CE802– Machine Learning and Data Mining

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

Students' name:

Francisco Parrilla A 1900939

Professor's name: Dr Luca Citi

Date: 15/01/2020

1. Pilot-Study Proposal (627 words)
2. Comparative Study (1453 words)

1. Pilot-Study Proposal

   Sometimes, companies are afraid to implement machine learning algorithms in their daily activities, either because the complexity of their current processes are complex and the implementation would be costly and long, or they are stuck to the old way of doing it manually or based on human criteria. Being able to identify misleading/fake information and letting people know about the veracity of information has been something that humans can perform easily, based on intuition and fact checking the information with a proper source, however, it can be time consuming. Currently, companies are looking for a way to use machine learning to identify account that post false information, and this project is aimed to find an optimal algorithm to identify users that are known for posting fake news based on the given data. The only two possible outcomes for this problem, either the user is classified as “troublesome” or not (TRUE and FALSE respectively), therefore, the type of predictive task that must be used is classification.

   Before training the machine learning algorithm, it is needed to identify key features that will tells us whether the veracity of the information. The following features are a suggestion of the type of features that will be informative, they are based on the S.H.A.R.E checklist provided by the UK Government [1] and they can be dependent on each other.

   - **Source** [1]

     It will be beneficial to rate or measure the reputation of the information source, whether is an organization or a person. A measure of veracity of the source, based on previous posts, who they are as a person or organization, use of credible sources.

   - **Headline** [1]

     Many news or stories posted on internet are only made to attract the public, so the use of incredible headlines is a lot, also known as click bait. A measure of how credible a headline might give a criterion to know if the new is fake.

   - **Analyze** [1]

     It is important to check the facts given in the information. The coverage of that information might be high; however, it does not make it true. Instead, the fact/s can be check with reliable sources or fact checking websites/organizations, and based on fact checking, give a measure of how veracious the information is.

- **Retouched** [1]

  A percentage or measure of how much the information has been manipulated, in the case of videos and images, how much it looks like it has been modified. Also, for texts, the information might be taken out of context, and therefore manipulated, usually with older information that is recycled [2].

- **Bias** [1]

  A measure of how bias is the user/organization is their current and previous posts, this might have a dependency with other features.

- **Error** [1]

  Many stories or information that is fake has grammatical errors, or can be found in not genuine websites, so a measure that tells whether the information has grammar errors or/ and is found in not genuine website.

Given that a labelled data set is given, and the ground truth (either TRUE or FALSE) is shown in it, the learning procedures can be narrowed down to supervised learning procedures, such as decision trees, k-NN, SVMs, logistic regression, Naïve Bayes. As a result, the following learning procedures are to be considered: decision trees, k-NN, SVM, Naïve Bayes classifier and logistic regression. To finally narrow down our possible learning techniques, it was taken into consideration an article *"Supervised Learning for Fake News Detection"* [3], where Random Forests and XGBoost, algorithms not mentioned yet, where the ones with the best performance, however, K-NN and SVM were taken into consideration for research purposes.

To measure the accuracy of the algorithms and how reliable they are, it is suggested to make use of a confusion matrix because the problem is a binary classification. With the information from the confusion matrix, the precision, recall and accuracy of each machine learning procedures can be calculated and have a comparison of each algorithm performance for these data set.

2. Comparative Study

2.1 Classification algorithms

After narrowing down the possible algorithms to use for this problem, it was decided to use a pruned Decision Tree, K-Nearest Neighbor, Support Vector Machine, Random Forest.Before training and comparing performance between each algorithm, is important to note that each algorithm has its advantages and disadvantages, and that the performance shown here is specific to this dataset.

Decision trees is a classification algorithm that is easy to understand and is able to deal with interaction among features [4] . Decision trees can handle numerical and/or categorical data, redundant attributes, and is able to work with noisy data, as it is non-parametric [4]. At the end, it can provide a good performance using small computational resources. However, when it comes to dataset with high dimensionality, decision trees might require more time to form a tree, and if not pruned correctly, it can over fit easily [4]. Furthermore, it is more effective on large dataset [5].

K-Nearest Neighbor is an instance-based algorithm [5]. These algorithms are often called lazy learners, because of the retention of all training examples, and that new classifiers are not build until an unseen sample is presented, therefore, training is fast and the theory behind it, is easy to understand and implement [5]. At the same time, it can handle training data that is noisy [5]. On the other hand, its performance is better when the target labels are more than two [5] and might vary with how big the dataset is [4], the value of k needs to be chosen, so it can be computational expensive to do cross validation [4], the algorithm is susceptible to the local structure of the data [5] and is susceptible to irrelevant features [6].

Support Vector Machine algorithm is relatively easy to train with small dataset [6]. At the same time, this algorithm tends to perform well on small datasets with high dimensional space and it is adaptable to numerical and numerical classification [7]. It is possible to get high accuracy with the appropriate choose of parameters, and at the same time avoid over fitting [5]. On the other hand, it can be time and resources consuming when using this algorithm with a large dataset and the performance can be affected when there are overlapping classes [7].

