1. For k = 1, 5, 9 and 15, build k-nearest neighbor classifiers from the training data. For each of these values of k, write down a table of training errors (error on the training data) and the validation errors (error on the validation data). Which of these classifiers performs the best on validation data? What is the test error of this classifier?

	k = 1	k = 5	k = 9	k = 15
Training Error	0.0	0.0565	0.0685	0.093
Validation Error	0.082	0.101	0.102	0.104

The running time for calculating each training error is around 7 minutes and the running time for calculating each validation error is around 3 minutes.

According to this table, the classifier that performs the best on validation data is the 1-nearest neighbor classifier and the test error of this classifier is 0.094.

2. Project the training, validation and test data onto the column space of this matrix, and repeat part (1) of the problem. For k = 1, 5, 9, 15, write down a table of the training and validation errors, as well as the test error of the classifier which performs best on the validation data. How is the classification accuracy affected by projection? How does the running time of your program change when you run it on projected data?

	k = 1	k = 5	k = 9	k = 15
Training Error	0.0	0.192	0.23	0.259
Validation Error	0.32	0.299	0.296	0.295

The running time for calculating each training error is around 40 seconds and the running time for calculating each validation error is around 20 seconds.

According to this table, the classifier that performs the best on validation data is the 15-nearest neighbor classifier and the test error of this classifier is 0.303.

Comparing the data from two tables, we can see that matrix projection significantly reduces the running time of the program at the cost of accuracy.

Also, for the second part, although 15-nearest neighbor classifier performs best on the validation data, 9-nearest neighbor classifiers actually gives me better test errors, which further proves the fact that matrix projection reduces the accuracy.

```python
import argparse
import numpy
import random

class Solver():
    def __init__(self):
        # read the arguments
        parser = argparse.ArgumentParser()
        parser.add_argument("training_set")
        parser.add_argument("validation_set")
        parser.add_argument("test_set")
        parser.add_argument("projection_matrix")
        args = parser.parse_args()
        self.training_file = args.training_set
        self.validation_file = args.validation_set
        self.test_file = args.test_set
        self.projection_file = args.projection_matrix
        # list that contains the training data
        self.training_data = []
        # list that contains the validation data
        self.validation_data = []
        # list that contains the test data
        self.test_data = []
        # list that contains the rows of the projection matrix
        self.projection_data = []
    def load(self):
        # open the training data file
        with open(self.training_file) as training_file:
            for line in training_file:
                data = line.split()
                for i in range(0, 785):
                    data[i] = int(data[i])
                # add the data to the training data list
                self.training_data.append(data)
        print("There are ", len(self.training_data), " training data")
        # open the validation data file
        with open(self.validation_file) as validation_file:
            for line in validation_file:
                data = line.split()
                for i in range(0, 785):
                    data[i] = int(data[i])
```

```python
                # add the data to the validation data list
                self.validation_data.append(data)
        print("There are ", len(self.validation_data), " validation data")
        # open the test data file
        with open(self.test_file) as test_file:
            for line in test_file:
                data = line.split()
                for i in range(0, 785):
                    data[i] = int(data[i])
                # add the data to the test data list
                self.test_data.append(data)
        print("There are ", len(self.test_data), " test data")
        # open the projection matrix file
        with open(self.projection_file) as projection_file:
            for line in projection_file:
                data = line.split()
                for i in range(0, 20):
                    data[i] = float(data[i])
                # add the data to the matrix data list
                self.projection_data.append(data)
        print("There are ", len(self.projection_data), " rows in the projection matrix")
    def project(self):
        m0 = numpy.array(self.projection_data)
        # project the training data
        training = []
        for i in self.training_data:
            training.append(i[:-1])
        m1 = numpy.array(training)
        m2 = numpy.matmul(m1, m0)
        training_proj = []
        for i in range(0, 2000):
            r = m2[i].tolist()
            r.append(self.training_data[i][-1])
            training_proj.append(r)
        self.training_data = training_proj
        # project the validation data
        validation = []
        for i in self.validation_data:
            validation.append(i[:-1])
        m1 = numpy.array(validation)
```

```python
        m2 = numpy.matmul(m1, m0)
        validation_proj = []
        for i in range(0, 1000):
            r = m2[i].tolist()
            r.append(self.validation_data[i][-1])
            validation_proj.append(r)
        self.validation_data = validation_proj
        # project the test data
        test = []
        for i in self.test_data:
            test.append(i[:-1])
        m1 = numpy.array(test)
        m2 = numpy.matmul(m1, m0)
        test_proj = []
        for i in range(0, 1000):
            r = m2[i].tolist()
            r.append(self.test_data[i][-1])
            test_proj.append(r)
        self.test_data = test_proj
    def getDistance(self, data1, data2):
        dist = 0.0
        # calculate the distance between two data
        d1 = numpy.array(data1[:-1])
        d2 = numpy.array(data2[:-1])
        dist = numpy.linalg.norm(d1 - d2)
        return dist
    def getKNeighbors(self, k, test_example):
        distances = []
        neighbors = []
        # calculate the distances from the test example to all the training data
        for i in self.training_data:
            distance = self.getDistance(test_example, i)
            distances.append((distance, i))
        # sort all the distances
        distances.sort()
        # get the k closest neighbors
        for i in range(0, k):
            neighbors.append(distances[i][1])
        return neighbors
```

```python
    def getPrediction(self, neighbors):
        # dictionary that maps the label to the number of times this label appears
        labels = {}
        # initialize the dictionary
        for i in neighbors:
            labels[i[-1]] = 0
        # update the dictionary
        for i in neighbors:
            labels[i[-1]] = labels[i[-1]] + 1
        max_count = 0
        # predict the majority
        for i in labels:
            if labels[i] > max_count:
                max_count = labels[i]
        # break tie randomly
        predictions = []
        for i in labels:
            if labels[i] == max_count:
                predictions.append(i)
        index = random.randint(0, len(predictions) - 1)
        return predictions[index]
    def getError(self, predictions, test_examples):
        if len(predictions) != len(test_examples):
            print("Size does not match")
            return
        # get the errors
        errors = 0.0
        for i in range(0, len(predictions)):
            if predictions[i] != test_examples[i][-1]:
                errors = errors + 1.0
        errors = errors / (float(len(test_examples)))
        return errors
```

```python
if __name__ == '__main__':
    # create the solver
    solver = Solver()
    # load the data
    print("loading data")
    solver.load()
    print("projecting data")
    # project the data onto the projection matrix
    solver.project()
    # list that contains all the predictions
    predictions = []
    # calculate the training error
    for i in solver.training_data:
        # get the k nearest neighbors
        #kNeighbors = solver.getKNeighbors(1, i)
        #kNeighbors = solver.getKNeighbors(5, i)
        #kNeighbors = solver.getKNeighbors(9, i)
        kNeighbors = solver.getKNeighbors(15, i)
        # get the prediction
        prediction = solver.getPrediction(kNeighbors)
        # add the prediction to the prediciton list
        predictions.append(prediction)
    # get the number of errors
    training_error = solver.getError(predictions, solver.training_data)
    # print out the errors
    print("Training error ", training_error)
    # calculate the validation error
    for i in solver.validation_data:
        # get the k nearest neighbors
        #kNeighbors = solver.getKNeighbors(1, i)
        #kNeighbors = solver.getKNeighbors(5, i)
        #kNeighbors = solver.getKNeighbors(9, i)
        kNeighbors = solver.getKNeighbors(15, i)
        # get the prediction
        prediction = solver.getPrediction(kNeighbors)
        # add the prediction to the prediciton list
        predictions.append(prediction)
    # get the number of errors
    validation_error = solver.getError(predictions, solver.validation_data)
```

```python
    # print out the errors
    print("Validation error ", validation_error)
    # calculate the test error
    for i in solver.test_data:
        # get the k nearest neighbors
        kNeighbors = solver.getKNeighbors(15, i)
        # get the prediction
        prediction = solver.getPrediction(kNeighbors)
        # add the prediction to the prediciton list
        predictions.append(prediction)
    # get the number of errors
    test_error = solver.getError(predictions, solver.test_data)
    # print out the errors
    print("Validation error ", test_error)
```