Software Project Final Report

Ryan Garry

Final Report

The final report involves testing the kthperclass classifier that you implemented and then comparing performance with the kNN classifier.

First, the kthperclass classifier is tested for **k=1** and **k=2**, using all 150 datapoints as training data. The test vectors to be used are given in Table D.

Table D. Iris dataset test vectors and results for kthperclass classification.

Test case	Petal length	Petal width	True Class	Pred Class k=1	Pred Class k=2
1	2.0	0.8	Setosa		
2	4.0	0.8	Versicolor		
3	6.5	2.5	Virginica		
4	4.5	1.7	Virginica		
5	4.8	1.8	Virginica		
6	5.0	1.8	Versicolor		
7	5.0	1.5	Virginica		

Provide the following for the test results.

1. Complete Table D with the kthperclass classifier results for k=1 and k=2.
2. For both k=1 and k=2, provide confusion matrices.
3. For both k=1 and k=2, fill in Table E, providing the overall probability of classification error as well as conditional classification error probabilities, conditioned on the true class.

Table E. Overall & conditional classification error probabilities for kthperclass classification trained with the irisf34 dataset.

kNN k value	Overall Pe	Pe | setosa	Pe | versicolor	Pe | virginica
1				
2				

Second, compare the classification error performance for the kNN and kthperclass classifiers. You will need to write two functions.

1. **Splitdata**: it will take the original data set and split it into a training set and a test set. For the Iris dataset, this will involve taking the 50 vectors of each class and using the first Nt for training and the remaining 50-Nt for test. Warning: because of the order of the vectors and labels in the dataset, one cannot simply keep the first 90 vectors as this would give 50 setosa vectors and 40 versicolor vectors.

2. **Kthperclass**: it will take a training set (features and labels) and a test set (feature vectors) and produce a set of detected (predicted) labels.

You will also need to write an overall script or set of scripts, using these and other functions, to evaluate the probability of classification error as a function of k. Use this script with Nt = 30 (30 training vectors per class, 20 test vectors per class) to compare

1. kNN classification, k=1 through 17, and
2. kthperclass classification, k=1 through 17

Results for Nt = 35 are shown in Fig. C**.**

Question: Explain why the probability of classification error appears to take on only certain values.

