In [1]:

```
from pandas import read_csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import warnings
from sklearn.metrics import accuracy_score
warnings.filterwarnings('ignore')
```

In [2]:

```
#Loading the dataset
df = pd.read_csv("C:/Users/SAURABH/Saurabh patil/DATA SCIENCE/RNN/zoo.csv")
df
```

Out[2]:

	animal name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	type
0	aardvark	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	1
1	antelope	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
2	bass	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
3	bear	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	1
4	boar	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
96	wallaby	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0	1	1
97	wasp	1	0	1	0	1	0	0	0	0	1	1	0	6	0	0	0	6
98	wolf	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
99	worm	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	7
100	wren	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0	0	2

101 rows × 18 columns

In [4]:

```
#Checking for null values & data types
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  type         101 non-null    int64
dtypes: int64(17), object(1)
memory usage: 14.3+ KB
```

Since number of columns are more, let's use PCA

In [5]:

```
#Scaling the data (leaving out the target variable, and the taking only the numerical data for input)
dfin= df.iloc[:,1:17]

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

sc.fit(dfin)
df_norm = sc.transform(dfin)
df_norm              #Normalized dataset
```

Out[5]:

```
array([[ 1.16139451, -0.49690399, -1.18522652, ..., -1.69841555,
        -0.38435306,  1.13818037],
       [ 1.16139451, -0.49690399, -1.18522652, ...,  0.58878406,
        -0.38435306,  1.13818037],
       [-0.86103386, -0.49690399,  0.84372057, ...,  0.58878406,
        -0.38435306, -0.87859537],
       ...,
       [ 1.16139451, -0.49690399, -1.18522652, ...,  0.58878406,
        -0.38435306,  1.13818037],
       [-0.86103386, -0.49690399,  0.84372057, ..., -1.69841555,
        -0.38435306, -0.87859537],
       [-0.86103386,  2.01246118,  0.84372057, ...,  0.58878406,
        -0.38435306, -0.87859537]])
```

In [6]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 16)
pca_values = pca.fit_transform(df_norm)
pca_values
```

Out[6]:

```
array([[ 2.52525106, -0.29027784,  1.24164209, ..., -0.40003753,
         [illegible]],
       [illegible]
       ...,
       [illegible]])
```

In [7]:

```
# The amount of variance that each PCA explains is
var = pca.explained_variance_ratio_
var
```

Out[7]:

```
array([0.29101491, 0.20882094, [illegible]
       0.00111144])
```

In [8]:

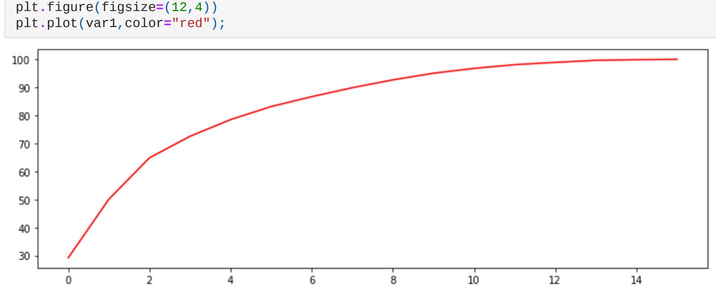
```
# Cumulative variance
var1 = np.cumsum(np.round(var,decimals = 4)*100)
var1
```

Out[8]:

```
array([29.16, 50.07, 64.85, 71.54, [illegible], 83.16, 86.68, 89.88, 92.65,
       94.92, 96.67, 97.99, 98.75, 99.4 , 99.89, 99.99])
```

In [9]:

```
# Variance plot for PCA components obtained
plt.figure(figsize=(12,4))
plt.plot(var1,color="red");
```



Selecting first 14 PCAs out of total 16

In [10]:

```
finalDf = pd.concat([pd.DataFrame(pca_values[:,0:14],columns=['pc1','pc2','pc3','pc4','pc5','pc6','pc7','pc8',
                     'pc9','pc10','pc11','pc12','pc13','pc14']), df[['type']]], axis = 1)
finalDf
```

Out[10]:

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	pc10	pc11	pc12	pc13	pc14	type
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4
3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
96	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
97	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	6
98	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1
99	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	7
100	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2

101 rows × 15 columns

In [11]:

```
array = finalDf.values
X = array[:,0:14]
Y = array[:,14]
```

Selecting the model validation technique

Trial 1 : Train Test split approach

In [13]:

```
from sklearn.model_selection import train_test_split
import numpy as np
test_size = 0.3
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
np.round(result, 4)
```

Out[13]: 0.9677

Trial 2 : Cross Validation approach

In [14]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import numpy as np
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression(max_iter=400)
results = cross_val_score(model, X, Y, cv=kfold)
print('Result:',np.round(results.mean(),4),'\n','\n','Standard dev:',np.round(results.std(),4))
```

```
Result: 0.96

 Standard dev: 0.0663
```

In [15]:

```
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
loocv = LeaveOneOut()
model = LogisticRegression(max_iter=400)
results = cross_val_score(model, X, Y, cv=loocv)
print('Result:',np.round(results.mean(),4),'\n','\n','Standard dev:',np.round(results.std(),4))
```

```
Result: 0.9604

 Standard dev: 0.195
```

Hence, Train Test Split is the best model vaidation technique here, so we'll proceed with that

KNN Classification

Let's use Grid search CV to find out best value for K

In [16]:

```
# Grid Search for Algorithm Tuning
import numpy
from pandas import read_csv
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

n_neighbors = numpy.array(range(1,40))
param_grid = dict(n_neighbors=n_neighbors)

model = KNeighborsClassifier()
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid.fit(X, Y)

print(grid.best_score_)
print(grid.best_params_)
```

```
0.9504761904761905
{'n_neighbors': 1}
```

In [17]:

```
import matplotlib.pyplot as plt
%matplotlib inline
# choose k between 1 to 40
k_range = range(1, 40)
k_scores = []
# use iteration to caclulator different k in models, then return the average accuracy based on the cross validation
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, Y, cv=5)
    k_scores.append(scores.mean())
# plot to see clearly
plt.figure(figsize=(15,7))
plt.plot(k_range, k_scores)
plt.axhline(y=0.9504761904761905, color='r', linestyle='--')
plt.axvline(x=1, color='r', linestyle='--')
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.show()
```

Hence K=1 is the best value, so we'll make the model using that.

In [18]:

```
#KNN Classification
model = KNeighborsClassifier(n_neighbors=1) #making the model
model.fit(X_train , Y_train) #training the model
y_pred = model.predict(X_test) #predicting on the test dataset
acc = accuracy_score(Y_test, y_pred) * 100
print("Accuracy =", acc)
```

```
Accuracy = 93.54838709677419
```

SVM Classification

In [19]:

```
# SVM Classification
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.preprocessing import StandardScaler

from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report


from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split, cross_val_score
```

Let's use Grid search CV to find out best value for params

In [20]:

```
clf = SVC()
param_grid = [{'kernel':['rbf'],'gamma':[0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2,0.1],'C':[1,10,100,1000] },
             {'kernel':['linear'],'C':[1,10,100,1000]}]
gsv = GridSearchCV(clf,param_grid,cv=10,n_jobs=-1)
gsv.fit(X,Y)

gsv.best_params_ , gsv.best_score_
```

Out[20]: ({'C': 1, 'kernel': 'linear'}, 0.97)

In [21]:

```
#SVM Clasification
clf = SVC(C= 1, kernel='linear') #building the model
clf.fit(X_train , Y_train) #training the model
y_pred = clf.predict(X_test) #predicting on test dataset
acc = accuracy_score(Y_test, y_pred) * 100
print("Accuracy =", acc)
```

```
Accuracy = 96.7741935483871
```

Now, let's try some Ensemble methods to see if we can further increase the accuracy of the model

Trial-1: Bagging

In [22]:

```
# Bagged Decision Trees for Classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

seed = 7
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
model.fit(X_train,Y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 96.7741935483871
```

Trial-2: Random Forest

In [23]:

```
# Random Forest Classification

from sklearn.ensemble import RandomForestClassifier

num_trees = 100
max_features = 3
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
model.fit(X_train,Y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 96.7741935483871
```

Trial-3: Boosting

In [24]:

```
# AdaBoost Classification

from sklearn.ensemble import AdaBoostClassifier
num_trees = 100
seed=7

model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
model.fit(X_train,Y_train)
y_pred = model.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 87.09677419354838
```

Trial-4: Stacking

In [25]:

```
# Stacking Ensemble for Classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
```

In [26]:

```
# create the sub models
estimators = []
model = LogisticRegression(max_iter=500)
estimators.append(('logistic', model))
model = DecisionTreeClassifier()
estimators.append(('cart', model))
model = SVC()
estimators.append(('svm', model))

# create the ensemble model
ensemble = VotingClassifier(estimators)
ensemble.fit(X_train,Y_train)
y_pred = ensemble.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 93.54838709677419
```

In [27]:

```
# create the sub models
estimators = []
model = LogisticRegression(max_iter=500)
estimators.append(('logistic', model))
model = DecisionTreeClassifier()
estimators.append(('cart', model))
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('Boosting', model))

# create the ensemble model
ensemble = VotingClassifier(estimators)
ensemble.fit(X_train,Y_train)
y_pred = ensemble.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 96.7741935483871
```

In [28]:

```
# create the sub models
estimators = []
model = LogisticRegression(max_iter=500)
estimators.append(('logistic', model))
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('boosting', model))
model = SVC()
estimators.append(('svm', model))

# create the ensemble model
ensemble = VotingClassifier(estimators)
ensemble.fit(X_train,Y_train)
y_pred = ensemble.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 96.7741935483871
```

In [29]:

```
# create the sub models
estimators = []
model = LogisticRegression(max_iter=500)
estimators.append(('logistic', model))
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
estimators.append(('boosting', model))


# create the ensemble model
ensemble = VotingClassifier(estimators)
ensemble.fit(X_train,Y_train)
y_pred = ensemble.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 87.09677419354838
```

In [30]:

```
# create the sub models
estimators = []
model = LogisticRegression(max_iter=500)
estimators.append(('logistic', model))
model = SVC()
estimators.append(('svm', model))

# create the ensemble model
ensemble = VotingClassifier(estimators)
ensemble.fit(X_train,Y_train)
y_pred = ensemble.predict(X_test)
acc = accuracy_score(Y_test, y_pred)*100
print('Accuracy:',acc)
```

```
Accuracy: 90.32258064516128
```

Hence, we can say that SVM is the best predicting model for this dataset

In [ ]: