Exercicio 3

Everton Schumacker Soares, 116724

October 23, 2016

1 Pré-processamento dos dados

A leitura de input é feita de acordo com o código abaixo. A função read_input() também faz o pré-processamento para que os dados possuam média 0 e desvio padão 1. **Como evidenciado pela saída, algumas colunas possuem desvio padrão 0, pois, como os dados da coluna estão muito próximos(quase constantes) após a injeção da média. Isso impossibilita a divisão por 0, necessário no algorítmo de scale(pré-processamento), o que torna impossível tornar o devio igual a 1.**

In [12]:
```
import numpy as np
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold

# Reads and preprocess the input
def read_input():
    f = open('secom.data', 'r')
    dataset = []

    for line in f:
        row = [float(x) for x in line.strip().split(' ')]
        dataset.append(row)

    # Replacing NaN with the column's mean
    means = np.nanmean(dataset, axis=0)
    dataset = np.array(dataset)
    for row in dataset:
        nan_index = np.isnan(row)
        row[nan_index] = means[nan_index]
```

```
        # Scaling dataset so the column has mean = 0 e std = 1
        dataset = preprocessing.scale(dataset, axis=0)

        # Reads labels
        f = open('secom_labels.data', 'r')
        labels = np.array([float(line.split(' ')[0]) for line in f])

        return (dataset, labels)


    # Main
    np.set_printoptions(precision=4, suppress=True)
    (dataset, labels) = read_input()
    print("Mean of the columns:")
    print(np.mean(dataset, axis=0))
    print("\nStandart Deviation of the columns:")
    print(np.std(dataset, axis=0))
```

```
Mean of the columns:
[ 0. -0.  0.  0. -0.  0. -0.  0. -0.  0. -0. -0. -0.  0.  0.  0.  0.  0.
  0. -0. -0.  0. -0. -0.  0.  0.  0.  0.  0. -0.  0. -0.  0.  0. -0. -0.
  0. -0.  0. -0.  0. -0.  0.  0. -0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 -0.  0.  0. -0.  0.  0.  0.  0.  0. -0.  0. -0.  0.  0. -0.  0. -0. -0.
  0.  0. -0.  0.  0. -0.  0. -0. -0. -0.  0.  0.  0.  0. -0. -0.  0. -0.
  0.  0. -0. -0.  0. -0. -0.  0.  0. -0.  0.  0. -0.  0.  0. -0.  0.  0.
 -0. -0. -0. -0.  0.  0.  0. -0. -0. -0.  0.  0.  0.  0. -0.  0.  0.  0.
 -0. -0. -0.  0.  0.  0. -0. -0. -0.  0. -0.  0. -0.  0. -0.  0.  0.  0.
  0. -0.  0. -0.  0.  0. -0.  0. -0. -0.  0. -0.  0.  0. -0. -0.  0.  0.
  0.  0.  0. -0. -0. -0.  0. -0.  0.  0. -0.  0. -0. -0.  0.  0.  0.  0.
 -0. -0. -0. -0. -0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -0. -0.
  0.  0.  0.  0. -0. -0.  0. -0. -0.  0. -0. -0. -0. -0.  0.  0.  0.  0.
  0.  0.  0. -0. -0.  0.  0. -0. -0. -0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -0. -0. -0. -0.  0.  0.  0. -0.
  0.  0.  0.  0. -0.  0. -0.  0.  0. -0. -0.  0.  0. -0. -0.  0. -0. -0.
 -0.  0. -0. -0. -0. -0.  0. -0. -0. -0. -0. -0.  0. -0.  0.  0.  0.  0.
  0.  0. -0.  0.  0. -0.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0.  0.  0.
  0. -0. -0.  0.  0.  0.  0.  0.  0.  0. -0. -0.  0.  0.  0.  0.  0.  0.
 -0.  0. -0.  0.  0. -0. -0. -0. -0. -0.  0. -0.  0. -0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
```

```
 -0.  0. -0. -0. -0. -0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -0.
 -0.  0.  0.  0.  0.  0.  0.  0. -0.  0.  0.  0. -0. -0. -0. -0. -0.  0.
  0.  0. -0. -0.  0.  0.  0. -0. -0. -0.  0.  0.  0. -0.  0. -0.  0.  0.
 -0. -0.  0.  0. -0. -0. -0. -0.  0. -0.  0.  0.  0.  0.]

Standart Deviation of the columns:
[ 1.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  0.  1.  1.  0.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.
  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  0.
  1.  1.  1.  1.  1.  1.  0.  1.  1.  0.  0.  0.  0.  0.  0.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  1.  1.  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  0.  0.  0.  1.  1.  1.  1.  1.  1.  0.  1.
  1.  0.  0.  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  0.  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  1.  1.
  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  0.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.
  0.  0.  1.  1.  1.  1.  1.  1.  0.  1.  1.  0.  0.  0.  0.  0.  0.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  0.  1.  1.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  1.  1.  0.  0.  0.  0.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

2 Cross-Validation

No código abaixo, cv_classification() é a função genérica que utiliza uma 5-fold cross validação estratificada externa para medir a acurácia de um classificador, após descobrir os hiperparâmetros de cada fold através do uso de uma 3-fold cross-validação interna.