Random Forest algorithm is an ensemble method that might surpass SVM in classification problems [4]. Its advantages are that it is a fast algorithm, handle noise, not susceptible to over fitting, and are easy to interpret [4]. However, increasing the number of trees will lead to a higher training and prediction time,

therefore, the complexity increases and most likely lead to more computational resources [4].

2.2 Imputation methods

Before training the algorithm, it is important to explore the data and search for anomalies, such as missing values. In our case, the dataset has missing values for one feature, which is a problem when using scikit-learn models, as they do not support having missing values.

In order to solve this, it is needed to impute the missing values, this study will compare performance results ignoring F20 feature and using imputation methods that consider F20 feature. The missing values will be replaced in two ways, the mean of all the remaining values in F20, and by using a K-nearest neighbor's imputation. The last method works by imputing the missing values with the mean value from its *"n"* nearest neighbors in the dataset. This will be done using scikit-learn imputation library.

2.3 Approach

The general approach will be to split the data intro training and testing, to get a confusion matrix and evaluate our model with label data that has not been seen by the model.

First, each algorithm will be trained without taking the F20 feature. First, if needed, the features will be normalized, this applies to K-NN and SVM algorithm, and then grid search will be used to get the "best" parameters. Next, with those parameters, a 10 folds cross validation will be done to get an average accuracy of the model based on the training data. Then, the model will predict the target values for our testing data and will compared it to the correct label. From those results, a confusion matrix and a classification report will be made based on the comparison of our predicted values and the ground truth.

After that, the F20 feature will be considered, but using both imputation methods, mean and K- nearest neighbor, and following the same method mentioned recently.

2.4 Results

2.4.1 Without F20 feature

Pruned decision tree

After using grid search to tun the following parameters, max_depth, min_samples_splits, min_samples_leaf and max_features, the best parameters were the following:

max_depth=7, min_samples_splits= 11, min_samples_leaf= 5
max_features = 'sqrt'

Average accuracy with a 10 k-fold cross validation: 0.613444

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	8	47
	FALSE	5	65

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.58	0.93	0.71	70
True	0.62	0.15	0.24	55
accuracy			0.58	125
macro avg	0.60	0.54	0.47	125
weighted avg	0.60	0.58	0.50	125

K-Nearest Neighbor

Best parameters with grid search and cross validation:

n_neighbors=42, weights= 'distance'

Average accuracy with a 10 k-fold cross validation: 0.60284495
Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	12	43
	FALSE	7	63

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.59	0.90	0.72	70
True	0.63	0.22	0.32	55
accuracy			0.60	125
macro avg	0.61	0.56	0.52	125
weighted avg	0.61	0.60	0.54	125

Support Vector Machine

Best parameters with grid search and cross validation:

C=1000.0, gamma= 0.01

Average accuracy with a 10 k-fold cross validation: 0.669487

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	33	22
	FALSE	21	49

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.69	0.70	0.70	70
True	0.61	0.60	0.61	55
accuracy			0.66	125
macro avg	0.65	0.65	0.65	125
weighted avg	0.66	0.66	0.66	125

Random Forest

Best parameters with grid search and cross validation:

max_depth=4, n_estimators= 550

Average accuracy with a 10 k-fold cross validation: 0.5893314

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	12	43
	FALSE	11	59

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.58	0.84	0.69	70
True	0.52	0.22	0.31	55
accuracy			0.57	125
macro avg	0.55	0.53	0.50	125
weighted avg	0.55	0.57	0.52	125

2.4.2 With F20 mean imputation

Pruned decision tree

After using grid search to tun the following parameters, max_depth, min_samples_splits, min_samples_leaf and max_features, the best parameters were the following:

max_depth=10, min_samples_splits= 4, min_samples_leaf= 6
Average accuracy with a 10 k-fold cross validation: 0.615931

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	33	22
	FALSE	33	37

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.63	0.53	0.57	70
True	0.50	0.60	0.55	55
accuracy			0.56	125
macro avg	0.56	0.56	0.56	125
weighted avg	0.57	0.56	0.56	125

K-Nearest Neighbor

Best parameters with grid search and cross validation:

n_neighbors=49, weights= 'distance'

Average accuracy with a 10 k-fold cross validation: 0.6136557

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	19	36
	FALSE	8	62

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.63	0.89	0.74	70
True	0.70	0.35	0.46	55
accuracy			0.65	125
macro avg	0.67	0.62	0.60	125
weighted avg	0.66	0.65	0.62	125

Support Vector Machine

Best parameters with grid search and cross validation:

C=100.0, gamma= 0.01

Average accuracy with a 10 k-fold cross validation: 0.648435

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	32	23
	FALSE	21	49

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.68	0.70	0.69	70
True	0.60	0.58	0.59	55
accuracy			0.65	125
macro avg	0.64	0.64	0.64	125
weighted avg	0.65	0.65	0.65	125

