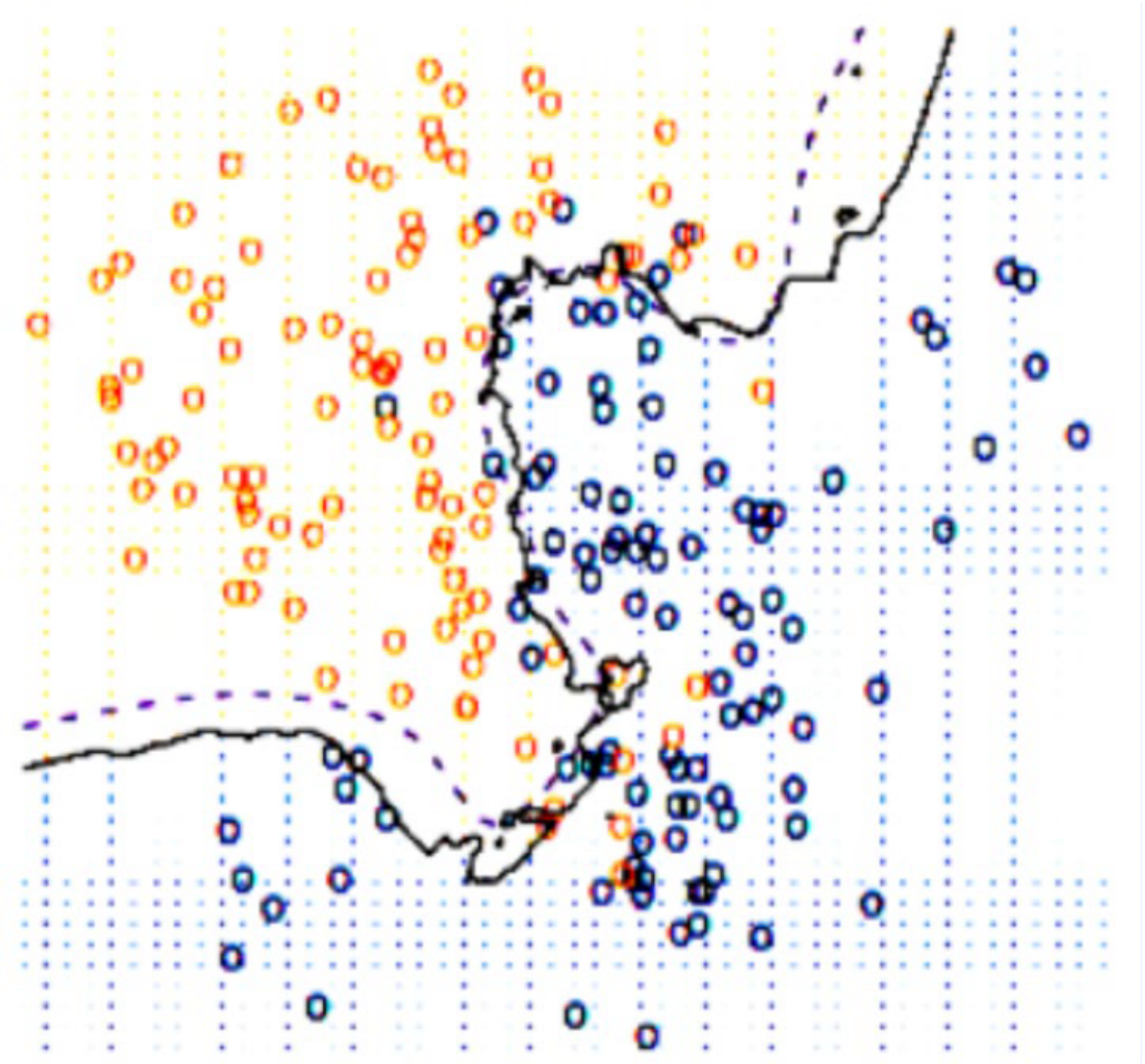
1. KNN

a. We discussed data normalization (also known as data standardization). Which of the following statements are true for doing data standardization on training and test sets.

