University of Essex
School of Computer Science and Electronic Engineering

Design and Application of a Machine Learning System for a Practical Problem

Submitted as part of the requirements for:

CE802 Machine Learning and Data Mining

Name: Ogulcan Ozer

Tutor: Dr. Luca Citi

Date: 07 December 2019

Table of Contents

1 Pilot-Study Proposal

Word count: 639

In this work as a consultant, we are expected to predict if a new restaurant opened in a new place will be profitable or not, by using the informative features and past information given by the restaurant owner. Since we are not trying to distinguish between given features or group them, clustering is not suitable for this work. Also, we are not doing market basket analysis and trying to find relations, so rules mining is not compatible either. What we need is a supervised learning algorithm that can take advantage of the historical information given and create a generalized model with high accuracy. This leaves us with classification and regression. In theory we could use regression and assume the target as a numerical value between 0-1 and predict our target class based on some threshold value. But classification provides us everything we need. Supervised learning that predicts a target class based on given features. Thus, Classification in this situation, looks like the best approach for the solution of this problem.

To predict our target, we need good and informative features. First feature requested would be socio-economic data of the area where the new restaurant is. This feature would help our algorithm understand in which way the wealth of the people living in that area affect the profit of a given restaurant. Second and the following feature would be the density of the population in that area. Because if not enough people live there, wealth of those people cannot contribute much to the profit of the restaurant. Another useful feature would be the nearest competitor. This feature would help our algorithm rule out if the changes in profit are caused by the competition. Because according to [1] food quality and the menu variety are the most important factors on choosing a restaurant. Since these variables are constant for our restaurant, knowing the closest competitor would indirectly affect the predictions of profit, because of these hidden but important variables. From this, next feature would be the average dining price of the nearest competitor. But this data can be hard to acquire. The last request would be- if the restaurants in the franchise differ in menu or variety- again the average dining price in a given restaurant.

The predictive task was chosen as classification. And there are many classification methods such as decision trees, MLP classifiers, k-NN, SVM. Decision trees are easy to implement, also easy to interpret. Creating IF-Then rules for DTs could be useful for the restaurant owner. But they are also simple and greedy structures which might be too basic for this problem. In the case of k-NN and SVM, both algorithms are similar. In fact, SVMs act like k-NNs but SVMs have a more strict procedure to choose the retained examples, which makes them more dense and compact [2]. Classical MLP classifiers also look like a good option, but they are hard to optimize. MLPs are sensitive and tend to overfit. Thus, for this case, the most appealing learning procedure is SVM.

Evaluation would depend on the size of the data provided by the owner. If the dataset is large, first choice would be splitting the dataset as- 70,15,15- training, validation and test data. This approach would increase the generalization of the model and help lower the over-fitting, since no data is shared among the sets. But if the dataset is not large, the approach would be splitting the dataset as- 2/3, 1/3- training and test data. After the split, the bigger chunk of data would be used to train and validate using k-fold cross validation. Then the smaller part would be used in testing to get an expected error/score. For the last resort, if the dataset is very small, approach would be k-fold cross validation with leave one out, even though there is too much shared data.

2 Comparative Study Word count: 985

For the comparative study, the chosen classification methods were decision tree classifier, support vector classifier and multi-layer perceptron classifier. The data given for the study was divided as training and testing with fractions 2/3 and 1/3 respectively. In the pre-processing part, data was scaled using the scale function of the sklearn library, which centres the data around the mean and scales to the unit variance. Then the data was shuffled in two different ways in different experiments to demonstrate the sensitivity of the MLP classifier, which will be explained with detail in the results part. After the pre-processing of the data, training set was used in 5-fold cross validation, and the test set was used to get the expected error of the classifiers for performance comparisons.

Hyperparameter optimization of the classifiers were done using the built in GridsearchCV function of sklearn library, which optimized the parameters by looking at the scores of the 5-fold CV results mentioned above. For the repeatability of the study, the randomness states of split and shuffle operations were set to 1. Also, the KFold function was defined as a global variable so that each classifier can use it and perform the same splits and shuffles on the training data. In the end, the only difference between the classifications were classifiers themselves and their parameters. At the end of a grid search, the best estimator of a given classifier was picked and tested. Results of the tests then were used to calculate the ROC curve for each classifier and the AUC values. Then the same results were used in McNemar`s test- chi square test with one degree of freedom-. Significance of the difference between each pair of classifiers were computed and confirmed against the significance level alpha.

1 Results

This section is divided into three parts. First two parts show how the results differ when the data is not shuffled and shuffled with random state 1. The last part is the average of the results gathered from five epochs with random shuffling to demonstrate how these classifiers behave on average. The aim of this division is to show how different results can be obtained for an MLP classifier by simply changing the order of the data. All the scoring values used are accuracy scores, and the parameters are for the best estimators of the validation set.

Results without shuffling.

Grid search results using 5-fold CV:

Classifier	Best Validation Score	Best Training Score	Parameters
SVC	0.873	1.0	C= 100, gamma = 0.1, kernel = radial basis function
DT	0.761	1.0(depth > 13)	max depth = 9
MLP	0.857	0.88525	alpha = 0.0001, hidden nodes = 4, solver = adam

Test report of the SVC:

CLASS	Precision	Recall	Accuracy
0/False	0.88	0.88	
1/True	0.85	0.85	
Average	0.87	0.87	**0.866**

Confusion Matrices:

	p=T	p=F
A=T	247	33
A=F	34	186

Test report of the DT:

CLASS	Precision	Recall	Accuracy
0/False	0.77	0.77	
1/True	0.71	0.71	
Average	0.74	0.74	**0.744**

	p=T	p=F
A=T	216	64
A=F	64	156

Test report of the MLP:

CLASS	Precision	Recall	Accuracy
0/False	0.89	0.89	
1/True	0.86	0.85	
Average	0.87	0.87	**0.874**

	p=T	p=F
A=T	249	31
A=F	32	188

Differences between algorithms:

Classifiers	Mcnemar stat.	p-value	Significance α=0.05
SVC vs DT	29.75206612	0.00000005	Significant difference
SVC vs MLP	0.16071429	0.68849974	Non-significant diff.
DT vs MLP	33.85123967	0.00000001	Significant difference

Results with shuffling (random state = 1).

Grid search results using 5-fold CV:

Classifier	Best Validation Score	Best Training Score	Parameters
SVC	0.873	1.0	C= 316.227766016, gamma = 0.1 kernel = radial basis function
DT	0.76	1.0 (depth >13)	max depth = 9
MLP	0.849	0.88925	alpha = 0.001, hidden nodes = 4, solver = adam

Test report of the SVC:

CLASS	Precision	Recall	Accuracy
0/False	0.92	0.89	
1/True	0.86	0.89	
Average	0.89	0.89	**0.89**

Confusion Matrices:

	p=T	p=F
A=T	253	32
A=F	23	192

Test report of the DT:

CLASS	Precision	Recall	Accuracy
0/False	0.78	0.80	
1/True	0.72	0.69	
Average	0.75	0.75	**0.754**

	p=T	p=F
A=T	228	57
A=F	66	149

Test report of the MLP:

CLASS	Precision	Recall	Accuracy
0/False	0.98	0.45	
1/True	0.58	0.99	
Average	0.80	0.68	**0.682**

	p=T	p=F
A=T	129	156
A=F	3	212

Differences between algorithms:

Classifiers	Mcnemar stat.	p-value	Significance α=0.05
SVC vs DT	40.08035714	< 0.00000001	Significant difference
SVC vs MLP	71.68243243	< 0.00000001	Significant difference
DT vs MLP	6.58602151	0.01027826	Significant difference

