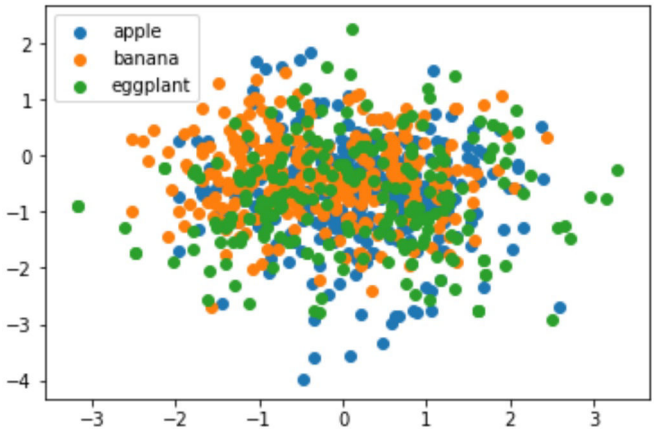
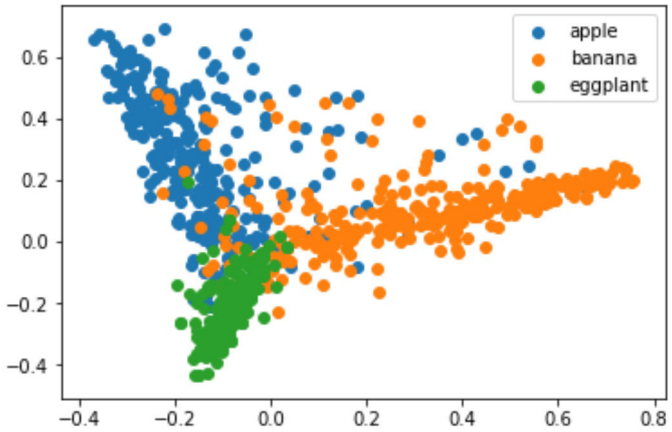
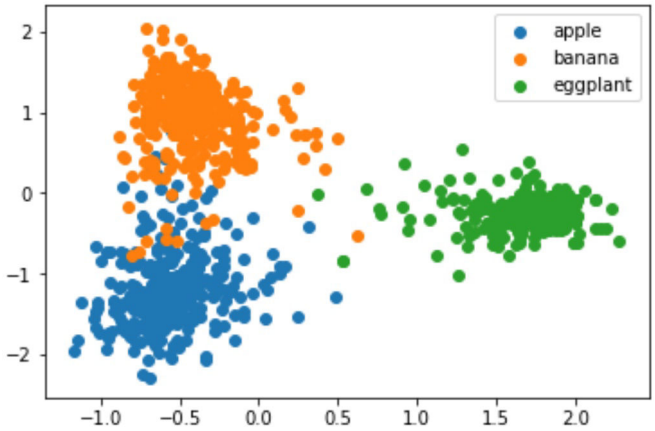
In [1]:
```
# utils.py

from numpy.random import uniform
import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

from sklearn.preprocessing import StandardScaler


def create_one_hot_label(Y,N_C):
        '''
        Input
        Y: list of class labels (int)
        N_C: Number of Classes

        Returns
        List of one hot arrays with dimension N_C

        '''

        y_one_hot = []
        for y in Y:

                one_hot_label = np.zeros(N_C)

                one_hot_label[y] = 1.0
                y_one_hot.append(one_hot_label)

        return  y_one_hot


def subtract_mean_from_data(X,Y):
        '''
        Input
        X: List of data points
        Y: list of one hot class labels

        Returns
        X and Y with mean subtracted

        '''

        ss_x = StandardScaler(with_std = False)
        ss_y = StandardScaler(with_std = False)

        ss_x.fit(X)
        X = ss_x.transform(X)

        ss_y.fit(Y)
        Y = ss_y.transform(Y)

        return X,Y


def compute_covariance_matrix(X,Y):
        '''
        Input
        X: List of data points
        Y: list of one hot class labels
```

