Predicting Lung Cancer with Machine Learning

October 9, 2022

1 Introduction

Lung cancer is the most common form of cancer worldwide. Over 200,000 new cases are diagnosed each year, accounting for 13% of all cancer diagnoses. Almost half of patients with lung cancer die within a year of diagnosis. As with all cancers, early diagnosis dramatically increases survival rates. Earlier stages of lung cancer are easier to treat and can reduce mortality rates by 14% to 20%[1]. Thus, the application of machine learning seems to be a reasonable use case for the diagnosis of lung cancer as it can achieve high effectiveness.

In this paper, I will try to determine whether a patient with certain features has lung cancer or not.

1.1 Report structure

The report continues with section 2, *"Problem statement"*, where the data set is introduced and all data points are clarified. Section 3, *"Methods"*, explains how and why the feature selection was made and how the datasets were constructed. It also discusses the two machine learning methods, *Logistic Regression* and *Support Vector Machine (SVM)*. In section 4, *Results*, the results are analysed and in section 5, *Conclusion*, a final conclusion is drawn.

2 Problem Formulation

2.1 Data set

The data set consists of 309 entries in an Excel spreadsheet, with patients between 21 and 87 years old. Each data point describes a patient with different characteristics. The data set is licensed under the *CCO: Public Domain* and was found on *Kaggle* [2]. For transparency, this dataset is used in another project found *here* [3], but as this work was done independently, it has not been copied in any way.

The dataset is complete and has no missing features. Each data point has 16 different attributes associated with it (gender, age, smoking, yellow fingers, chronic diseases, *see appendix for more...*). I was also able to find 33 duplicates, which I eliminated. As you may have noticed, the dataset is not too large and contains only 276 entries. How this problem was fixed is explained in more detail in the section "Splitting the data".

2.2 Label and feature clarification

The label for this prediction is whether a patient has lung cancer or not (yes/no), which means that this problem is a classification problem. I decided to change the dataset for the labels from *yes* and *no* to *-1* and *1* to make the prediction easier and because I will use the SVC hinge loss function in the second part. All attributes except gender and age are represented with *1* or *2*, which corresponds to *no* and *yes* respectively. The gender is represented with *M* or *F*, which means male or female respectively. The age is the actual age as an integer. The histogram shows that the data is largely balanced, except for a few outliers such as *fatigue* and *lung cancer*. Since I will be using Logistic Regression, these outliers will not significantly affect our results [4][5].

3 Methods

3.1 Feature selection and engineering

For the features, I selected all attributes except for the lung cancer column, as it is used for the label. For the selected features, I have changed the data points from *1* and *2* to *0* and *1* for convention purposes and ease of understanding. It is important to note here that with the exception of age, all characteristics can be represented in binary, meaning, a normalisation is not required. After looking at the correlation heat map of the characteristic, I could not identify any bad data sets. All data points appear to be in the range of -0.75 and 0.75, which is a common indicator of correlation. Also, all the identifiers, gender, age, smoking, yellow fingers, (...) seem intuitively important in determining whether a patient has lung cancer or not. After some manual tests with *Variance Inflation Factor (VIF)* I could confirm that all 15 characteristics seem to improve the accuracy of the prediction.

3.2 Data Splitting

At first the data is split into two parts, the validation/training set and the test set. I chose the ratio to be 80 to 20 which was discussed in the lecture. This helps us to split the whole data into two parts so that the test set is not used in either the training set or the validation set. This was achieved by using the *train_test_split* function of the *sklearn* library. The remaining part (80%) of the dataset will be used to train and validate the machine learning models. As mentioned earlier, the dataset is not very large. I decided to use the k-fold cross-validation method. This method is suitable for data sets with fewer entries, to split it up into training and validation sets. The algorithm works by randomly dividing the datasets into equal parts and using one set as the test set and the rest as the training set. I decided to split the data into 5 parts (k=5) to achieve an 80 to 20 (training to validation) ratio, which is very common. Our goal is to classify whether a patient has lung cancer. This can be described as a binary classification problem.

3.3 Logistic Regression

For the first machine learning model, I chose Logistic Regression because it seemed reasonable for a simple binary classification problem. Logistic Regression uses a linear hypothesis space and works by setting a limiter between the data points. In a 2D space,

you would put a line between the given data points. In 3D space, the points would be separated by a plane, and in higher dimensions you would use a hyperplane to describe the separation.