- i. Calculate the mean and standard deviation of the whole dataset (training and test set). And apply the mean and standard deviation to both training set and test set.
- **False: Training data should not be standardized on data that will be used for testing. Doing so could cause data leakage and testing data could cause negative influence to the standardized data in the training set.**
- ii. Calculate the mean and standard deviation of the training dataset. And apply the mean and standard deviation of training set to both training set and test set.
- **True: Training data should be used to standardize both training and test data as it does not cause any leakage of data and represents a more realistic scenario.**
- iii. Calculate the mean and standard deviation of the training dataset. And do not need to do the data standardization for test set.
- **False: Not applying the training set standardization to the test set would cause the model not to work because this data is on a different scale.**
- iv. Calculate the mean and standard deviation of the training dataset and test set respectively. And apply the the mean and standard deviation to training and test set respectively.
- **False: Standardizing the test data using it's own data would cause performance issues as the training data should be representative of the training data.**

b. Which value of k in the k-nearest neighbors algorithm generates the solid decision boundary depicted here? There are only 2 classes. (Ignore the dashed line, which is the Bayes decision boundary.) k = 1 k = 10 k = 2 k = 100.



i. K = 1

ii. K = 2

iii. K = 10

iv. K = 100

ANSWER: K=1 - As the value of K decreases, the model will try to fit cloesly to the specific datapoints and result in a lot of noise/standard deviation. The image shows the decision boundry representative of a lower K value because of it's closeness to specific points of data. If, however, the decision boundry was a smoother line then we could assume a larger K value.

k-Nearest Neighbor (kNN)

The kNN classifier consists of two stages: training and testing. During training, the classifier takes the training data and simply remembers it. During testing, kNN classifies a test image by comparing it to all training images and selecting the majority label among the k most similar training examples. The value of k is computed by cross-validation. In this exercise you will implement these steps and gain proficiency in writing efficient, vectorized code. You will select the best value of k by cross-validation. You will be using a version of the CIFAR-10 object recognition dataset for this exercise.