In [2]:
```
# projection.py

from numpy.random import uniform
from numpy.random import randn
import random
import time

import matplotlib.pyplot as plt

from scipy.linalg import eig
from scipy.linalg import sqrtm
from numpy.linalg import inv
from numpy.linalg import svd

from utils import create_one_hot_label
from utils import subtract_mean_from_data
from utils import compute_covariance_matrix

import numpy as np
import numpy.linalg as LA

import sys
from numpy.linalg import svd


class Project2D():

        '''
        Class to draw projection on 2D scatter space
        '''

        def __init__(self,projection, clss_labels):

                self.proj = projection
                self.clss_labels = clss_labels


        def project_data(self,X,Y,white=None):

                '''
                Takes list of state space and class labels
                State space should be 2D
                Labels shoud be int
                '''

                p_a = []
                p_b = []
                p_c = []

                # Project all Data
                proj = np.matmul(self.proj,white)

                X_P = np.matmul(proj,np.array(X).T)

                for i in range(len(Y)):

                        if Y[i] == 0:
                                p_a.append(X_P[:,i])
                        elif Y[i] == 1:
                                p_b.append(X_P[:,i])
                        else:
                                p_c.append(X_P[:,i])
```


apple
banana
eggplant
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3


apple
banana
eggplant
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8


apple
banana
eggplant
2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

In [3]:
```
# Ridge_Model

from numpy.random import uniform
import random
import time

import numpy as np
import numpy.linalg as LA

import sys

from sklearn.linear_model import Ridge

from utils import create_one_hot_label


class Ridge_Model():

        def __init__(self,class_labels):

                ###RIDGE HYPERPARAMETER
                self.lmda = 1.0
                self.class_labels = class_labels
                self.ridge_model = Ridge(self.lmda)


        def train_model(self,X,Y):
                ''''
                FILL IN CODE TO TRAIN MODEL
                MAKE SURE TO ADD HYPERPARAMTER TO MODEL

                '''

                X = np.array(X)
                y_one_hot = create_one_hot_label(Y, len(self.class_labels))
                self.ridge_model.fit(X, y_one_hot)


        def eval(self,x):
                ''''
                Fill in code to evaluate model and return a prediction
                Prediction should be an integer specifying a class
                '''
                x = x.reshape(1, -1)
                y = self.ridge_model.predict(x)
                return np.argmax(y)
```

In [ ]:
```
# LDA

import random
import time

import glob
import os
import pickle
import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys
from numpy.linalg import inv
from numpy.linalg import det
from sklearn.s m import  inearS C
from projection import Project2 , Projections

from utils import subtract_mean_from_data
from utils import compute_covariance_matrix


class LDA_Model():

        def __init__(self,class_labels):

                ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COV
ARIANCE MATRIX TO PREVENT IT BEING SINGULAR ###
                self.reg_cov = 0.001
                self.N M_C  SS S = len(class_labels)


        def train_model(self,X,Y):
                ''''
                FILL IN CODE TO TRAIN MODEL
                MAKE SURE TO ADD HYPERPARAMTER TO MODEL

                '''
                ps = [ [] for j in range(self.N M_C  SS S) ]
                for i, y in enumerate(Y):
                        ps[y].append(X[i])

                self.mean_list = []
                for lst in ps:
                        self.mean_list.append( np.mean(np.array(lst), axis=0) )

                Sigma_XX = compute_covariance_matrix(X, X)
                Sigma_XX  = self.reg_cov   np.identity(Sigma_XX.shape[0])
                self.Sigma_inv = inv(Sigma_XX)


        def eval(self,x):
                ''''
                Fill in code to evaluate model and return a prediction
                Prediction should be an integer specifying a class
                '''
                x = x.reshape(1, -1)
                y =
                for i in range(self.N M_C  SS S):
                        x_demeaned = x - self.mean_list[i]
                        f = - x_demeaned.dot(self.Sigma_inv).dot(x_demeaned.T)
```