3.3.1 Loss function for Logistic Regression

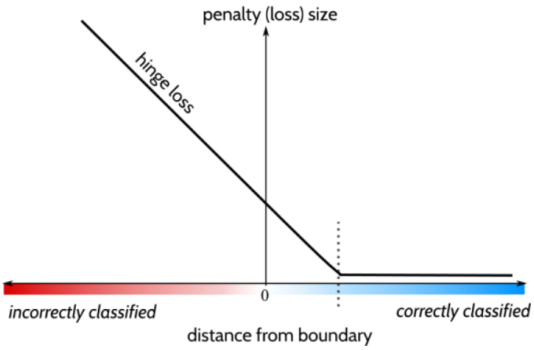
For the loss function, I chose the Logistic Loss function, as it is the common and proven function for Logistic Regression. Also it was easy to use as it is already implemented in the used library.

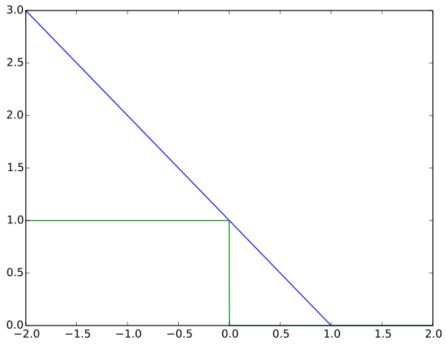
3.4 Support Vector Machine (SVM)

For the second machine learning model, we chose to use the Support Vector Machine (SVM) classifier class, Support Vector Classification (SVC), for this simple binary classification task. For SVC, it also uses a linear hypothesis space that maps $h(x) = w^{T(x)}$, identical to that of the Logistic Regression method used previously. The decision to opt for this method was so that we can compare the classification methods and evaluate which has better performance. We did not choose to make use of LinearSVC or SGDClassifier over the basic SVC due to the dataset being not too large.

3.4.1 Loss function for SVC

We also decided on using hinge loss to calculate the loss of our SVC method. The motivation for making use of hinge loss as the function for this method hinge loss function, is mainly because it is widely known as the loss function tailored for SVM and also is easily accessible with sklearn.metrics. Furthermore, it also makes sense to use this loss function for this binary classification task. A visualization and representation of the hinge loss function is shown in the following figures.



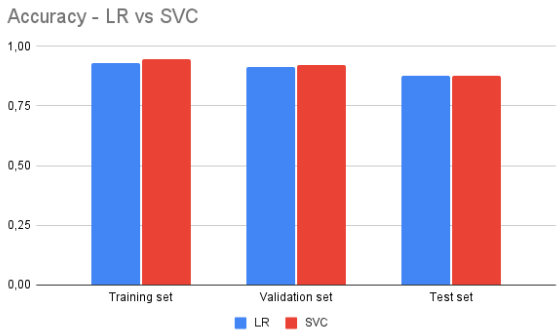


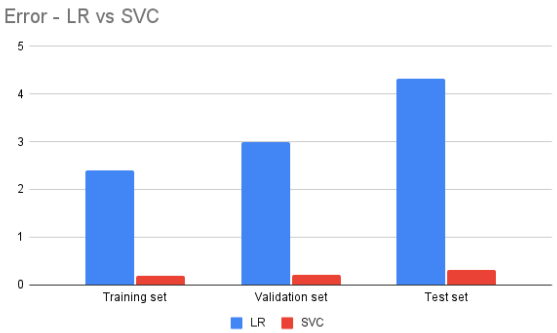
From these figures, we can see that for observations that are of a margin distance of greater than or equal to 1, the hinge loss is valued at zero. While for observations of margin distance less than 1, the hinge loss value incurs a loss the increases linearly. To put simply, while the SVC bears the similarity with Logistic Regression in that it aims to separate both classes with a line, the difference lies in this hinge loss function, that aims to maximize the margin distance between each data point and the separating line.

4 Results

To evaluate and compare the two models, we have calculated the errors and the accuracy scores for each training, validation and test set, obtaining the following results shown in

the charts below.





Accuracy	LR	SVC
Training set	0,9306818182	0,9465909091
Validation set	0,9136363636	0,9227272727
Test set	0,875	0,8785714286

Error	LR	SVC
Training set	2,394204253	0,1806818182
Validation set	2,982945208	0,2045454545
Test set	4,317415586	0,3

As we can see from the charts and table above, both methods performed well in this binary classification to predict persons with lung cancer, with the training, validation and test accuracies for both Logistic Regression and SVC valued around 90%. Additionally, from our results, we can clearly see how the absolute errors for SVC using hinge loss is significantly smaller than that of Logistic Regression. From these results, we therefore come to the decision that SVC is the better method for this prediction of lung cancer binary classification task.