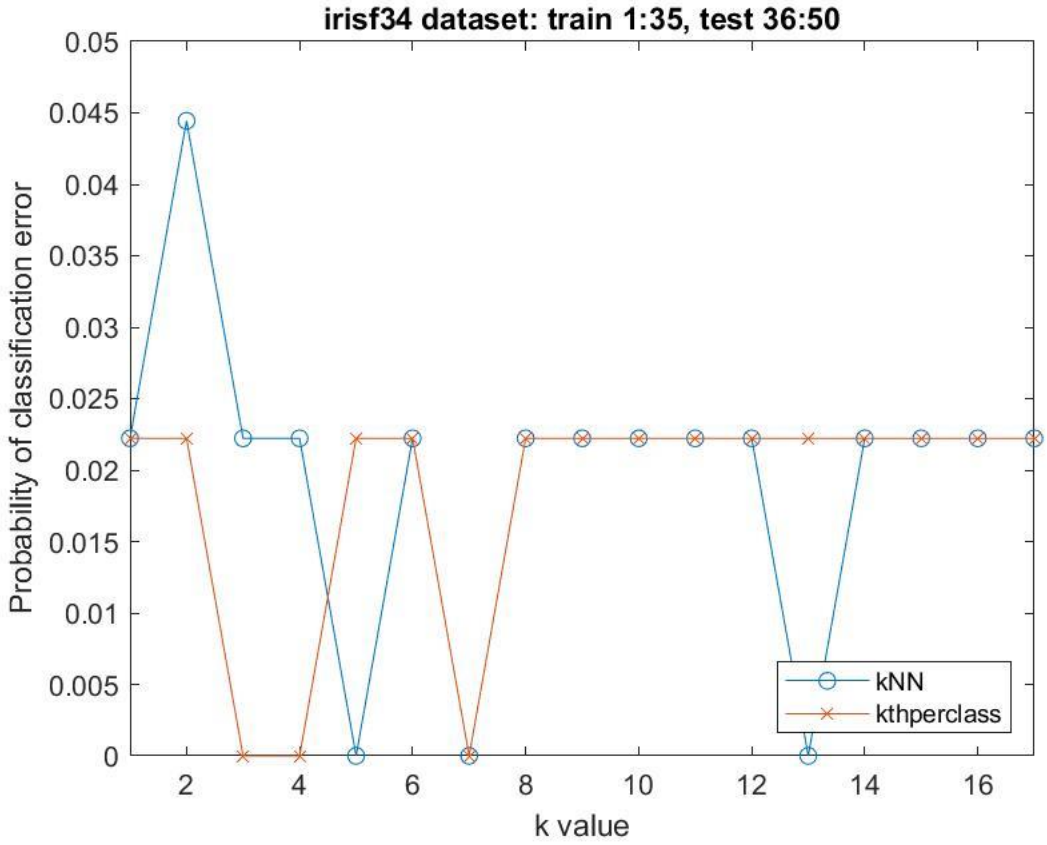

Figure D. Performance results for 35 training vectors per class, 15 test vectors per class.

Test Results for kthperclass classifier

1. Results for the kNN classifier for k=1 and k=2 are given in Table 1.
2. Confusion matrices for k=1 and k=2 are given in Figs. 1 and 2, respectively.
3. Overall and conditional probability of classification error values are given in Table 2.

Table 1. Iris dataset test vectors and results for kthperclass classification for final report.

Test case	Petal length	Petal width	True Class	Pred Class k=1	Pred Class k=2
1	2.0	0.8	Setosa	Setosa	Setosa
2	4.0	0.8	Versicolor	Versicolor	Versicolor
3	6.5	2.5	Virginica	Virginica	Virginica
4	4.5	1.7	Virginica	Virginica	Versicolor
5	4.8	1.8	Virginica	Versicolor	Virginica
6	5.0	1.8	Versicolor	Versicolor	Virginica
7	5.0	1.5	Virginica	Virginica	Versicolor

Table 2. Overall & conditional classification error probabilities for kthperclass classification trained with the irisf34 dataset.

kthperclass k value	Overall Pe	Pe | setosa	Pe | versicolor	Pe | virginica
1	1/7	0	0	1/4
2	3/7	0	1/2	2/4

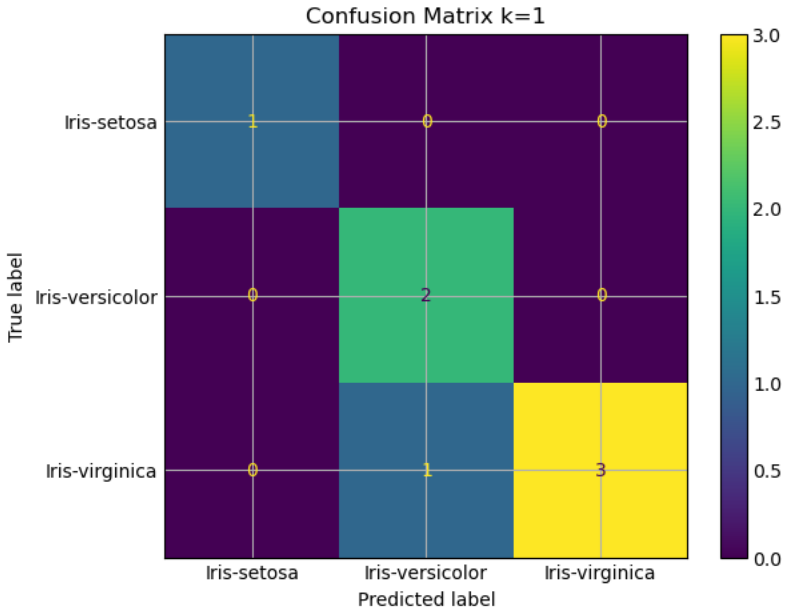

Figure 1. Confusion matrix for kthperclass classifier and k=1.

