RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

UNIVERSITÉ CÔTE D'AZUR

STATISTICAL LEARNING PROJECT

Prof. Sylvain Rubenthaler

Bhargav Ramudu Manam

Victor Enyinnaya Orji

Odunayo Olajumoke Okebunmi

Toluwaloju Rotimi

OJO Suleiman Adebowale

8. mars 2022

1 Introduction

What is Machine Learning?

> A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
> -Tom Mitchell, 1997

1.1 Multi-Class Classification:

Classification is a task that requires the use of machine learning algorithms that learn how to assign a class label to examples from the problem domain. When we have only 2 possible labels, we say that we want to solve a binary classification problem. When we have more than 2 possible labels, we say that we want to solve a multi-class classification problem.
Multi-Class classification problems can be solved by a variety of machine learning algorithms ranging from Logistic Regression to deep learning networks. Not all algorithms are capable of handling multiple classes natively, algorithms like Logistic Regression, and Support Vector Machine were designed for binary classification and do not natively support classification tasks with more than two classes. There are various techniques one could use to perform multi-class classification with these algorithms.
Such strategies are:

- One-versus-one (OVO).
- One-Versus-Rest (OVR).

1.2 Problem Statement:

In this project, we will be training 5 classifiers, one for each classes(C_0, C_1, C_2, C_3, C_4). To classify an observation, we try to get the decision score from each classifier for that observation and choose the class whose classifier has the highest score. We will try this strategy on the following algorithms: Logistics regression, Support Vector Machine(SVM), Kernelized SVM and lastly neural networks (NN). We also compare which algorithm performs best by evaluating the accuracy, confusion matrix, precision, recall and F1 score depending on the data volume (number of samples and features) and data quality (outliers and imbalanced data).

2 Data Overview

In this project, we will be using the Urban Tree Attribute Schema dataset, which is collected by activists and science students from Saint-Germain-en-Laye. The dataset contains information on the location, species, dimensions, specificities and state of health of the tree heritage of the municipality of Saint-Germain-en-Laye. The target variable is classification diagnostic, each tree is labeled with classes from C_0 to C_4. The goal is to try various classification algorithms mentioned in the introductory section using the "one vs all" strategy.

The sample of 709 instances in the dataset is considered small by the machine learning standards. The **cote-voirie** has 545 non-null and **controle** has 11 non-null and **situation** has 708 non-null. Each data has 57 attributes. Following the instructions, all the columns that come after **Q** th column are erased.

```
df=pd.read_csv('/content/drive/MyDrive/colab/PROJECT/sgl-arbres-urbains-wgs84.csv')
dataset= df.iloc[:,:17]
#rENAME COLUMNS
dataset.rename(columns={"ID_ARBRE":"TREE_ID", "commune":"municipality", "quartier":"district",
                        "cote_voirie":"road_side","matricule_arbre":"tree_number",
                        "genre_arbre":"tree_genus", "espece_arbre":"tree_species",
                        "controle":"control","type_sol":"soil_type", "surf_permeable":"permeable_surf",
                  "date_plantation":"planting_date","classe_age":"age_class", "hauteur":"height",
                  "classe_hauteur":"height_class", "diametre":"diameter"},inplace=True)
#Target variable
label = df["classification_diagnostic"]

#Join the target variable with the dataset to have a single dataframe
dataset["classification_diagnostic"]=label
```

Top ten rows of the dataset using the **DataFrame's head()** method:

ID_ARBRE	commune	quartier	site	cote_voirie	matricule_arbre	genre_arbre	espece_arbre	controle	situation	type_sol	surf_permeable	date_plan
78551-Arbres-001	Saint Germain en Laye	Quartier 2 - Alsace - Pereire	Carrefour RN13	NaN	7	Betula	Alba	NaN	Groupe	P	5.0	
78551-Arbres-002	Saint Germain en Laye	Quartier 2 - Alsace - Pereire	Carrefour RN13	NaN	8	Betula	Alba	NaN	Groupe	P	5.0	
78551-Arbres-003	Saint Germain en Laye	Quartier 2 - Alsace - Pereire	Carrefour RN13	NaN	9	Betula	Alba	NaN	Groupe	P	5.0	
78551-Arbres-004	Saint Germain en Laye	Quartier 2 - Alsace - Pereire	Carrefour RN13	NaN	1	Carpinus	Betulus - L. - Fastigiata	NaN	Bosquet	G	100.0	
78551-Arbres-005	Saint Germain en Laye	Quartier 2 - Alsace - Pereire	Carrefour RN13	NaN	2	Carpinus	Betulus - L. - Fastigiata	NaN	Bosquet	G	100.0	
78551-Arbres-006	Saint Germain en Laye	Quartier 2 - Alsace - Pereire	Carrefour RN13	NaN	3	Betula	sp. - -	NaN	Groupe	G	100.0	
78551-Arbres-007	Saint Germain en Laye	Quartier 2 - Alsace - Pereire	Carrefour RN13	NaN	4	Betula	sp. - -	NaN	Groupe	G	100.0	
78551-	Saint	Quartier 2	Carrefour									

