HW1 **Jonas Korbinian Julian Pfeiffer**
COMPSCI 589 Machine Learning JPfeiffer - 31195456
Benjamin M. Marlin Due Date: 02/09/17

1. Classifier Selection

k - Nearest Neighbor

K-Nearest Neighbor (KNN) is a machine learning algorithm traditionally used for supervised learning. Although modified versions can be used for outlier detection or unsupervised learning. The traditional KNN classifier takes the k closest labeled data points to a certain data point into account. k being an integer specified by the user. The voted average of the k nearest neighbors set the label of the new data point. Different distance measures can be used to calculate the distance between the respective data points. One such metric is the Manhattan distance which is of the form

$$d(\mathbf{a}, \mathbf{b}) = \sum_{i}^{I} |a_i - b_i| \tag{1}$$

d being the distance between the data points $\mathbf{a}$ and $\mathbf{b}$. $\mathbf{a}$ and $\mathbf{b}$ are vectors of size I. It is called Manhattan distance because the distance between to data points is calculated by summing up the difference between the respective values of each dimension. In a 2 dimensional space this would be like walking through the block structure of Manhattan in New York City. In order to get from one intersection to another, one would need to first walk in a straight line in one dimension and then another straight line in the other dimension as buildings are in the way of walking a straight line between start and destination.
The hyperparameters of KNN are k and the distance metric. k being the number of nearest neighbors which should be used for the voting of the predicted label. Traditionally, in a binary classification problem an odd integer is chosen for k so that the classification does not have a tied voting, thus having to choose a random label. The distance metric is the second hyperparameter. Manhattan distance, as described previously is one possible parameter setting, but any distance metric can be used to calculate the distance between two data points.
I chose KNN as a classification model as it usually gives a good prediction on small datasets and it is very straight forward to understand. The amount of data is in the data set given is not very high, making KNN a possible classifier.

Decision Tree

Decision Tree classification is probably one of the most intuitive models to understand after it has been trained. The idea is to build a tree which at every leaf node will have a feature which splits up in different possible branches. This can either be done with categorical values {*"Small", "Medium", "Large"*} or number regions {*[0;1), [1;2.5), [2.5;5]*}. The traditional decision tree algorithm starts building the tree from top down, deciding on the splitting criterion on a greedy basis. The splitting criterion is one of the hyperparameters to be set. The basis on which the tree is split is usually based on node purity. A node is the purest when all underlying data points are of only one class, as opposed to it being the impurest when there is an equal amount of all labels at the node. Thus at each node the following node purity should be maximized. This can be done for example with the gini impurity measurement.

$$I(\mathbf{f}) = \sum_{i}^{J} f_i(1 - f_i) \tag{2}$$

Where $\mathbf{f}$ is the probability of a data point to be a certain label. The gini impurity is calculated by summing over all labels J and multiplying the probability of a data point being label i on a certain node with the converse probability.
There are a lot of hyperparameters for the decision tree that can be tuned. The most important ones besides the impurity metric are the depth of the tree and the number of samples on a leaf node. Both parameters are very import because when not defined the tree is bound to overfit. Depth of the tree defines how far the tree is allowed to span. The higher depth is, the more features are used and at a certain point only a small amount of training data is actually affected by the branch, making the decision very uncertain. The number of data points per node targets the problem from the other direction. There can only be a split on the data if at least the defined amount of data points are left in the successing node.
I chose the decision tree classification model as the understanding of the trained model is intuitively understandable. Other than that, no normalization is needed, and the prediction time is very low. Decision trees are a very good baseline for comparison with other classifiers.

