Estimação, Deteção e Análise II

02 – Classificação Avançada

Redes neuronais

Support Vector Machines

Vizinho mais próximo

Neural networks

1. Instalar o package keras

2. Definir as libraries a importar

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
```

3. Ler o ficheiro pima-indians-diabetes.data.csv que se encontra no Moodle e dividir em variáveis independentes (X) e dependente (Y)

```
dataset = loadtxt('caminho_fich/pima-indians-diabetes.data.csv', delimiter=',')
X = dataset[:,0:8]
y = dataset[:,8]
```

4. Definir o modelo:

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

5. Compilar o modelo

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

6. Fazer fit ao modelo

```
model.fit(X, y, epochs=150, batch_size=10)
```

7. Avaliar o modelo:

```
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy * 100))
```

8. Calcular as previsões, arredondar (para dar 0 ou 1) e visualizar

```
predictions = model.predict (X)
rounded = [round(x[0]) for x in predictions]
print(predictions)
```

9. Visualizar o modelo (texto)

```
print(model.summary())
```

10. Visualizar o modelo (gráfico 'model_plot.png' guardado no projeto)

```
11. plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

12. Correr novamente algumas vezes. O que acontece à accuracy? Porquê?

13. Alterar os parâmetros do modelo para verificar o efeito em termos de accuracy

Support Vector Machines

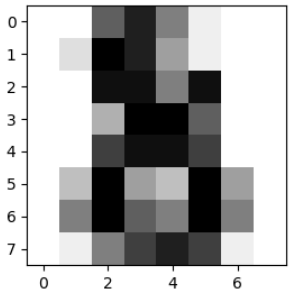
1. Definir as libraries a importar

```
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn import datasets
```

2. Carregar o dataset digits

```
digits = datasets.load_digits()
```

Este dataset[1] tem o objetivo de identificar algarismos manuscritos, em cada dígito é representado por uma imagem como, por exemplo:



3. De forma a utilizar este dataset, é necessário "achatá-lo". Ao mesmo tempo, separa-se em *features* (X) e target (y):

```
n_samples = len(digits.images)
X = digits.images.reshape((n_samples, -1))
y = digits.target
```

4. Uma vez que as SVM requerem diversos parâmetros, vamos utilizar uma grid search para escolher a melhor combinação de parâmetros para este dataset em particular. Para isso, precisamos de separar o dataset (features e target) em treino (80% dos dados) e teste (20%):

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.2,
train_size=0.8)
```

5. De seguida definimos os parâmetros a que queremos fazer *tunning* e as métricas a utilizar no processo:

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

scores = ['precision', 'recall']
```

6. Por fim, executamos a *grid search* com *cross validation*:

```
for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(
        SVC(), tuned_parameters, scoring='%s_macro' % score
    )
    clf.fit(X_train, y_train)

    print("Best parameters set found on development set:")
```

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.6299&rep=rep1&type=pdf

```
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
```

7. O output será:

```
# Tuning hyper-parameters for precision

Best parameters set found on development set:

{'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on development set:

0.987 (+/-0.008) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.967 (+/-0.015) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.985 (+/-0.011) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.984 (+/-0.010) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.985 (+/-0.011) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.983 (+/-0.011) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.985 (+/-0.011) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.983 (+/-0.011) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.973 (+/-0.019) for {'C': 1, 'kernel': 'linear'}
0.973 (+/-0.019) for {'C': 10, 'kernel': 'linear'}
0.973 (+/-0.019) for {'C': 100, 'kernel': 'linear'}
0.973 (+/-0.019) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        34
           1       1.00      1.00      1.00        45
           2       1.00      1.00      1.00        37
           3       1.00      1.00      1.00        37
           4       1.00      1.00      1.00        32
           5       1.00      1.00      1.00        39
           6       1.00      1.00      1.00        43
           7       1.00      1.00      1.00        30
           8       1.00      1.00      1.00        33
           9       1.00      1.00      1.00        30

    accuracy                           1.00       360
   macro avg       1.00      1.00      1.00       360
weighted avg       1.00      1.00      1.00       360


# Tuning hyper-parameters for recall

Best parameters set found on development set:
```

```
{'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on development set:

0.987 (+/-0.008) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.965 (+/-0.016) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.984 (+/-0.011) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.984 (+/-0.010) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.984 (+/-0.011) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.983 (+/-0.012) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.984 (+/-0.011) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.983 (+/-0.012) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.973 (+/-0.019) for {'C': 1, 'kernel': 'linear'}
0.973 (+/-0.019) for {'C': 10, 'kernel': 'linear'}
0.973 (+/-0.019) for {'C': 100, 'kernel': 'linear'}
0.973 (+/-0.019) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        34
           1       1.00      1.00      1.00        45
           2       1.00      1.00      1.00        37
           3       1.00      1.00      1.00        37
           4       1.00      1.00      1.00        32
           5       1.00      1.00      1.00        39
           6       1.00      1.00      1.00        43
           7       1.00      1.00      1.00        30
           8       1.00      1.00      1.00        33
           9       1.00      1.00      1.00        30

    accuracy                           1.00       360
   macro avg       1.00      1.00      1.00       360
weighted avg       1.00      1.00      1.00       360
```