info() method to get a quick synopsis of the resultant dataset:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 709 entries, 0 to 708
Data columns (total 18 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   TREE_ID                     709 non-null    object
 1   municipality                709 non-null    object
 2   district                    709 non-null    object
 3   site                        709 non-null    object
 4   road_side                   545 non-null    object
 5   tree_number                 709 non-null    int64
 6   tree_genus                  709 non-null    object
 7   tree_species                701 non-null    object
 8   control                     11 non-null     float64
 9   situation                   708 non-null    object
 10  soil_type                   709 non-null    object
 11  permeable_surf              709 non-null    float64
 12  planting_date               709 non-null    int64
 13  age_class                   709 non-null    object
 14  height                      709 non-null    int64
 15  height_class                709 non-null    object
 16  diameter                    709 non-null    float64
 17  classification_diagnostic   699 non-null    object
dtypes: float64(3), int64(3), object(12)
memory usage: 99.8+ KB
```

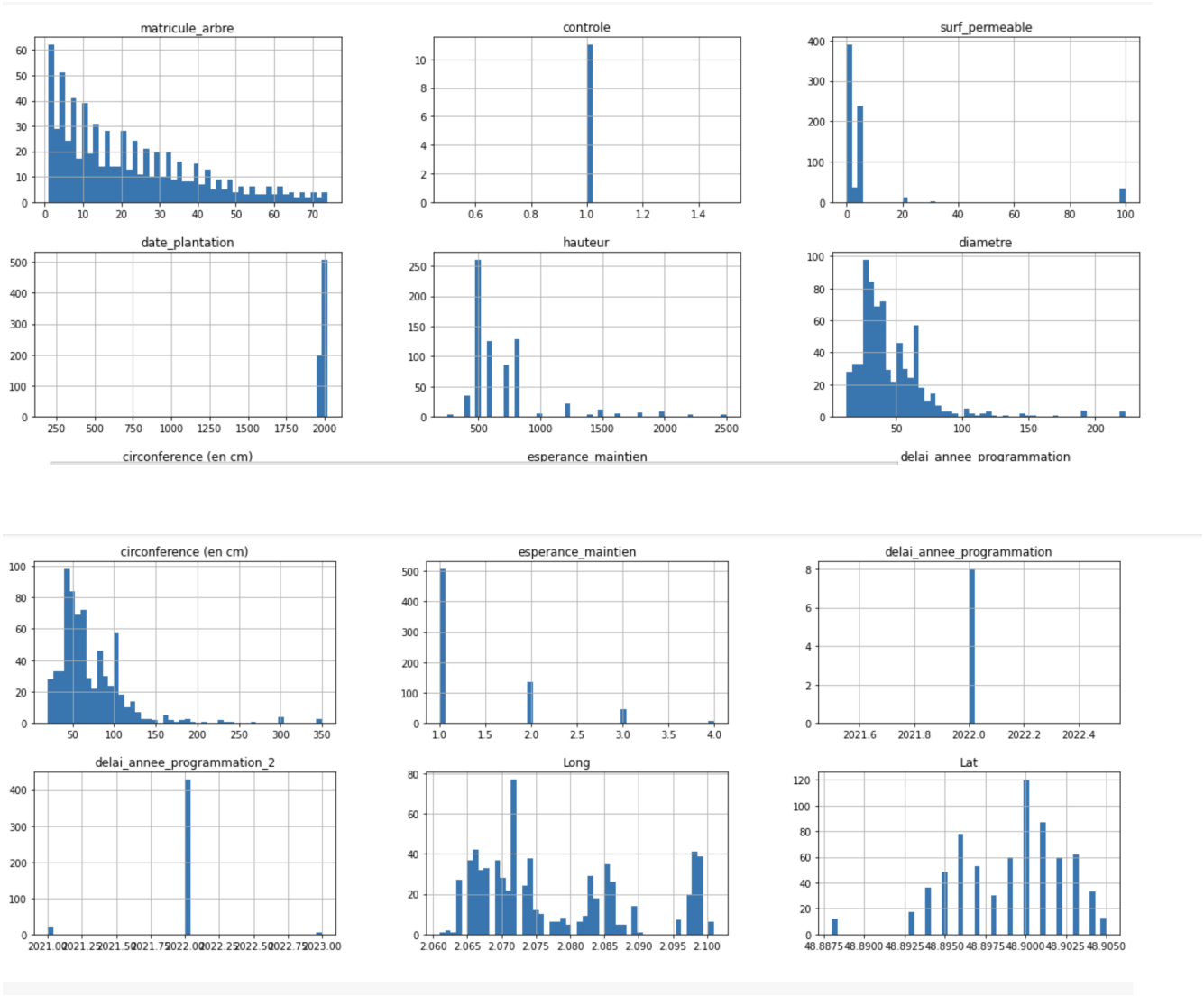
We noticed that almost all our attributes are objects, except for the case of **tree number, control, permeable surf, height** and **diameter**.

describe() method for the summary of the numerical attributes:

	tree_number	control	permeable_surf	planting_date	height	diameter
count	709.000000	11.0	709.000000	709.000000	709.000000	709.000000
mean	21.843441	1.0	7.198872	1996.530324	692.524683	45.520559
std	17.787176	0.0	21.032527	69.052103	333.023261	27.141451
min	1.000000	1.0	0.000000	200.000000	250.000000	12.732395
25%	7.000000	1.0	1.000000	1980.000000	500.000000	28.647890
50%	17.000000	1.0	1.500000	2000.000000	600.000000	38.197186
75%	33.000000	1.0	4.000000	2010.000000	800.000000	57.295780
max	74.000000	1.0	100.000000	2019.000000	2500.000000	222.816920

Mynd 1: Summary of each numerical attributes

hist() method for plotting a histogram for each numerical attribute:



Mynd 2: A histogram for each numerical attribute

corr() method for computing correlation coefficient among the numerical attributes:

	tree_number	control	permeable_surf	planting_date	height	diameter
tree_number	1.000000	NaN	-0.217633	-0.007850	-0.164903	-0.109965
control	NaN	NaN	NaN	NaN	NaN	NaN
permeable_surf	-0.217633	NaN	1.000000	-0.071507	0.604309	0.408231
planting_date	-0.007850	NaN	-0.071507	1.000000	-0.090306	-0.155416
height	-0.164903	NaN	0.604309	-0.090306	1.000000	0.759239
diameter	-0.109965	NaN	0.408231	-0.155416	0.759239	1.000000

Since correlation coefficient ranges from -1 to 1,we see that there is a strong correlation between **permeable-surf**, **height** and **diameter**.

3 Data Preprocessing

3.1 Handling missing data

Since most machine learning algorithms cannot work with missing features, in this section, we handle the missing variables by computing the mode (highest occurrence) on the whole set and use it to replace the missing values. we use from **Scikit-Learn** the class **SimpleImputer**,by creating a **SimpleImputer** instance and specifying our strategy to be "**most-frequent**", we then fit and replace the missing values with the learned mode using the **fit-transform()** method.

```
imp_mode = SimpleImputer(missing_values=np.nan, strategy="most_frequent")
dataset=imp_mode.fit_transform(dataset)
column_names=['TREE_ID','municipality','district','site','road_side',
              'tree_number','tree_genus','tree_species','control','situation',
              'soil_type','permeable_surf','planting_date','age_class','height',
              'height_class', 'diameter','classification_diagnostic']
df_imp=pd.DataFrame(dataset,columns=column_names)
df_imp['classification_diagnostic'].value_counts()
```

3.2 Transforming categorical data into numerical data

Here we will try to handle the text attributes and categorical attributes, that is features with object data types (as in section 2, when we used the info() method). We will then convert these categories from text to numbers using **Scikit-Learn OneHotEncoder** class to convert the categorical values to one-hot vectors. We will also use the **scikit-Learn LabelEncoder** to transform our target variable.

3.3 Feature Scaling

We use the **StandardScaler** to standardize our feature matrix by subtracting the mean and scaling to unit variance. We normalise X so that the values lie between -1 and 1. We do this so that we can get all the features into a similar range. We will use the following equation:

$$Z = \frac{X - \mu}{\sigma} \tag{1}$$

The goal of performing standardization is to bring down all the features to a common scale without distorting the differences in the range of the values. This process of rescaling the features is such that they have mean of 0 and variance as 1.

```
#cartegorical data
enc = OneHotEncoder(handle_unknown="ignore",sparse=False)
X_t=enc.fit_transform(X)

#label data
label= preprocessing.LabelEncoder()
y_t=label.fit_transform(y)


df1 = pd.DataFrame(X_t)
df2 = pd.DataFrame(y_t)

#SCALING THE FEATURE
scaler = preprocessing.StandardScaler().fit(X_t)
X_scaled = scaler.transform(X_t)
```

3.4 Splitting dataset.

We will be using the **Scikit-Learn** function **train-test-split** to split our dataset into training set in order to train model parameters and test set in order to get some final performance metric. We will be using 20% of our dataset as the test set.