Random Forest

Best parameters with grid search and cross validation:

max_depth=11, n_estimators= 450

Average accuracy with a 10 k-fold cross validation: 0.642532

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	26	29
	FALSE	18	52

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.64	0.74	0.69	70
True	0.59	0.47	0.53	55
accuracy			0.62	125
macro avg	0.62	0.61	0.61	125
weighted avg	0.62	0.62	0.62	125

2.4.3 With F20 k-nearest neighbor imputation

Pruned decision tree

After using grid search to tun the following parameters, max_depth, min_samples_splits, min_samples_leaf and max_features, the best parameters were the following:

max_depth=5, min_samples_splits= 19, min_samples_leaf= 8

Average accuracy with a 10 k-fold cross validation: 0.602702

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	4	51
	FALSE	6	64

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.56	0.91	0.69	70
True	0.40	0.07	0.12	55
accuracy			0.54	125
macro avg	0.48	0.49	0.41	125
weighted avg	0.49	0.54	0.44	125

K-Nearest Neighbor

Best parameters with grid search and cross validation:
n_neighbors=49, weights= 'distance'

Average accuracy with a 10 k-fold cross validation: 0.6056899

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	9	46
	FALSE	5	65

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.59	0.93	0.72	70
True	0.64	0.16	0.26	55
accuracy			0.59	125
macro avg	0.61	0.55	0.49	125
weighted avg	0.61	0.59	0.52	125

Support Vector Machine

Best parameters with grid search and cross validation:
C=100.0, gamma= 0.01

Average accuracy with a 10 k-fold cross validation: 0.6694167

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	33	22
	FALSE	22	48

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.69	0.69	0.69	70
True	0.60	0.60	0.60	55
accuracy			0.65	125
macro avg	0.64	0.64	0.64	125
weighted avg	0.65	0.65	0.65	125

Random Forest

Best parameters with grid search and cross validation:

max_depth=16, n_estimators= 550

Average accuracy with a 10 k-fold cross validation: 0.626386

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	28	27
	FALSE	17	53

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.66	0.76	0.71	70
True	0.62	0.51	0.56	55
accuracy			0.65	125
macro avg	0.64	0.63	0.63	125
weighted avg	0.64	0.65	0.64	125

2.5 Feature Selection

Since the dataset has a considerable number of features, it might be useful to do feature selection to reduce the high dimensionality of the dataset and reduce training and see if the prediction error on the testing set reduces. While searching how to do feature selection, not just by looking a feature importance, but also filtering the correlation between features, a code from GitHub was found, created

by Will Koehrsen [8]. He created a class that does feature selection based on features with high percentage of missing values, collinear features, zero importance features, low importance features, at the end, those features can be removed from the dataset and train your model with the remaining one.

Based on this class, 9 features were removed based on best score. See code for better understanding.

The best accuracy at classifying the testing set was Support Vector Machine with the following parameters:
C = 100.0 and gamma= 0.01

Average accuracy after doing cross validation: 0.716927

Confusion matrix:

		Predicted	
	n =125	TRUE	FALSE
Actual	TRUE	39	16
	FALSE	16	54

***Prediction made on the testing set and compared to ground truth**

Classification report:

	precision	recall	f1-score	support
False	0.77	0.77	0.77	70
True	0.71	0.71	0.71	55
accuracy			0.74	125
macro avg	0.74	0.74	0.74	125
weighted avg	0.74	0.74	0.74	125

However, more research and experimenting must be done with feature selection.

2.6 Conclusion

The last model will be used to make prediction because of the high accuracy on unseen data, compared to other ones. However, features will be removed from the dataset given for predictions.

References

[1] HM Government, "Share Checklist," 2019. [Online]. Available: https://sharechecklist.gov.uk/?gclid=Cj0KCQiA0NfvBRCVARIsAO4930kI2xTxaxy4rnCNAtLGIWEUYnmROosK0uw9s68rQEUi1by4OqX7G4UaAl-eEALw_wcB. [Accessed 24 December 2019].

[2] C. Nagler, "4 Tips for Spotting a Fake News Story," 2019. [Online]. Available: https://www.summer.harvard.edu/inside-summer/4-tips-spotting-fake-news-story. [Accessed 24 December 2019].

[3] J. C. Reis, A. Correia, F. Murai, A. Veloso and F. Benevenuto, "Supervised Learning for Fake News Detection," *IEEE Computer Society,* vol. 34, no. 2, pp. 76-81, 2019.

[4] A. Singh, N. Thakur and A. Sharma, "A Review of Supervised Machine Learning," in *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016.

[5] S. D. Jadhav and H. Channe, "Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques," *International Journal of Science and Research,* vol. 5, no. 1, pp. 1842-1845, 2013.

[6] A. E. Mohamed, "Comparative Study of Four Supervised Machine Learning Techniques for," *International Journal of Applied Science and Technology ,* vol. 7, no. 2, 2017.