5 Conclusion

In conclusion, we were able to predict, with an accuracy of 87.9%, for the test set, wether a patient with certain symptoms has lungcancer or not.

A dataset with 16 different parameters was used to make out the prediction. The data was split into training, validation and a testset, with usage of the k-fold method to increase accuarcy, since the dataset was not too big with 276 entries. The accuracy was determined via the average of all the foldings. After comparing Logistic Regression with SVM, we found out that SVM is more suitable. It can be observed that for all the datasets, SVC performed better than the Logistic Regression model, with a 94.7% training, 92.3% validation and 87.9% test accuracy, compared to the 93.1% training, 91.4% validation and 97.5% accuracy of the latter.

Nevertheless, an accuracy of 87.9% may not be sufficient to reliably detect lung cancer. Even a few misdiagnoses of whether a patient may not have lung cancer can be fatal. Therefore, the prediction of lung cancer needs to be further improved. Testing different machine learning models could provide more insight into which model is best suited for this classification problem. In addition, a larger data set would likely further improve the accuracy of the prediction.

6 References

- [1] Lung cancer fact sheet website: https://www.lung.org/lung-health-diseases/lung-disease-lookup/lung-cancer/resource-library/lung-cancer-fact-sheet
- [2] Data set from Kaggle: https://www.kaggle.com/datasets/mysarahmadbhat/lung-cancer
- [3] Other Kaggle project with same data set: https://www.kaggle.com/code/gaganmaahi224/lung-cancer-5ml-models-full-analysis-plotly
- [4] How to handle unbalanced sets tutorial : https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html
- [5] Unbalanced data in Logistic Regression: https://stats.stackexchange.com/questions/6067/does-an-unbalanced-sample-matter-when-doing-logistic-regression
- [6] Heatmap Tutorial *Medium* an seaborn library: https://medium.com/@szabo.bibor/how-to-\\create-a-seaborn-correlation-heatmap-in-python-834c0686b88e

7 Code Appendics

Predicting Lung Cancer with Machine Learning

October 9, 2022

[ ]:
```
import pandas as pd
%config Completer.use_jedi = False
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, log_loss, hinge_loss
from sklearn.model_selection import train_test_split, KFold
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.svm import SVC
```

[ ]:
```
# Read in the data stored in the file 'survey lung cancer.csv'
df = pd.read_csv('survey lung cancer.csv')
df.head(5)
```

[ ]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	\
0	M	69	1	2	2	1	
1	M	74	2	1	1	1	
2	F	59	1	1	1	2	
3	M	63	2	2	2	1	
4	F	63	1	2	1	1	

	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	\
0	1	2	1	2	2	2	
1	2	2	2	1	1	1	
2	1	2	1	2	1	2	
3	1	1	1	1	2	1	
4	1	1	1	2	1	2	

	SHORTNESS OF BREATH	SWALLOWING DIFFICULTY	CHEST PAIN	LUNG_CANCER
0	2	2	2	YES
1	2	2	2	YES
2	2	1	2	NO
3	1	2	2	NO
4	2	1	1	NO

[ ]:
```
# replace all M/F with 1/0
df.GENDER.replace(['M', 'F'], [1, 0], inplace=True)

# replace all YES/NO with 1/0
df.LUNG_CANCER.replace(['YES', 'NO'], [1, 0], inplace=True)
```