```
X_train, X_test, y_train, y_test = train_test_split(X_t, y_t, test_size=0.2, random_state=0)
print(y_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(567,)
(567,)
(142, 989)
(142,)
```

The training and test data contain contains 567 and 142 samples respectively.

4 Logistic Regression

Logistic Regression is a supervised learning technique, y (dependent variable) can take only a finite number of predefined values (quantitative) or labels/classes (qualitative).
Suppose let us consider the simple case of binary classification. Then, it is assumed that there is an underlying unknown function.

$$f(\mathbf{X}) : \mathbb{R}^2 \Rightarrow \{\mathbf{Y}, \mathbf{N}\} \tag{2}$$

that separates the two classes (even unseen data) in the best possible manner. The function $f(x)$ separates the input space into regions.

Objective: Adjust a function $\hat{f(x)}$ to data, such that $\hat{f(x)}$ is close in some sense to the unknown $f(x)$.

Assume one input feature x with N observations, one output feature y with two possible labels. Without loss of generality we can assume $y_i \in \{0, 1\}$.

Can we use a linear regression prediction model?

Answer: Large errors for large absolute values of x.

Can we transform the linear regression output to have more adapted predictions?

Answer: Use an increasing smooth function $f(z)$ such that:

$$\begin{aligned} \lim_{z \to -\infty} f(z) &= 0 \\ \lim_{z \to +\infty} f(z) &= 1 \end{aligned} \tag{3}$$

Sigmoidal functions, i.e. S-shaped functions, can be used. It implies the smooth versions of the sign function.

Examples:

- Gompertz: $f(z) = \exp[-\exp(-z)]$
- Hyperbolic tangent with offset: $f(z) = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{z}{2}\right)$
- Gaussian cumulative distribution: $f(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp\left(-\frac{1}{2}t^2\right) \mathrm{d}t$
- Logistic function: $f(z) = \frac{1}{1+\exp(-z)}$

Applyıng it to the output of the linear regression model we have

$$f_{\boldsymbol{\beta}}(x) = \frac{1}{1 + \exp\left[-\left(\beta_0 + \beta_1 x\right)\right]} \tag{4}$$

This gives a smooth transition from 0 to 1 , without large errors for large absolute values of $\mathbf{x}$, interpreted as a probability $\mathbb{P}(y = 1 \mid x) = f_\beta(x)$

How do we learn the model parameters?

Can we use the least squares approach?

For N observations, we would solve the following optimization problem to retrieve $\boldsymbol{\beta}$:

$$minimize \quad J(\boldsymbol{\beta}) = \sum_{i=1}^{N} j\left(y_i, f_{\boldsymbol{\beta}}\left(x_i\right)\right) \tag{5}$$

with respect to β

where

$$j\left(y_i, f_\beta\left(x_i\right)\right) = \varepsilon_i^2 = \left(y_i - f_\beta\left(x_i\right)\right)^2$$

How do we learn the model parameters? Which cost function should be used?

$$j\left(y, f_\beta(x)\right) = -y \log\left(f_\beta(x)\right) - (1-y) \log\left(1 - f_\beta(x)\right) \tag{6}$$

which is called the **log-loss function**.

How do we learn the model parameters?

The optimization problem corresponding to the logistic regression parameter learning:

$$minimize \quad J(\boldsymbol{\beta}) = \sum_{i=1}^{N} j\left(y_i, f_\beta\left(x_i\right)\right) \quad wrt \;\; \boldsymbol{\beta} \tag{7}$$

where $\quad j\left(y_i, f_\beta\left(x_i\right)\right) = -y_i \log\left(f_\beta\left(x_i\right)\right) - (1-y_i) \log\left(1 - f_\beta\left(x_i\right)\right)$

with $f_{\boldsymbol{\beta}}\left(x_i\right) = \frac{1}{1+\exp[-(\beta_0+\beta_1 x_i)]}$

Unfortunately, there is no closed-form solution for this problem. However, it is a smooth convex optimization problem. Efficient optimization algorithms can be used to solve it, e.g. gradient descent method or Newton's method.

4.1 Model Performance

Back to our code, for the logistic regression we will write a simple function using the **scikit learn** library for One-vs-the-rest (OvR) multiclass strategy, this strategy consists of fitting one classifier per class. For each classifier, the class is fitted against all the other classes. We will use the **LogisticRegression Model** and then proceed to defining our model strategy **OneVsRestClassifier()**, we will fit the model on the training set and then make predictions on our test set. Finally, we calculate the model accuracy.

```
#using the split training set
# define model
model = LogisticRegression()

# define the ovr strategy
ovr = OneVsRestClassifier(model)
# fit model
ovr.fit(X_train, y_train)
# make predictions
yhat = ovr.predict(X_test)
print(yhat)
#Acuracy
print("model score: %.3f" % ovr.score(X_test, y_test))
```

```
[1 1 0 0 1 1 1 0 0 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 2 1 0 0 0 0 1 1 1 0 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1
 0 1 1 0 1 0 1 4 1 0 0 1 1 0 1 0 0 2 1 1 2 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0
 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 1 2 0 1 0 1 1 1 0 0 0 1 1 1 1 1]
model score: 0.775
```

We have the accuracy to be 77.5%, Accuracy is defined as:

$$Accuracy = \frac{CorrectPredictions}{TotalPredictions}$$

Because we have unequal number of labels in the sample, Accuracy seems to be useless, we could see from the output, that the dataset is skewed, i.e the number of samples in one class outnumber the number of sample in other class by a huge number,therefore we will be using other evaluation metrics, since accuracy is not a good representation of the model.

Imbalanced Classification:

Imbalanced classification is a predictive modelling problem where the distribution of the labels in the sample across classes is not equal. Imbalance occurs when more classes have very low proportions in the training data as compared to the other.

Because of the imbalanced labels, we consider other performance indicators such as Confusion Matrix, Precision, Recall and F1 Score.

Precision is defined as:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

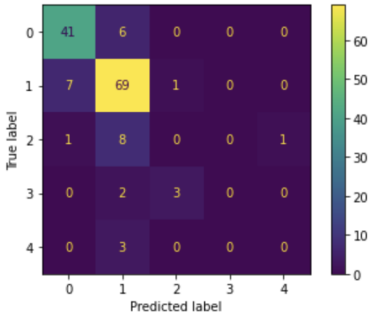
Recall is defined as:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

F1 score can be seen as the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. F1 score is defined as:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
predictions = ovr.predict(X_test)
cm = confusion_matrix(y_test, yhat, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.show()
```



Precision

```
from sklearn.metrics import precision_score,recall_score
precision_score(y_test, yhat, average=None,zero_division=1)
```

```
array([0.83673469, 0.78409091, 0.        , 1.        , 0.        ])
```

Recall

```
recall_score(y_test, yhat, average=None, zero_division=1)
```

```
array([0.87234043, 0.8961039 , 0.        , 0.        , 0.        ])
```

F1 score

```
print(f1_score(y_test, yhat, average=None))
```

```
[0.85416667 0.83636364 0.         0.         0.        ]
```

Cross Validation and Hyper-Parameter Optimization:

Variance problem: A situation where the accuracy obtained on one test set is very different to accuracy obtained on another test set using the same algorithm.

