```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Dec  1 23:10:31 2018

@author: ingridkasbah
"""


#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Dec  2 19:53:15 2018

@author: ingridkasbah
"""


from scipy import stats
from sklearn.metrics import fbeta_score, accuracy_score
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClas
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import VotingClassifier
from sklearn.feature_selection import SelectFromModel

# linear algebra

import numpy as np

# data processing
import pandas as pd

    # data visualization
import seaborn as sns
from matplotlib import pyplot as plt
from matplotlib import style

# Algorithms
```

```
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
from dataprep import dataprep
from classify import classify

traindata = 'data/train.csv'# Load  the train data
testdata='data/test.csv'   #load test data
X_train, Y_train, X_test,train_df,test_df= dataprep(traindata,testdata)

'''
#try with cross_val
from sklearn.cross_validation import StratifiedKFold
cross_validation = StratifiedKFold(Y_train, n_folds=5)
'''

#Stochastic gradient descent with cross validation
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
scores_sgd = cross_val_score(sgd, X_train, Y_train, cv=10, scoring = "acc
print("Scores:", scores_sgd)
print("Mean:", scores_sgd.mean())
print("Standard Deviation:", scores_sgd.std())



#Random Forest with cross validation
rf = RandomForestClassifier(n_estimators=100)
scores_rf=cross_val_score(rf, X_train, Y_train, cv=10, scoring = "accurac
print("Scores:", scores_rf)
print("Mean:", scores_rf.mean())
print("Standard Deviation:", scores_rf.std())

#Logistic Regression with cross validation
logreg = LogisticRegression()
scores_logreg=cross_val_score(logreg, X_train, Y_train, cv=10, scoring =
print("Scores:", scores_logreg)
print("Mean:", scores_logreg.mean())
print("Standard Deviation:", scores_logreg.std())
logreg.fit(X_train,Y_train)
Y_pred_log=logreg.predict(X_test)

#K Nearest Neighbors with cross validation
knn = KNeighborsClassifier(n_neighbors = 3)
scores_knn=cross_val_score(knn, X_train, Y_train, cv=10, scoring = "accur
print("Scores:", scores_knn)
print("Mean:", scores_knn.mean())
print("Standard Deviation:", scores_knn.std())

#Gaussian Naive Bayes with cross validation
gaussian = GaussianNB()
scores_gauss=cross_val_score(gaussian, X_train, Y_train, cv=10, scoring =
print("Scores:", scores_gauss)
print("Mean:", scores_gauss.mean())
print("Standard Deviation:", scores_gauss.std())

#Perceptron with cross validation
perceptron = Perceptron(max_iter=5)
scores_per=cross_val_score(perceptron, X_train, Y_train, cv=10, scoring =
print("Scores:", scores_per)
print("Mean:", scores_per.mean())
print("Standard Deviation:", scores_per.std())

#Decision Tree with cross validation
decision_tree = DecisionTreeClassifier()
scores_dt=cross_val_score(decision_tree, X_train, Y_train, cv=10, scoring
print("Scores:", scores_dt)
print("Mean:", scores_dt.mean())
print("Standard Deviation:", scores_dt.std())

#Support Vector Machine with cross validation
linear_svc = LinearSVC()
scores_svc=cross_val_score(linear_svc, X_train, Y_train, cv=10, scoring =
print("Scores:", scores_svc)
print("Mean:", scores_svc.mean())
print("Standard Deviation:", scores_svc.std())

#adaboost with cross validation

ada_boost = AdaBoostClassifier(random_state =42)
scores_ada=cross_val_score(ada_boost, X_train, Y_train, cv=10, scoring =
print("Scores:", scores_ada)
print("Mean:", scores_ada.mean())
print("Standard Deviation:", scores_ada.std())

#check for best model
results = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree','AdaBoost'],
    'Score': [scores_svc.mean(), scores_knn.mean(), scores_logreg.mean(),
              scores_rf.mean(), scores_gauss.mean(), scores_per.mean(),
              scores_sgd.mean(), scores_dt.mean(),scores_ada.mean()]})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(10)
```

```
#Linear Support Vector Machine Tuning:
linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)
acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
Y_predict_lsvc=linear_svc.predict(X_test)
SVCpipe = Pipeline([('scale', StandardScaler()),
                   ('SVC',LinearSVC())])

#  Perform grid search on the classifier
#using GridSearchCV()
parameters = {'SVC__C':np.arange(0.01,100,10)}
grid_svc = GridSearchCV(SVCpipe,parameters,cv=5,return_train_score=True)
grid_svc.fit(X_train,Y_train)
print(grid_svc.best_params_)

best_svc = grid_svc.best_estimator_
scores = cross_val_score(best_svc, X_train, Y_train, cv=10, scoring = "ac


# train on full X_train
svc_trainfull = (best_svc.fit(X_train, y=Y_train))
Y_predict_svc=best_svc.predict(X_test)
acc_linear_svc = round(svc_trainfull.score(X_train, Y_train) * 100, 2)
#Y_predictions=svc_training.predict(X_test)
from sklearn import model_selection

#Choosing models to use for ensemble learning
estimators = []
model1 = LogisticRegression()
estimators.append(('lr', model1))
model3 = SVC()
estimators.append(('svm', model3))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X_train, Y_train, cv=
print(results.mean())
ultimate_model=ensemble.fit(X_train,Y_train)
ultimate_model.score(X_train,Y_train)

Y_pred_ensemble=ultimate_model.predict(X_test)
Y_pred_ensemble
output2=test_df
output=test_df
output2['survived']=Y_pred_ensemble
output['survived']=Y_pred_ensemble

#Optimizing ADABOOST
# Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
# Initialize the classifier
clf = AdaBoostClassifier(random_state =42)
clf.fit(X_train,Y_train)
# Create the parameters list you wish to tune, using a dictionary if need
# parameters = {'n_estimators' : [50, 100, 200], 'learning_rate' : [0.1,


# Perform grid search on the classifier using 'scorer' as the scoring met
grid_obj = GridSearchCV(clf, parameters)
scores_adaopt = cross_val_score(grid_obj, X_train, Y_train, cv=10, scorin
#Fit the grid search object to the training data and find the optimal par
grid_fit = grid_obj.fit(X_train, Y_train)

best_clf = grid_fit.best_estimator_

#train on the whole model

model2 = best_clf.fit(X_train, Y_train)
best_predictions = best_clf.predict(X_train)

Y_predict_ada=best_clf.predict(X_test)
acc_ada=accuracy_score(Y_train, best_predictions)


#training random forest again
random_forest = RandomForestClassifier(n_estimators=100, oob_score = True
rf.fit(X_train, Y_train)
Y_rf_pred=rf.predict(X_test)

rf.score(X_train, Y_train)
Y_predict_rf=rf.predict(X_test)
acc_random_forest = round(rf.score(X_train, Y_train) * 100, 2)
print(round(acc_random_forest,2,), "%")

#Feature selection to prevent overfitting
#Feature Importance
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.rou
importances = importances.sort_values('importance',ascending=False).set_i
importances.head(15)

#dropping not important features
train_df  = train_df.drop("not_alone", axis=1)
test_df  = test_df.drop("not_alone", axis=1)

train_df  = train_df.drop("Parch", axis=1)
test_df  = test_df.drop("Parch", axis=1)

#training the model again
random_forest = RandomForestClassifier(n_estimators=100, oob_score = True
random_forest.fit(X_train, Y_train)
random_forest.score(X_train, Y_train)

acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
print(round(acc_random_forest,2,), "%")

#Random forest tuning using grid search
param_grid = { "criterion" : ["gini", "entropy"], "min_samples_leaf" : [1
rf = RandomForestClassifier(n_estimators=100, max_features='auto', oob_sc
clf = GridSearchCV(estimator=rf, param_grid=param_grid, n_jobs=-1)
clf.fit(X_train, Y_train)
clf.best_params_

random_forest = RandomForestClassifier(criterion = "gini",
                                       min_samples_leaf = 1,
                                       min_samples_split = 10,
                                       n_estimators=100,
                                       max_features='auto',
                                       oob_score=True,
                                       random_state=1,
                                       n_jobs=-1)
scores_rf=cross_val_score(random_forest, X_train, Y_train, cv=10, scoring
random_forest.fit(X_train, Y_train)
random_forest.score(X_train, Y_train)

print("oob score:", round(random_forest.oob_score_, 4)*100, "%")
predictions=random_forest.predict(X_train)
confusion_matrix(Y_train, predictions)
Y_predictions=random_forest.predict(X_test)
```