[7] K. Stahl, "California State University Stanislaus," 20 April 2018. [Online]. Available: https://www.csustan.edu/sites/default/files/groups/University%20Honors%20Program/Journals/02_stahl.pdf. [Accessed 11 January 2020].

Appendix

The code was divided into 4 jupyter notebooks.

- Jupyter notebook without F20

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
```

```
#Read the dataset
data_full = pd.read_csv("CE802_Ass_2019_Data.csv")

#Lets drop F20 feature
data_nof20 = pd.read_csv("CE802_Ass_2019_Data.csv")
data_nof20.drop(columns=['F20'], inplace = True)

#Lets separate the features and target
data_nof20_class = data_nof20["Class"]
data_nof20_feat = data_nof20.drop(columns = ["Class"],axis = 1)

#Lets first normalize the features for K-NN and SVM
scaler = StandardScaler()
scaler.fit(data_nof20_feat)
data_nof20_feat = scaler.transform(data_nof20_feat)

#Splitting of data to see model accuracy after cross validation and gridse
arch
data_feat_train, data_feat_test, data_class_train, data_class_test = train
_test_split(data_nof20_feat,data_nof20_class,test_size=0.25,stratify=data_
nof20_class,random_state=1234)
```

#Pruned Decision Tree

```
#Decision tree using grid search
clf_tree = tree.DecisionTreeClassifier(criterion = 'entropy',ccp_alpha=0.015,
random_state=1234)
param_grid = {'max_depth': np.arange(4,21),'min_samples_split': np.arange(4,2
1),'min_samples_leaf': np.arange(4,21),
```

```
              'max_features': ['sqrt','auto','log2']}
tree_gridcv = GridSearchCV(clf_tree,param_grid,cv=10 ,n_jobs=-1)
tree_gridcv.fit(data_feat_train,data_class_train)

print("Best parameters: " + str(tree_gridcv.best_params_))
print("Best score: " + str(tree_gridcv.best_score_))


#Now with these parameters, lets perform cross validation
clf_tree_prunned = tree.DecisionTreeClassifier(criterion = 'entropy',ccp_a
lpha=0.015,
                                               max_depth= tree_gridcv.best
_params_['max_depth'],
                                               min_samples_leaf= tree_grid
cv.best_params_['min_samples_leaf'],
                                               min_samples_split=tree_grid
cv.best_params_['min_samples_split'],
                                               max_features=tree_gridcv.be
st_params_['max_features'],random_state=1234)

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_tree = cross_val_score(clf_tree_prunned,data_feat_train,data_class_t
rain,cv=10,n_jobs=-1)
print('Average accuracy:', np.mean(score_tree))
print(score_tree.std())
#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
clf_tree_prunned.fit(data_feat_train,data_class_train)
tree_pred = clf_tree_prunned.predict(data_feat_test)
print(confusion_matrix(data_class_test, tree_pred))
print(classification_report(data_class_test, tree_pred))
```

#K-NN

```
knn_gridcv = KNeighborsClassifier()
#create a dictionary with the number of neighbors to try
param_gridsearch = {'n_neighbors': np.arange(1,80),'weights':['uniform','dist
ance']}

knn_gridsearch = GridSearchCV(knn_gridcv,param_gridsearch,cv=10)
knn_gridsearch.fit(data_feat_train,data_class_train)
print("Best parameters: " + str(knn_gridsearch.best_params_))
print("Best score: "+ str(knn_gridsearch.best_score_))


#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
knn_model = KNeighborsClassifier(n_neighbors = knn_gridsearch.best_params_
['n_neighbors'],
```

```
                                    weights=knn_gridsearch.best_params_['weig
hts'])

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_knn = cross_val_score(knn_model,data_feat_train,data_class_train,cv=
10,n_jobs=-1)
print('Average accuracy:', np.mean(score_knn))
print(score_knn.std())
#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
knn_model.fit(data_feat_train,data_class_train)
knn_pred = knn_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, knn_pred))
print(classification_report(data_class_test, knn_pred))
```

#Support Vector Machine

```
clf_svm = svm.SVC()
param_grid = {'C': np.logspace(-1, 3, 9),
              'gamma': np.logspace(-7, -0, 8)}

svm_gridsearch = GridSearchCV(clf_svm,param_grid,n_jobs=-1, cv = 10)
svm_gridsearch.fit(data_feat_train,data_class_train)

print("Best parameters: " + str(svm_gridsearch.best_params_))
print("Best score : " + str(svm_gridsearch.best_score_))
```

```
#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
svm_model = svm.SVC(C = svm_gridsearch.best_params_['C'],gamma=svm_gridsea
rch.best_params_['gamma'])

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_svm = cross_val_score(svm_model,data_feat_train,data_class_train,cv=
10,n_jobs=-1)
print('Average accuracy:', np.mean(score_svm))
print(score_svm.std())
#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
svm_model.fit(data_feat_train,data_class_train)
svm_pred = svm_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, svm_pred))
print(classification_report(data_class_test, svm_pred))
```