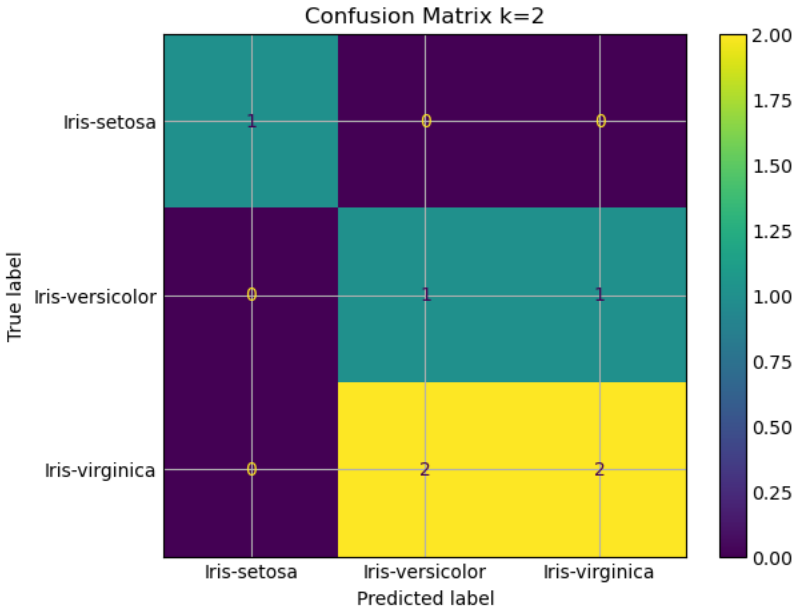

Figure 2. Confusion matrix for kthperclass classifier and k=2.

Results comparing kNN and kthperclass classifiers

Classification error as a function of k is shown for the two classifiers in Fig. 3.

Question: Explain why the probability of classification error appears to take on only certain values.

Answer: Since probability of error is estimated by dividing the number of misclassified test vectors by the total number of test vectors, they will always be discrete values aka fractions.

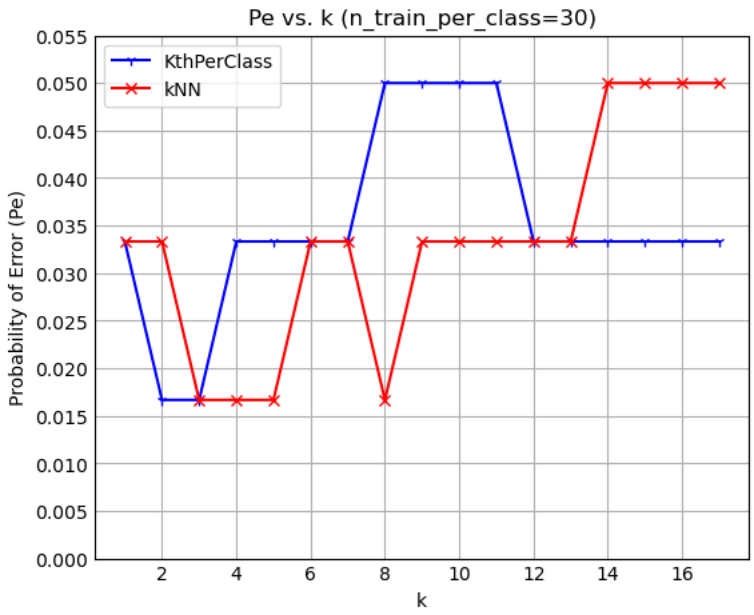

Figure 3. Pe vs. k for the kNN and kthperclass classifiers using the iris dataset (first 30 vectors per class for training and remaining 20 vectors per class for testing).

Appendix A: code listing of function to split the data.

```python
def split_data(X,y,n_train: int) -> (pd.DataFrame, pd.DataFrame, pd.DataFrame, pd.DataFrame, np.array):
    """Splits data into train and test with n_train vectors
    per class and the remaining test vectors
    Returns x_train, y_train, x_test, y_test"""
    unique_y_values = y.squeeze().unique()
    x_columns = list(X.columns)
    y_columns = list(y.columns)
    # combine X and y into one dataframe
    df = pd.concat([X, y], ignore_index=False, axis=1)
    x_train = pd.DataFrame(columns = x_columns)
    y_train = pd.DataFrame(columns = y_columns)
    x_test = pd.DataFrame(columns = x_columns)
    y_test = []
    for i in unique_y_values:
            #filter by class and find the length of each class
            filtered_df = df[df[y_columns[0]] == i]
            class_len = len(filtered_df)
            n_test = class_len - n_train
            # head finds the first n_train vectors and tail finds the last n_test vectors
            x_train = pd.concat([x_train, filtered_df[x_columns].head(n_train)])
            y_train = pd.concat([y_train, filtered_df[y_columns].head(n_train)])
            x_test = pd.concat([x_test, filtered_df[x_columns].tail(n_test)])
            y_test.append(filtered_df[y_columns].tail(n_test).values.tolist())
    x_train = x_train.astype(int)
    x_test = x_test.astype(int)
    y_train = y_train.astype(int)
    y_test = np.array(y_test).flatten()
    return x_train, y_train, x_test, y_test

```

Appendix B: code listing of function to implement the kthperclass classifier

```python
class KthPerClass():
    """Classifier based on kNN that finds the kth nearest training vector per class
    based on the manhattan distance and in the event of a tie chooess the class that
    appears earlier in the training data"""

    def __init__(self, k: int = 1) -> None:
        self.k = k

    def fit(self, X: pd.DataFrame, y: pd.DataFrame) -> None:
        """Stores X=features and y=labels as training data"""
        assert type(X) is pd.core.frame.DataFrame, 'X is not a DataFrame'
        assert type(y) is pd.core.frame.DataFrame, 'y is not a DataFrame'
        self._x_train = X
        self._y_train = y
        self._x_train_columns = list(self._x_train.columns)
        self._y_train_columns = list(self._y_train.columns)
        self.unique_y_values = self._y_train.squeeze().unique()

    def predict(self, X: pd.DataFrame) -> np.ndarray:
        """Returns the predicted labels y_pred for the test vectors X"""
        # create a new col called distance using manhattan distance
        assert type(X) is pd.core.frame.DataFrame, 'X is not a DataFrame'
        x_test = X.values # ndarray
        y_list = []
        for x in x_test:
            y_distance = self._y_train.copy()
            # calculate manhattan distance (L1 norm) between
            # each test vector and the training data
            y_distance['distance'] = self._x_train.apply(
                lambda row: linalg.norm(x-row, ord=1),
                    axis=1)
            # for each class: filter by class and sort the distance in ascending order
            kth_distance_per_class = []
            for i in self.unique_y_values:
                # filter by class then sort by distance metric least to greatest
                filtered_y_distance = y_distance[y_distance[self._y_train_columns[0]] == i]
                sorted_y_distance = filtered_y_distance.sort_values('distance',ignore_index=True)
                # grab the kth nearest distance for each class
                kth_distance = sorted_y_distance['distance'].iloc[self.k-1]
                kth_distance_per_class.append(kth_distance)
            # choose the distance that is the smallest with min()
            # then grab the index which corresponds to the kth nearest class
            y = kth_distance_per_class.index(min(kth_distance_per_class))
            y_list.append(y)
        y_pred = np.array(y_list)
        return y_pred
```