**Titulo:** Exercício 3 - kNN, SVM, Redes Neurais, Random Forest e Gradient Boosting
**Autor:** Juan Sebastián Beleño Díaz
**Data:** 21 de Outubro de 2016

# Introdução

Neste trabalho é feita uma comparação na precisão de diferentes classificadores(kNN, SVM, Redes Neurais, Random Forest e Gradient Boosting). Todos os classificadores usam uma validação cruzada externa de 5 folds para achar a média da precisão e uma validação cruzada interna de 3 folds para escolher os hiperparâmetros.

# Dados

Os arquivos usados neste trabalho são secom.data e secom_labels.data que pertencem ao conjunto de dados SECOM. O conjunto de dados do SECOM foram dados coletados dum processo de fabricação de semicondutores complexo. O arquivo *secom.data* contem os dados principais usados neste trabalho. O arquivo *secom_labels.data* na primeira coluna contem as classes dos dados de *secom.data*.

# Preparação dos dados

Antes de começar trablahar com os dados é preciso incluir as dependecias do projeto:

```
# Loading the libraries
import numpy as np
import pandas as pd

from sklearn.model_selection import StratifiedKFold
from sklearn.decomposition import PCA
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

Existem muitas maneiras de abrir os arquivos e obter os dados, mas neste caso foi usado *pandas* para obter o dataframe diretamente desde a URL.

```
# Defining the URIs with raw data
url_parameters =
'https://archive.ics.uci.edu/ml/machine-learning-databases/secom/secom.data'
url_results =
'https://archive.ics.uci.edu/ml/machine-learning-databases/secom/secom_labels.data'

# Reading the files with the raw data
```

```
df_parameters = pd.read_csv(url_parameters, header = 0, delimiter = " ")
df_results = pd.read_csv(url_results, header = 0, delimiter = " ")

# Getting classes from result
df_classes = df_results.iloc[:, 0:1]
df_classes = np.ravel(df_classes)
```

No código embaixo foram declaradas as variáveis que seram usadas pelos classificadores, incluindo os possíveis hiperparâmetros.

```
# Number of columns and rows in the raw data
n_columns = df_parameters.shape[1]
n_rows = df_parameters.shape[0]

# Precision mean for all models
knn_precision = 0
svm_precision = 0
neural_net_precision = 0
random_forest_precision = 0
gbm_precision = 0

# Folds variables
n_external_folds = 5
n_internal_folds = 3

# 80% of variance in the PCA
variance_percentage_pca = 0.8
n_components_pca = 0

# k values for kNN
knn_parameters = {'n_neighbors':[1, 5, 11, 15, 21, 25]}

# parameters for SVM
svm_parameters = {'kernel':['rbf'], 'C':[2**(-5), 2**(0), 2**(5), 2**(10)],
'gamma':[2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)]}

# Number of neurons in the hidden layer for Neural nets
neural_nets_parameters = {'hidden_layer_sizes':[10, 20, 30, 40]}

# Random Forest parameters
random_forest_parameters = {'max_features':[10, 15, 20, 25], 'n_estimators':[100,
200, 300, 400]}

# Parameters for Gradient Boosting Machine
gbm_parameters ={'learning_rate':[0.1, 0.05], 'max_depth':[5], 'n_estimators':[30,
70, 100]}
```

# Definição de classificadores

Depois de preparar os dados, já podemos empezar declarar os classificadores em funções para melhorar a modularidade do código.

## Classificador kNN

```
def get_precision_kNN_PCA(parameters, vp_pca, train_params, test_params,
train_classes, test_classes, n_folds):

    # Applying the PCA keeping the variance over vp_pca %
    pca = PCA(n_components = vp_pca)
    pca.fit(train_params)
    params_reduced_train = pca.transform(train_params)
    params_reduced_test = pca.transform(test_params)

    # GridSearch over the kNN parameters using a 3 KFold
    # The cv parameter is for Cross-validation
    # We find the hyperparameters here
    knn = KNeighborsClassifier()
    clf_knn = GridSearchCV(knn, parameters, cv=n_folds)
    clf_knn.fit(params_reduced_train, train_classes)

    # Getting the best hyperparameters
    knn_best_hyperparams = clf_knn.best_params_

    # Create the best kNN model
    knn_tuned =
KNeighborsClassifier(n_neighbors=knn_best_hyperparams['n_neighbors'])
    knn_tuned.fit(params_reduced_train, train_classes)

    # Get the precision of the model
    knn_tuned_score = knn_tuned.score(params_reduced_test, test_classes)

    return knn_tuned_score
```

## Classificador SVM

```
def get_precision_svm(parameters, train_params, test_params, train_classes,
test_classes, n_folds):

    # GridSearch over the SVM parameters using a 3 KFold
    # The cv parameter is for Cross-validation
    # We find the hyperparameters here
    svm = SVC()
    clf_svm = GridSearchCV(svm, parameters, cv=n_folds)
    clf_svm.fit(train_params, train_classes)

    # Getting the best hyperparameters
    svm_best_hyperparams = clf_svm.best_params_

    # Create the best SVM model
    svm_tuned = SVC(C = svm_best_hyperparams['C'], kernel =
svm_best_hyperparams['kernel'], gamma = svm_best_hyperparams['gamma'])
    svm_tuned.fit(train_params, train_classes)

    # Getting the model precision
    svm_tuned_score = svm_tuned.score(test_params, test_classes)
```

```
    return svm_tuned_score
```

## Classificador Redes Neurais

```
def get_precision_neural_nets(parameters, train_params, test_params, train_classes,
test_classes, n_folds):

    # GridSearch over the Neural Nets parameters using a 3 KFold
    # The cv parameter is for Cross-validation
    # We find the hyperparameters here
    nn = MLPClassifier()
    clf_nn = GridSearchCV(nn, parameters, cv=n_folds)
    clf_nn.fit(train_params, train_classes)

    # Getting the best hyperparameters
    nn_best_hyperparams = clf_nn.best_params_

    # Create the best Neural Net model
    nn_tuned = MLPClassifier(hidden_layer_sizes =
nn_best_hyperparams['hidden_layer_sizes'])
    nn_tuned.fit(train_params, train_classes)

    # Getting the model precision
    nn_tuned_score = nn_tuned.score(test_params, test_classes)

    return nn_tuned_score
```

## Classificador Random Forest

```
def get_precision_random_forest(parameters, train_params, test_params,
train_classes, test_classes, n_folds):

    # GridSearch over the Random Forest parameters using a 3 KFold
    # The cv parameter is for Cross-validation
    # We find the hyperparameters here
    rf = RandomForestClassifier()
    clf_rf = GridSearchCV(rf, parameters, cv=n_folds)
    clf_rf.fit(train_params, train_classes)

    # Getting the best hyperparameters
    rf_best_hyperparams = clf_rf.best_params_

    # Create the best Random Forest model
    rf_tuned = RandomForestClassifier(max_features =
rf_best_hyperparams['max_features'], n_estimators =
rf_best_hyperparams['n_estimators'])
    rf_tuned.fit(train_params, train_classes)

    # Getting the model precision
    rf_tuned_score = rf_tuned.score(test_params, test_classes)

    return rf_tuned_score
```

## Classificador GBM

```
def get_precision_gbm(parameters, train_params, test_params, train_classes,
test_classes, n_folds):

    # GridSearch over the Grid Boosting parameters using a 3 KFold
    # The cv parameter is for Cross-validation
    # We find the hyperparameters here
    gbm = GradientBoostingClassifier()
    clf_gbm = GridSearchCV(gbm, parameters, cv=n_folds)
    clf_gbm.fit(train_params, train_classes)

    # Getting the best hyperparameters
    gbm_best_hyperparams = clf_gbm.best_params_

    # Create the best Grid Boosting model
    gbm_tuned = GradientBoostingClassifier(learning_rate =
gbm_best_hyperparams['learning_rate'], max_depth =
gbm_best_hyperparams['max_depth'], n_estimators =
gbm_best_hyperparams['n_estimators'])
    gbm_tuned.fit(train_params, train_classes)

    # Getting the model precision
    gbm_tuned_score = gbm_tuned.score(test_params, test_classes)

    return gbm_tuned_score
```

# Processamento dos classificadores

Neste ponto o código faz uma validaçaõ cruzada externa de 5 folds, uma imputação de dados usando a média da colunas, normaliza os dados e faz uma validação cruzada interna de 3 folds sobre cada classificador para achar a precisão de cada um deles. As precisões são somadas em variaveis para depois calcular a média dos classificadores segundo a validação cruzada externa.