```
In [15]: def cv_clasification(dataset, labels, params, est):
             five_fold = StratifiedKFold(n_splits=5)    # Creates a external 5-fold
             best_params = []
             score = []

             # For each external test fold searches the hiperparameters
             # through 3-fold cross validation
             for (trainset, testset) in five_fold.split(dataset, labels):
                 # 3-fold cross validation
                 three_fold_cv = GridSearchCV(estimator=est, param_grid=params, n_
                 three_fold_cv.fit(dataset[trainset], labels[trainset])
                 best_params.append(three_fold_cv.best_params_)
                 # Accuracy of the i-th external test fold
                 score.append(three_fold_cv.score(dataset[testset], labels[testset]

             # Transforms a list of dictionaries into a single dictionary where
             # each key is a searched parameter and the value is a list of the
             # best parameters found
             bp = {}
             for d in best_params:
                 for key in d:
                     try:
                         bp[key].append(d[key])
                     except KeyError:
                         bp[key] = [d[key]]

             # returns used parameters and accuracy
             return (bp, np.mean(score))
```

3 kNN com PCA

Para o classificador k-Neirest Neighbors, foi utilizado o dataset transformado com PCA (com 80% da variância), como indicado abaixo. Os melhores valores de K (para cada um dos 5 folds) encontrados estão indicados abaixo junto à acurácia obtida pelo classificador.

```
In [19]: def kNN(dataset, labels):
             print("kNN Classifier")

             # Apply PCA with 80% of variance in dataset
             pca = PCA(n_components=0.8)
             pca.fit(dataset)
             dataset_pca = pca.transform(dataset)
             print("Number of components = {}".format(len(pca.explained_variance_ra
             # kNN Classification
             knn = KNeighborsClassifier()
             param_grid = {'n_neighbors':[1, 5, 11, 15, 21, 25]}
             (best_params, score) = cv_clasification(dataset_pca, labels, param_gri
```

```
            print("Best values found for K: {}".format(best_params['n_neighbors']))
            print('kNN accuracy = {:.4f}\n'.format(score))
            return

        # Classification
        kNN(dataset, labels)
```

```
kNN Classifier
Number of components = 90
Best values found for K: [15, 11, 15, 25, 25]
kNN accuracy = 0.9298
```

4 SVM RBF

Para o classificador SVM com kernel de 'rbf', os reusultados de C e gamma para cada um dos 5 folds, bem como a accurácia média da classficação, estão reportados abaixo:

```
In [22]: def SVM(dataset, labels):
            print("SVM classifier")

            svm_rbf = svm.SVC()
            param_grid = {
                'kernel':['rbf'],
                'C':[pow(2,-5), pow(2,0), pow(2,5), pow(2,10)],
                'gamma':[pow(2,-15), pow(2,-10), pow(2,-5), pow(2,0), pow(2,5)]
            }

            (best_params, score) = cv_clasification(dataset, labels, param_grid, s

            print("Best values found for C: {}".format(best_params['C']))
            print("Best values found for gamma: {}".format(best_params['gamma']))
            print("SVM accuracy: {:.4f}\n".format(score))
            return

        # Classification
        SVM(dataset, labels)
```

```
SVM classifier
Best values found for C: [0.03125, 0.03125, 0.03125, 0.03125, 0.03125]
Best values found for gamma: [3.0517578125e-05, 3.0517578125e-05, 3.0517578125e-05,
SVM accuracy: 0.9336
```

5 Rede Neural

Para uma neural network, foram investigados valores de 10, 20, 30 e 40 neurônios na hidden layer. Os valores de hiperparâmetro selecionados pela 3-fold cross-validação para cada um dos 5 folds externos está indicado abaixo. Bem como a acurácia média da rede neural.

```
In [28]: def neural_network(dataset, labels):
             print("Neural Network classifier")

             nn = MLPClassifier()
             param_grid = {'hidden_layer_sizes':[10, 20, 30, 40]}

             (best_params, score) = cv_clasification(dataset, labels, param_grid, r

             print("Best numbers of neurons: {}".format(best_params['hidden_layer_s
             print("Neural Network accuracy: {:.4f}\n".format(score))
             return

         # Classification
         neural_network(dataset, labels)
```

```
Neural Network classifier
Best numbers of neurons: [30, 30, 40, 40, 40]
Neural Network accuracy: 0.8010
```

6 Random Forest

Para a Random Forest, foram investigados valores de n_featrues = 10, 15, 20, 25 e ntrees = 100, 200, 300, 400. Os melhores hiperparâmetros escolhidos pela cross-validação interna estão indicadas abaixo, assim como a acurácia média obtida pela cross-validação externa.

```
In [31]: def random_forest(dataset, labels):
             print("Random Forest classifier")

             rf = RandomForestClassifier()
             param_grid = {
                 'max_features':[10, 15, 20, 25],     # number of features
                 'n_estimators':[100, 200, 300, 400] # number of trees
             }

             (best_params, score) = cv_clasification(dataset, labels, param_grid, r

             print("Best values for n_features: {}".format(best_params['max_feature
             print("Best values for ntrees: {}".format(best_params['n_estimators'])
             print("Random Forest accuracy: {:.4f}\n".format(score))
             return
```

```
        # Classification
        random_forest(dataset, labels)
```

```
Random Forest classifier
Best values for n_features: [10, 10, 10, 10, 10]
Best values for ntrees: [100, 100, 400, 100, 400]
Random Forest accuracy: 0.9330
```

7 Gradient Boosting Machine

Para o GBM, foram investigados valores de hiperparâmetro: numero de arvores = 30, 70, e 100; learning rate de 0.1 e 0.05 e profundidade da árvore=5. Os melhores valores de hiperparâmetro, bem como a acurácia média, estão indicados abaixo:

```
In [33]: def GBM(dataset, labels):
             print("GBM classifier")
             cbm = GradientBoostingClassifier()
             param_grid = {
                 'n_estimators':[30, 70, 100], # number of trees
                 'learning_rate':[0.1, 0.05],
                 'max_depth':[5]               # depth of the trees
             }

             (best_params, score) = cv_clasification(dataset, labels, param_grid, c

             print("Best numbers of trees: {}".format(best_params['n_estimators']))
             print("Best learning rates: {}".format(best_params['learning_rate']))
             print("GBM accuracy: {:.4f}\n".format(score))
             return

         # Classification
         GBM(dataset, labels)
```

```
GBM classifier
Best numbers of trees: [100, 70, 70, 30, 70]
Best learning rates: [0.05, 0.1, 0.05, 0.05, 0.05]
GBM accuracy: 0.8413
```

8 Conclusão

As acurácias obtidas foram: 92.98% para o kNN, 93.36% para o SVM, 80.10% para a Neural Network, 93.30% para a Random Forest e 84.13% para o GBM. Os melhores classificadores foram

o SVM (Support Vector Machine) e o Random Forest, praticamente empatas (uma vez que uma diferença de 0.06% não possui grande relevância estatistica). O kNN também pode ser comsiderado como estando empatado com os dois melhores classificadores, uma vez que uma diferença de pouco mais de 30% não possui muita relevância estatísca. O bom resultado obtido pelo kNN pode ter tido grande contribuição do uso de PCA, uma vez que muitas colunas possuiam desvio padrão 0, não sendo, portanto, dimensões relevantes para o classificador. Em seguida, o segundo pior resultado é o do GBM, tendo uma acurácia aproximadamente 9% menor que as obtidas pelo kNN, SVM e Random Forest. A pior taxa de acerto foi a apresentada pela Rede Neural, com apenas 80.10% de acerto (mais de 10% menor que a dos três melhores classificadores e 4% menor que a do GBM). O desempenho relativamente baixo da Neural Network pode ter sido influenciado pela configuração da rede, pois a precisão di algorítimo é altamente dependente da topologial da rede. Sendo assim, a exploração de outros hiperparâmetro (como número de hidden layers, critério de convergência e número de neurônio por layer) talvez possa gerar uma acurácia melhor.