```
rf = RandomForestClassifier(criterion='entropy',random_state=1234)
param_grid = {'n_estimators':[400,450,500,550,600],'max_depth': np.arange(
4,20)}
```

```
#'max_depth': np.arange(4,19),'min_samples_split': np.arange(4,19),'min_sa
mples_leaf': np.arange(4,25)}

rf = GridSearchCV(rf, param_grid,cv=10,n_jobs=-1)
rf.fit(data_feat_train,data_class_train)

print("Best parameters: "+ str(rf.best_params_))
print("Best score: " + str(rf.best_score_))
```

```
#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
rf_model = rf = RandomForestClassifier(criterion='entropy',n_estimators= r
f.best_params_['n_estimators'],
                                       max_depth=rf.best_params_['max_depth
'],random_state=1234)


#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_rf = cross_val_score(rf_model,data_feat_train,data_class_train,cv=10
,n_jobs=-1)
print('Average accuracy:', np.mean(score_rf))
print(score_rf.std())
#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
rf_model.fit(data_feat_train,data_class_train)
rf_pred = rf_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, rf_pred))
print(classification_report(data_class_test, rf_pred))
```

- Jupyter notebook with F20 mean imputation

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
data_full = pd.read_csv("CE802_Ass_2019_Data.csv")

#Replace missing values with mean

data_full.fillna(data_full['F20'].mean(),inplace = True)
```

```
#Lets separate the features and target
data_f20mean_class = data_full["Class"]
data_f20mean_feat = data_full.drop(columns = ["Class"],axis = 1)

#Lets first normalize the features for K-NN and SVM
scaler = StandardScaler()
scaler.fit(data_f20mean_feat)
data_f20mean_feat = scaler.transform(data_f20mean_feat)

#Splitting of data to see model accuracy after cross validation and gridsearc
h
data_feat_train, data_feat_test, data_class_train, data_class_test = train_te
st_split(data_f20mean_feat,data_f20mean_class,test_size=0.25,stratify=data_f2
0mean_class,random_state=1234)
```

#Pruned Decision Tree

```
#Decision tree using grid search
clf_tree = tree.DecisionTreeClassifier(criterion = 'entropy',random_state=123
4)
param_grid = {'max_depth': np.arange(4,21),'min_samples_split': np.arange(4,2
1),'min_samples_leaf': np.arange(4,21)}
tree_gridcv = GridSearchCV(clf_tree,param_grid,cv=10 ,n_jobs=-1)
tree_gridcv.fit(data_feat_train,data_class_train)

print("Best parameters: " + str(tree_gridcv.best_params_))
print("Best score: " + str(tree_gridcv.best_score_))
#Now with these parameters, lets perform cross validation
clf_tree_prunned = tree.DecisionTreeClassifier(criterion = 'entropy',rando
m_state=1234,
                                               max_depth= tree_gridcv.best
_params_['max_depth'],
                                               min_samples_leaf= tree_grid
cv.best_params_['min_samples_leaf'],
                                               min_samples_split=tree_grid
cv.best_params_['min_samples_split'] )

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_tree = cross_val_score(clf_tree_prunned,data_feat_train,data_class_t
rain,cv=10,n_jobs=-1)
print('Average accuracy:', np.mean(score_tree))

#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
clf_tree_prunned.fit(data_feat_train,data_class_train)
tree_pred = clf_tree_prunned.predict(data_feat_test)
print(confusion_matrix(data_class_test, tree_pred))
print(classification_report(data_class_test, tree_pred))
```

#K-NN

```
knn_gridcv = KNeighborsClassifier()
#create a dictionary with the number of neighbors to try
param_gridsearch = {'n_neighbors': np.arange(1,80),'weights':['uniform','dist
ance']}

knn_gridsearch = GridSearchCV(knn_gridcv,param_gridsearch,cv=10)
knn_gridsearch.fit(data_feat_train,data_class_train)
print("Best parameters: " + str(knn_gridsearch.best_params_))
print("Best score: "+ str(knn_gridsearch.best_score_))
```

```
#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
knn_model = KNeighborsClassifier(n_neighbors = knn_gridsearch.best_params_
['n_neighbors'],
                                 weights=knn_gridsearch.best_params_['weig
hts'])

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_knn = cross_val_score(knn_model,data_feat_train,data_class_train,cv=
10,n_jobs=-1)
print('Average accuracy:', np.mean(score_knn))

#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
knn_model.fit(data_feat_train,data_class_train)
knn_pred = knn_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, knn_pred))
print(classification_report(data_class_test, knn_pred))
```

```
#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
svm_model = svm.SVC(C = svm_gridsearch.best_params_['C'],gamma=svm_gridsea
rch.best_params_['gamma'])

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_svm = cross_val_score(svm_model,data_feat_train,data_class_train,cv=
10,n_jobs=-1)
print('Average accuracy:', np.mean(score_svm))

#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
svm_model.fit(data_feat_train,data_class_train)
svm_pred = svm_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, svm_pred))
print(classification_report(data_class_test, svm_pred))
```