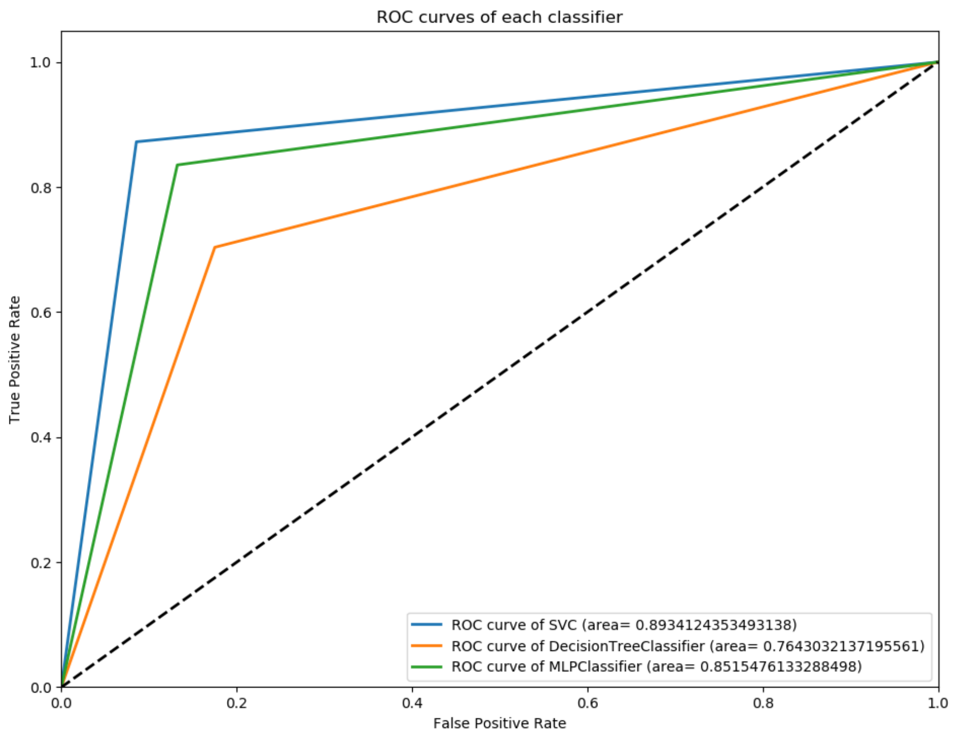

Figure 2 – ROC curves for the results without shuffling.

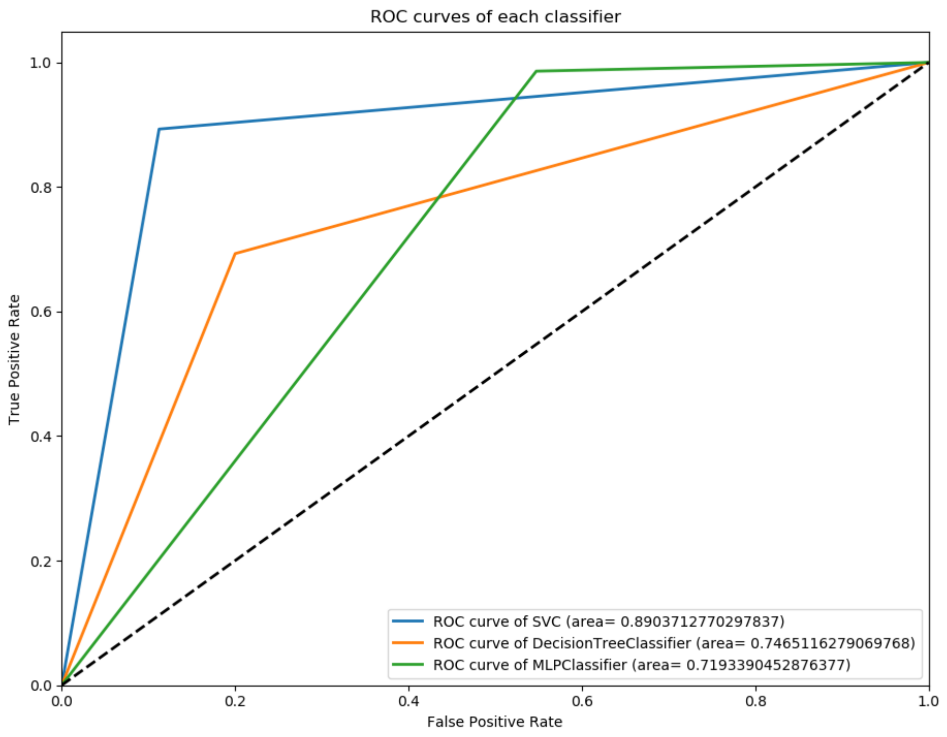

Figure 1 – ROC curves for the results with shuffling (random state = 1)

Average of 5 epochs with shuffling.