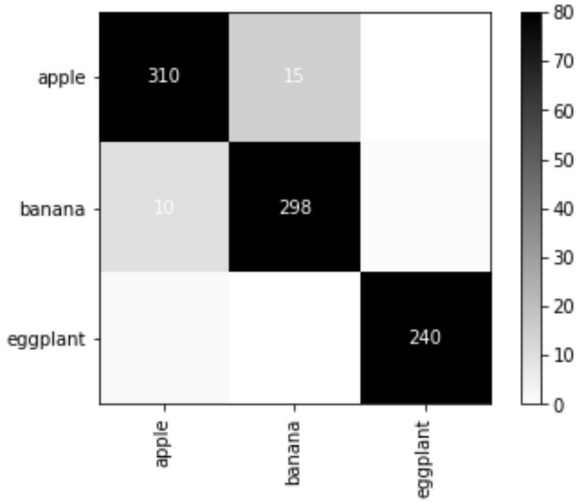
In [ ]:
```
# QDA

import random
import time

import numpy as np
import numpy.linalg as LA

from numpy.linalg import inv
from numpy.linalg import det

from projection import Project2 , Projections

from utils import subtract_mean_from_data
from utils import compute_covariance_matrix


class  DA_Model():

        def __init__(self,class_labels):

                ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COV
ARIANCE MATRIX TO PREVENT IT BEING SINGULAR ###
                self.reg_cov = 0.01
                self.N M_C  SS S = len(class_labels)


        def train_model(self,X,Y):
                ''''
                FILL IN CODE TO TRAIN MODEL
                MAKE SURE TO ADD HYPERPARAMTER TO MODEL

                '''
                ps = [ [] for j in range(self.N M_C  SS S) ]
                for i, y in enumerate(Y):
                        ps[y].append(X[i])

                self.mean_list = []
                self.Sigma_inv_list = []
                for lst in ps:
                        self.mean_list.append( np.mean(np.array(lst), axis=0) )
                        Sigma_XX = compute_covariance_matrix(lst, lst)
                        Sigma_XX  = self.reg_cov   np.identity(Sigma_XX.shape[0]
)
                        self.Sigma_inv_list.append(inv(Sigma_XX))


        def eval(self,x):
                ''''
                Fill in code to evaluate model and return a prediction
                Prediction should be an integer specifying a class
                '''
                x = x.reshape(1, -1)
                y =
                for i in range(self.N M_C  SS S):
                        x_demeaned = x - self.mean_list[i]
                        f = - x_demeaned.dot(self.Sigma_inv_list[i]).dot(x_demea
ned.T)
                        y[i] = f.flatten()[0]
                return max(y,  ey=lambda x: y[x])
```

In [ ]:
```
# SVM

from numpy.random import uniform
import random
import time

import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys

from sklearn.s m import  inearS C
from projection import Project2 , Projections

from utils import create_one_hot_label


class   M_Model():

        def __init__(self,class_labels,projection=None):

                ###SLACK HYPERPARAMETER
                self.C = 1.0
                self.class_labels = class_labels
                self.svc_model =  inearS C(C=self.C)


        def train_model(self,X,Y):
                ''''
                FILL IN CODE TO TRAIN MODEL
                MAKE SURE TO ADD HYPERPARAMTER TO MODEL

                '''
                X = np.array(X)
                self.svc_model.fit(X, Y)


        def eval(self,x):
                ''''
                Fill in code to evaluate model and return a prediction
                Prediction should be an integer specifying a class
                '''
                x = x.reshape(1, -1)
                y = self.svc_model.predict(x)
                return y[0]
```

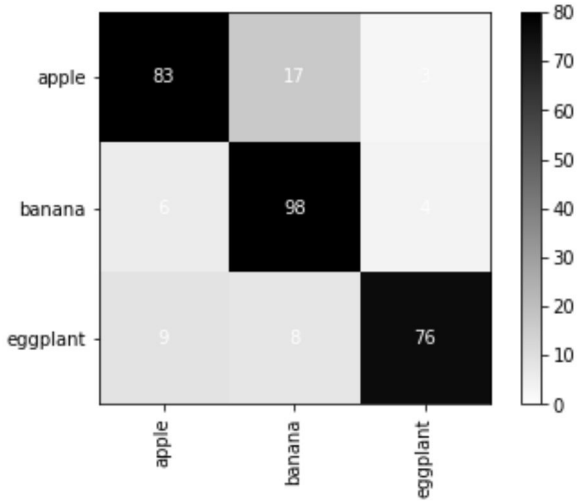
In [17]:
```
# confusion_mat.py

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import random
import IPython

def main():
    """
    Result
    Plot RANDOM confusion matrix (true labels vs. predicted labels)
    """
    true_labels = [random.randint(1, 10) for i in range(100)]
    predicted_labels = [random.randint(1, 10) for i in range(100)]

    # Plot confusion matrix (true labels vs. predicted labels)
    plot = getConfusionMatrixPlot(true_labels, predicted_labels)
    plot.show()


def getConfusionMatrix(true_labels, predicted_labels):
    """
    Input
    true_labels: actual labels
    predicted_labels: model's predicted labels

    Output
    cm: confusion matrix (true labels vs. predicted labels)
    """

    # Generate confusion matrix using sklearn.metrics
    cm = confusion_matrix(true_labels, predicted_labels)
    return cm


def plotConfusionMatrix(cm, alphabet):
    """
    Input
    cm: confusion matrix (true labels vs. predicted labels)
    alphabet: names of class labels

    Output
    Plot confusion matrix (true labels vs. predicted labels)
    """

    fig = plt.figure()
    plt.clf()                       # Clear plot
    ax = fig.add_subplot(111)       # Add 1x1 grid, first subplot
    ax.set_aspect(1)
    res = ax.imshow(cm, cmap=plt.cm.binary,
                    interpolation='nearest', vmin=0, vmax=80)

    plt.colorbar(res)               # Add color bar

    width = len(cm)                 # Width of confusion matrix
    height = len(cm[0])             # Height of confusion matrix

    # Annotate confusion entry with numeric value
    for x in range(width):
        for y in range(height):
            ax.annotate(str(cm[x][y]), xy=(y, x), horizontalalignment='center',
                        verticalalignment='center', color=getFontColor(cm[x][y])
```

In [7]:
```
# linear_classification.py

from numpy.random import uniform
import random
import time

import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys

from projection import Project2D, Projections
from confusion_mat import getConfusionMatrixPlot

from ridge_model import Ridge_Model
from  da_model import  DA_Model
from lda_model import LDA_Model
from svm_model import SVM_Model


CLASS_LA ELS = ['apple','banana','eggplant']


class Model():
        """ Generic wrapper for specific model instance. """

        def __init__(self, model):
                """ Store specific pre-initialized model instance. """

                self.model = model


        def train_model(self,X,Y):
                """ Train using specific model's training function. """

                self.model.train_model(X,Y)


        def test_model(self,X,Y):
                """ Test using specific model's eval function. """

                labels = []                                             # List o
f actual labels
                p_labels = []                                   # List of model'
s predictions
                success = 0                                             # Number
of correct predictions
                total_count = 0                                 # Number of imag
es

                for i in range(len(X)):

                        x = X[i]                                        # Test i
nput
                        y = Y[i]                                        # Actual
label
                        y_ = self.model.eval(x)         # Model's prediction
                        labels.append(y)
                        p_labels.append(y_)
```

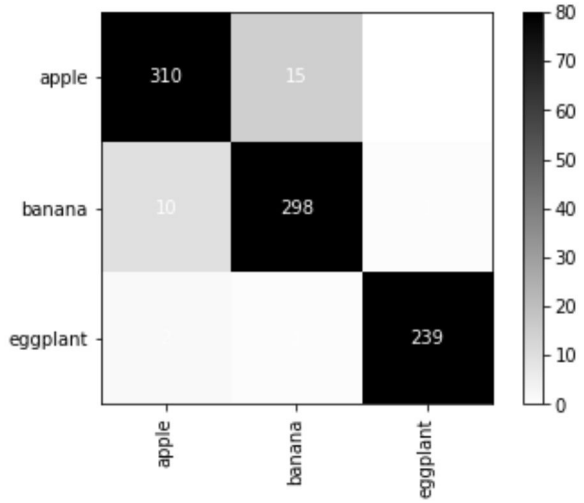
```
[[310  15   0]
 [ 10 2 8   1]
 [  2   0 240]]
```



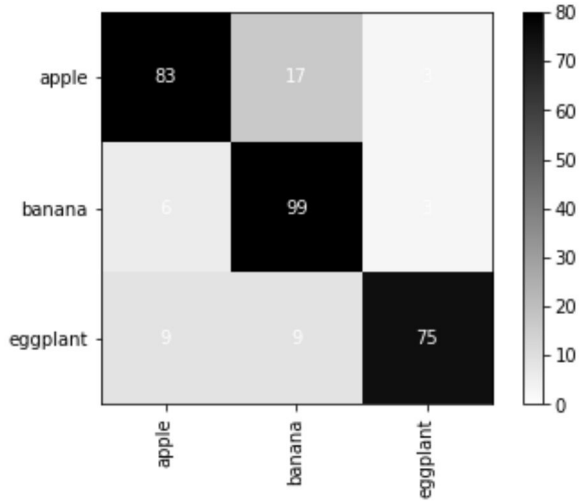

```
[[83 17  3]
 [ 6  8  4]
 [    8 76]]
```



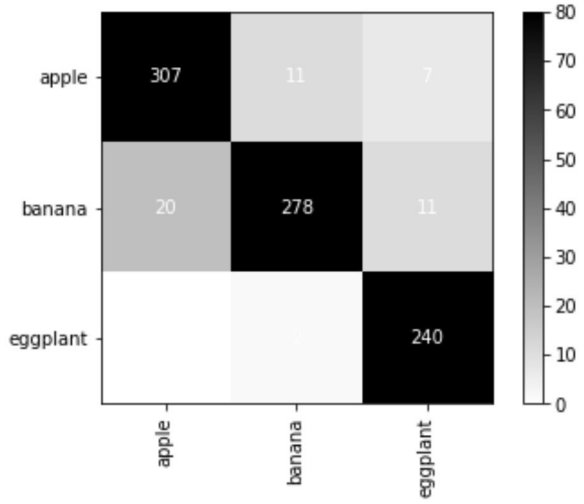

```
[[310  15   0]
 [ 10 2 8   1]
 [  2   1 23 ]]
```



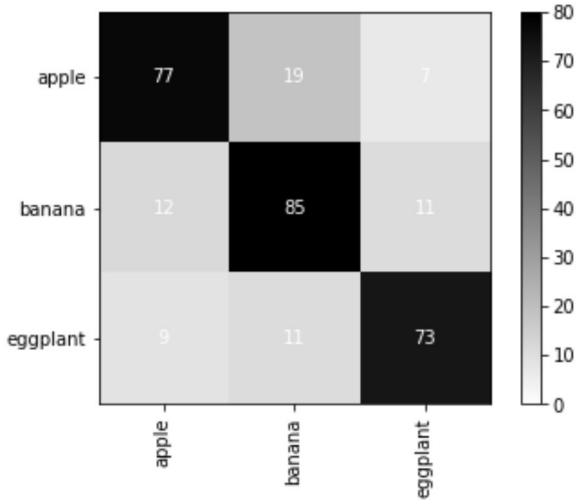

```
[[83 17  3]
 [ 6     3]
 [       75]]
```



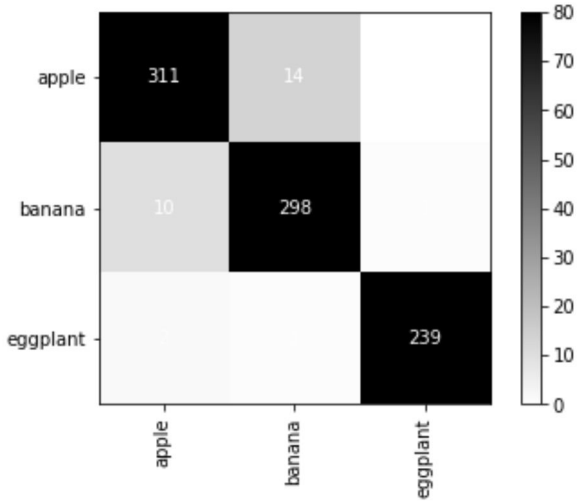

```
[[307  11   7]
 [ 20 278  11]
 [  0   2 240]]
```



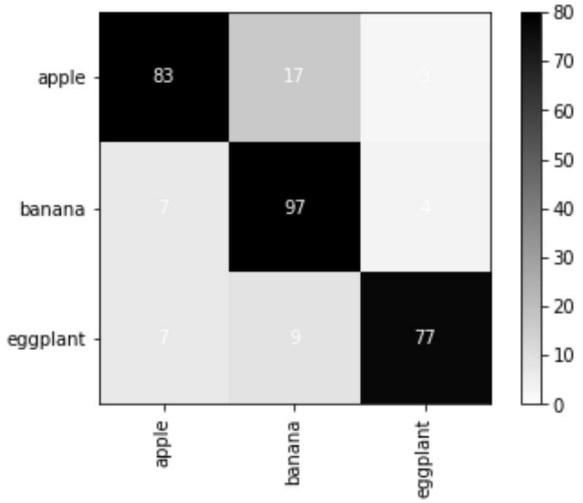
```
[[77 1   7]
 [12 85 11]
 [   11 73]]
```