In order to avoid the variance problem, we consider the method of cross validation, this method ensure that every set is at least used once for training and once for testing set, the final result is the average of results obtained using all folds, this way we get rid of the variance, using the standard deviation of the results obtained from each fold.

We also used the Grid search parameter selection since random selection of Hyper-parameter can be exhaustive, we used this method to automatically select the best parameter for our model.

```
#using the grid searchCV
#define models and parameters of lOGISTIC REGRESSION
solvers = ['newton-cg', 'lbfgs', 'liblinear'] #it allow us to see useful differences in performance or convergence with different solvers (solver).
penalty = ['l2']  #Regularization (penalty)
c_values = [100, 10, 1.0, 0.1, 0.01] #The C parameter controls the penality strength
model = LogisticRegression(multi_class='ovr')
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_scaled, y_t)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.766311 using {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.765842 (0.034798) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
0.766311 (0.035008) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
0.720268 (0.043202) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
0.764903 (0.036228) with: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}
0.764903 (0.036228) with: {'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}
0.721677 (0.042884) with: {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}
0.763964 (0.034842) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.763964 (0.034842) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.723085 (0.043895) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.765372 (0.036442) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-cg'}
0.765372 (0.036442) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
0.725433 (0.045287) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
0.763970 (0.038272) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
0.763970 (0.038272) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
0.728256 (0.047022) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

5 Support Vector Machine (SVM)

Support Vector Machine (SVM) is used for both classification and regression problems. It is a linear model that solves linear and nonlinear problems. The algorithm takes the input data and outputs a line or hyperplane (decision boundary) which separates data into classes.

Inputs: set of training pair samples features $x_1, x_2, ..., x_n$, and the output result y. There can be lots of input features x_i

Outputs: set of weights w (or w_i) one for each feature and its linear combination can be used to predict the value of y.

Linear SVM:

It is used to classify a dataset into two classes by using a straight line, although in practice, the data might not be linearly separable. A two dimensional data is separated by a line and a multidimensional data is separated by a hyperplane.

In the two dimensional case, the function of the line is of the form

$$y = ax + b \tag{8}$$

which can be written as

$$x_2 = ax_1 + b \tag{9}$$

where $x_2 = y$ and $x_1 = x$. If we take $x = (x_1, x_2)$ and $w = (a, -1)$, we have $w.x + b = 0$. Once we have hyperplane, it can be used to make predictions, we define it as

$$h(x_i) = \begin{cases} +1, & \text{if } w.x + b \geq 0 \\ -1 & \text{if } w.x + b < 0 \end{cases}$$

SVM helps us to find an ideal linear classifier between the two lines called the optimal hyperplane, it finds the points called the support vectors that are closest to the line from both classes. Then compute the distance between the line and the support vectors. This distance is called the margin, we always choose a hyperplane such that the distances to the closest points on each side are equal.

Finding the values of w and b of the optimal hyperplane, we solve the following optimization problem, with the constraint:

$$\begin{gathered} \min_{w,b} \frac{1}{2}\|w\|^2 \\ \text{subject to } : y_i(w \cdot x + b) - 1 \geq 0, i = 1 \dots m \end{gathered} \tag{10}$$

This is the SVM optimization problem. To solve this problem we introduce the multipliers of Lagrange.

The Lagrange function is given as

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i \left[y_i(w \cdot x + b) - 1\right] \tag{11}$$

The dual problem can then be written as :

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\ w.w - \sum_{i=1}^{m} \alpha_i \left[y_i(w \cdot x + b) - 1\right] \tag{12}$$

we then have

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y_i x_i = 0 \tag{13}$$

$$\nabla_b \mathcal{L}(w, b, \alpha) = -\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{14}$$

equation (7) then gives that

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{15}$$

and equation (8)

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{16}$$

substituting into the Lagrangian function we get

$$\begin{gathered} \max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \\ \text{subject to} \qquad \sum_{i=1}^{m} a_i y_i = 0 \\ 0 \leq \alpha_i \leq C, i = 1 \ldots m \end{gathered} \tag{17}$$

Where C helps to control how the SVM handles the errors, small values of C results in a wider margin. Hence we find C such that it will not make the solution affected by the noise of the data.

5.1 Model Performance

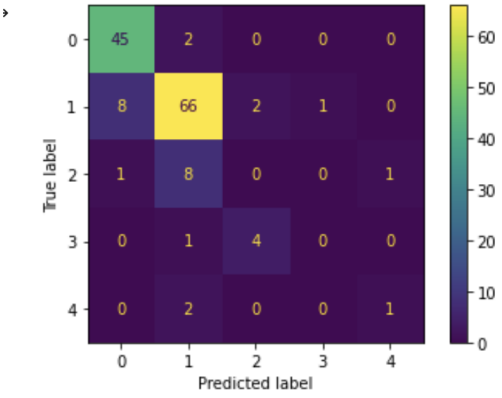
Linear models are quick to train and predict. We can adapt it to a very large data set. For this model, the accuracy of the test set is 78.9 %.

```
X_train, X_test, y_train, y_test = train_test_split(X_t, y_t, test_size=0.2, random_state=0)
# Run classifier
classifier = OneVsRestClassifier(SVC(kernel='linear', probability=True))
clf=classifier.fit(X_train, y_train)
print("model score: %.3f" % classifier.score(X_test, y_test))
yhat=classifier.predict(X_test)
print(yhat)
y_score = clf.decision_function(X_test)
```

```
model score: 0.789
[1 2 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 2 1 0 1 0 0 1 4 1 0 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 3 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1
 0 1 0 0 1 0 1 4 1 0 0 1 1 0 1 0 0 2 1 1 2 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0
 1 1 1 0 0 0 0 1 0 0 1 1 1 0 1 1 2 0 1 0 1 2 1 0 0 0 1 1 1 1 1]
```

Confuson Matrix For The SVM

```
cm = confusion_matrix(y_test, yhat, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classifier.classes_)
disp.plot()
plt.show()
```




```
from sklearn.metrics import precision_score,recall_score
print(precision_score(y_test, yhat, average=None,zero_division=1))
print(recall_score(y_test, yhat, average=None, zero_division=1))
print(f1_score(y_test, yhat, average=None))
```

```
[0.83333333 0.83544304 0.         0.         0.5        ]
[0.95744681 0.85714286 0.         0.         0.33333333]
[0.89108911 0.84615385 0.         0.         0.4        ]
```

6 Kernelized SVM

Kernel function is a method used to take data as input and transform it into the required form of processing data. "Kernel" is used due to a set of mathematical functions used in Support Vector Machine providing the window to manipulate the data. So kernel functions generally transform the training set of data so that a non-linear decision surface is able to transform into a linear equation in a higher number of dimension spaces. Basically, it turns the inner product between two points in a standard feature dimension.

Relating this to machine Learning, in most of the cases of SVM, the optimal boundary is nonlinear and this is a limitation to solving peculiar machine learning problems. In order to tackle the problem of the linear separator, the idea of the kernel function is considered which would look at the problem in a higher dimensional space. We thus create a non-linear model in the initial space. The re-described space H is so named because we will choose maps ϕ such that H is a Hilbert space. In practice, this means that H can be considered to be the map $\phi : R^d \longrightarrow H$. We use the same algorithm for Linear SVM in H to learn a model that explains y as a linear function of the coordinates of $\phi(x)$. We thus create a nonlinear problem in the initial space. To do this, we replace x by $\phi(x)$ in the optimisation problem below:

$$\begin{cases} \min_{w,b} & \frac{1}{2}\|w\|^2 \\ & \qquad\qquad i = 1, n \\ \text{avec} & y_i \left(w^\top x_i + b\right) \geq 1 \end{cases} \tag{18}$$

This is because the SVM model is not in the initial space but in the re-described space H

There are several Kernel functions we could choose from like Polynomial (poly), Radial basis function (rbf) and Sigmoid Kernel.

We conduct the **Grid Search Optimization** as described in the below code for finding the best Kernel function.

```
#Using the gridSearchCV
# define model and parameters
model = SVC()
kernel = ['poly', 'rbf', 'sigmoid']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
# define grid search
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0
grid_result = grid_search.fit(X_t, y_t)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Output

```
Best: 0.823689 using {'C': 1.0, 'gamma': 'scale', 'kernel': 'poly'}
0.804440 (0.038851) with: {'C': 50, 'gamma': 'scale', 'kernel': 'poly'}
0.801147 (0.037408) with: {'C': 50, 'gamma': 'scale', 'kernel': 'rbf'}
0.700892 (0.048727) with: {'C': 50, 'gamma': 'scale', 'kernel': 'sigmoid'}
0.804440 (0.038851) with: {'C': 10, 'gamma': 'scale', 'kernel': 'poly'}
0.801147 (0.037408) with: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
0.752643 (0.043986) with: {'C': 10, 'gamma': 'scale', 'kernel': 'sigmoid'}
0.823689 (0.036368) with: {'C': 1.0, 'gamma': 'scale', 'kernel': 'poly'}
0.815674 (0.035281) with: {'C': 1.0, 'gamma': 'scale', 'kernel': 'rbf'}
0.767693 (0.037266) with: {'C': 1.0, 'gamma': 'scale', 'kernel': 'sigmoid'}
0.750329 (0.044122) with: {'C': 0.1, 'gamma': 'scale', 'kernel': 'poly'}
0.753146 (0.039488) with: {'C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'}
0.734809 (0.033460) with: {'C': 0.1, 'gamma': 'scale', 'kernel': 'sigmoid'}
0.575453 (0.004408) with: {'C': 0.01, 'gamma': 'scale', 'kernel': 'poly'}
0.575453 (0.004408) with: {'C': 0.01, 'gamma': 'scale', 'kernel': 'rbf'}
0.575453 (0.004408) with: {'C': 0.01, 'gamma': 'scale', 'kernel': 'sigmoid'}
```

Using **Grid Search Optimization**, the best Kernel function is found to be with parameters 'C': 1.0, 'gamma': 'scale', 'kernel': 'poly'.