#Random Forest

```
rf = RandomForestClassifier(criterion='entropy',random_state=1234)
param_grid = {'n_estimators':[400,450,500,550,600],'max_depth': np.arange(4,2
0)}
#'max_depth': np.arange(4,19),'min_samples_split': np.arange(4,19),'min_sampl
es_leaf': np.arange(4,25)}

rf = GridSearchCV(rf, param_grid,cv=10,n_jobs=-1)
rf.fit(data_feat_train,data_class_train)

print("Best parameters: "+ str(rf.best_params_))
print("Best score: " + str(rf.best_score_))
```

```
#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
rf_model = rf = RandomForestClassifier(criterion='entropy',n_estimators= r
f.best_params_['n_estimators'],
                                       max_depth=rf.best_params_['max_depth
'],random_state=1234)


#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_rf = cross_val_score(rf_model,data_feat_train,data_class_train,cv=10
,n_jobs=-1)
print('Average accuracy:', np.mean(score_rf))
print(score_rf.std())
#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
rf_model.fit(data_feat_train,data_class_train)
rf_pred = rf_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, rf_pred))
print(classification_report(data_class_test, rf_pred))
```

- Jupyter notebook with F20 K-NN imputation

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
```

```
from sklearn.metrics import accuracy_score

#Read the dataset
data_full = pd.read_csv("CE802_Ass_2019_Data.csv")
data_class = data_full["Class"]
data_full = data_full.drop(columns = ["Class"],axis = 1)
#Replace missing values with K-Nearest Neighbor imputation method
data_features= data_full.to_numpy()
imputer = KNNImputer(n_neighbors=2, weights="uniform")
data_features = imputer.fit_transform(data_features)
data_feat = pd.DataFrame(data_features, index=range(data_features.shape[0]
),
                         columns=range(data_features.shape[1]))

#Lets first normalize the features for K-NN and SVM
scaler = StandardScaler()
scaler.fit(data_feat)
data_feat = scaler.transform(data_feat)

#Splitting of data to see model accuracy after cross validation and gridse
arch
data_feat_train, data_feat_test, data_class_train, data_class_test = train
_test_split(data_feat,data_class,test_size=0.25,stratify=data_class,random
_state=1234)
```

#Pruned Decision Tree

```
#Decision tree using grid search
clf_tree = tree.DecisionTreeClassifier(criterion = 'entropy',random_state=123
4)
param_grid = {'max_depth': np.arange(4,21),'min_samples_split': np.arange(4,2
1),'min_samples_leaf': np.arange(4,21)}
tree_gridcv = GridSearchCV(clf_tree,param_grid,cv=10 ,n_jobs=-1)
tree_gridcv.fit(data_feat_train,data_class_train)

print("Best parameters: " + str(tree_gridcv.best_params_))
print("Best score: " + str(tree_gridcv.best_score_))
```

```
#Now with these parameters, lets perform cross validation
clf_tree_prunned = tree.DecisionTreeClassifier(criterion = 'entropy',rando
m_state=1234,
                                               max_depth= tree_gridcv.best
_params_['max_depth'],
                                               min_samples_leaf= tree_grid
cv.best_params_['min_samples_leaf'],
                                               min_samples_split=tree_grid
cv.best_params_['min_samples_split'] )

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
```

```
score_tree = cross_val_score(clf_tree_prunned,data_feat_train,data_class_t
rain,cv=10,n_jobs=-1)
print('Average accuracy:', np.mean(score_tree))

#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
clf_tree_prunned.fit(data_feat_train,data_class_train)
tree_pred = clf_tree_prunned.predict(data_feat_test)
print(confusion_matrix(data_class_test, tree_pred))
print(classification_report(data_class_test, tree_pred))
```

#K-NN

```
knn_gridcv = KNeighborsClassifier()
#create a dictionary with the number of neighbors to try
param_gridsearch = {'n_neighbors': np.arange(1,80),'weights':['uniform','dist
ance']}

knn_gridsearch = GridSearchCV(knn_gridcv,param_gridsearch,cv=10)
knn_gridsearch.fit(data_feat_train,data_class_train)
print("Best parameters: " + str(knn_gridsearch.best_params_))
print("Best score: "+ str(knn_gridsearch.best_score_))
```

```
#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
knn_model = KNeighborsClassifier(n_neighbors = knn_gridsearch.best_params_
['n_neighbors'],
                                 weights=knn_gridsearch.best_params_['weig
hts'])

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_knn = cross_val_score(knn_model,data_feat_train,data_class_train,cv=
10,n_jobs=-1)
print('Average accuracy:', np.mean(score_knn))

#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
knn_model.fit(data_feat_train,data_class_train)
knn_pred = knn_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, knn_pred))
print(classification_report(data_class_test, knn_pred))
```

#Support Vector Machine

```
clf_svm = svm.SVC()
param_grid = {'C': np.logspace(-1, 3, 9),
              'gamma': np.logspace(-7, -0, 8)}

svm_gridsearch = GridSearchCV(clf_svm,param_grid,n_jobs=-1, cv = 10)
```

```
svm_gridsearch.fit(data_feat_train,data_class_train)

print("Best parameters: " + str(svm_gridsearch.best_params_))
print("Best score : " + str(svm_gridsearch.best_score_))


#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
svm_model = svm.SVC(C = svm_gridsearch.best_params_['C'],gamma=svm_gridsea
rch.best_params_['gamma'])

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_svm = cross_val_score(svm_model,data_feat_train,data_class_train,cv=
10,n_jobs=-1)
print('Average accuracy:', np.mean(score_svm))

#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
svm_model.fit(data_feat_train,data_class_train)
svm_pred = svm_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, svm_pred))
print(classification_report(data_class_test, svm_pred))
```

#Random Forest

```
rf = RandomForestClassifier(criterion='entropy',random_state=1234)
param_grid = {'n_estimators':[400,450,500,550,600],'max_depth': np.arange(4,2
0)}
#'max_depth': np.arange(4,19),'min_samples_split': np.arange(4,19),'min_sampl
es_leaf': np.arange(4,25)}

rf = GridSearchCV(rf, param_grid,cv=10,n_jobs=-1)
rf.fit(data_feat_train,data_class_train)

print("Best parameters: "+ str(rf.best_params_))
print("Best score: " + str(rf.best_score_))


#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
rf_model = rf = RandomForestClassifier(criterion='entropy',n_estimators= r
f.best_params_['n_estimators'],
                                       max_depth=rf.best_params_['max_depth
'],random_state=1234)


#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_rf = cross_val_score(rf_model,data_feat_train,data_class_train,cv=10
,n_jobs=-1)
```

```
print('Average accuracy:', np.mean(score_rf))
print(score_rf.std())
#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
rf_model.fit(data_feat_train,data_class_train)
rf_pred = rf_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, rf_pred))
print(classification_report(data_class_test, rf_pred))
```

-Jupyter notebook with feature selection

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from feature_selector import FeatureSelector #credits to Will Koehrsen
#his class is on github https://github.com/WillKoehrsen/feature-selector/b
lob/master/Feature%20Selector%20Usage.ipynb
```

```
#Read the dataset
data_full = pd.read_csv("CE802_Ass_2019_Data.csv")
data_class = data_full["Class"]
data_full = data_full.drop(columns = ["Class"],axis = 1)
```

```
#dictionary with the parameters to loop for
parameters_features = {'missing_threshold': np.arange(0.35,0.45,0.05),
                       'correlation_threshold': np.arange(0.35,0.75,0.05),
                       'task': 'classification',
                       'eval_metric': 'auc',
                       'cumulative_importance': np.arange(0.65,1,0.05)}
```

```
#the following for with search for the best score based on a svm classifie
r
#and with the best score, it will find the best parameters to do feature s
election
score = 0.0000000
for idx,x in enumerate(parameters_features['missing_threshold']):
    for idy,y in enumerate(parameters_features['correlation_threshold']):
        for idz,z in enumerate(parameters_features['cumulative_importance'
]):
            fs.identify_all(selection_params = {'missing_threshold': param
eters_features['missing_threshold'][idx],
```

```
                                                'correlation_threshold': p
arameters_features['correlation_threshold'][idy],
                                                'task': 'classification',
'eval_metric': 'auc',
                                                 'cumulative_importance':
parameters_features['cumulative_importance'][idz]})
            train_removed_all_once = fs.remove(methods = 'all', keep_one_h
ot = True)

            data_features= train_removed_all_once.to_numpy()

            imputer = KNNImputer(n_neighbors=2, weights="uniform")
            data_features = imputer.fit_transform(data_features)
            data_feat = pd.DataFrame(data_features, index=range(data_featu
res.shape[0]),
                                     columns=range(data_features.shape[1]
))

            #Lets first normalize the features for K-NN and SVM
            scaler = StandardScaler()
            scaler.fit(data_feat)
            data_feat = scaler.transform(data_feat)

            #Splitting of data to see model accuracy after cross validatio
n and gridsearch
            data_feat_train, data_feat_test, data_class_train, data_class_
test = train_test_split(data_feat,data_class,test_size=0.25,stratify=data_
class,random_state=1234)

            clf_svm = svm.SVC()
            param_grid = {'C': np.logspace(-1, 3, 9),
                          'gamma': np.logspace(-7, -0, 8)}

            svm_gridsearch = GridSearchCV(clf_svm,param_grid,n_jobs=-1, cv
= 10)
            svm_gridsearch.fit(data_feat_train,data_class_train)
            if svm_gridsearch.best_score_ >= score:
                score = svm_gridsearch.best_score_
                parameters_values = [parameters_features['missing_threshol
d'][idx],parameters_features['correlation_threshold'][idy],parameters_feat
ures['cumulative_importance'][idz]]
            #print("Best parameters: " + str(svm_gridsearch.best_params_))
            #print("Best score : " + str(svm_gridsearch.best_score_))
print(score)
print(parameters_values)
```

```
fs = FeatureSelector(data = data_full, labels = data_class)
fs.identify_all(selection_params = {'missing_threshold': 0.399999999999999
97,
                                    'correlation_threshold': 0.7,
                                    'task': 'classification',
                                    'eval_metric': 'auc',
```

```
                                        'cumulative_importance': 0.75000000000
00001})


train_removed_all_once = fs.remove(methods = 'all', keep_one_hot = True)
print(train_removed_all_once)
data_features= train_removed_all_once.to_numpy()

imputer = KNNImputer(n_neighbors=2, weights="uniform")
data_features = imputer.fit_transform(data_features)
data_feat = pd.DataFrame(data_features, index=range(data_features.shape[0]
),
                                         columns=range(data_features.shape[1]
))
#Lets first normalize the features for K-NN and SVM
scaler = StandardScaler()
scaler.fit(data_feat)
data_feat = scaler.transform(data_feat)

#Splitting of data to see model accuracy after cross validation and gridse
arch
data_feat_train, data_feat_test, data_class_train, data_class_test = train
_test_split(data_feat,data_class,test_size=0.25,stratify=data_class,random
_state=1234)

clf_svm = svm.SVC()
param_grid = {'C': np.logspace(-1, 3, 9),
              'gamma': np.logspace(-7, -0, 8)}

svm_gridsearch = GridSearchCV(clf_svm,param_grid,n_jobs=-1, cv = 10)
svm_gridsearch.fit(data_feat_train,data_class_train)
print("Best parameters: " + str(svm_gridsearch.best_params_))
print("Best score : " + str(svm_gridsearch.best_score_))
#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
svm_model = svm.SVC(C = svm_gridsearch.best_params_['C'],gamma=svm_gridsea
rch.best_params_['gamma'])

#Now lets used cross validation in the whole data set, but with the best p
arameters by gridsearch
score_svm = cross_val_score(svm_model,data_feat_train,data_class_train,cv=
10,n_jobs=-1)
print('Average accuracy:', np.mean(score_svm))

#Now lets compute the confussion matrix by splitting the data into trainni
ng and testing
svm_model.fit(data_feat_train,data_class_train)
svm_pred = svm_model.predict(data_feat_test)
print(confusion_matrix(data_class_test, svm_pred))
print(classification_report(data_class_test, svm_pred))
```