1 Data analysis

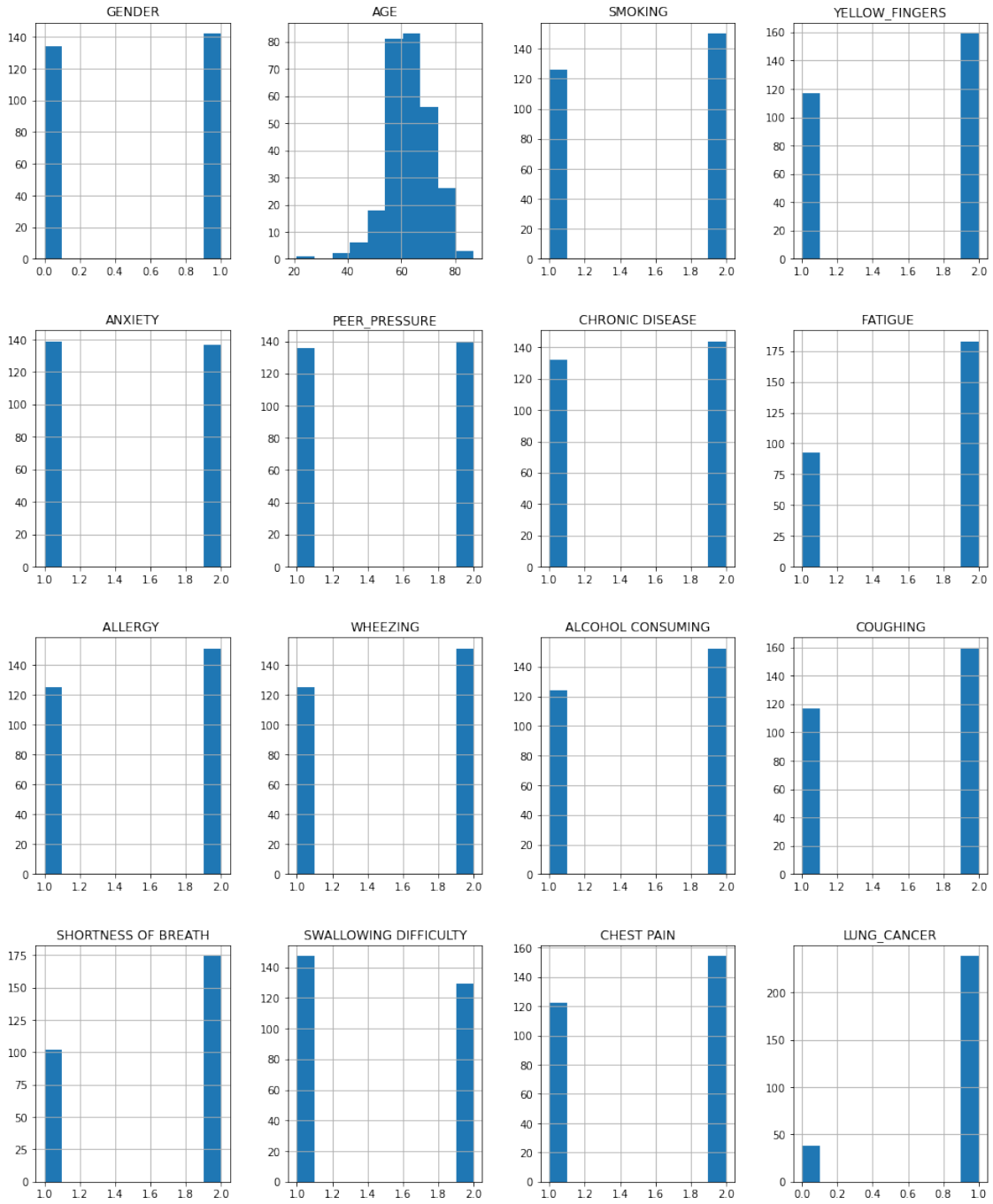
[ ]:
```
print("Number of duplicates: ", df.duplicated().sum())
df = df.drop_duplicates()
print("Number of duplcates after drop ", df.duplicated().sum())
```

```
Number of duplicates:  33
Number of duplcates after drop  0
```

1.1 Histrogram

[ ]:
```
# Data histogram
df.hist(figsize=(16,20))
```

[ ]:
```
array([[<AxesSubplot:title={'center':'GENDER'}>,
        <AxesSubplot:title={'center':'AGE'}>,
        <AxesSubplot:title={'center':'SMOKING'}>,
        <AxesSubplot:title={'center':'YELLOW_FINGERS'}>],
       [<AxesSubplot:title={'center':'ANXIETY'}>,
        <AxesSubplot:title={'center':'PEER_PRESSURE'}>,
        <AxesSubplot:title={'center':'CHRONIC DISEASE'}>,
        <AxesSubplot:title={'center':'FATIGUE '}>],
       [<AxesSubplot:title={'center':'ALLERGY '}>,
        <AxesSubplot:title={'center':'WHEEZING'}>,
        <AxesSubplot:title={'center':'ALCOHOL CONSUMING'}>,
        <AxesSubplot:title={'center':'COUGHING'}>],
       [<AxesSubplot:title={'center':'SHORTNESS OF BREATH'}>,
        <AxesSubplot:title={'center':'SWALLOWING DIFFICULTY'}>,
        <AxesSubplot:title={'center':'CHEST PAIN'}>,
        <AxesSubplot:title={'center':'LUNG_CANCER'}>]], dtype=object)
```

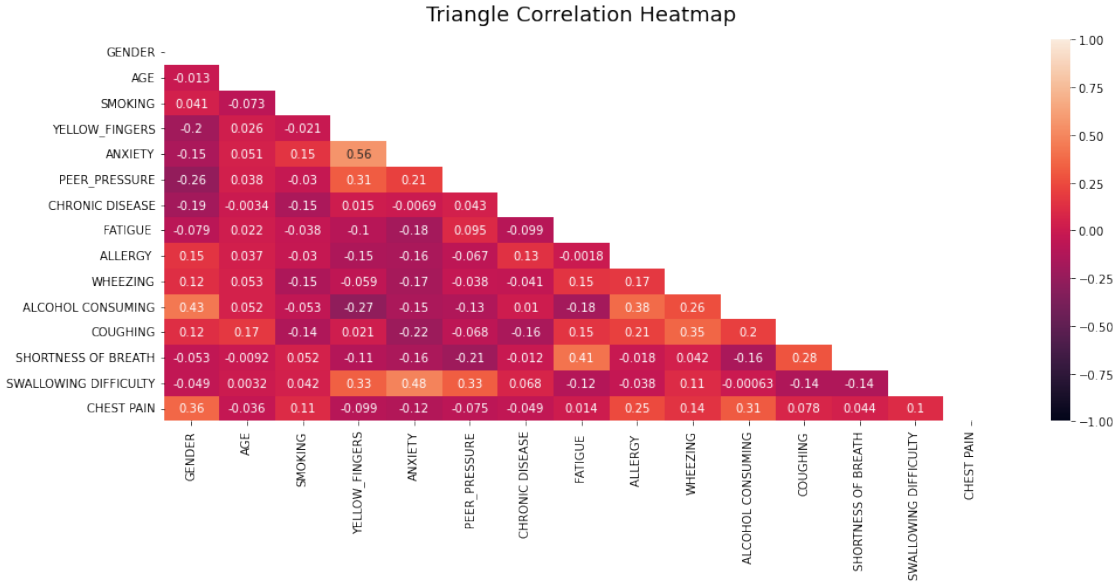


1.2 Heatmap

[ ]:
```
# create X and y sets
X = df.drop(columns="LUNG_CANCER").to_numpy()
X_ht = df.drop(columns="LUNG_CANCER", axis= 1)
y = df['LUNG_CANCER'].to_numpy().reshape(-1,)
```

[ ]:
```
# Heatmap of dataframe
plt.figure(figsize=(16, 6))
mask = np.triu(np.ones_like(X_ht.corr()))
htmap = sns.heatmap(X_ht.corr(),vmin= -1, vmax= 1,  annot= True , mask = mask)
htmap.set_title('Triangle Correlation Heatmap', fontdict={'fontsize':18},␣
↪pad=16);

#https://medium.com/@szabo.bibor/
↪how-to-create-a-seaborn-correlation-heatmap-in-python-834c0686b88e
```



1.3 Virtual Inflation Factor VIF

Was just used for some manual testing

[ ]:
```
X_t = df.drop(columns="LUNG_CANCER", axis=1)
vif_data = pd.DataFrame()
vif_data["feature"] = X_t.columns
vif_data["VIF"] = [variance_inflation_factor(X_t.values, i) for i in␣
↪range(len(X_t.columns))]
print(vif_data.sort_values(by = ["VIF"]))

#https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/
```

```
                feature        VIF
0                GENDER   3.012213
2               SMOKING  10.738372
6     CHRONIC DISEASE    10.896006
8               ALLERGY   13.490502
```

```
5           PEER_PRESSURE  13.564503
9               WHEEZING  13.986291
14            CHEST PAIN  14.003207
13  SWALLOWING DIFFICULTY  15.218347
11              COUGHING  17.344194
10     ALCOHOL CONSUMING  17.545363
12   SHORTNESS OF BREATH  17.666498
7               FATIGUE   17.770848
3         YELLOW_FINGERS  19.138132
4                ANXIETY  19.508213
1                    AGE  41.961929
```

2 Machine Learning part

2.1 Data splitting

[ ]:
```
# splitting out the test set, to be used later in comparing models
X_reduced, X_test, y_reduced, y_test = train_test_split(X, y, test_size=0.2,␣
↪shuffle=True)
```

2.2 Logistic Regression

[ ]:
```
# Spitting the data in 5 sets
kf = KFold(n_splits=5, shuffle=True, random_state=69)

#results for train error
train_res_acc = []
train_res_err = []

# results for validation set
val_res_acc = []
val_res_err = []

# results for test set
test_res_acc = []
test_res_err = []

# iteration for k-fold
for train_index, val_index in kf.split(X_reduced):
    X_train, X_val= X_reduced[train_index], X_reduced[val_index]
    y_train, y_val= y_reduced[train_index], y_reduced[val_index]

    # make logistic regression model
    linreg = LogisticRegression(max_iter=100000)
    linreg.fit(X_train, y_train)
```

```
    # predict training set
    # calculate accuracy and logistic loss
    y_pred_train = linreg.predict(X_train)
    train_acc = accuracy_score(y_train, y_pred_train)
    train_err = log_loss(y_train, y_pred_train)

    # predict validation set
    # calculate accuracy and logistic loss
    y_pred_val = linreg.predict(X_val)
    val_acc = accuracy_score(y_val, y_pred_val)
    val_error = log_loss(y_val, y_pred_val)

    # predict test set
    # calculate accuracy and logistic loss
    y_pred_test = linreg.predict(X_test)
    test_acc = accuracy_score(y_test, y_pred_test)
    test_error = log_loss(y_test, y_pred_test)


    # append train results for later
    train_res_acc.append(train_acc)
    train_res_err.append(train_err)

    # append validation results for later
    val_res_acc.append(val_acc)
    val_res_err.append(val_error)

    # append test results for later
    test_res_acc.append(test_acc)
    test_res_err.append(test_error)




# print results
print("Logistic Regression")
print("Training accuracy: ", np.mean(train_res_acc))
print("Training error: ", np.mean(train_res_err))
print("Validation accuracy: ", np.mean(val_res_acc))
print("Validation error: ", np.mean(val_res_err))
print("Test accuracy: ", np.mean(test_res_acc))
print("Test error: ", np.mean(test_res_err))
```

```
Logistic Regression
Training accuracy:  0.9318181818181819
Training error:  2.354952917718011
```

```
Validation accuracy:  0.9090909090909092
Validation error:  3.1399396466465297
Test accuracy:  0.8892857142857142
Test error:  3.8240102063407724
```

2.3 SVC

[ ]:
```
# Spitting the data in 5 sets
kf = KFold(n_splits=5, shuffle=True, random_state=69)

#results for train error
train_res_acc2 = []
train_res_err2 = []

# results for validation set
val_res_acc2 = []
val_res_err2 = []

# results for test set
test_res_acc2 = []
test_res_err2 = []

# iteration for k-fold
for train_index, val_index in kf.split(X_reduced):
    X_train, X_val= X_reduced[train_index], X_reduced[val_index]
    y_train, y_val= y_reduced[train_index], y_reduced[val_index]


    #make SVC model
    svc = SVC(gamma='auto',kernel='linear')
    svc.fit(X_train,y_train)

    # predict training set
    # calculate accuracy and logistic loss
    y_pred_train2 = svc.predict(X_train)
    train_acc2 = accuracy_score(y_train, y_pred_train2)
    train_err2 = hinge_loss(y_train, y_pred_train2)

    # predict validation set
    # calculate accuracy and logistic loss
    y_pred_val2 = svc.predict(X_val)
    val_acc2 = accuracy_score(y_val, y_pred_val2)
    val_error2 = hinge_loss(y_val, y_pred_val2)

    # predict test set
    # calculate accuracy and logistic loss
    y_pred_test2 = svc.predict(X_test)
```

```
    test_acc2 = accuracy_score(y_test, y_pred_test2)
    test_error2 = hinge_loss(y_test, y_pred_test2)



    # append train results for later
    train_res_acc2.append(train_acc2)
    train_res_err2.append(train_err2)

    # append validation results for later
    val_res_acc2.append(val_acc2)
    val_res_err2.append(val_error2)

    # append test results for later
    test_res_acc2.append(test_acc2)
    test_res_err2.append(test_error2)

# print results
print("SVC")
print("Training accuracy: ", np.mean(train_res_acc2))
print("Training error: ", np.mean(train_res_err2))
print("Validation accuracy: ", np.mean(val_res_acc2))
print("Validation error: ", np.mean(val_res_err2))
print("Test accuracy: ", np.mean(test_res_acc2))
print("Test error: ", np.mean(test_res_err2))
```

```
SVC
Training accuracy:  0.9431818181818181
Training error:  0.19772727272727275
Validation accuracy:  0.9181818181818182
Validation error:  0.22272727272727272
Test accuracy:  0.8821428571428571
Test error:  0.24285714285714283
```

[ ]:

[ ]: