Assignment 1.3

Wednesday, January 31, 2024 5:44 PM

3. I. Full code in q-3.py * data set is randomized with set seed before execution *

a)

```python
def knn(num_neighbour, metric, training_set, validation_set, test_set, sklearn = False):
    validation_set, validation_class = np.hsplit(validation_set, np.array([3]))
    validation_class = validation_class.reshape(-1)

    training_set, training_class = np.hsplit(training_set, np.array([3]))
    training_class = training_class.reshape(-1)

    test_set, test_class = np.hsplit(test_set, np.array([3]))
    test_class = test_class.reshape(-1)

    validation_error = 0

    # use sklearn if metric is None
    if sklearn:
        neigh = KNeighborsClassifier(n_neighbors=num_neighbour, metric=metric)
        neigh.fit(training_set, training_class)

    # calculate error in validation set
    for i, x in enumerate(validation_set):
        if sklearn:
            if neigh.predict(x.reshape(1, -1))[0] != validation_class[i]:
                class_sum = -1
            else:
                class_sum = 1
        else:
            k_neighbours = []

            # add distances of between validation point and all training points into min heap
            for j, y in enumerate(training_set):
                distance = metric(x, y)
                heapq.heappush(k_neighbours, (distance, training_class[j]))

            # use min heap to find k closest training class points from validation point
            class_sum = 0
            for k in range(num_neighbour):
                distance, classification = heapq.heappop(k_neighbours)

                # class_sum finds if majority of neighbours will belong to the correct validation class
                if classification == validation_class[i]:
                    class_sum += 1
                else:
                    class_sum -= 1

        # if the major of neighbours not in same class as the validation point, that means there will be
        # classification error
        if class_sum < 0:
            validation_error += 1

    # same operations as validation set
    test_error = 0
    for i, x in enumerate(test_set):
        if sklearn:
            if neigh.predict(x.reshape(1, -1))[0] != test_class[i]:
                class_sum = -1
            else:
                class_sum = 1
        else:
            k_neighbours = []
            for j, y in enumerate(training_set):
                distance = metric(x, y)
                heapq.heappush(k_neighbours, (distance, training_class[j]))

            class_sum = 0
            for k in range(num_neighbour):
                distance, classification = heapq.heappop(k_neighbours)
                if classification == test_class[i]:
                    class_sum += 1
                else:
                    class_sum -= 1

        if class_sum < 0:
            test_error += 1

    return validation_error, test_error
```

b)

```python
def euclid_dist(x, y):
    res = 0
    for i in range(x.shape[0]):
        res += pow((x[i] - y[i]), 2)

    return math.sqrt(res)
```

c)

```python
def cosine_sim(x, y):
    # subtract 1 for distance
    return 1 - (dot(x, y) / (norm(x) * norm(y)))
```

II a)

Error rate = $\frac{\text{\# of error}}{\text{\# in validation set}}$

	Validation Error #	Validation Error rate
Euclidian	28	36.84%
Cosine	33	43.42%
SkL Euclidian	28	36.84%
SkL Cosine	33	43.42%

b)

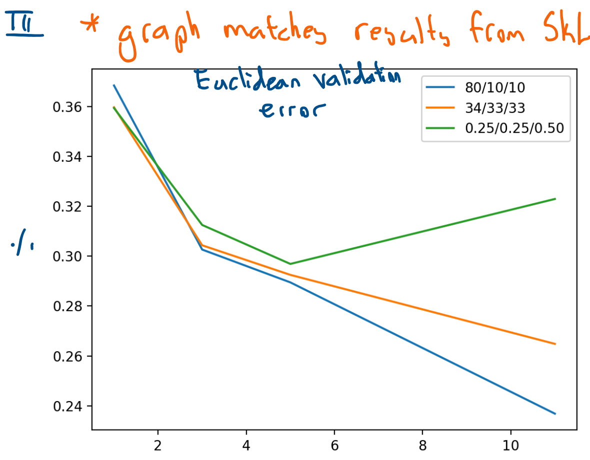
	Validation Error #	Validation Error rate
Euclidian	91	35.97%
Cosine	109	43.08%
SkL Euclidian	91	35.97%
SkL Cosine	109	43.08%

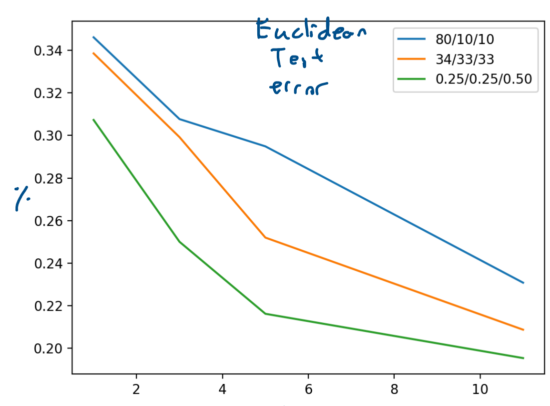
c)

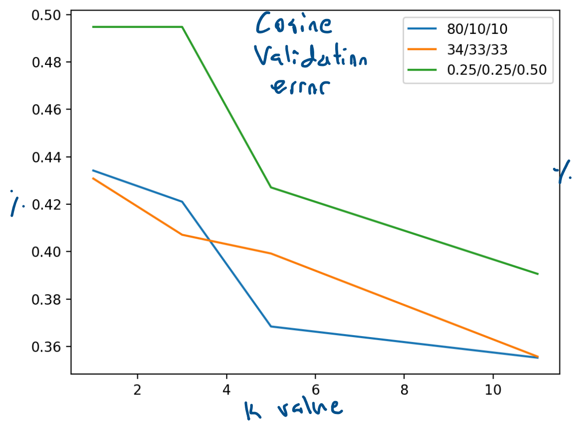
	Validation Error #	Validation Error rate
Euclidian	69	35.94%
Cosine	95	49.48%
SkL Euclidian	69	35.94%
SkL Cosine	95	49.48%

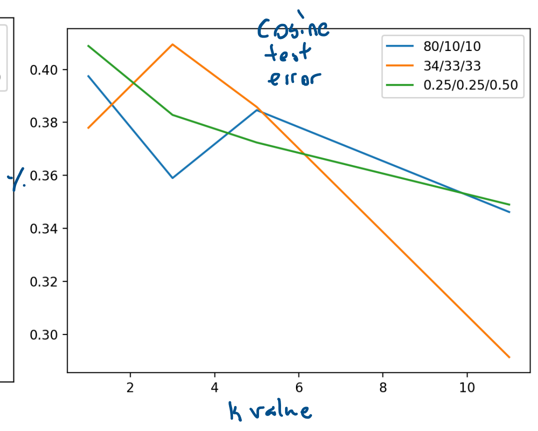
d) The combination of the split 25/25/50 and Euclidian distance as the metric works the best since it has the lowest error rate out of all the combination. Although it might be thought that a split of 80/10/10 would do the best, the results might be due to the number of neighbour only being 1. Any single data point heavily infuences the prediction, so the number of data points in the training set matters less.

III * graph matches results from SkLearn implementation for all k values *









The combination of k = 11, split = 80/10/10 and metric = euclidean distance results in the best model. It's validation is the lowest amongs models at ≈ 24%, and has fairly low test error rate. This is expected as with the number of neighbours increasing, singular datapoints have less of an impact, which gives the model with the higher amount of training data the advantage. This fact is reflected on the graphs above, as all errors decreased with increasing number of neighbours.