HW_4_SP22

February 27, 2022

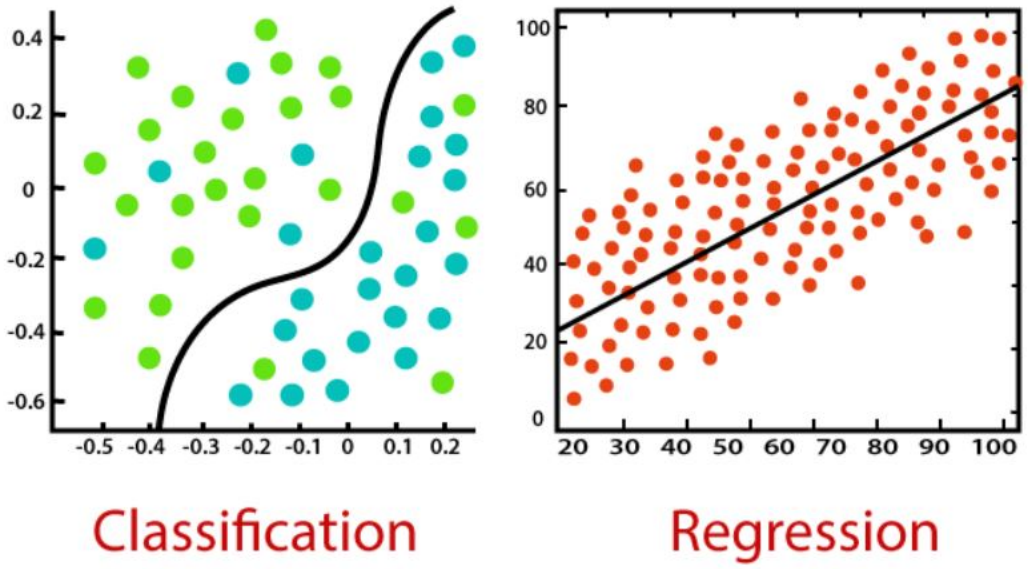
import matplotlib.pyplot as plt import plotly.express as px import pandas as pd # ~~~ pyforest auto-imports - don't write above this line # Classification Algorithms: Logistic Regression and Support Vector Machine

0.1 1 Explain what is classification and how it is different from regression

Classification is the process of finding a model that separates input data into multiple discrete classes or labels. In other words, a classification problem determines whether or not an input value can be part of a pre-identified group.

Consider the same dataset of all the students at a university. A classification task would be to use parameters, such as a student's weight, major, and diet, to determine whether they fall into the "Above Average" or "Below Average" category. Note that there are only two discrete labels in which the data is classified.

A classification algorithm is evaluated by computing the accuracy with which it correctly classified its input.



The following are the differences between regression and classification The **main difference between Regression and Classification algorithms** that **Regression algorithms**

are used to predict the continuous values such as price, salary, age, etc. and **Classification algorithms** are used to predict/Classify the discrete values such as Male or Female, True or False, Spam or Not Spam, etc.

Parameter	CLASSIFICATION	REGRESSION
Basic	The mapping function is used for mapping values to predefined classes.	Mapping Function is used for the mapping of values to continuous output.
Involves prediction of	Discrete values	Continuous values
Nature of the predicted data	Unordered	Ordered
Method of calculation	by measuring accuracy	by measurement of root mean square error
Example Algorithms	Decision tree, logistic regression, etc.	Regression tree (Random forest), Linear regression, etc.

0.2 2 Explain what is Logistic regression, its working and how it is different from linear regression

- This type of statistical analysis (also known as **logit model**) is often used for predictive analytics and modeling, and extends to applications in machine learning. In this analytics approach, the dependent variable is finite or categorical: either A or B (binary regression) or a range of finite options A, B, C or D (multinomial regression). It is used in statistical software to understand the relationship between the dependent variable and one or more independent variables by estimating probabilities using a logistic regression equation.
- This type of analysis can help you **predict the likelihood of an event happening or a choice being made.** For example, you may want to know the likelihood of a visitor choosing an offer made on your website — or not (dependent variable). Your analysis can look at known characteristics of visitors, such as sites they came from, repeat visits to your site, behavior on your site (independent variables).
- **Logistic regression** models help you determine a probability of what type of visitors are likely to accept the offer — or not. As a result, you can make better decisions about promoting your offer or make decisions about the offer itself.

How does Logistic Regression Work?

The logistic regression equation is quite similar to the linear regression model.

Consider we have a model with one predictor "x" and one Bernoulli response variable "$\hat{y}$" and p is the probability of $\hat{y}$=1. The linear equation can be written as:

$$p = b_0+b_1x \quad \text{--------> eq 1}$$

- The right-hand side of the equation (b0+b1x) is a linear equation and can hold values that exceed the range (0,1). But we know probability will always be in the range of (0,1).
- To overcome that, we predict odds instead of probability.
- **Odds: The ratio of the probability of an event occurring to the probability of an event not occurring.**
- **Odds = p/(1-p)**
- The equation 1 can be re-written as:

$$p/(1-p) = b_0+b_1x \quad \text{--------> eq 2}$$

- Odds can only be a positive value, to tackle the negative numbers, we predict the **logarithm of odds.**
- **Log of odds = ln(p/(1-p))**
- The equation 2 can be re-written as:

$$\ln(p/(1-p)) = b_0+b_1x \quad \text{--------> eq 3}$$

- To recover p from equation 3, we apply exponential on both sides.