AdaBoost

Adaboost is a type of ensemble classification algorithm. Ensembles are algorithms which rely on weak models, like decision trees or in the most extreme case even decision stumps, to make weak predictions. The theory relies more or less on the wisdom of the crowd, saying that if you ask many stupid people a question, in average you will get the same, or even a better answer than if you would ask an expert.

$$F_t(\mathbf{x}) = \sum_{t}^{T} f_t(\mathbf{x}) \tag{3}$$

$$E_t(\mathbf{x}) = \sum_{i}^{I} E[F_{t-1}(x_i) + \alpha h(x_i)] \tag{4}$$

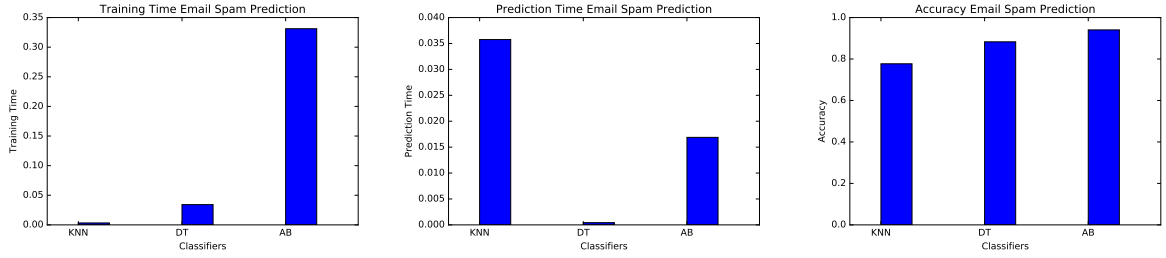
Adaboost successively trains a weak model and measures its accuracy. In a broad sense (3) shows the general boosting of algorithms, where f_t is one trained model. The subsequent expected value E_t is calculated by minimizing the previous model predictions F_{t-1} and the hypothesis h of data point x_i with a variable weighting α. A user specified T models are trained, thus all of them in the end predicting the label of x_i.
AdaBoost uses a weak model, let us say a decision stump, and classifies the dataset into two parts. The accuracy of each set is subsequently stored, while wrongly classified data points indirectly receive a higher weighting. With this modified data set a new classifier is trained and everything is repeated T times. For the prediction of new unlabeled data the precision of every model is accumulated with the respective predicted class so that in the end the most likely label is set.
The most important hyperparameters is the base model, which per default is the decision tree, and the number of estimators after which the ensemble learner will stop generating new models. Of course the hyperparameters of the base model can be tweaked aswell.
I chose AdaBoost as it is commonly known that this is one of the best out-of-the-box classifiers. The training and prediction time are acceptable. The best feature of AdaBoost is that you can set the base model according the the data set at hand, making AdaBoost a reasonable choice for almost every dataset.