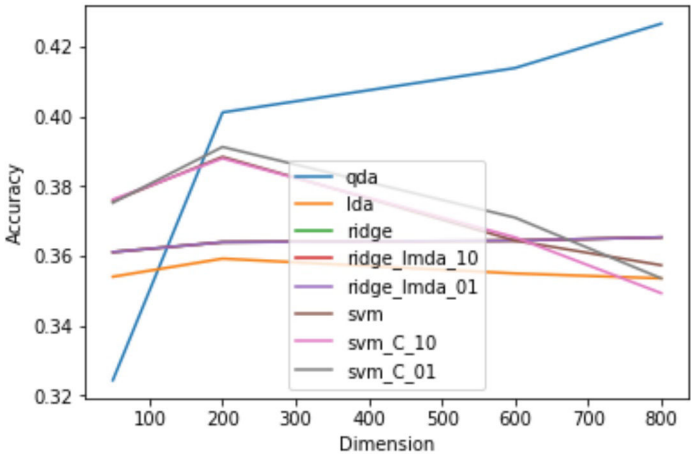

```
[[311  14   0]
 [ 10 2 8   1]
 [  2   1 23 ]]
```

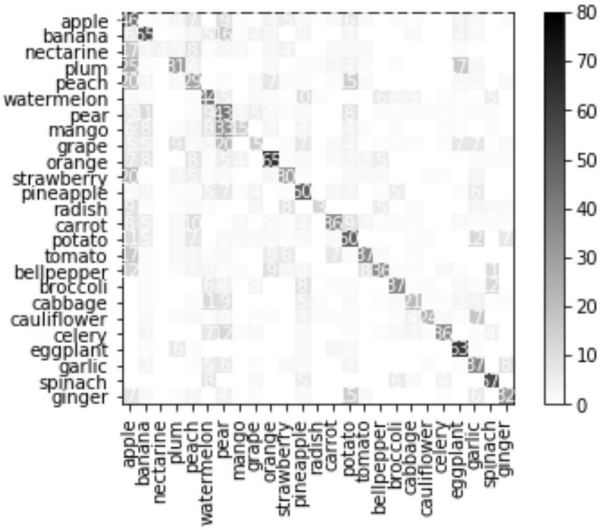



```
[[83 17  3]
 [ 7  7  4]
 [ 7    77]]
```


apple
banana
eggplant
83
17
3
7
97
4
7
9
77
80
70
60
50
40
30
20
10
0
apple
banana
eggplant

In [8]:
```
# hyper_search.py

import IPython
from numpy.random import uniform
import random
import time

import glob
import os
import pickle
import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA

import sys


from projection import Project2D, Projections

from confusion_mat import getConfusionMatrix
from confusion_mat import plotConfusionMatrix

from ridge_model import Ridge_Model
from qda_model import QDA_Model
from lda_model import LDA_Model
from svm_model import SVM_Model


CLASS_LABELS = ['apple','banana','nectarine','plum','peach','watermelon','pear',
'mango','grape','orange','strawberry','pineapple',
        'radish','carrot','potato','tomato','bellpepper','broccoli','cabbage','c
auliflower','celery','eggplant','garlic','spinach','ginger']


def eval_model(X,Y,k,model_key,proj):
        # PROJECT DATA
        cca_proj,white_cov = proj.cca_projection(X,Y,k=k)

        X_p = proj.project(cca_proj,white_cov,X)
        X_val_p = proj.project(cca_proj,white_cov,X_val)

        # TRAIN MODEL
        model = models[model_key]

        model.train_model(X_p,Y)
        acc,cm = model.test_model(X_val_p,Y_val)

        return acc,cm


class Model():
        """ Generic wrapper for specific model instance. """


        def __init__(self,model):
                """ Store specific pre-initialized model instance. """

                self.model = model


        def train_model(self,X,Y):
```


0.42
0.40
0.38
0.36
0.34
0.32
Accuracy
100
200
300
400
500
600
700
800
Dimension
qda
lda
ridge
ridge_lmda_10
ridge_lmda_01
svm
svm_C_10
svm_C_01


apple
banana
nectarine
plum
peach
watermelon
pear
mango
grape
orange
strawberry
pineapple
radish
carrot
potato
tomato
bellpepper
broccoli
cabbage
cauliflower
celery
eggplant
garlic
spinach
ginger
80
70
60
50
40
30
20
10
0