$$\exp(\ln(p/(1-p))) = \exp(b_0+b_1x)$$
$$e^{\ln(p/(1-p))} = e^{(b0+b1x)}$$

- From the inverse rule of logarithms,

$$p/(1-p) = e^{(b0+b1x)}$$

- Simple algebraic manipulations

$$p = (1-p) * e^{(b0+b1x)}$$
$$p = e^{(b0+b1x)} - p * e^{(b0+b1x)}$$

- Taking p as common on the right-hand side

$$p = p * ((e^{(b0+b1x)})/p - e^{(b0+b1x)})$$
$$p = e^{(b0+b1x)} / (1 + e^{(b0+b1x)})$$

- Dividing numerator and denominator by e(b0+b1x) on the right-hand side

$$p = 1 / (1 + e^{-(b0+b1x)})$$

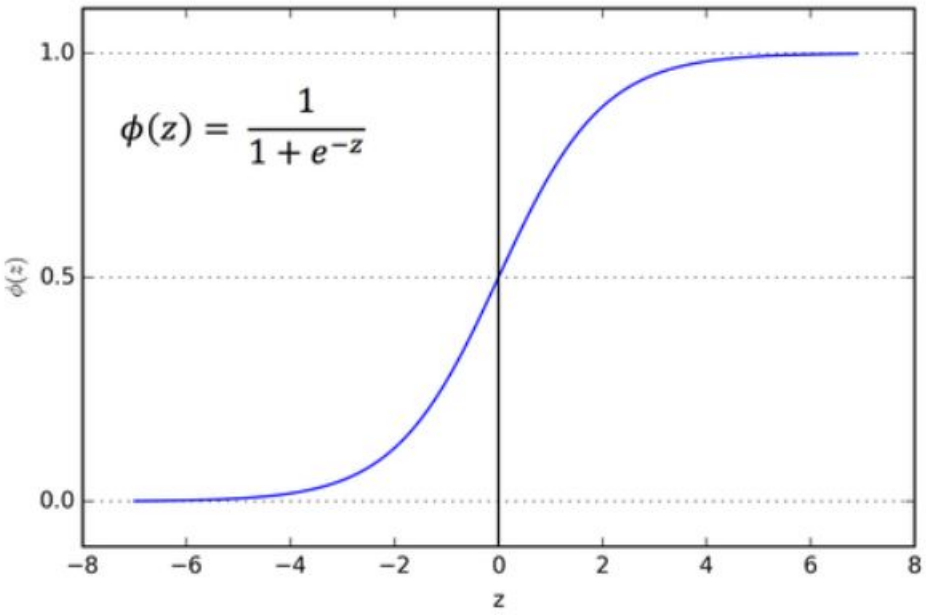
- Similarly, the equation for a logistic model with 'n' predictors is as below:

$$p = 1/\ (1 + e^{-(b0+b1x1+b2x2+b3x3+----+bnxn)})$$

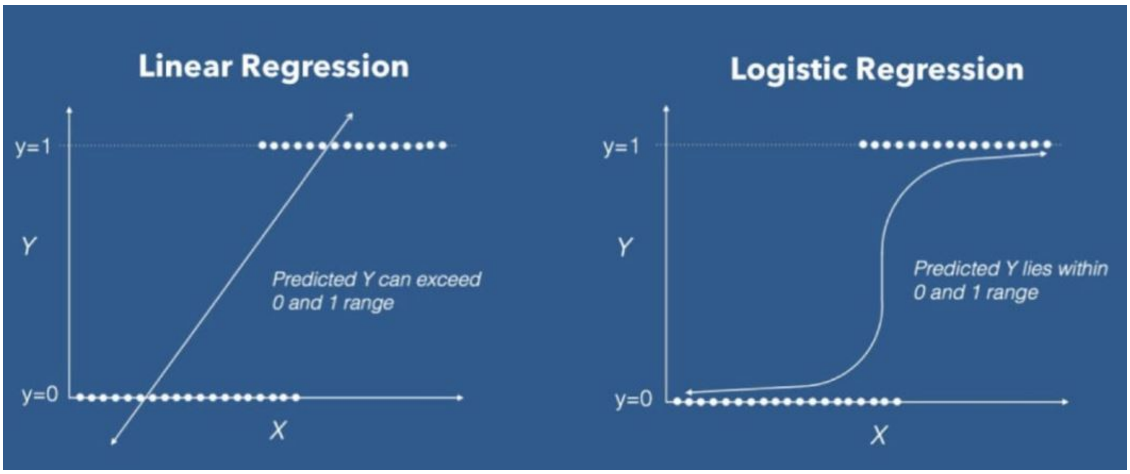
- The right side part is the sigmoid function. It helps to squeeze the output to be in the range between 0 and 1.

Sigmoid Function:

- The sigmoid function is useful to map any predicted values of probabilities into another value between 0 and 1.