```
# -------------------------- Here goes the magic -----------------------------

# Define the external K-Fold Stratified
external_skf = StratifiedKFold(n_splits = n_external_folds)
external_skf.get_n_splits(df_parameters, df_classes)

# Iterate over external data
for external_train_index, external_test_index in external_skf.split(df_parameters,
df_classes):

    # Split the external training set and the external test set
    external_params_train = df_parameters.iloc[external_train_index, :]
    external_classes_train = df_classes[external_train_index]
    external_params_test = df_parameters.iloc[external_test_index, :]
    external_classes_test = df_classes[external_test_index]
```

```
    # ********************** Imputation of data ***************************
    imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
    imp.fit(external_params_train)

    # Appling the imputation
    imp_external_params_train = imp.transform(external_params_train)
    imp_external_params_test = imp.transform(external_params_test)

    # Scaling the data
    scaler = StandardScaler().fit(imp_external_params_train)
    scaled_external_params_train = scaler.transform(imp_external_params_train)
    scaled_external_params_test = scaler.transform(imp_external_params_test)

    # Cleaning NaN for bad scaling
    # clean_external_params_train = np.nan_to_num(scaled_external_params_train)
    # clean_external_params_test = np.nan_to_num(scaled_external_params_test)


    # Getting the kNN precision for this fold keeping PCA with 80% of the variance
and internal CV 3-Fold
    knn_score = get_precision_kNN_PCA(knn_parameters, variance_percentage_pca,
scaled_external_params_train, scaled_external_params_test, external_classes_train,
external_classes_test, n_internal_folds)
    # 0.929769178069

    # Getting the precision of SVM with kernel RBF using a 3-Fold internal CV
    svm_score = get_precision_svm(svm_parameters, scaled_external_params_train,
scaled_external_params_test, external_classes_train, external_classes_test,
n_internal_folds)
    # 0.93359083412

    # Getting the precision of neural nets
    neural_net_score = get_precision_neural_nets(neural_nets_parameters,
scaled_external_params_train, scaled_external_params_test, external_classes_train,
external_classes_test, n_internal_folds)
    # 0.853934283295

    # Getting the precision of random forest
    random_forest_score = get_precision_random_forest(random_forest_parameters,
scaled_external_params_train, scaled_external_params_test, external_classes_train,
external_classes_test, n_internal_folds)
    # 0.93359083412

    # Getting the precision of gradient boosting
    gbm_score = get_precision_gbm(gbm_parameters, scaled_external_params_train,
scaled_external_params_test, external_classes_train, external_classes_test,
n_internal_folds)
    # 0.839945990058

    # Stacking the precision
    knn_precision = knn_precision + knn_score
    svm_precision = svm_precision + svm_score
    neural_net_precision = neural_net_precision + neural_net_score
    random_forest_precision = random_forest_precision + random_forest_score
    gbm_precision = gbm_precision + gbm_score
```

# Resultados

```
knn_precision = knn_precision/n_external_folds
svm_precision = svm_precision/n_external_folds
neural_net_precision = neural_net_precision/n_external_folds
random_forest_precision = random_forest_precision/n_external_folds
gbm_precision = gbm_precision/n_external_folds

print('Accuracy kNN: ', knn_precision)
print('Accuracy SVM: ', svm_precision)
print('Accuracy Neural Networks: ', neural_net_precision)
print('Accuracy Random Forest: ', random_forest_precision)
print('Accuracy Gradient Boosting: ', gbm_precision)
```

```
Accuracy kNN:  0.929769178069
Accuracy SVM:  0.93359083412
Accuracy Neural Networks:  0.847556729745
Accuracy Random Forest:  0.931680006094
Accuracy Gradient Boosting:  0.842485594827
```

# Conclusões

O melhor classificador para o conjunto de dados SECOM é o SVM com uma acurácia de **93.359%**.