In [86]:

```
# packages
%matplotlib notebook
import numpy as np
import pandas as pd
from collections import Counter
import warnings
warnings.filterwarnings("ignore")
```

In [87]:

```
df = pd.read_csv("zoo.csv")
```

In [88]:

```
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
```

In [89]:

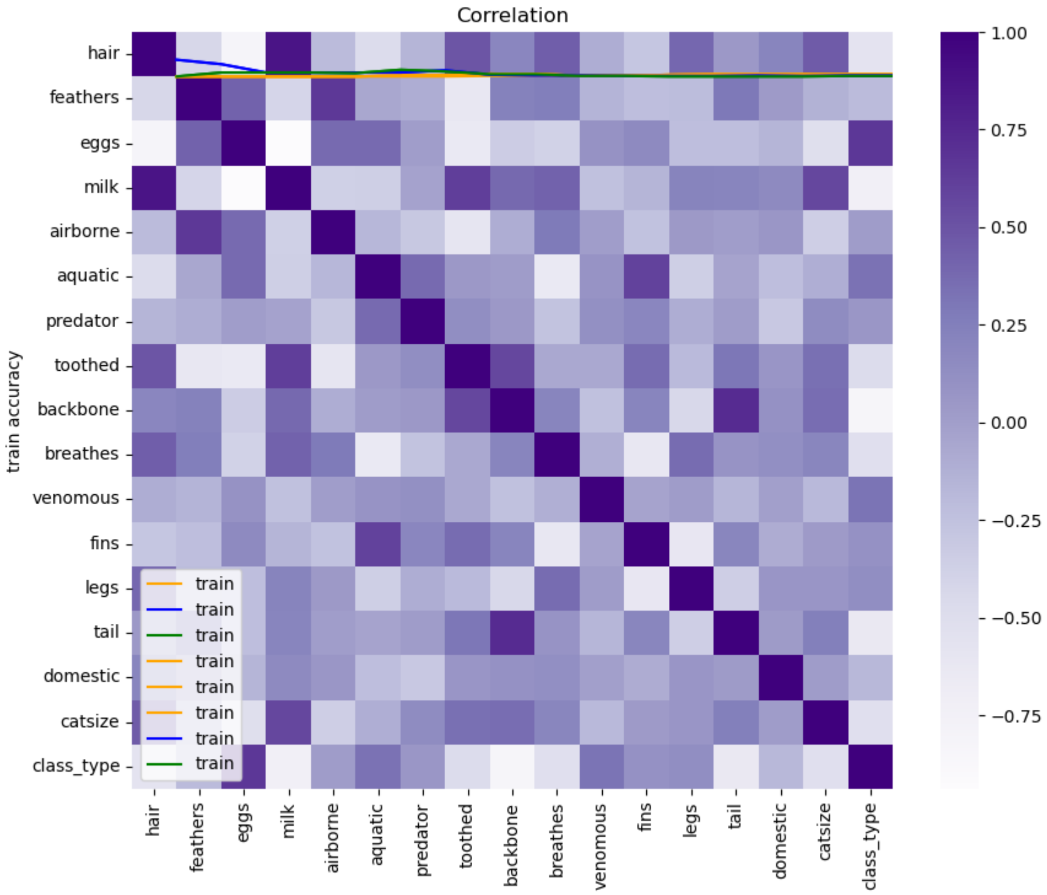
```
df.head()
```

Out[89]:

	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breat
0	aardvark	1	0	0	1	0	0	1	1	1	
1	antelope	1	0	0	1	0	0	0	1	1	
2	bass	0	0	1	0	0	1	1	1	1	
3	bear	1	0	0	1	0	0	1	1	1	
4	boar	1	0	0	1	0	0	1	1	1	

In [90]:
```
#correlation value between features
import seaborn as sns
import matplotlib.pyplot as plt
corr = ds.corr()
fig = plt.figure(figsize=(10,8))
r = sns.heatmap(corr, cmap='Purples')
r.set_title("Correlation")
```

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



Out[90]:
```
Text(0.5, 1.0, 'Correlation')
```

In [91]:
```
# data preperation
y = ds["class_type"].values
x_ds=ds.drop(["animal_name"],axis=1)
x = (x_ds-np.min(x_ds))/(np.max(x_ds)-np.min(x_ds))
from sklearn.model_selection import train_test_split
x_tn, x_ts, y_tn, y_ts = train_test_split(x,y,test_size = 0.2,random_state=1)
```

In [92]:
```
x_tn
```

Out[92]:
```
    hair  feathers  eggs  milk  airborne  aquatic  predator  toothed  backbone  breathes  venomou
32   1.0       0.0   0.0   1.0       0.0      0.0       0.0      1.0       1.0       1.0        0
40   1.0       0.0   1.0   0.0       1.0      0.0       0.0      0.0       0.0       1.0        0
39   1.0       0.0   1.0   0.0       1.0      0.0       0.0      0.0       0.0       1.0        1
38   0.0       0.0   1.0   0.0       0.0      1.0       1.0      1.0       1.0       0.0        0
46   0.0       0.0   1.0   0.0       0.0      1.0       1.0      0.0       0.0       0.0        0
...  ...       ...   ...   ...       ...      ...       ...      ...       ...       ...
75   1.0       0.0   0.0   1.0       0.0      1.0       1.0      1.0       1.0       1.0        0
 9   1.0       0.0   0.0   1.0       0.0      0.0       0.0      1.0       1.0       1.0        0
72   0.0       0.0   0.0   0.0       0.0      0.0       1.0      0.0       0.0       1.0        1
12   0.0       0.0   1.0   0.0       0.0      1.0       1.0      1.0       1.0       0.0        0
37   0.0       1.0   1.0   0.0       1.0      0.0       1.0      0.0       1.0       1.0        0

80 rows × 17 columns
```

In [93]:
```
# KNN package
from sklearn.neighbors import KNeighborsClassifier
# Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=2 )
```

In [94]:
```
# Train the model using the training sets
knn.fit(x_tn, y_tn)
```

Out[94]:
```
KNeighborsClassifier(n_neighbors=2)
```

In [95]:
```
# accuracy or score #train
knn.score(x_tn, y_tn)
```

Out[95]:
```
0.9625
```

In [96]:
```
# accuracy or score #test
knn.score(x_ts, y_ts)
```

Out[96]:
```
0.9047619047619048
```

In [97]:
```
from sklearn.linear_model import LogisticRegression
# creating linear regression object.
lgrgmodel = LogisticRegression()
```

```
In [98]: # Train the model using the train sets.
         lgrgmodel.fit(x_tn,y_tn)

Out[98]: LogisticRegression()
```

```
In [99]: # score of LR (Accuracy) #train
         lgrgmodel.score(x_tn, y_tn)

Out[99]: 0.975
```

```
In [100]: # score of LR (Accuracy) #test
          lgrgmodel.score(x_ts, y_ts)

Out[100]: 0.9523809523809523
```

```
In [101]: # SVM package
          from sklearn.svm import SVC
          svm=SVC(random_state=1)
```

```
In [102]: #train model using train set
          svm.fit(x_tn,y_tn)

Out[102]: SVC(random_state=1)
```

```
In [103]: # calc score of SVM (Accuracy)
          print("train accuracy:",svm.score(x_tn,y_tn))
          print("test accuracy:",svm.score(x_ts,y_ts))

          train accuracy: 0.9875
          test accuracy: 0.9047619047619048
```

```
In [ ]: from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import f_classif
```

```
In [ ]:
```

In [115]:

```
#improve the accuracy of prediction.#KNN

print("==================================#KNN")
accuracy_list_train_KNN = []
number_of_features_KNN = np.arange(1,18,1)
for each in number_of_features_KNN:
    x_new_KNN = SelectKBest(f_classif, k = each).fit_transform(x_tn, y_tn)
    knn.fit (x_new_KNN, y_tn)
    accuracy_list_train_KNN.append(knn.score(x_new_KNN, y_tn))

plt.plot(number_of_features_KNN,accuracy_list_train_KNN,color="orange", label="tr
plt.xlabel("number of features")
plt.ylabel("train accuracy")
plt.legend()
plt.show()

#improve the accuracy of prediction.#LR

print("==================================#LR")
accuracy_list_train_LR = []
number_of_features_LR = np.arange(1,18,1)
for each in number_of_features_LR:
    x_new_LR = SelectKBest(f_classif, k = each).fit_transform(x_tn, y_tn)
    lgrgmodel.fit(x_new_LR, y_tn)
    accuracy_list_train_LR.append(lgrgmodel.score(x_new_LR, y_tn))

plt.plot(number_of_features_LR,accuracy_list_train_LR,color="blue", label="train"
plt.xlabel("number of features")
plt.ylabel("train accuracy")
plt.legend()
plt.show()

#improve the accuracy of prediction.#SVM

print("==================================#SVM")
accuracy_list_train = []
number_of_features = np.arange(1,18,1)
for each in number_of_features:
    x_new = SelectKBest(f_classif, k = each).fit_transform(x_tn, y_tn)
    svm. fit (x_new, y_tn)
    accuracy_list_train.append(svm.score(x_new, y_tn))


plt.plot(number_of_features,accuracy_list_train,color="green", label="train")
plt.xlabel("number of features")
plt.ylabel("train accuracy")
plt.legend()
plt.show()
```

```
==================================#KNN
==================================#LR
==================================#SVM
```

In [105]:

```
#best accuracy

#KNN
print("=================================#KNN")
d_KNN = {'best features number': number_of_features_KNN, 'train_score': accuracy_
df_KNN = pd.DataFrame(data=d_KNN)

print("max accuracy:",df_KNN["train_score"].max())

print("max accuracy id:",df_KNN["train_score"].idxmax() )

#LR
print("=================================#LR")
d_LR = {'best features number': number_of_features_LR, 'train_score': accuracy_li
df_LR = pd.DataFrame(data=d_LR)

print("max accuracy:",df_LR["train_score"].max())

print("max accuracy id:",df_LR["train_score"].idxmax() )

#SVM
print("=================================#SVM")
d = {'best features number': number_of_features, 'train_score': accuracy_list_tra
df = pd.DataFrame(data=d)

print("max accuracy:",df["train_score"].max())

print("max accuracy id:",df["train_score"].idxmax() )
```

```
=================================#KNN
max accuracy: 1.0
max accuracy id: 0
=================================#LR
max accuracy: 0.9875
max accuracy id: 10
=================================#SVM
max accuracy: 1.0
max accuracy id: 0
```

In [106]:

```

#KNN
print("=================================#KNN")
print("max accuracy values: \n",df_KNN.iloc[9])
#LR
print("=================================#LR")
print("max accuracy values: \n",df_LR.iloc[9])

#SVM
print("=================================#SVM")
print("max accuracy values: \n",df.iloc[9])
```

```
=================================#KNN
max accuracy values: 
 best features number    10.0000
train_score             0.9875
Name: 9, dtype: float64
=================================#LR
max accuracy values: 
 best features number    10.000
train_score             0.975
Name: 9, dtype: float64
=================================#SVM
max accuracy values: 
 best features number    10.0000
train_score             0.9875
Name: 9, dtype: float64
```

In [107]:

```
# Arrange the train and test dataset including best features
#KNN

selector = SelectKBest(f_classif, k = 10)

x_new_KNN = selector.fit_transform(x_tn, y_tn)
x_new_test_KNN=selector. fit_transform(x_ts, y_ts)

names_train_KNN = x_tn.columns.values[selector.get_support()]
names_test_KNN = x_ts.columns.values[selector.get_support() ]

print("x train features:",names_train_KNN)
print("x test features:",names_test_KNN)
```

```
x train features: ['hair' 'feathers' 'eggs' 'milk' 'airborne' 'toothed' 'backbo
ne'
 'breathes' 'tail' 'class_type']
x test features: ['hair' 'feathers' 'eggs' 'milk' 'airborne' 'toothed' 'backbon
e'
 'breathes' 'tail' 'class_type']
```

In [108]:

```
# Arrange the train and test dataset including best features
#LR

selector = SelectKBest(f_classif, k = 10)

x_new_LR = selector.fit_transform(x_tn, y_tn)
x_new_test_LR=selector. fit_transform(x_ts, y_ts)

names_train_LR = x_tn.columns.values[selector.get_support()]
names_test_LR = x_ts.columns.values[selector.get_support() ]

print("x train features:",names_train_LR)
print("x test features:",names_test_LR)
```

```
x train features: ['hair' 'feathers' 'eggs' 'milk' 'airborne' 'toothed' 'backbo
ne'
 'breathes' 'tail' 'class_type']
x test features: ['hair' 'feathers' 'eggs' 'milk' 'airborne' 'toothed' 'backbon
e'
 'breathes' 'tail' 'class_type']
```

In [109]:

```
# Arrange the train and test dataset including best features
#SVM

selector = SelectKBest(f_classif, k = 10)

x_new = selector.fit_transform(x_tn, y_tn)
x_new_test=selector. fit_transform(x_ts, y_ts)

names_train = x_tn.columns.values[selector.get_support()]
names_test = x_ts.columns.values[selector.get_support() ]

print("x train features:",names_train)
print("x test features:",names_test)
```

```
x train features: ['hair' 'feathers' 'eggs' 'milk' 'airborne' 'toothed' 'backbo
ne'
 'breathes' 'tail' 'class_type']
x test features: ['hair' 'feathers' 'eggs' 'milk' 'airborne' 'toothed' 'backbon
e'
 'breathes' 'tail' 'class_type']
```

In [110]:

```
#Re-train and re-calculate the model accuracy using the new arrangement of featur
#KNN
#import backage
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

knn = KNeighborsClassifier( )
knn.fit(x_new, y_tn)

#Calc score or (Accuracy)
print("train accuracy: ",knn.score(x_new_KNN, y_tn))
print("test accuracy: ",knn.score(x_new_test_KNN,y_ts))
```

```
train accuracy:  0.975
test accuracy:  0.9047619047619048
```

In [111]:

```
#Re-train and re-calculate the model accuracy using the new arrangement of featur
#LR
#import backage
from sklearn.linear_model import LogisticRegression

lgrgmodel = LogisticRegression()

lgrgmodel.fit(x_new, y_tn)

#Calc score or (Accuracy)
print("train accuracy: ",lgrgmodel.score(x_new_LR, y_tn))
print("test accuracy: ",lgrgmodel.score(x_new_test_LR,y_ts))
```

```
train accuracy:  0.975
test accuracy:  0.8571428571428571
```

In [112]:

```
#Re-train and re-calculate the model accuracy using the new arrangement of featur
#SVM

from sklearn.svm import SVC

svm=SVC( random_state=1)

svm. fit (x_new, y_tn)

print("train accuracy:",svm.score(x_new, y_tn) )
print("test accuracy:",svm.score(x_new_test,y_ts) )
```

```
train accuracy: 0.9875
test accuracy: 0.6666666666666666
```