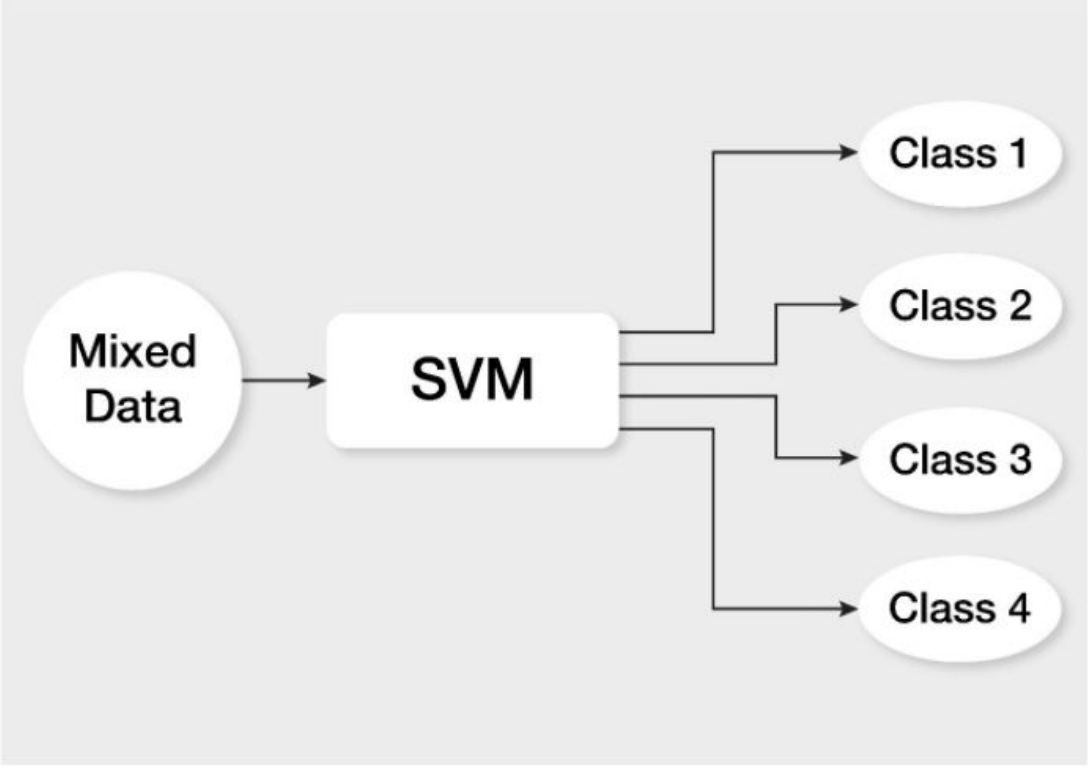


- We started with a linear equation and ended up with a logistic regression model with the help of a sigmoid function.
  - **Linear model: $\hat{y}$ = b0+b1x**
  - **Sigmoid function: (z) = 1/(1+e−z)**
  - **Logistic regression model: $\hat{y}$ = (b0+b1x) = 1/(1+e-(b0+b1x))**
- So, unlike linear regression, **we get an 'S' shaped curve** in logistic regression.



0.3 3 Explain what is Linear SVM and its working

- **SVM or Support Vector Machine:** is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems.
- The idea of SVM is simple: The algorithm creates **a line or a hyperplane which separates the data into classes.**



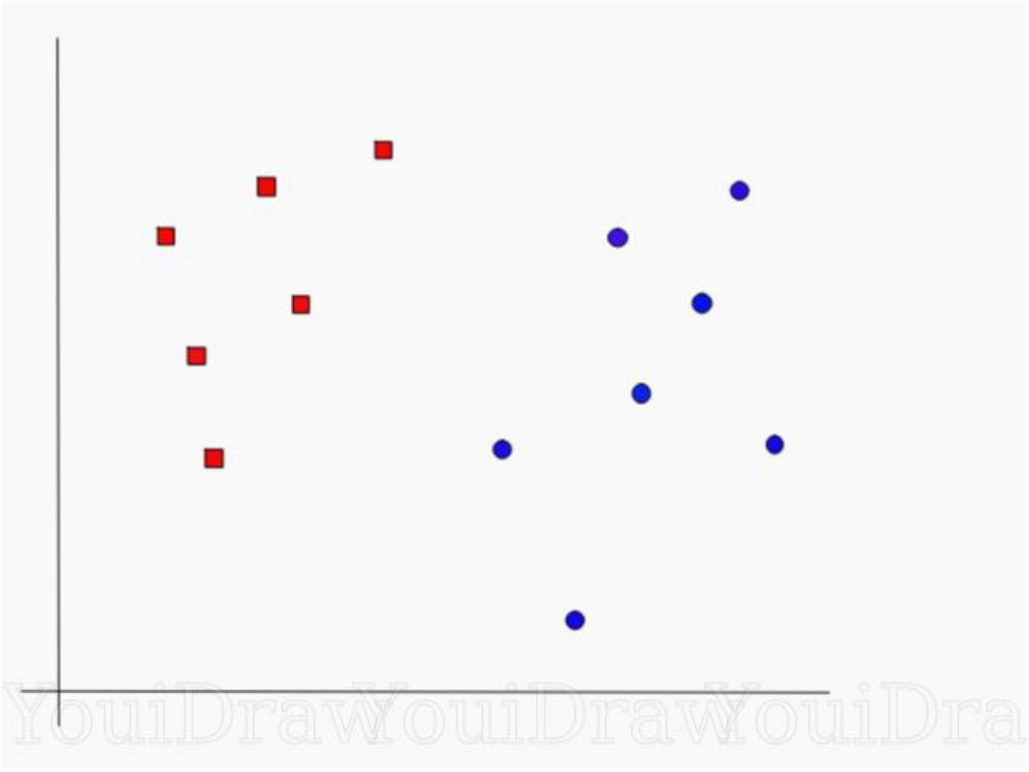
SVM classifier

THEORY:

- At first approximation what SVMs do is to find a separating line(or hyperplane) between