Mais exemplos de utilização de SVM: https://towardsdatascience.com/the-complete-guide-to-support-vector-machine-svm-f1a820d8af0b

K Nearest Neighbors

1. Definir as libraries a importar

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Carregar o dataset

```
df = pd.read_csv('caminho_fich/titanic_train.csv', sep=";")
```

3. Definir as variáveis independentes (X) e dependente (y)

```
X = df.drop('Survived', axis=1)
y = df.Survived
```

4. Dividir X e y em treino e teste

```
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=42)
```

5. Definir que colunas devem ser consideradas numéricas e categóricas

```
numerical = ['Age', "SibSp", "Parch", 'Fare']
categorical = ["Pclass", "Sex"]
```

6. Definir scaling

```
#defining train and test index variables for casting the scaled numerical values in a
dataframe
X_train_index = X_train.index
X_test_index = X_test.index

#instantiating OneHotEncoder and defining the train and test features to be encoded
ohe = OneHotEncoder()
X_train_ohe = X_train[categorical]
X_test_ohe = X_test[categorical]

#fitting the encoder to the train set and transforming both the train and test set
X_train_encoded = ohe.fit_transform(X_train_ohe)
X_test_encoded = ohe.transform(X_test_ohe)

#instantiating StandardScaler and defining continous variables to be scaled
ss = StandardScaler()
X_train_cont = X_train[numerical].astype(float)
X_test_cont = X_test[numerical].astype(float)

#scaling the continuous features and casting results as dataframes
X_train_scaled = pd.DataFrame(ss.fit_transform(X_train_cont),
columns=X_train_cont.columns, index=X_train_index)
X_test_scaled = pd.DataFrame(ss.transform(X_test_cont), columns=X_test_cont.columns,
index=X_test_index)

#defining the columns for the train and test splits
train_columns = ohe.get_feature_names(input_features=X_train_ohe.columns)
test_columns = ohe.get_feature_names(input_features=X_test_ohe.columns)

#casting the encoded X_train and X_test as dataframes
X_train_processed = pd.DataFrame(X_train_encoded.todense(), columns=train_columns,
index=X_train_index)
X_test_processed = pd.DataFrame(X_test_encoded.todense(), columns=test_columns,
index=X_test_index)

#combining the encoded and scaled dataframes for a preprocessed
#X_train and X_test
X_train = pd.concat([X_train_scaled, X_train_processed], axis=1)
X_test = pd.concat([X_test_scaled, X_test_processed], axis=1)
```

7. Definir uma função para avaliação

```
def evaluation(k,d,type,y, y_hat):
    precision = round(precision_score(y, y_hat)*100,1)
    recall = round(recall_score(y, y_hat)*100,1)
    accuracy = round(accuracy_score(y, y_hat)*100,1)
    f1 = round(f1_score(y, y_hat)*100,1)
    return [k,d,type,recall,accuracy,precision,f1]
```

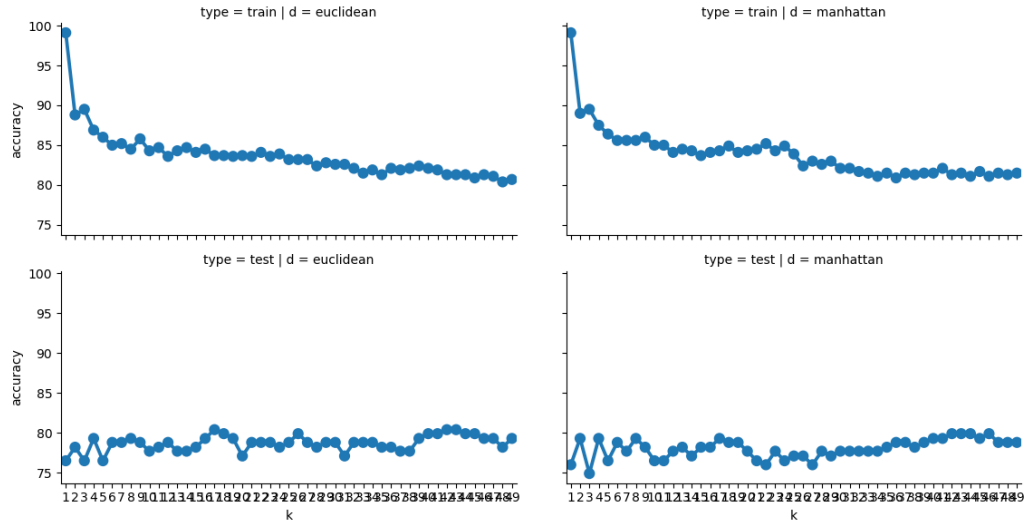
8. Instanciar e correr o Knn com todos os k ϵ [1,50] e utilizando distância euclidiana e de manhattan

```
possible_ks = range(1, 50)
possible_distances = ['euclidean', 'manhattan']
results = []
for d in possible_distances:
    for k in possible_ks:
        knn = KNeighborsClassifier(n_neighbors=k, metric=d)
        knn.fit(X_train, y_train)
        y_hat_train = knn.predict(X_train)
        y_hat_test = knn.predict(X_test)
        results.append(evaluation(k,d,"train",y_train, y_hat_train))
        results.append(evaluation(k, d, "test", y_test, y_hat_test))
```

9. Visualizar graficamente os resultados em termos accuracy (treino e teste)

```
results                                =                                pd.DataFrame(results,
columns=["k","d","type","recall","accuracy","precision","f1"])
g = sns.FacetGrid(results, col="d", row="type")
g.map(sns.pointplot, "k", "accuracy")
plt.show()
```

O resultado deverá ser:



10. Qual é o k que permite uma melhor performance? Podemos confirmar:

```
res_test = results[(results['type']=="test")]
print(res_test.loc[[res_test["accuracy"].idxmax()]])
```

Resultado:

```
     k          d  type  recall  accuracy  precision    f1
33  17  euclidean  test    70.8      80.4       78.5  74.5
```

11. Podemos agora criar o "melhor" modelo para depois poder aplicar aos dados de teste. Num novo ficheiro, repetimos o processo, mas desta vez definindo o k como 17 e a distância euclidiana. Podemos, então, calcular as previsões para cada um dos exemplos do conjunto de teste.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.neighbors import KNeighborsClassifier

df_train = pd.read_csv('caminho_fich/titanic_train.csv', sep=";")
df_train = df_train[['Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex']]

X_train = df_train.drop('Survived', axis=1)
y_train = df_train.Survived

df_test = pd.read_csv('C:/Users/Catarina/Dropbox/Aulas/UPT/2020_2021/2_semestre/EDA2/02-
ClassificacaoAvancada/titanic_test.csv', sep=";")
df_test = df_test[['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex']]

X_test = df_test

#importing scaler and encoder

#defining numerical and categorical features in dataframe
numerical = ['Age', "SibSp", "Parch", 'Fare']
categorical = ["Pclass", "Sex"]

#defining train and test index variables for casting the scaled
```

```
#numerical values in a dataframe
X_train_index = X_train.index
X_test_index = X_test.index

#instantiating OneHotEncoder and defining the train and test
#features to be encoded
ohe = OneHotEncoder()
X_train_ohe = X_train[categorical]
X_test_ohe = X_test[categorical]

#fitting the encoder to the train set and transforming both
#the train and test set
X_train_encoded = ohe.fit_transform(X_train_ohe)
X_test_encoded = ohe.transform(X_test_ohe)

#instantiating StandardScaler and defining continous variables
#to be scaled
ss = StandardScaler()
X_train_cont = X_train[numerical].astype(float)
X_test_cont = X_test[numerical].astype(float)

#scaling the continuous features and casting results as dataframes
X_train_scaled               =               pd.DataFrame(ss.fit_transform(X_train_cont),
columns=X_train_cont.columns,
                              index=X_train_index)
X_test_scaled = pd.DataFrame(ss.transform(X_test_cont), columns=X_test_cont.columns,
                             index=X_test_index)

#defining the columns for the train and test splits
train_columns = ohe.get_feature_names(input_features=X_train_ohe.columns)
test_columns = ohe.get_feature_names(input_features=X_test_ohe.columns)

#casting the encoded X_train and X_test as dataframes
X_train_processed   =   pd.DataFrame(X_train_encoded.todense(),   columns=train_columns,
index=X_train_index)
X_test_processed    =    pd.DataFrame(X_test_encoded.todense(),    columns=test_columns,
index=X_test_index)

#combining the encoded and scaled dataframes for a preprocessed
#X_train and X_test
X_train = pd.concat([X_train_scaled, X_train_processed], axis=1)
X_test = pd.concat([X_test_scaled, X_test_processed], axis=1)

# instantiating and fitting K Neighbors Classifier
knn = KNeighborsClassifier(n_neighbors=17, metric='euclidean')
knn.fit(X_train, y_train)
# predictions =
print(knn.predict(X_test))
```

Resultado:

```
[0 0 0 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 1 0 0 0 1
 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1
 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1
 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 0
 1 0 0 0 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0
 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 1
 1 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0
 1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 1 1 0 0 1 0 0 1
 1 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 1 0 1 1 0 0 0 1 1 1 1 1 0]
```