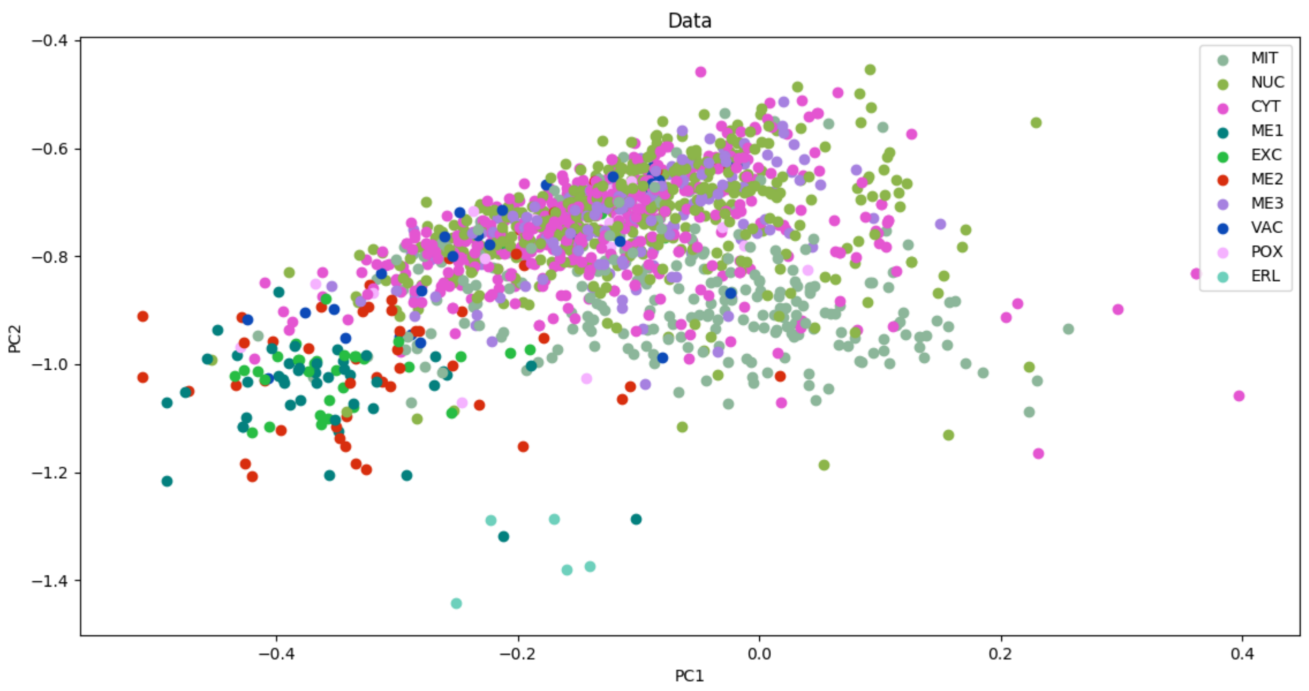
Classification

Classification is a problem that can be tackled with supervised learning. In supervised learning, our objective is to build a model, using the training data, with which we will then be able to make accurate predictions on new data that has the same characteristics as the training set that we used. The purpose of this project is to develop such algorithms from scratch. For that purpose a dataset with known labels was used (http://mlearn.ics.uci.edu/databases/yeast/yeast.data) where 1484 yeast proteins were classified into 10 classes with an accuracy of 55%.

In order to visualize the data PCA was performed. The results were plotted in accordance to the class of each data point and are demonstrated below:

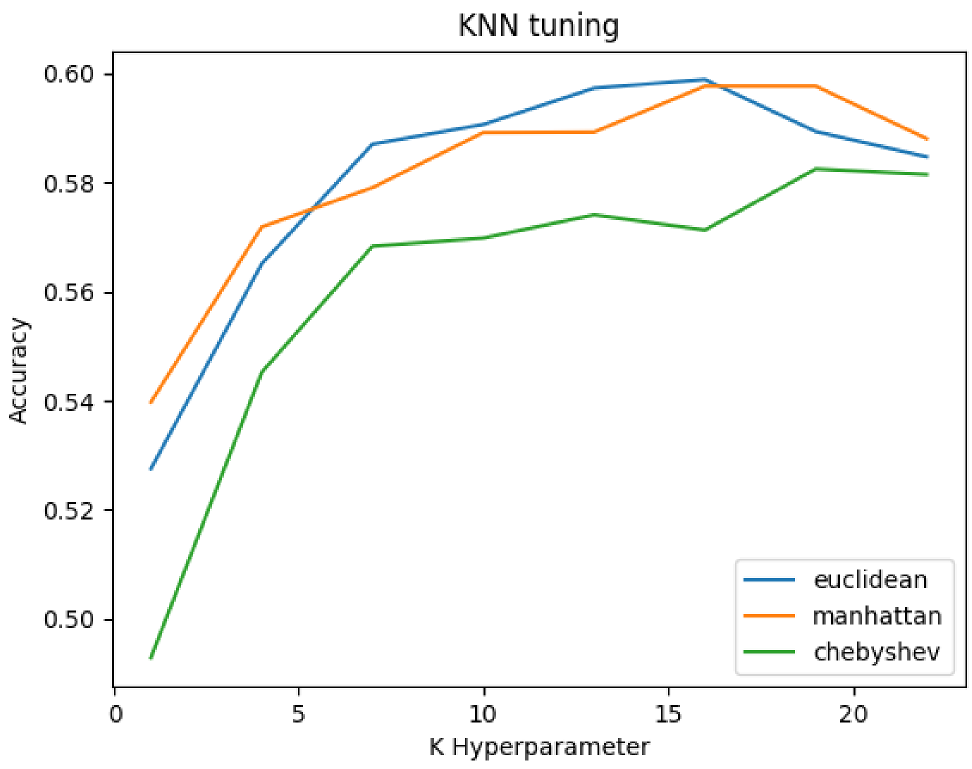


It is clear that the classes are highly intersected and it would be a real challenge for a classifier to identify the correct class of each data point. With 10 classes and that level of intersection, even a 55% accuracy is impressive.

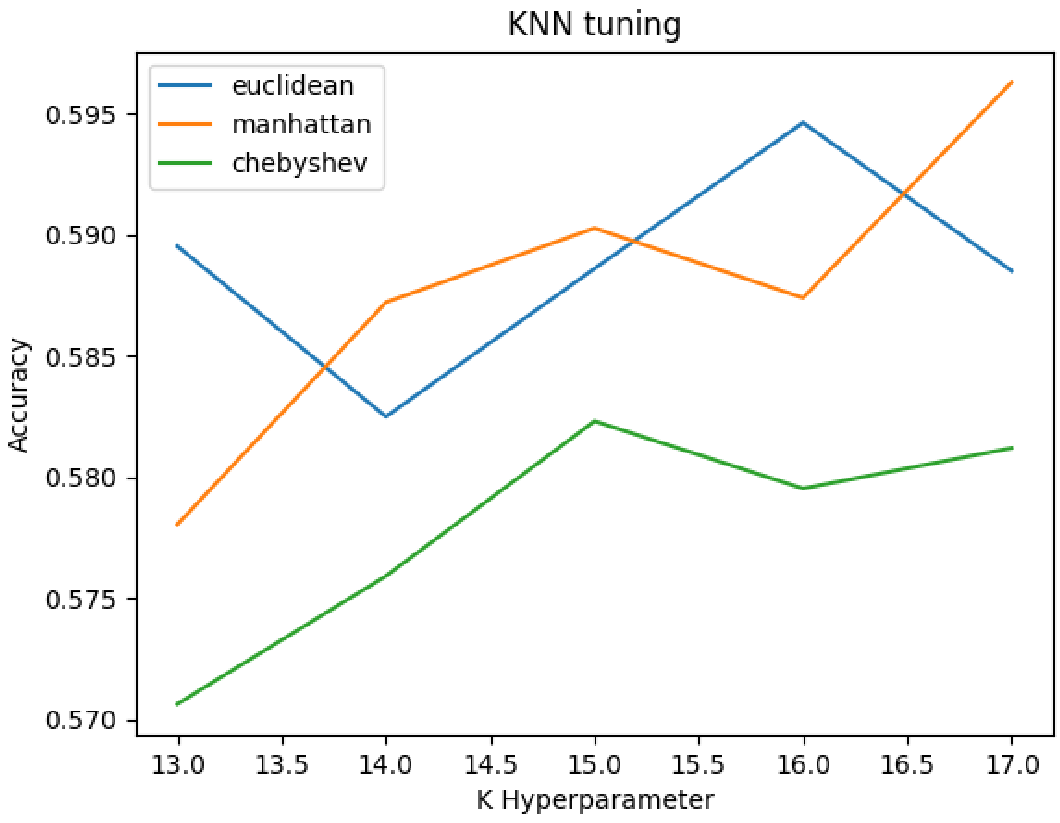
The first algorithm that was developed is K-nearest neighbors (KNN). KNN is an instance-based learning algorithm which means that it doesn't generalize from the training dataset but instead memorizes it in order to predict the classes of new data. When new data is introduced, each data point will be matched to the label that corresponds to the most frequent label among the K nearest training data points. This basically means that the closest K data points to the new data point will define its class, by majority rule. Choosing a huge number of K reduces performance as it increases underfitting (imagine that choosing K equal to the number of the training data points will match every new data point to the most frequent class, which would produce a bad accuracy even on the training set itself) while choosing a very low number causes overfitting (it should be obvious as the decision boundaries will focus more on single points). The right choice of k is crucial to find a good balance between over- and underfitting. The big downsides of this algorithm is the computation time required which increases linearly as the training sample size increases as well as its lacking performance on datasets with a high amount of features.