2. Classifier Relationships

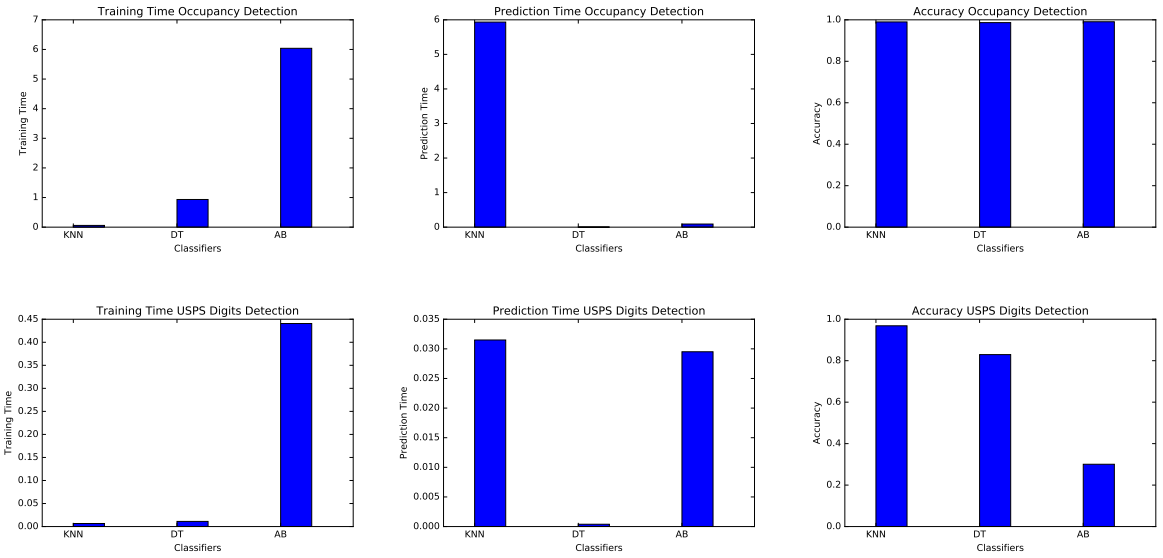
Decision trees (DT) and default AdaBoosts (AB), and KNNs do not have very much in common. While DTs and ABs will only use a certain amount of features for classification (depth of tree) KNNs use all features to calculate the distance. AB, in its default version, on the other hand has a lot in common with a DT. AB is in an extreme case a weighted combination of decision stumps, having similar splitting rules as a DT. Big difference between those two is the weighted factor of AB.
The main difference between DT and AB, and KNN classifiers is the amount of training time and prediction time. While KNN practically has no training time (only the whole training data needs to be stored) the training of a DT and AB is comparatively higher depending on the depth of the tree and the size of the training set. But when the DT and AB is modeled, the prediction is very fast as only the features of a data point needs to be "walked" through the tree or for AB all decision stumps need to be set and weighted to find the prediction. On the other hand, KNN needs a lot of time to predict as the distance of every test data point needs to be calculated to every train data point. This is computationally very expensive, making KNN impractical for large datasets with many predictions. The modeling time of DT and AB is exponentially dependent on the depth and amount of estimators.
While the decision boundary of AB and DT are very similar, being very quadratic, having split rules that cut the hyperspace into pieces in parallel to the dimensions, the boundaries of KNNs are round, surrounding the closest data points.
In general one must say that depending on the problem KNN and DT are very different and for different problems either algorithm will be better. On the other hand, AB is in general a very good alround classifier.
The biggest problem that KNN has is the exponential distance with respect to dimensionality. That means, the more features a data set has, the further the data points are from each other, decreasing the density and making prediction based on KNN hard. For these circumstances DT is often the better choice, as the dimensionality has little influence on the complexity of modeling.
On the other hand DTs easily overfit the trainings data with respect to the depth of the tree. If a high k value is chosen for KNN this is not a problem.

3. Default Classifier Accuracy

3. a and b



In overall average the best classifier is the decision tree. That is only the case because in the case of the USPS dataset the AdaBoost algorithm was extremely bad. Adaboost outperformed the other algorithms by very much on the other datasets. The fastest algorithm is also the decision tree. It never takes more than a second for training and predicting. In my opinion still



AdaBoost is the best classifier. Although in the default configuration it performed very badly on the USPS dataset, I was able to tweek the hyperparameters in a way that it came close to the accuracy of KNN.

4. Hyperparameter Selection Method - Grid Search Cross Validation

4.a

Grid Search Cross Validation is a brute force algorithm for finding the best hyperparameters from a defined set. The user needs to think of all the hyperparameter combinations he/she wants to test for a specific classifier and list them in an array. The user also defines into how many buckets the dataset should be split, and thus how many times the algorithm should be tested with different datasets for each parameter setting. For cross validation the training data is subsequently randomly shuffled and split into the defined set. The buckets are then put back together with always one bucket being spared and set as the validation data. If the user chooses a cross validation parameter $cs = 3$ the dataset is split into 3 buckets $[bucket1, bucket2, bucket3]$. The first training set will therefore consist of $bucket1$ and $bucket2$ while the validation set will consist of $bucket3$. The second training set: $bucket1$ $bucket3$, validation: $bucket2$. Third training: $bucket2$ and $bucket3$ validation $bucket1$. Each possible combined parameter setting will be trained 3 times and the accuracy is calculated for every prediction. The average of all 3 outcomes will be the accuracy of the parameter setting. The parameter setting with the highest accuracy will be returned to the user. Instead of accuracy, other measures like "F1", "precision", "recall", etc. can be set.

4.b

I chose the Grid Search Cross Validation hyperparameter optimization approach because it is the only approach which will definitely find the best parameters combination. Other optimization approaches like "Bayesian Optimization", "Random Search" are based on probability and assume certain distribution of the trainings data which in most cases is not actually given. The problem with GridSearch is that is computationally expensive as it trains a new model with

every parameter combination. Other optimization techniques often have similar results in less training cycles. But for getting the global optimum, GridSearch is the only possible method.

4.c

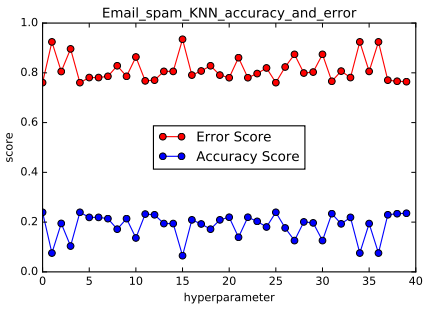
See code "four_and_five_hyperparameters.py"
Class: GridSearchCV_new()

5. Optimizing Hyperparameters

I have optimized the parameters for all datasets for all classifiers. I will report on KNN on the Email dataset. I chose this setting for reporting as KNN performed exceptionally bad with the default parameters on this dataset, and therefore the accuracy, when optimized would probably increase. Also, the dataset has an acceptable amount of dimensions and a binary labeling, making KNN a possible classifier.

5. a and b

The accuracy on the validation set on Email with classifier KNN and hyperparameters k=1 and distance function = "Manhattan" had the highest accuracy of 94.31%. Unfortunately on the test set on Kaggle this parameter setting only scored 80.795%. There is a high discrepancy between the validation set and the test set, which is not usual. Maybe there is an error in the GridSearch where in k=1 the data point is predicted by its own copied data point. But if this were the case, the validation accuracy should by 1.00. This high discrepancy definitely speaks for overfitting of the model.



5.c

There is a definite increase in accuracy on the validation set between the default KNN classifier (0.83329) and the optimized (0.943127). Therefore I see an improvement.

6. Get Creative

For optimization of the classifiers the only new algorithm that I used was random forest. This classifier is a combination of bagging and small decision trees. Random samples of the dataset are "bagged" and small decision trees are trained for these datasets. These low level classifiers subsequently are weighted by their accuracy and used for predicting the label of unseen data.
I used the random forest classifier as the base estimator for AdaBoost for all datasets. With this combination and hyperparameter tuning I was able to score the highest on the Kaggle competition for the Email and Occupancy dataset.
The parameters as seen in Table 1 and Table 2 were chosen. Unfortunately I was not able to score higher than the default parameters of KNN on the USPS dataset.

Appendix

Code for Question 3

File: three_default_classifier_accuracy.py
Execute:
- question_3()
- data_set_for_bargraph()

Code for Question 4 and 5

File: four_and_five_hyperparameters.py
Execute:
- optimize_knn_email()
- acc_default_knn_email()
- optimize_knn_occupancy()
- optimize_knn_USPS()
- optimize_decision_tree_Email()
- optimize_decision_tree_occupancy()
- optimize_decision_tree_USPS()
- optimize_AdaBoost_Email()
- optimize_AdaBoost_occupancy()
- optimize_AdaBoost_USPS()

Code for Question 6

File: six_get_creative.py
Execute:
- adaBoost_final_optimization_email()
- adaBoost_final_optimization_occupancy()
- adaBoost_final_optimization_USPS()

Optimal Hyperparameters

Classifier	AdaBoost
Base Estimator (BE)	RandomForest
Nr. of Estimators	16
BE Criterion	Gini
BE Max Depth	9
BE Min Samples Leaf	2
BE Min Samples Split	4

Table 1: Email Optimal Hyperparameters

Classifier	AdaBoost
Base Estimator (BE)	RandomForest
Nr. of Estimators	15
BE Criterion	Gini
BE Min Samples Leaf	1
BE Min Samples Split	2

Table 2: Occupancy Optimal Hyperparam.