6.1 Model Performance

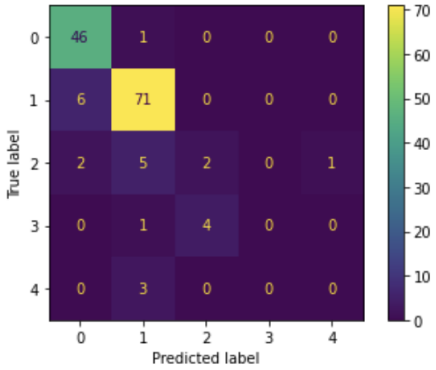
Our Kernelized SVM model using the Kernel function as above gives an Accuracy of 83.8 %, which is same as SVM model. It is therefore advised to look into the other performance indicators for better evaluation of the model.

```
[62] X_train, X_test, y_train, y_test = train_test_split(X_t, y_t, test_size=0.2, random_state=0)
     # Run classifier
     classifier = OneVsRestClassifier(SVC(kernel='poly',C=1.0,gamma='scale', probability=True))
     clf=classifier.fit(X_train, y_train)
     print("model score: %.3f" % classifier.score(X_test, y_test))
     yhat=classifier.predict(X_test)
     print(yhat)
     y_score = clf.decision_function(X_test)
```

```
model score: 0.838
[1 1 0 0 1 1 1 0 0 1 1 0 0 1 1 0 2 1 0 0 1 0 1 1 1 2 1 0 1 0 0 1 1 1 0 1 1
 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1
 1 1 0 0 1 0 1 4 1 0 0 1 1 0 1 0 0 2 1 2 1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0
 1 1 1 0 0 0 0 1 0 0 1 1 1 0 1 1 2 0 1 0 1 2 1 0 0 0 1 1 1 1 1]
```

Confuson Matrix and other performance indicators for Kernelized SVM:

```
cm = confusion_matrix(y_test, yhat, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classifier.classes_)
disp.plot()
plt.show()
```




```
from sklearn.metrics import precision_score,recall_score
print(precision_score(y_test, yhat, average=None,zero_division=1))
print(recall_score(y_test, yhat, average=None, zero_division=1))
print(f1_score(y_test, yhat, average=None))
```

```
[0.85185185 0.87654321 0.33333333 1.         0.        ]
[0.9787234  0.92207792 0.2        0.         0.        ]
[0.91089109 0.89873418 0.25       0.         0.        ]
```

7 Artificial Neural Networks (ANN)

An artificial neural network is a machine learning model that contains collections of weights and mathematical operations, arranged in such a way to loosely replicate the functioning of human brain. ANN can be seen as a mathematical function that takes in one or more tensors as inputs and predicts one or more tensors as outputs. The arrangement of operations that connects these inputs to outputs is referred to as the architecture of the neural network.

What is Deep learning?

Deep learning refers to neural networks with more than one hidden layer. Deep learning can be used to solve tasks such as classifying billions of images(e.g Google Images), powering speech recognition services(e.g, Apple's Siri), recommending the best videos to watch to hundreds of users everyday(e.g Youtube), or learning to beat the world champion at the game of Go (DeepMind's AlphaGo) that many other types of algorithms cannot solve. Deep learning work is computationally expensive,people used Graphics processing unit (GPU).
An ANN is made up of 3 layers,The input Layers, Hidden (Intermediate) Layers, and Output Layers.
The output value O is calculated as follows:

$$O = f(w_0 + \sum_{i=1}^{n} w_i x_i) \tag{19}$$

The function f is the activation function.

Why do neural networks need an activation function?

The activation function helps in modeling complex relations between the input and the output, that is an activation function is used to put non-linearity to the neural network. Without activation function, no matter how many hidden layers we put in the neural network, every neuron will behave in the same way. Some of the commonly used activation functions are as follows:

$$Sigmoid : h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{20}$$

$$Tanh : f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

(21)

$$ReLU \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{22}$$
$$= \max\{0, x\} = x\mathbf{1}_{x>0}$$

$$Linear: \quad f(x) = x \tag{23}$$

What does it mean "to train a neuron"?

Training a neuron simply means choosing weights w and bias b so that we get the desired output for all N inputs x. To achieve this, we minimize a loss function, loss function allow us to measure how far we are from the output of our neural network. Gradient descent methods such as stochastic gradient descent(SGD), adaptive gradient descent, AdaGrad, RSMSprop, SGDNesterov, AdaDelta allow us to minimize the cost and error. All of these methods require us to evaluate the partial derivative (the gradient) with respect to the model parameters w and b. Our goal is to gradually tweak w and b so that the overall loss function keeps getting smaller and smaller across all x inputs.

How to train Neural Network

Step 1: Using the input set X. Set the activation function a for the input layer.

$z = w * x + b$

$a = \sigma(z)$

Step 2: For each layer compute,

$z^l = w^l * x^l + b^l$

$a^l = \sigma(z^l)$

Step 3: Compute our Error Vector

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right)$$

Where :

$$\nabla_a c = \left(a^L - y\right)$$

Expressing the rate of Change of C w.r.t output activation's.

$$\Rightarrow \delta^L = \left(a^L - y\right)\sigma'\left(z^L\right)$$

Step 4: Propagate the error backwards for each layer $L-1$,$L-2$,...

$$\begin{cases} \delta^L = \left(w^{L+1}\right)^\top \delta^{L+1} \odot \sigma'\left(z^L\right) \\ \left(w^{L+1}\right)^\top \end{cases}$$

Then the gradient of the cost function for each layer is given by: $L-1, L-2, \ldots$

$$\begin{cases} \frac{\partial c}{\partial w_{jk}^{L}} = a_k^{L-1}\delta_j^L \\ \frac{\partial c}{\partial b_j} = \delta_j^L \end{cases}$$

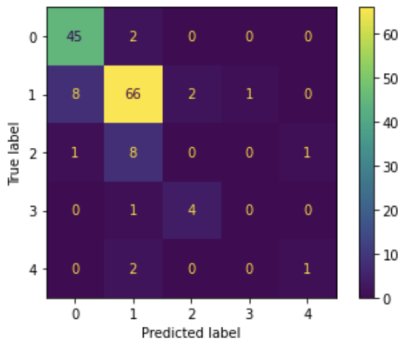
This allows us to adjust the weights and biases to help minimize the cost.