The second algorithm that was developed is a Naive Bayes Classifier. Much more efficient in speed, the Naive Bayes generalizes by estimating the probability of belonging to a class given the values of the features. In order to estimate that probability, the probability density function of the Gaussian distribution is used. Within each class the data is summarized by the mean and standard deviation for each feature (in total, #features * #classes = #distributions). The probability density function will give us an estimate of how likely it is for a new data point to belong to the Gaussian distribution with these parameters (and therefore how likely it is to belong in that class). This is an estimate of the conditional probability to witness these values of the feature given that the data point belongs in that class. The product of these estimates is multiplied with the marginal probability of the class and finally a comparison of the final results will determine which class matches the data point. The big downside of Naive Bayes is the assumption of independence between the features as in Biology, more often than not, the features are not independent but rather influence each other.

To find which algorithm configuration performs best in our data 10-fold validation was used. This type of validation includes splitting in 10 parts (after the order has been randomized) and for each iteration choose one part to be the "validation" set and the other 9 parts to be the training set. In order to validate the accuracy of different KNN configurations the following approach was implemented: KNN was run repeatedly with k's from 1 to 25 by a step of 3 and with 3 different distance metrics (euclidean, manhattan, chebyshev) and the corresponding accuracies were kept and plotted. The results are presented in the figure below:



Now that we have some insight on what range of K the algorithm performs best, we repeat the process with a step of one in the range of [13,17]:



The manhattan distance metric seems to perform best with a K set to 17.

Naive Bayes has no hyperparameters, so one 10fold run was performed and the output accuracy was 0.5387037037037037. It is obvious that Naive Bayes is outperformed by most configurations of the KNN. This makes sense considering the assumption of independence that was mentioned before. Repeating the process with a new randomization of the dataset columns (samples) may alter the results, but it shouldn't deviate that much.

The leave one out method was also tried out. For this method only 1 sample is kept as validation in each iteration and the rest of the samples are used for training. However in general the LOO method is only to be used when the number of samples is small to have as big training sample as possible. Since we have a big sample size, this is not required and will also take a lot of time to run. Nevertheless it was tried out and the Naive Bayes produced ~50% accuracy while the 17NN with manhattan metric produced 59.3%.

Before doing any of the things mentioned above a 10% of the data was hidden and the rest 90% was used through all the algorithms and validations. This was done so that we can have a completely unbiased estimate of the accuracy given the algorithm and configuration that the validation test has indicated to be the best. The accuracy with which the 17NN_manhattan algorithm managed to classify the test set was 0.5973154362416108 however this number would vary through different randomizations of the base dataset. To have a more accurate feel of the true accuracy, the whole process should be repeated as to try out different parts of the randomized data as test set through an iterative process (nested cross validation).