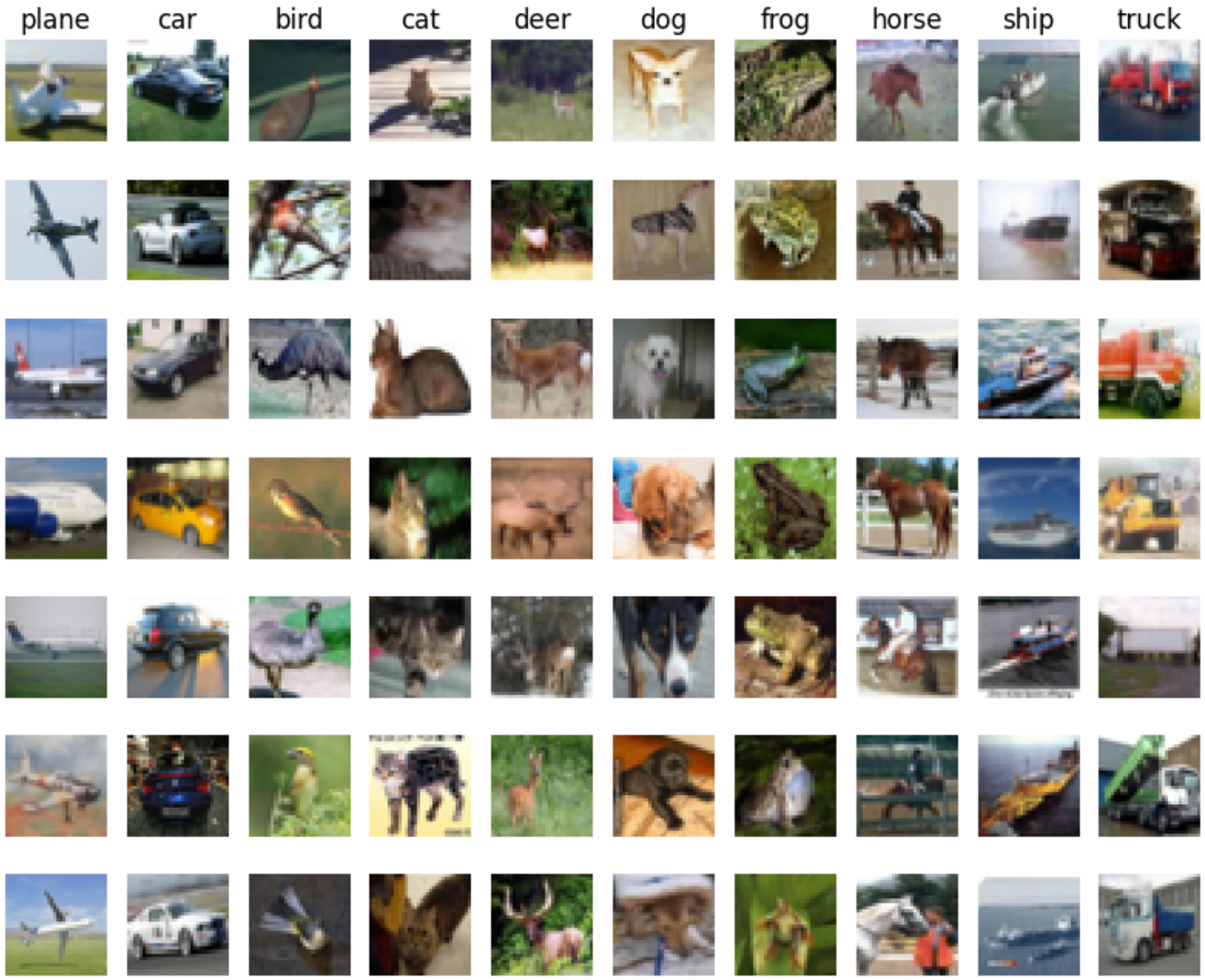
In [ ]:
```
import random
import numpy as np
from data_utils import load_CIFAR10
import matplotlib.pyplot as plt

# This is a bit of magic to make matplotlib figures appear inline in the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
# Load the raw CIFAR-10 data.
# Run the CIFAR-10 dataset load script in the folder datasets, before you run this cell

cifar10_dir = './datasets/cifar-10-batches-py'
X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

# As a sanity check, we print out the size of the training and test data.
print('Training data shape: %s' %(X_train.shape,))
print('Training labels shape: %s' %(y_train.shape,))
print('Test data shape: %s' %(X_test.shape,))
print('Test labels shape: %s' %(y_test.shape))
```

```
Training data shape: (50000, 32, 32, 3)
Training labels shape: (50000,)
Test data shape: (10000, 32, 32, 3)
Test labels shape: 10000
```

In [ ]:
```
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
```

```
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
num_classes = len(classes)
samples_per_class = 7
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```



In [ ]:
```
# Subsample the data for more efficient code execution in this exercise
num_training = 5000
mask = range(num_training)
X_train = X_train[mask]
y_train = y_train[mask]

num_test = 500
mask = range(num_test)
X_test = X_test[mask]
y_test = y_test[mask]
```

```
# Reshape the image data into rows
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
print(X_train.shape, X_test.shape)
```

```
(5000, 3072) (500, 3072)
```

Question1 : Creating a knn classifier

Remember that training a kNN classifier is a no-op. The classifier simply remembers the data and does no further processing, define the KNN classifier by using sklearn KNeighborsClassifier

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# TODO :: define the knn classifier, expect 2 lines of the code
knn = KNeighborsClassifier(
    n_neighbors=5,
    p=2,
    metric='minkowski'
)

knn.fit(X_train, y_train)
```

Out[ ]:
```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

In [ ]:
```
# TODO :: predict on test set, expect 1 line of the code
y_test_pred = knn.predict(X_test)
```

In [ ]:
```
# Compute and print the fraction of correctly predicted examples
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accuracy))
```

```
Got 139 / 500 correct => accuracy: 0.278000
```

Question 2 : Choosing k by cross-validation

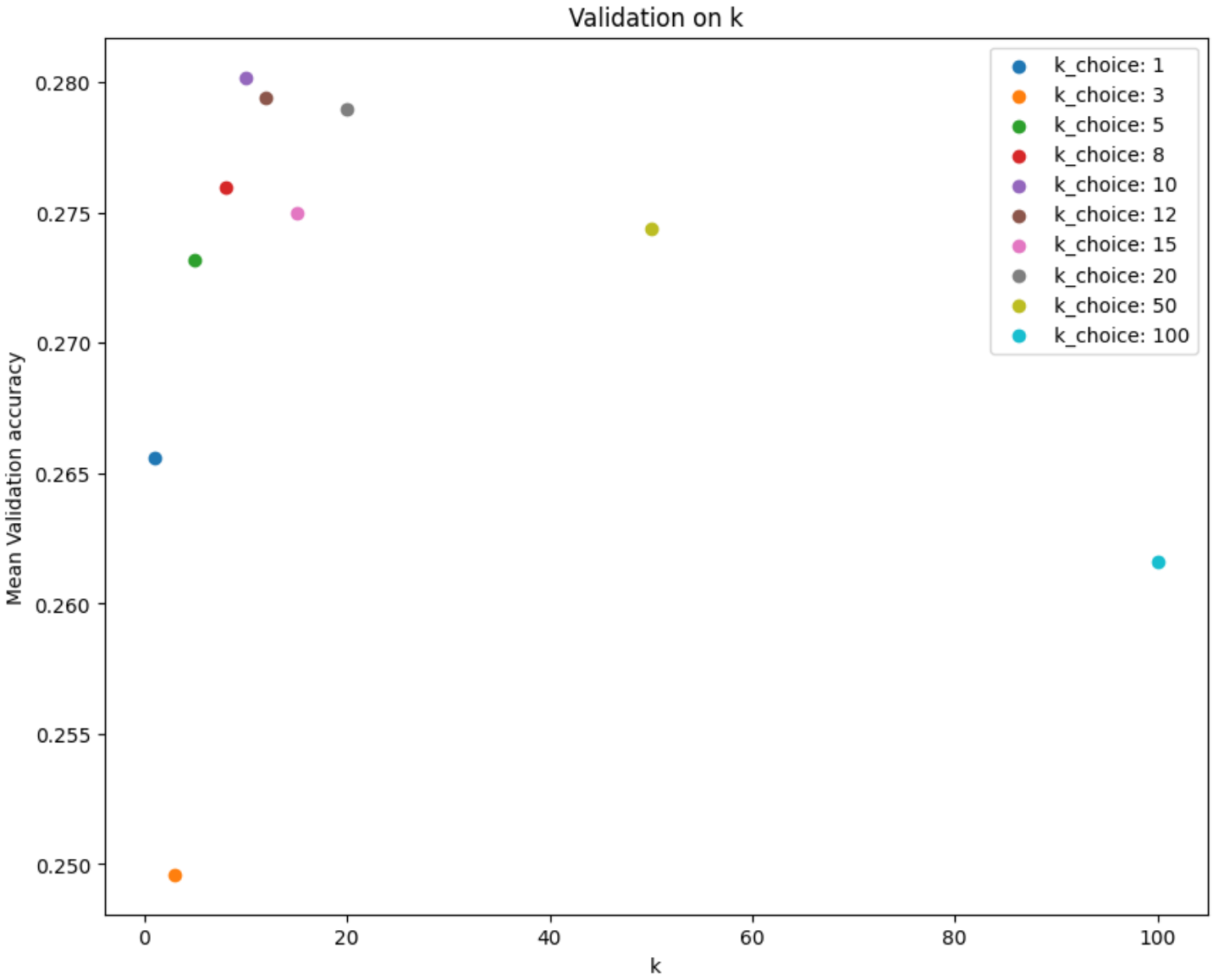
We have implemented the k-Nearest Neighbor classifier but we set the value k = 5 arbitrarily. We will now determine the best value of this hyperparameter with cross-validation. What is the best value for k ?

In [ ]:
```
from sklearn.model_selection import train_test_split
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_folds = []
```

```
y_train_folds = []
################################################################################
# TODO:                                                                        #
# Split up the training data into training and validation set. After splitting,#
# X_train_folds and y_train_folds should each be lists of length num_folds,    #
# where y_train_folds[i] is the label vector for the points in X_train_folds[i]#
# Hint: Look up the numpy array_split function.                                #
################################################################################
X_train_folds = np.array_split(X_train, num_folds)
y_train_folds = np.array_split(y_train, num_folds)

################################################################################
#                                END OF YOUR CODE                              #
################################################################################

# A dictionary holding the accuracies for different values of k that we find
# when running validation. After running validation, k_to_accuracies[k] shold be
# the accuracy for using k value for n_neighbors
k_to_accuracies = {}

################################################################################
# TODO:                                                                        #
# Perform validation to find the best value of k. For each                     #
# possible value of k, run the k-nearest-neighbor algorithm num_folds times,   #
# where in each case you use all but one of the folds as training data and the #
# last fold as a validation set. Store the accuracies for all fold and all     #
# values of k in the k_to_accuracies dictionary.
# expect 8 lines of the code
################################################################################
for k in k_choices:
    accuracies = []
    for fold in range(num_folds):

        # set current fold iteration as the test set and all other data as training
        val_fold_index = fold
        train_fold_indices = [i for i in range(num_folds) if i != val_fold_index]

        # get training data
        X_train_fold = np.concatenate([X_train_folds[i] for i in train_fold_indices])
        y_train_fold = np.concatenate([y_train_folds[i] for i in train_fold_indices])

        # get testing data
        X_val_fold = X_train_folds[val_fold_index]
        y_val_fold = y_train_folds[val_fold_index]

        # train
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train_fold, y_train_fold)

        # store accuracy scores as list
```

```
        accuracy = knn.score(X_val_fold, y_val_fold)
        accuracies.append(accuracy)

    # set key k_choices to the respective accuracies
    k_to_accuracies[k] = accuracies
```

In [ ]:
```
# plot the raw observations
for k in k_choices:
  accuracy = k_to_accuracies[k]
  # k_list = [k, k, k, k, k]
  plt.scatter(k, np.mean(accuracy), label=f'k_choice: {k}')

# plot the trend line with error bars that correspond to standard deviation
plt.title('Validation on k')
plt.xlabel('k')
plt.ylabel('Mean Validation accuracy')
plt.legend()
plt.show()
```



In [ ]:
```
max(k_to_accuracies, key=lambda k: np.mean(k_to_accuracies[k]))
```

Out[ ]: 10

In [ ]:
```
# Based on the cross-validation results above, choose the best value for k,
# retrain the classifier using all the training data, and test it on the test
# data. You should be able to get above 28% accuracy on the test data.
```

```
X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
# TODO :: use the best k to train and predict on the test set, expect 4 lines of code
best_k = max(k_to_accuracies, key=lambda k: np.mean(k_to_accuracies[k]))
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# TODO :: Compute and display the accuracy, expect 3 lines of code
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f'accuracy score for {best_k}: {accuracy}')
```

```
accuracy score for 10: 0.3386
```