Grid search results using 5-fold CV:

Classifier	Avg. Best Score
SVC	**0.879**
DT	**0.7542**
MLP	**0.8486**

Test results:

CLASSIFIER	Avg. Precision	Avg. Recall	Avg. Support	Avg. AUC	Avg. Accuracy
SVM	0.878	0.878	500	0.874204	**0.8756**
DT	0.776	0.772	500	0.770638	**0.7732**
MLP	0.844	0.83	500	0.824669	**0.83**

Average Differences:

Classifiers	Avg. Significance α=0.05
SVC vs DT	Significant difference
SVC vs MLP	Significant difference
DT vs MLP	Significant difference

2 Conclusions

First thing that stands out in the results is the score of the decision tree classifier. Decision tree is a greedy algorithm, which builds the tree by choosing the best split in a given moment. This approach causes DTs to overfit. This also can be seen in the results. While the training score of the tree in the part one is 1.0, score of the best estimator chosen by the grid search function- acquired from 5-fold CV- is 0.761. As we can see infer from the results, the tree is pruned to depth 9 from 13 to prevent overfitting, and the test accuracy results confirm this, since the resulting scores are very similar. I believe the reason for this poor expected performance are the features. Using continuous numerical values and splitting the features in numerical ranges causes information loss, and this results in tree being non-robust. Decision trees are for categorical values and they do not look for numerical relations between values. [3] Because of the inherent structure of DTs, I believe it is hard to build a DT for this problem that can generalize.

Second important result and the reason why there are three different experiments is because how MLPs behave. MLPs are sensitive to noise, scaling and the order of the training data. By looking at the results we can see the expected score of the first experiment is very close to its validation score. But in the second experiment, we have an overfitted MLP with a big difference between its expected and validation scores. Thus, a third experiment was performed, to get a more realistic result on how these classifiers behave on average. Even though

MLPs also tend to overfit, I believe that the accuracy of the MLP can be increased with more data and tweaking.

SVM was the highest scoring classifier throughout the experiments. Given the fact that how robust and compact the algorithm is, this was not a surprise. Even though some overfitting can be inferred form the training scores, the grid search function gives us a n optimized estimator. We can see from the validation and test scores, how well the SVM performs and generalizes. I believe the reason for this is tightly related to the algorithm itself. Since SVMs optimize both for error minimization and margin maximization, it produces a model that can generalize well. And the fact that it can handle both classification and regression problems make this algorithm a great machine learning tool.

Comparisons of the algorithms were done using the ROC curves and their AUC values. And after that McNemar`s test was performed. Resulting ROC curves and the AUC values for the first part can be seen in Figure 1. Results tell us that the SVM is better than MLP and DT, and MLP is better than DT. But the ROC curves and AUC values do not tell how significant this difference is. Very small percentage of difference in performance is most likely significant if the amount of the testing data is huge. [4] Thus, McNemar`s test was applied to each pair of classifiers. We can see from the results that there is no significant difference between the SVM and the MLP classifiers. In the second part, order of the curves and the AUC values are similar but the McNemar results tell us that the performances this time differ significantly. For the last part of the experiment, results averaged over five epochs seems to be in between first and second experiments, and on average there is significant difference between the performances of these three classifiers.

3 References

[1] R. C. Lewis, "Restaurant Advertising: Appeals and Consumers' Intentions," *Journal of Advertising Research,* vol. 21, no. 5, p. 71, 1981.

[2] A. Ethem, *Introduction to Machine Learning*. Cambridge, Massachusetts: The MIT Press, 2010.

[3] P.D.Scott & L.Citi, "Decision Trees Lecture Notes.", slide 24, 2018

[4] Adrian F. Clark, "Evaluating Vision Systems Lecture Notes", p. 4, 2018

4 Appendix

```
#_______________________________________________________________________
_________
# CE802 Machine Learning and Data Mining | Ogulcan Ozer. | 25 December
2018
#_______________________________________________________________________
_________
import graphviz, os, sklearn, numpy as np, pandas as pd,
matplotlib.pyplot as plt
from statsmodels.stats.contingency_tables import mcnemar
from sklearn import tree
from sklearn.utils import shuffle
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV, KFold,
cross_val_score, train_test_split
from sklearn.metrics import accuracy_score , recall_score ,
precision_score , classification_report, confusion_matrix, roc_curve,
auc
from sklearn.preprocessing import StandardScaler

kf_CV = KFold(n_splits=5, shuffle=True)
s_p = 2/3
#-----------------------------------------------------------------------
---------
# Functions
#-----------------------------------------------------------------------
---------
def read_csv():
    dft =
pd.read_csv(os.path.join(os.path.dirname(__file__),"data_ass.csv"))
    dft = shuffle(dft)

    data_scale = dft.iloc[:,:-1]
    scaler = StandardScaler()
    scaler.fit(data_scale)

    train= dft.iloc[:(int(len(dft.index)*s_p)),:-1]
    test= dft.iloc[(int(len(dft.index)*s_p)):,:-1]
    tr_target = dft.iloc[:int(len(dft.index)*s_p),-1].astype(int)
    te_target = dft.iloc[int(len(dft.index)*s_p):,-1].astype(int)
    dfp =
pd.read_csv(os.path.join(os.path.dirname(__file__),"data_predict.csv"))
    predict = dfp.iloc[:,:-1]

    train = pd.DataFrame(scaler.transform(train))
    test = pd.DataFrame(scaler.transform(test))
    predict = pd.DataFrame(scaler.transform(predict))
    return train, test, tr_target, te_target, predict

def _svm(tr, ta, pr):
    #Set the local variables
    data_train, data_target, data_predict = tr, ta, pr
    #Set the parameter ranges for the gridsearch
    Cs = np.logspace(-1, 3, 9)
    Gs = np.logspace(-7, -0, 8)
    #Create an svm classifier with placeholder parameters.
    cls_svm = SVC(gamma=0.0001, C=100.)
    #Initialize gridsearch with our parameters
    cls_svm = GridSearchCV(estimator=cls_svm, param_grid=dict(C=Cs, 
gamma=Gs),return_train_score=True,scoring='accuracy',cv=kf_CV, n_jobs=-
1)
    #Fit the data
    cls_svm.fit(data_train, data_target)
    cls_tostring(cls_svm)
    return cls_svm


def _pdt(tr, ta, pr):
    #Set the local variables
    data_train, data_target, data_predict = tr, ta, pr
    #Set the parameter ranges for the gridsearch
    md = np.arange(1, 20)
     #Create a decision tree with placeholder parameters.
    cls_dt = tree.DecisionTreeClassifier(criterion = "entropy", 
max_depth=3)
    #Initialize gridsearch with our parameters
    cls_dt = GridSearchCV(estimator=cls_dt, 
param_grid=dict(max_depth=md),return_train_score=True,scoring='accuracy
',cv=kf_CV, n_jobs=-1)
    #Fit the data
    cls_dt.fit(data_train, data_target)
    cls_tostring(cls_dt)
    return cls_dt

def _mlp(tr, ta, pr):
    #Set the local variables
    data_train, data_target, data_predict = tr, ta, pr
    #Set the parameter ranges for the gridsearch
    a = 10.0 ** -np.arange(1, 7)
    h = [(2,),(4,),(14,),(28,),(42,),(56,)]
    s = ['sgd','adam']
    #Create an mlp classifier using stochastic gradient descent with 
momentum
    cls_mlp = MLPClassifier(solver='sgd', learning_rate= 
'constant',momentum= .9, nesterovs_momentum= False, learning_rate_init= 
0.2, alpha=1e-5, hidden_layer_sizes=(28,), random_state=1)
    #Initialize gridsearch with our parameters
    cls_mlp = GridSearchCV(estimator=cls_mlp, param_grid=dict(alpha=a, 
hidden_layer_sizes=h, 
solver=s),return_train_score=True,scoring='accuracy',cv=kf_CV, n_jobs=-
1)
    #Fit the data
    cls_mlp.fit(data_train, data_target)
    cls_tostring(cls_mlp)
    return cls_mlp

def cls_tostring(cls):
        best_s = cls.best_score_
        best_p = cls.best_params_
        print("Best training score for the gridsearch of the 
"+cls.best_estimator_.__class__.__qualname__+":")
        print(np.amax(cls.cv_results_['mean_train_score']))
        print(cls.cv_results_)
        print("Best validation score for the gridsearch of the 
"+cls.best_estimator_.__class__.__qualname__+":")
        print(best_s)
        print("Parameters for the best score: ")
        print(best_p)
        return

def _compare(clss, test, target):
        results = []
        for i in range(0,len(clss)):
            results.append(pd.DataFrame(clss[i].predict(test)))

        for i in range(0,len(results)):
            print("Test results of the 
"+clss[i].best_estimator_.__class__.__qualname__+":")
            cls_report = 
classification_report(data_te_target,results[i])
            print(cls_report)
            print("accuracy : " + 
str(accuracy_score(data_te_target,results[i])))
            tn, fp, fn, tp = confusion_matrix(data_te_target, 
results[i]).ravel()
            print(pd.DataFrame(confusion_matrix(data_te_target, 
results[i], labels=[0, 1]), index=['a = T', 'a = F'], columns=['p = T', 
'p = F']))

        print("Differences between algorithms :")
        done = set()
        for i in range(0,len(clss)):
            for j in range(0,len(clss)):
                if(i != j and (i+j not in done)):

print(clss[i].best_estimator_.__class__.__qualname__+" and 
"+clss[j].best_estimator_.__class__.__qualname__)
                    mn = mc_nemar(results[i],results[j],target)
                    done.add(i+j)
                    print('statistic=%.8f, p-value=%.8f' % 
(mn.statistic, mn.pvalue))
                    alpha = 0.05
                    if mn.pvalue > alpha:
                       print('Difference is non-significant')
                    else:
                       print('Significant difference')

        roc_auc(clss,results,target)

def mc_nemar(c1,c2,target):

    ss = sf = fs = ff = 0
    for i in range (0,len(target)):
        t = int(target.iloc[i])
        c1r = int(c1.iloc[i])
        c2r = int(c2.iloc[i])
        if((t == c1r) and (t == c2r)):
            ss = ss + 1
        elif((t != c1r) and (t != c2r)):
            ff = ff + 1
        elif((t==c1r) and (t != c2r)):
            sf = sf + 1
        else:
            fs = fs + 1
    table = [[ss, sf],[fs, ff]]
    result = mcnemar(table, exact=False, correction=True)
    return result

def roc_auc(clss, res, target):
    fpr = dict()
    tpr = dict()
    auc_roc = dict()
    for i in range(0,3):
        fpr[i], tpr[i], _ = roc_curve(target, res[i])
        auc_roc[i] = auc(fpr[i],tpr[i])

    plt.figure()
    for i in range(0,len(res)):
        lbl = f"ROC curve of
{clss[i].best_estimator_.__class__.__qualname__} (area= {auc_roc[i]})"
        plt.plot(fpr[i], tpr[i], lw=2, label = lbl )
        print('AUC of '+clss[i].best_estimator_.__class__.__qualname__)
        print(auc_roc[i])

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curves of each classifier')
    plt.legend(loc="lower right")
    plt.show()

#----------------------------------------------------------------------
---------
# Main program.
#----------------------------------------------------------------------
---------

#Read and parse the training and prediction data.
data_train, data_test, data_tr_target,data_te_target, data_predict =
read_csv()

#Classifier list to hold best fitted classifiers.
classifiers = []

#Append the returned classifiers.
classifiers.append(_svm(data_train, data_tr_target, data_predict))
classifiers.append(_pdt(data_train, data_tr_target, data_predict))
classifiers.append(_mlp(data_train, data_tr_target, data_predict))

#Compare the classifiers.
_compare(classifiers,data_test,data_te_target)

results =
pd.read_csv(os.path.join(os.path.dirname(__file__),"data_predict.csv"))
result_predict =
pd.DataFrame(classifiers[0].best_estimator_.predict(data_predict))
```

```
for i in range(0,len(result_predict.index)):
    if(result_predict.iloc[i,0]==1):
        results.loc[i,'Class']= 'True'
    else:
        results.loc[i,'Class']= 'False'

results.to_csv('results.csv',sep=',',index=False)

#-----------------------------------------------------------------------
---------
# End of Program
#-----------------------------------------------------------------------
---------
```