- Jupyter notebook making prediction

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.impute import KNNImputer
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from feature_selector import FeatureSelector #credits to Will Koehrsen
#his class is on github https://github.com/WillKoehrsen/feature-selector/b
lob/master/Feature%20Selector%20Usage.ipynb
```

```
#Read the dataset that is for training the model
data_training = pd.read_csv("CE802_Ass_2019_Data.csv")
data_class_train = data_training["Class"]
data_full_train = data_training.drop(columns = ["Class","F1","F2","F5","F8
","F11","F12","F15","F17","F18"],axis = 1)

imputer = KNNImputer(n_neighbors=2, weights="uniform")
data_features = imputer.fit_transform(data_full_train)
data_feat = pd.DataFrame(data_features, index=range(data_features.shape[0]
),
                                        columns=range(data_features.shape[1]
))
#Lets first normalize the features for K-NN and SVM
scaler = StandardScaler()
scaler.fit(data_feat)
data_feat = scaler.transform(data_feat)

#Training model
svm_model = svm.SVC(C= 100.0, gamma= 0.01)
svm_model.fit(data_feat,data_class_train)
```

```
#Read the dataset that is going to be predicted
data_pred= pd.read_csv("CE802_Ass_2019_Test.csv")
data_pred = data_pred.drop(columns = ["Class","F1","F2","F5","F8","F11","F
12","F15","F17","F18"],axis = 1)

imputer = KNNImputer(n_neighbors=2, weights="uniform")
data_features = imputer.fit_transform(data_pred)
data_pred = pd.DataFrame(data_features, index=range(data_features.shape[0]
),
                                        columns=range(data_features.shape[1]
))
```

```
#Lets first normalize the features for K-NN and SVM
scaler = StandardScaler()
scaler.fit(data_pred)
data_pred = scaler.transform(data_pred)
```

```
svm_pred = svm_model.predict(data_pred)
pred =  pd.DataFrame(svm_pred,columns=['Class'])
final_pred = pd.read_csv('CE802_Ass_2019_Test.csv')
final_pred.drop(columns = ['Class'],inplace = True)
submit_csv = pd.concat([final_pred,pred],axis=1)
submit_csv.to_csv("CE802_Ass_2019_Test.csv", index = False)
```