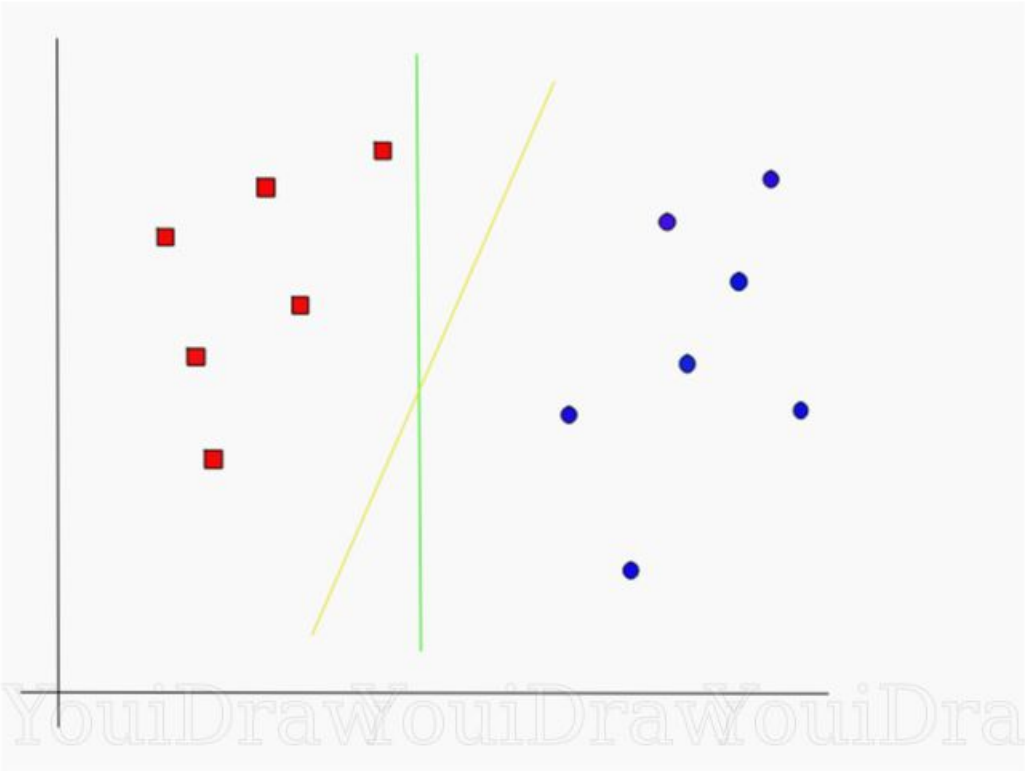
data of two classes. SVM is an algorithm that takes the data as an input and outputs a line that separates those classes if possible.

- Lets begin with a problem. Suppose you have a dataset as shown below and you need to classify the red rectangles from the blue ellipses(let's say positives from the negatives). So your task is to find an ideal line that separates this dataset in two classes (say red and blue).



Find an ideal line/ hyperplane that separates this dataset into red and blue categories

- Not a big task, right?
- But, as you notice there isn't a unique line that does the job. In fact, we have an infinite lines that can separate these two classes. So how does SVM find the ideal one??? Let's take some probable candidates and figure it out ourselves.

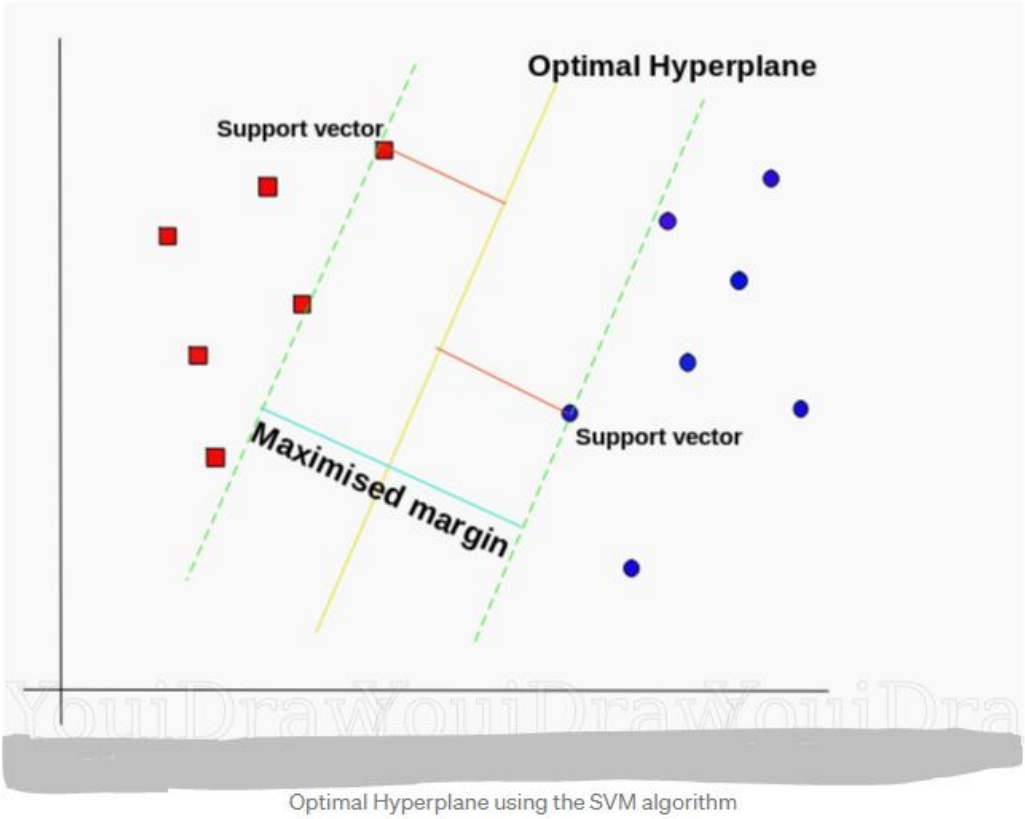


Which line according to you best separates the data???

- We have two candidates here, the green colored line and the yellow colored line. Which line according to you best separates the data?
- If you selected the yellow line then congrats, because thats the line we are looking for. It's visually quite intuitive in this case that the yellow line classifies better. But, we need something concrete to fix our line.
- The green line in the image above is quite close to the red class. Though it classifies the current datasets it is not a generalized line and in machine learning our goal is to get a more generalized separator.

SVM's way to find the best line

- According to the SVM algorithm we find the points closest to the line from both the classes.These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin.
- Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.



Optimal Hyperplane using the SVM algorithm

Thus SVM tries to make a decision boundary in such a way that the separation between the two classes(that street) is as wide as possible.

0.4 4 What do you mean by kernel functions?

- Kernel is a way of computing the dot product of two vectors x and y in some (possibly very high dimensional) feature space, which is why kernel functions are sometimes called "generalized dot product".
- In machine learning, a "kernel" is usually used to refer to the kernel trick, a method of using a linear classifier to solve a non-linear problem. It entails transforming linearly inseparable data like (Fig. 3) to linearly separable ones (Fig. 2). The kernel function is what is applied on each data instance to map the original non-linear observations into a higher-dimensional space in which they become separable.

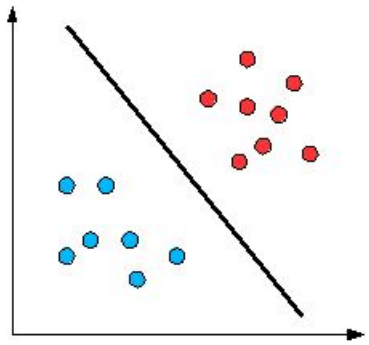


Fig. 2

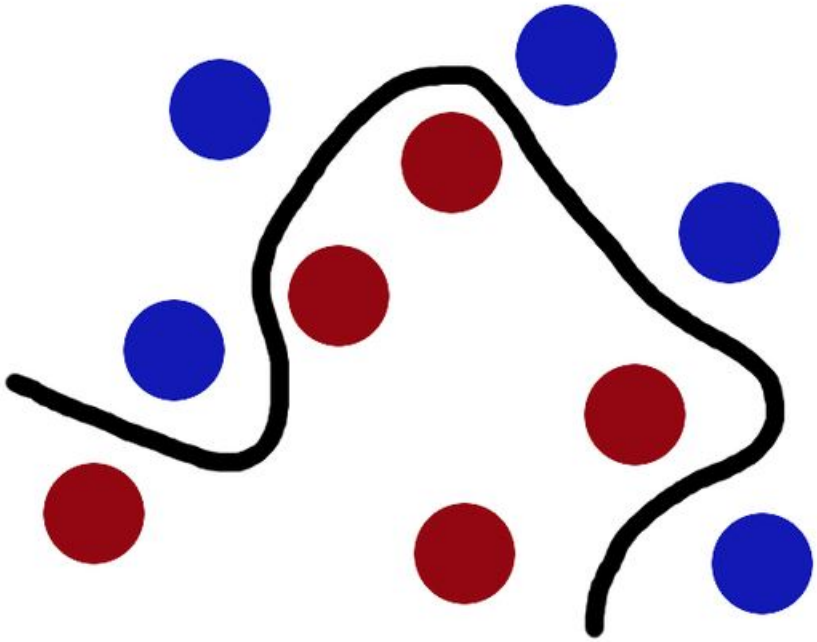
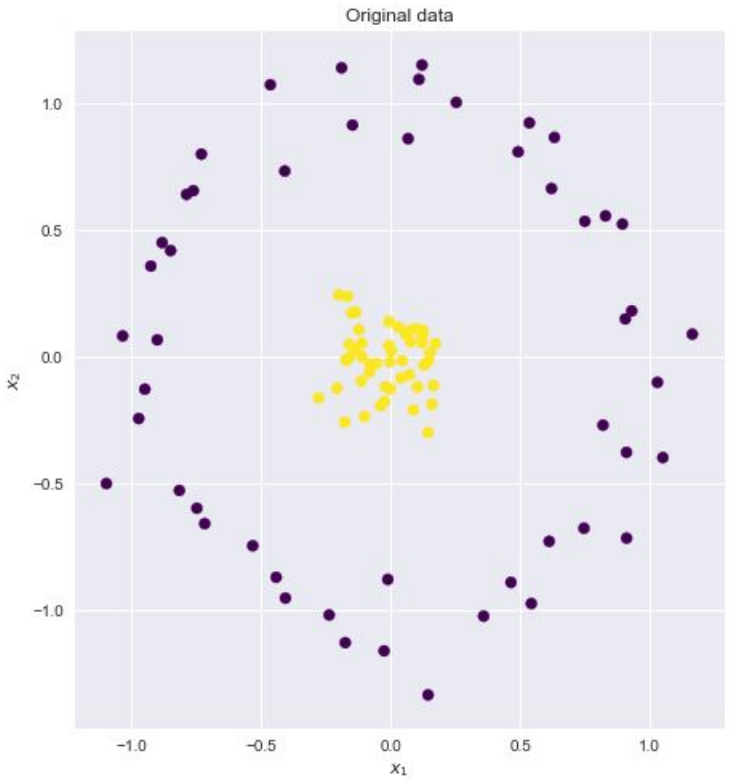


Fig. 3

Consider the following dataset where the yellow and blue points are clearly not linearly separable in two dimensions.



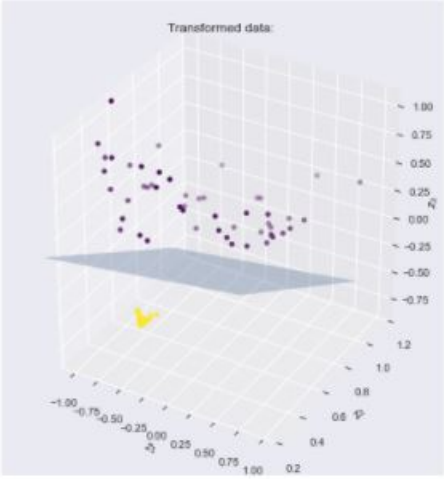
If we could find a higher dimensional space in which these points were linearly separable, then we could do the following:

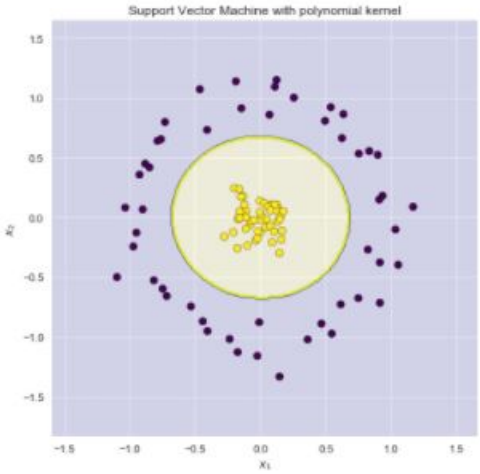
- Map the original features to the higher, transformer space (feature mapping)
- Perform linear SVM in this higher space
- Obtain a set of weights corresponding to the decision boundary hyperplane
- Map this hyperplane back into the original 2D space to obtain a non linear decision boundary

There are many higher dimensional spaces in which these points are linearly separable.

Visualizing the feature map and the resulting boundary line

- Left-hand side plot shows the points plotted in the transformed space together with the SVM linear boundary hyperplane
- Right-hand side plot shows the result in the original 2-D space





0.5 5 Discuss how SVM makes use of kernel functions

- Briefly speaking, a **kernel is a shortcut that helps us do certain calculation faster which otherwise would involve computations in higher dimensional space.**
- **Mathematical definition:** K(x, y) = <f(x), f(y)>. Here K is the kernel function, x, y are n dimensional inputs. f is a map from n-dimension to m-dimension space. < x,y> denotes the dot product. usually m is much larger than n.
- **Intuition:** normally calculating <f(x), f(y)> requires us to calculate f(x), f(y) first, and then do the dot product. These two computation steps can be quite expensive as they involve manipulations in m dimensional space, where m can be a large number. But after all the trouble of going to the high dimensional space, the result of the dot product is really a scalar: we come back to one-dimensional space again! Now, the question we have is: do we really need to go through all the trouble to get this one number? do we really have to go to the m-dimensional space? The answer is no, if you find a clever kernel.

Simple Example: *x = (x1, x2, x3); y = (y1, y2, y3). Then for the function f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3), the kernel is K(x, y) = (<x, y>)².*

Let's plug in some numbers to make this more intuitive: suppose x = (1, 2, 3); y = (4, 5, 6). Then:
f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)
f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)
<f(x), f(y)> = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024

A lot of algebra, mainly because f is a mapping from 3-dimensional to 9 dimensional space.

Now let us use the kernel instead:
K(x, y) = (4 + 10 + 18) ^2 = 32² = 1024
Same result, but this calculation is so much easier.

- **Additional beauty of Kernel:** kernels allow us to do stuff in infinite dimensions! Sometimes going to higher dimension is not just computationally expensive, but also impossible. f(x) can be a mapping from n dimension to infinite dimension which we may have little idea of how to deal with. Then kernel gives us a wonderful shortcut.
- **Relation to SVM: now how is related to SVM?** The idea of SVM is that y = w phi(x) +b, where w is the weight, phi is the feature vector, and b is the bias. if y> 0, then we classify datum to class 1, else to class 0. We want to find a set of weight and bias such that the margin is maximized. Previous answers mention that kernel makes data linearly separable for SVM. I think a more precise way to put this is, kernels do not make the data linearly separable. The feature vector phi(x) makes the data linearly separable. Kernel is to make the calculation process faster and easier, especially when the feature vector phi is of very high dimension (for example, x1, x2, x3, …, x_D^n, x1^2, x2^2, …., x_D^2).
- **Why it can also be understood as a measure of similarity:** if we put the definition of kernel above, <f(x), f(y)>, in the context of SVM and feature vectors, it becomes <phi(x), phi(y)>. The inner product means the projection of phi(x) onto phi(y). or colloquially, how much overlap do x and y have in their feature space. In other words, how similar they are.

[ ]:

0.6 6 Discuss the following terms: Accuracy, Precision, Recall, F1 score, Specificity, Sensitivity, AUROC, PRAUC

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

0.7 7 perform classification on zoo dataset from kaggle using logistic regression after performing appropriate data pre processing and hyper parameter tuning and evaluate using the technique you feel is fit for this task and give your comments.

[98]:
```
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, KFold, cross_validate
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
```

[64]:
```
#save the csv in a dataframe
zoo_df = pd.read_csv("D:/Masters/SJSU/Academics/sem_2/CMPE_257_ML/Assignments/
↪HW4/Data/zoo.csv")
```

[65]:
```
zoo_df.head()
```

[65]:

	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize
0	aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1
1	antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1
2	bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0
3	bear	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1
4	boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1

```
   class_type
0           1
1           1
2           4
3           1
4           1
```

[66]:
```
# shape of the entire dataset
zoo_df.shape
```

[66]:
```
(101, 18)
```

[67]:
```
zoo_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   animal_name  101 non-null    object
 1   hair         101 non-null    int64
 2   feathers     101 non-null    int64
 3   eggs         101 non-null    int64
 4   milk         101 non-null    int64
 5   airborne     101 non-null    int64
 6   aquatic      101 non-null    int64
 7   predator     101 non-null    int64
 8   toothed      101 non-null    int64
 9   backbone     101 non-null    int64
 10  breathes     101 non-null    int64
 11  venomous     101 non-null    int64
 12  fins         101 non-null    int64
 13  legs         101 non-null    int64
 14  tail         101 non-null    int64
 15  domestic     101 non-null    int64
 16  catsize      101 non-null    int64
 17  class_type   101 non-null    int64
dtypes: int64(17), object(1)
memory usage: 14.3+ KB
```

Observation:

- There are no Null values

[100]:
```
# summary of the dataset

zoo_df.describe()
```

[100]:
```
          feathers        eggs    airborne     aquatic    predator     toothed  \
count  101.000000  101.000000  101.000000  101.000000  101.000000  101.000000
mean     0.198020    0.584158    0.237624    0.356436    0.554455    0.603960
std      0.400495    0.495325    0.427750    0.481335    0.499505    0.491512
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
50%      0.000000    1.000000    0.000000    0.000000    1.000000    1.000000
75%      0.000000    1.000000    0.000000    1.000000    1.000000    1.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

          backbone    breathes    venomous        fins        legs        tail  \
count  101.000000  101.000000  101.000000  101.000000  101.000000  101.000000
mean     0.821782    0.792079    0.079208    0.168317    2.841584    0.742574
std      0.384605    0.407844    0.271410    0.376013    2.033385    0.439397
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000    1.000000    0.000000    0.000000    2.000000    0.000000
50%      1.000000    1.000000    0.000000    0.000000    4.000000    1.000000
75%      1.000000    1.000000    0.000000    0.000000    4.000000    1.000000
max      1.000000    1.000000    1.000000    1.000000    8.000000    1.000000

          domestic     catsize  class_type
count  101.000000  101.000000  101.000000
mean     0.128713    0.435644    2.831683
std      0.336552    0.498314    2.102709
min      0.000000    0.000000    1.000000
25%      0.000000    0.000000    1.000000
50%      0.000000    0.000000    2.000000
75%      0.000000    1.000000    4.000000
max      1.000000    1.000000    7.000000
```

[69]:
```
# check the target feature

zoo_df['class_type'].value_counts()
```

[69]:
```
1    41
2    20
4    13
7    10
6     8
3     5
5     4
Name: class_type, dtype: int64
```

[70]:
```
# plot the target variable distribution

x = zoo_df['class_type'].value_counts().index.tolist()
y = zoo_df['class_type'].value_counts().tolist()
```

```
fig = px.bar(x=x, y=y, color=x, title="Animal Class Type Distribution",
             labels={
                'x': 'Animal Class',
                'y': 'count'
                },)
fig.show()
```

```
<IPython.core.display.Javascript object>
```

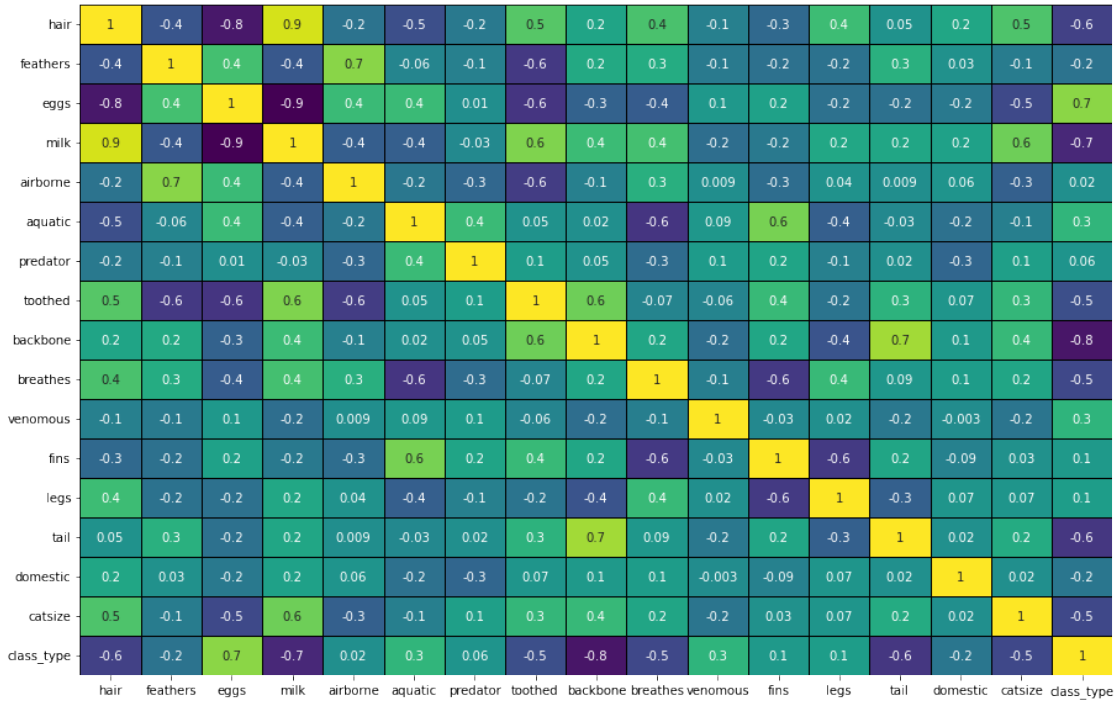
[71]:
```
zoo_df['animal_name'].value_counts()
```

[71]:
```
frog        2
pony        1
sealion     1
seal        1
seahorse    1
           ..
gorilla     1
goat        1
gnat        1
girl        1
wren        1
Name: animal_name, Length: 100, dtype: int64
```

[72]:
```
# correlation heatmap

fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(zoo_df.corr(), annot=True, fmt='.1g', cmap="viridis", cbar=False,␣
 ↪linewidths=0.5, linecolor='black');
```

```
<IPython.core.display.Javascript object>
```



Observation:

- Dropping **hair** feature since it's high correlated with milk feature
- Dropping **milk** feature since it's highly correlated with eggs
- Dropping **animal_name** as it is insignificant

[73]:
```
# drop the above columns

zoo_df.drop(['animal_name','hair', 'milk'], axis=1, inplace=True)
```

[74]:
```
# drop the target feature from the input data

X = zoo_df.drop("class_type", axis=1)
X.head()
```

[74]:

	feathers	eggs	airborne	aquatic	predator	toothed	backbone	breathes	\
0	0	0	0	0	1	1	1	1	
1	0	0	0	0	0	1	1	1	
2	0	1	0	1	1	1	1	0	
3	0	0	0	0	1	1	1	1	
4	0	0	0	0	1	1	1	1	

	venomous	fins	legs	tail	domestic	catsize
0	0	0	4	0	0	1

```
1          0     0     4     1          0          1
2          0     1     0     1          0          0
3          0     0     4     0          0          1
4          0     0     4     1          0          1
```

[75]:
```
# saving the target feature in y variable

y = zoo_df["class_type"]
y.head()
```

[75]:
```
0    1
1    1
2    4
3    1
4    1
Name: class_type, dtype: int64
```

Splitting the data into training and test datasets

Here, we are trying to predict the class type of the animal using the given data. Hence, the class_type will be the y label and rest of the data will be the X or the input data

[86]:
```
# split the dataset into train test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4,␣
 ↪random_state = 0)

print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(60, 14)
(60,)
(41, 14)
(41,)
```

[87]:
```
# Train the logsitic regression model

LRmodel = LogisticRegression(max_iter = 1500)
LRmodel.fit(X_train, y_train)
```

[87]:
```
LogisticRegression(max_iter=1500)
```

[88]:
```
# Predicting for the training set
y_train_pred = LRmodel.predict(X_train)
```

[89]:
```
# Checking the evaluation metrics of the train set
```

```
print(classification_report(y_train, y_train_pred))
```

```
              precision    recall  f1-score   support

           1       0.96      1.00      0.98        25
           2       1.00      1.00      1.00        13
           3       0.00      0.00      0.00         1
           4       1.00      1.00      1.00         5
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         5
           7       1.00      1.00      1.00         8

    accuracy                           0.98        60
   macro avg       0.85      0.86      0.85        60
weighted avg       0.97      0.98      0.98        60
```

```
D:\IDEs\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

D:\IDEs\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

D:\IDEs\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

[90]:
```
#calculating the confusion matrix for train set

confusion_matrix(y_train, y_train_pred)
```

[90]:
```
array([[25,  0,  0,  0,  0,  0,  0],
       [ 0, 13,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  5,  0,  0,  0],
       [ 0,  0,  0,  0,  3,  0,  0],
       [ 0,  0,  0,  0,  0,  5,  0],
       [ 0,  0,  0,  0,  0,  0,  8]], dtype=int64)
```

[91]:
```
# Predicting for the test set

y_test_pred = LRmodel.predict(X_test)
```

[92]:
```
# Checking the evaluation metrics of the test set

print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           1       0.88      0.94      0.91        16
           2       0.88      1.00      0.93         7
           3       0.00      0.00      0.00         4
           4       0.80      1.00      0.89         8
           5       1.00      1.00      1.00         1
           6       0.75      1.00      0.86         3
           7       1.00      0.50      0.67         2

    accuracy                           0.85        41
   macro avg       0.76      0.78      0.75        41
weighted avg       0.78      0.85      0.81        41
```

```
D:\IDEs\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

D:\IDEs\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

D:\IDEs\Anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318:
UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

[93]:
```
#calculating the confusion matrix for test set

confusion_matrix(y_test, y_test_pred)
```

[93]:
```
array([[15,  1,  0,  0,  0,  0,  0],
       [ 0,  7,  0,  0,  0,  0,  0],
```

```
       [ 2,  0,  0,  2,  0,  0,  0],
       [ 0,  0,  0,  8,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0,  3,  0],
       [ 0,  0,  0,  0,  0,  1,  1]], dtype=int64)
```

[96]:
```
# Now instead of train and test set, using cross validation

# since sv is 4 it'll give 4 different models

# CV is used to check homogeneity, overfitting or high variance or to tune␣
 ↪hyperparameters

model = LogisticRegression(max_iter = 1000)
scores = cross_validate(model, X, y, cv = 4, scoring = 'accuracy',␣
 ↪return_train_score = True)
scores
```

[96]:
```
{'fit_time': array([0.02692986, 0.02293873, 0.02094412, 0.0249331 ]),
 'score_time': array([0.00099754, 0.0009973 , 0.00099754, 0.00099754]),
 'test_score': array([0.96153846, 1.        , 0.84      , 0.88      ]),
 'train_score': array([0.97333333, 0.97368421, 0.98684211, 0.98684211])}
```

Observation:

- The accuracy of the **2nd model has the highest accuracy** on both train and test data
- **3rd model is performing worst.** We can infer that during that split the train and test data are not equally distributed

[99]:
```
#Method 2 - performing cross validation using Kfold with code

# Kfold gives the indices of training and testing dataset.

# This is similar to cross_validate()

acc_score = []

cv = KFold(n_splits = 4, random_state = 100, shuffle = True)

#loop runds for 4 times since n_splits = 4
for train_index , test_index in cv.split(X):

    X_train , X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train , y_test = y[train_index] , y[test_index]

    model = LogisticRegression(max_iter = 1000)
    model.fit(X_train,y_train)
    pred_values = model.predict(X_test)
```

```
    acc = accuracy_score(y_test, pred_values)
    acc_score.append(acc)

avg_acc_score = sum(acc_score)/4

print('accuracy of each fold - {}'.format(acc_score))
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
accuracy of each fold - [0.9615384615384616, 0.92, 0.88, 0.92]
Avg accuracy : 0.9203846153846154
```

Observation:

- The accuracy for the 3rd fold is **88%** so we can infer that the train and test split were not equally distributed
- The avg accuracy is **92%** which is quite good

[ ]:

0.8 8 perform classification on MNIST dataset using Linear SVM and then using Kernels and compare the methods

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

Sources to learn more about the topics:

https://www.ibm.com/topics/logistic-regression

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

https://towardsdatascience.com/kernel-function-6f1d2be6091

https://www.quora.com/What-are-kernels-in-machine-learning-and-SVM-and-why-do-we-need-them/answer/Lili-Jiang?srid=oOgT

[ ]: