be used to evaluate the final performance of the models later, and it will have no involvement with the training, nor the validation set, as to make the assessment unbiased and genuine.

3.2 Logistic Regression

Firstly, I will be using Logistic Regression, a binary classification model, falling under the category of linear predictor maps, to solve the problem, as the data points can be classified to two categories, whether the patient has died (1) or is alive (0), which is why I will be picking the 'DEATH_EVENT' to be the deciding label of my model. The model itself is ideal due to its simplicity, and since it assumes that there is a linear relationship between features and labels, it is sufficient for my dataset and goal.

For choosing the adequate loss function, I have chosen logistic loss, as that would fit with my model more, and is frequently applied in binary classification problems, adhering to standard practice in the field. Logistic loss effectively measures the difference between predicted probabilities and the true binary outcomes. Additionally, the loss function is chosen as it allowed the use of a ready-made scikit-learn library for logistic regression.

3.3 Support Vector Classification

Support Vector Machines or Support Vector Classifiers (SVC) are models used for binary classification problems, similarly to Logistic Loss, where data points can be divided into two separate categories. SVC works by finding the optimal hyperplane that maximizes the margin of separation between classes, which makes it ideal to examine as a model, or in other words, it is great at handling high dimensional spaces. Like previously done in the Logistic Regression model, whether the patient has died (1) or is alive (0) will be our two classes. SVC excels in the field of dealing with smaller datasets, as its computational complexity correlates to the size of the dataset. Moreover, it is capable of handling both linear and non-linear relationships between features and labels.

The hypothesis space uses the same one as Logistic Regression [4] [Sec. 3.2]. I will be choosing the linear hyperparameter as kernel, as it performs the best, which I've investigated in the code included below.

Using the hinge loss function imported into the scikit-learn library already, it becomes the most straightforward option to apply as a loss function, while also being the most common choice for SVC, as hinge Loss penalizes misclassified points, and those within the margin.

4 Results

To choose the best machine learning method, we must examine the accuracy scores, also known as training-validation-test errors of each model, with each split set.

	Logistic Regression	**Support Vector Classification**
Training Accuracy	*86%*	*87%*
Validation Accuracy	*83%*	*80%*
Test Accuracy	*87%*	*87%*

Initially, looking at the *training accuracy*, both models appear to perform adequately, with only a single percentage difference where SVC performs slightly better than Logistic Regression. By the *validation accuracy*, Logistic Regression performs better (83% against 80%), which suggests that it has a slight advantage in terms of generalisation of the data [4].

Despite this, by utilizing our final test set— which was 10% of the total dataset [Sec. 3.1], independent of the other sets— it can be observed that performance is identical to both, with a *test accuracy* of 87%. For this reason, I will examine the loss errors further to avoid any false conclusions. (Log Loss for Logistic Regression, Hinge Loss for SVC)

	Logistic Regression	**Support Vector Classification**
Training Error Loss	*38%*	*40.7%*
Validation Error Loss	*46.9%*	*54%*
Test Error Loss	*37.9%*	*37.9%*

Looking upon the *validation error loss*, the difference becomes significantly higher between the two, which strengthens the argument that Logistic Regression generalizes the data better, as Logistic Regression has a lower validation error loss (46.9%) compared to SVC (54%).

Nonetheless, interestingly, the *test error losses* stay the same between them, at 37.9%.

5 CONCLUSIONS

The *test accuracy* and *test error losses* of the models are similar, which from we can draw the assumption that both models perform well on new, unseen data. Neither model is overfitting or underfitting; comparably, this makes them both great candidates, but Logistic Regression is slightly preferable: as reflected by the training-validation accuracy, we can derive that it generalizes data better. The smaller gap of 3% for Logistic Regression between training and validation percentages, compared to the larger differences between these two indicators (7%) for the SVC model mean that it is prone to overfitting [4]. This makes **Logistic Regression** more consistent and reliable, a suitable choice.

While the results are promising, they are not completely optimal. To further improve this research, a larger dataset would prove to be useful, as the current amount of datapoints is not sufficient by quantity. Depending on the ratio of the random data-split of the dataset, the accuracy results may vary due to the previous reason, therefore it is not noticing the underlying patterns in the data effectively. But, in future uses, a k-fold cross-validation method would prove to make a robust analysis, evening out any fluctuation in accuracy. Furthermore, other models, such as Random Forests (due to e.g. the model providing insights into feature importance), and K-Nearest Neighbours (due to e.g. the model not relying too much on data distribution) could potentially prove to perform better than the previous two models, as both are used for classification problems.

In conclusion, patient mortality can be predicted with reasonable validity, however, it calls for a need of refinement, should it be applicable to real-world scenarios for diagnosis. It is important to note that using these models alone is not sufficient, and there can be false negatives upon evaluation; it is more useful as a pre-emptive measure or tool to highlight to doctors which patients would need more attention, should it be overlooked by previous diagnoses.

6 REFERENCES, SOURCES

[1] Heart Failure Prediction Dataset (2020) sourced from UC Irvine Machine Learning Repository https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data

[2] 'Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone' (2020 February 3) by D. Chicco, Giuseppe Jurman https://www.semanticscholar.org/paper/Machine-learning-can-predict-survival-of-patients-Chicco-Jurman/e64579d8593140396b518682bb3a47ba246684eb

[3] Article on 'What is data splitting?' by Alexander S. Gillis 'https://www.techtarget.com/searchenterpriseai/definition/data-splitting#:~:text=Data%20should%20be%20split%20so,optimal%20for%20small%20data%20sets.

[4] 'Machine Learning: The Basics' (2024 September 21) by Alexander Jung https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

Appendix

October 5, 2024

[2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns  #data visualization library
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report  # evaluation metrics
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import log_loss, hinge_loss

df = pd.read_csv('heart_failure_clinical_records_dataset.csv')
df.head(5)
```

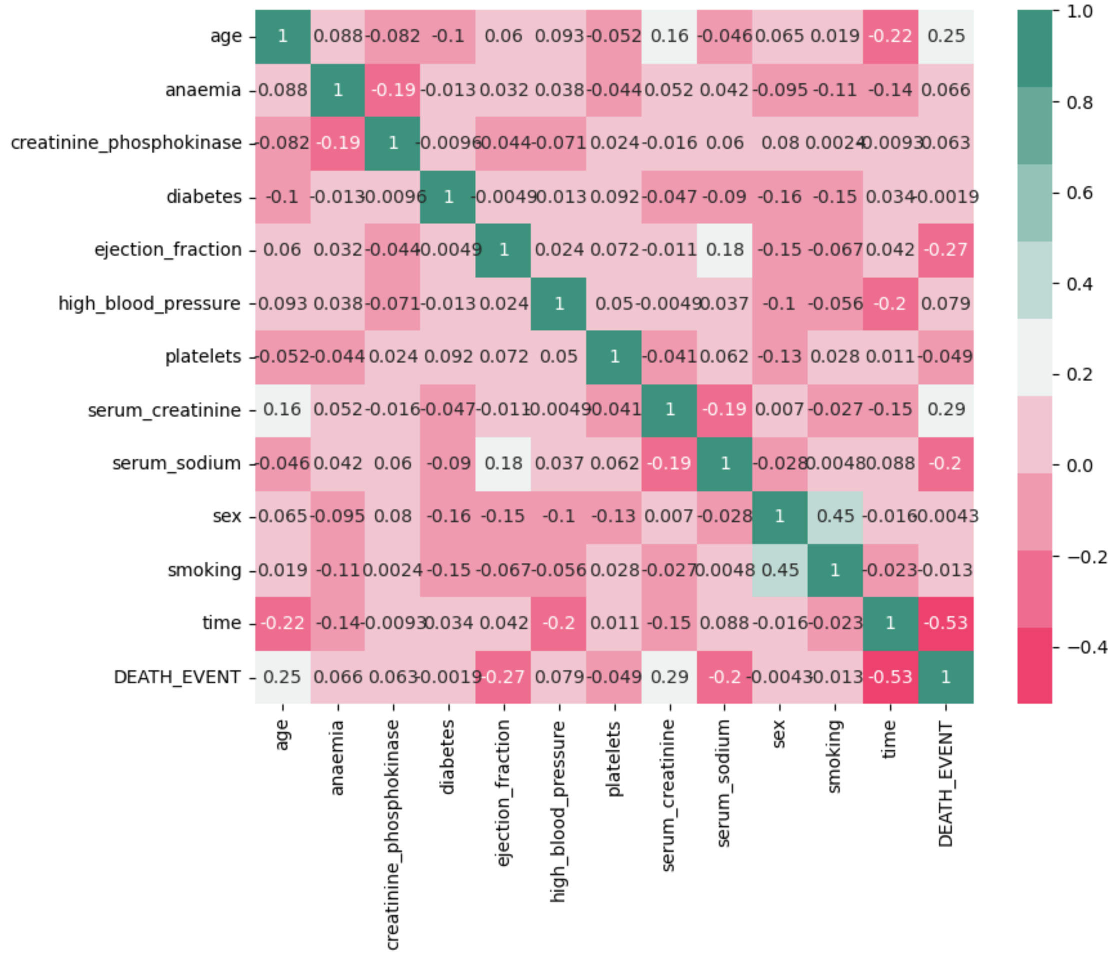
[2]:
```
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
0  75.0        0                       582         0                 20
1  55.0        0                      7861         0                 38
2  65.0        0                       146         0                 20
3  50.0        1                       111         0                 20
4  65.0        1                       160         1                 20

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                    1  265000.00               1.9           130    1
1                    0  263358.03               1.1           136    1
2                    0  162000.00               1.3           129    1
3                    0  210000.00               1.9           137    1
4                    0  327000.00               2.7           116    0

   smoking  time  DEATH_EVENT
0        0     4            1
1        0     6            1
2        1     7            1
3        0     7            1
4        0     8            1
```

[3]:
```python
# Correlation matrix
corr_matrix = df.corr()
```

```
cmap = sns.diverging_palette(2, 165, s=80, l=55, n=9)
plt.figure(figsize=(9,7))
sns.heatmap(corr_matrix, annot=True, cmap=cmap)
plt.show()
```



0.1 Logistic Regression

[4]:
```
#only taking into account the serum_creatine, ejection fraction and time factors
```

[5]:
```
features = ['time', 'ejection_fraction','serum_creatinine']
X = df[features]
y = df['DEATH_EVENT']
```

[6]:
```
#X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,␣
 ↪random_state=4)## <-- previous code from stage1
# 90-10 at first
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
 ↪1, random_state=4)
#then we split the remaining 20 from 90, as 90 * 0.2222 = 19.998 (not exact)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.2222, random_state=44)

#X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
 ↪2, random_state=4) 60 20 20
#X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.25, random_state=42)
```

[7]:
```
logregmodel = LogisticRegression()
logregmodel.fit(X_train, y_train)

y_pred_train = logregmodel.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)
print(f"Train Accuracy: {train_accuracy:.2f}")

y_val_pred = logregmodel.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {val_accuracy:.2f}")

y_test_pred = logregmodel.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.2f}")

confmat = confusion_matrix(y_val, y_val_pred)
print("Confusion Matrix:\n", confmat)

y_train_prob = logregmodel.predict_proba(X_train)
y_val_prob = logregmodel.predict_proba(X_val)
y_test_prob = logregmodel.predict_proba(X_test)

train_log_loss = log_loss(y_train, y_train_prob)
val_log_loss = log_loss(y_val, y_val_prob)
test_log_loss = log_loss(y_test, y_test_prob)

print("Training Log Loss:", train_log_loss)
print("Validation Log Loss:", val_log_loss)
print("Test Log Loss:", test_log_loss)
```

```
Train Accuracy: 0.86
Validation Accuracy: 0.83
Test Accuracy: 0.87
```

```
Confusion Matrix:
 [[32  4]
 [ 6 18]]
Training Log Loss: 0.3803872165432644
Log Loss: 0.46942344928618945
Test Log Loss: 0.37934672531631014
```

Investigation of putting emphasis on getting more false positives. - discarded due to worse accuracy and non reducable amount of false negatives

[8]:
```python
logregmodel = LogisticRegression(class_weight={0: 0.2, 1: 1})
logregmodel.fit(X_train, y_train)

y_pred_train2 = logregmodel.predict(X_train)
train_accuracy2 = accuracy_score(y_train, y_pred_train2)
print(f"Train Accuracy: {train_accuracy2:.2f}")

y_val_pred2 = logregmodel.predict(X_val)
val_accuracy2 = accuracy_score(y_val, y_val_pred2)
print(f"Validation Accuracy: {val_accuracy2:.2f}")

y_test_pred2 = logregmodel.predict(X_test)
test_accuracy2 = accuracy_score(y_test, y_test_pred2)
print(f"Test Accuracy: {test_accuracy2:.2f}")

confmat2 = confusion_matrix(y_val, y_val_pred2)
print("Confusion Matrix:\n", confmat2)
```

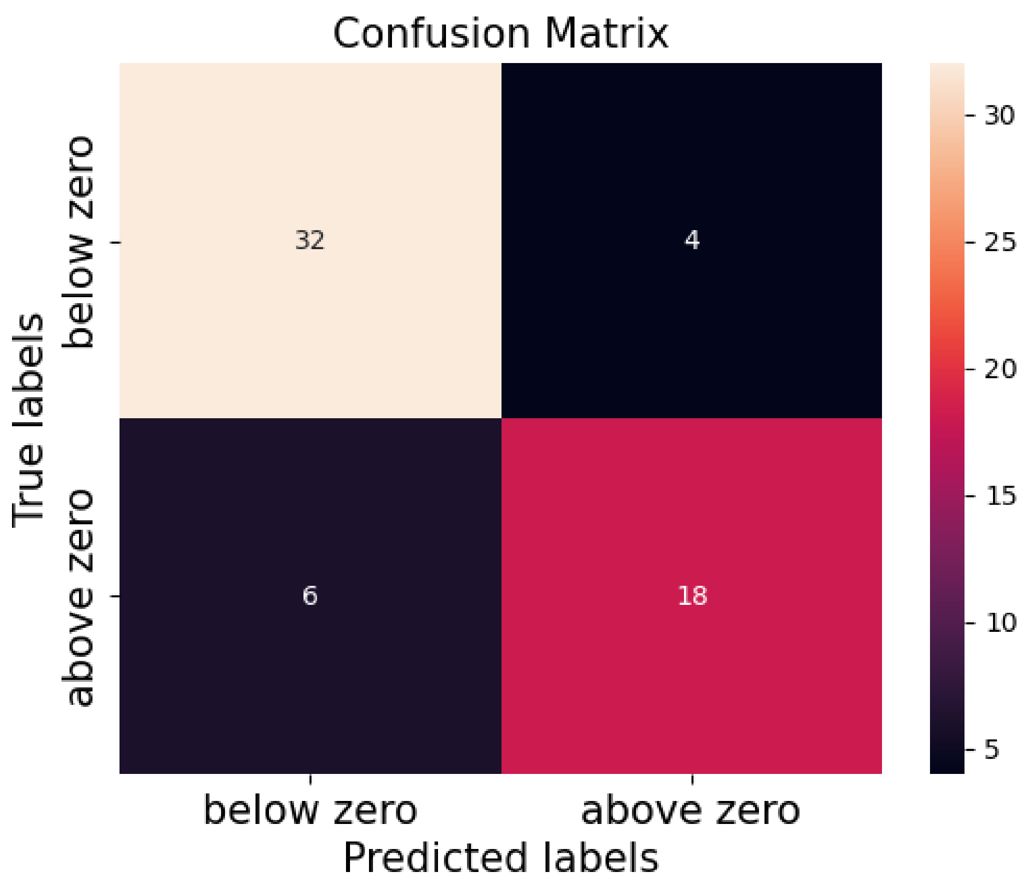
```
Train Accuracy: 0.72
Validation Accuracy: 0.70
Test Accuracy: 0.73
Confusion Matrix:
 [[20 16]
 [ 2 22]]
```

[26]:
```python
ax= plt.subplot()

sns.heatmap(confmat, annot=True, fmt='g', ax=ax )

ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title('Confusion Matrix',fontsize=15)
ax.xaxis.set_ticklabels(['below zero', 'above zero'],fontsize=15)
ax.yaxis.set_ticklabels(['below zero', 'above zero'],fontsize=15)
```

[26]:
```
[Text(0, 0.5, 'below zero'), Text(0, 1.5, 'above zero')]
```



```
[ ]:
```

0.2 Support Vector Machine

Let's examine which kernel would work the best

```
[59]: kernels = ['linear', 'poly', 'rbf', 'sigmoid']

      testaccuracies = {}
      for kernel in kernels:
          model = SVC(kernel=kernel)
          model.fit(X_train, y_train)
          y_pred = model.predict(X_test)
          test_accuracy = accuracy_score(y_test, y_pred)
          testaccuracies[kernel] = test_accuracy

      for kernel, result in testaccuracies.items():
          print(f"Kernel: {kernel}")
          print(f"  Test Accuracy: {result:.4f}")
```

```
    print("----------")
```

```
Kernel: linear
  Test Accuracy: 0.8667
----------
Kernel: poly
  Test Accuracy: 0.8333
----------
Kernel: rbf
  Test Accuracy: 0.8667
----------
Kernel: sigmoid
  Test Accuracy: 0.7667
----------
```

The linear/rbf one will suffice for our model.

[16]:
```
svm = SVC(kernel='linear', C=1.0)

svm.fit(X_train, y_train)

y_pred_train = svm.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)
print(f"Train Accuracy: {train_accuracy:.2f}")

y_val_pred = svm.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {val_accuracy:.2f}")

y_test_pred = svm.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Test Accuracy: {test_accuracy:.2f}")

confmat = confusion_matrix(y_val, y_val_pred)
print("Confusion Matrix:\n", confmat)

y_train_decision = svm.decision_function(X_train)
y_val_decision = svm.decision_function(X_val)
y_test_prob = logregmodel.predict_proba(X_test)

train_hinge_loss = hinge_loss(y_train, y_train_decision)
val_hinge_loss = hinge_loss(y_val, y_val_decision)
test_log_loss = log_loss(y_test, y_test_prob)

print("Training Hinge Loss:", train_hinge_loss)
print("Validation Hinge Loss:",val_hinge_loss)
print("Test Log Loss:", test_log_loss)
```

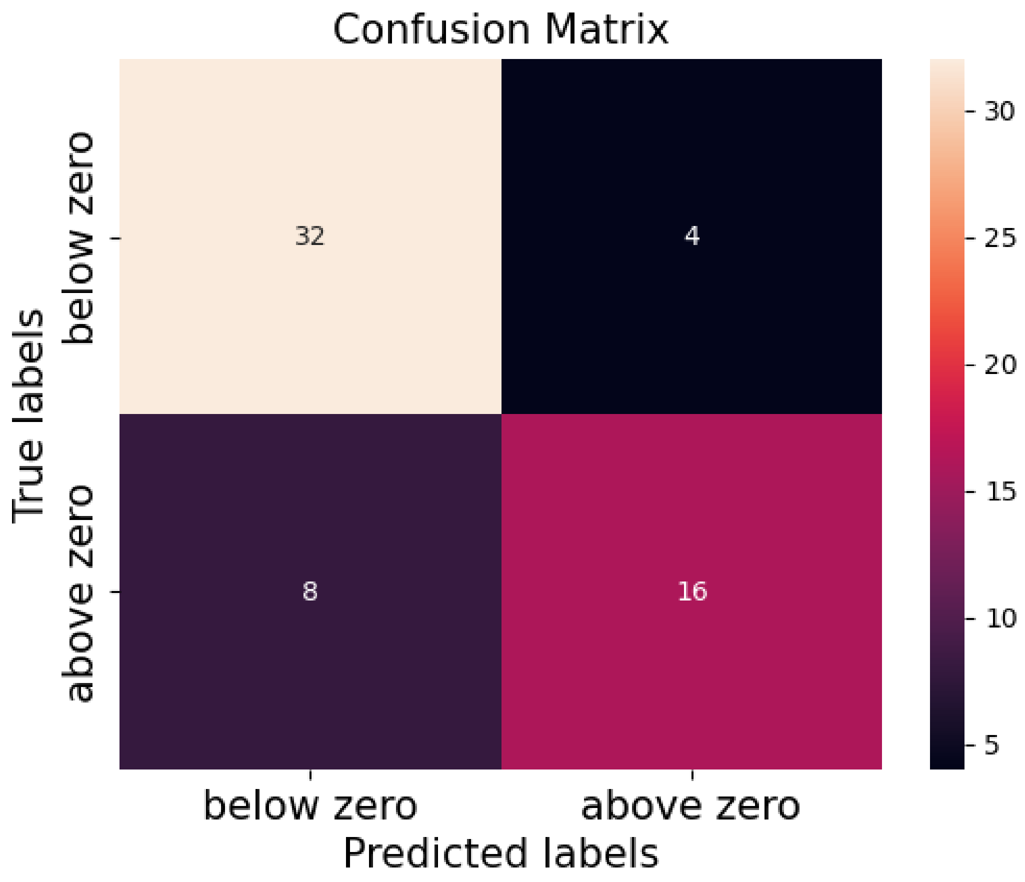
```
Train Accuracy: 0.87
Validation Accuracy: 0.80
Test Accuracy: 0.87
Confusion Matrix:
 [[32  4]
 [ 8 16]]
Training Hinge Loss: 0.4078068034036303
Validation Hinge Loss: 0.5405625252836793
Test Log Loss: 0.6091824228257186
```

[17]:
```
ax= plt.subplot()

sns.heatmap(confmat, annot=True, fmt='g', ax=ax )

ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title('Confusion Matrix',fontsize=15)
ax.xaxis.set_ticklabels(['below zero', 'above zero'],fontsize=15)
ax.yaxis.set_ticklabels(['below zero', 'above zero'],fontsize=15)
```

[17]:
```
[Text(0, 0.5, 'below zero'), Text(0, 1.5, 'above zero')]
```


Confusion Matrix
below zero
above zero
True labels
below zero
above zero
Predicted labels