7.1 Model Performance

```
from sklearn.neural_network import MLPClassifier
nn = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=1)
nn.fit(X_train, y_train)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,solver='lbfgs')
print("model score: %.3f" % nn.score(X_test, y_test))
yhat=clf.predict(X_test)
print(yhat)
```

```
model score: 0.782
[1 2 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 2 1 0 1 0 0 1 4 1 0 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 3 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1
 0 1 0 0 1 0 1 4 1 0 0 1 1 0 1 0 0 2 1 1 2 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0
 1 1 1 0 0 0 0 1 0 0 1 1 1 0 1 1 2 0 1 0 1 2 1 0 0 0 1 1 1 1 1]
```

```
cm = confusion_matrix(y_test, yhat, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classifier.classes_)
disp.plot()
plt.show()
```



Precision, Recall and F1 score

```
from sklearn.metrics import precision_score,recall_score
print(precision_score(y_test, yhat, average=None,zero_division=1))
print(recall_score(y_test, yhat, average=None, zero_division=1))
print(f1_score(y_test, yhat, average=None))
```

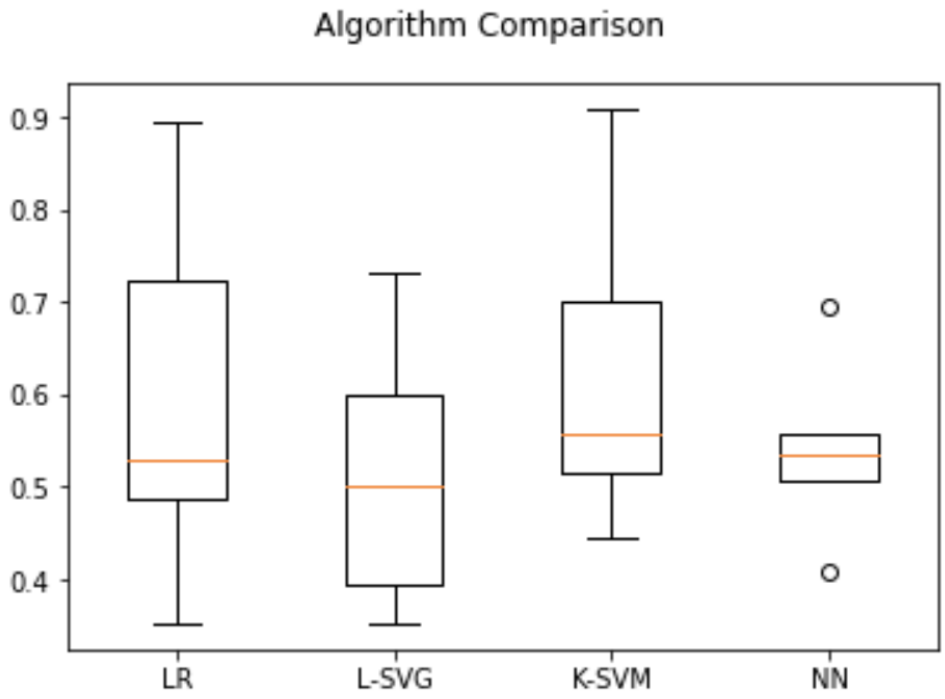
```
[0.83333333 0.83544304 0.         0.         0.5       ]
[0.95744681 0.85714286 0.         0.         0.33333333]
[0.89108911 0.84615385 0.         0.         0.4       ]
```

8 Comparison of the results and conclusion

8.1 Comparison of Performance Indicators

Metric	Logistic %	Linear SVM %	Kernel SVM%	Neural Network %	Best Performance
Accuracy	77.5	78.9	83.8	78.2	Kernel SVM
Precision	83.67	83.33	85.18	83.33	Kernel SVM
Recall	87.23	95.74	97.87	95.74	Kernel SVM
F1 score	85.41	89.10	91.08	89.10	Kernel SVM

8.2 Cross-Validation (5 splits) results using Box plot



Conclusion:

Hence, it is evident from section 8.1 and 8.2 that Kernel SVM is best for our Multi-class classification problem.

9 References

- Prof. Sylvain Rubenthaler, An Introduction to Machine Learning with Probabilities, in R (2021-2022)

- Geron, Aureìlien. 2019. Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed. CA 95472: O'Reilly.

- V Kishore Ayyadevara and Yeshwanth Reddy, Modern Computer Vision with PyTorch.

- Abhishek Thakur, Approaching (almost) any Machine Learning Problem.

- https://data-flair.training/blogs/svm-kernel-functions/

- https://shuzhanfan.github.io/2018/05/understanding-mathematics-behind-support-vector-machines/