Statistics and Machine Learning II

Classification

Coursework: Supervised and Unsupervised classification

Luis Da Silva

February 28, 2019

1 Introduction

Long study hours to get specific field knowledge and lots of experience allows doctors to perform highly complex pattern recognition procedures to determine whether or not a patient may be sick. Nevertheless, the mind may be tricked by some kind of cognitive bias[1] which may lead to a wrong conclusion being drawn on the condition of a person. Clearer, higher quality and opportune analysis are therefore necessary to support medical professionals on their decision-making process.

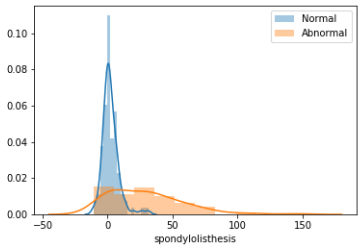
On this coursework, I am going to explore the performance of different supervised and unsupervised learning classification techniques to assess this problem on a biomedical dataset which contains measures commonly used to classify orthopaedic patients into normal or abnormal[1].

2 EDA and preprocessing

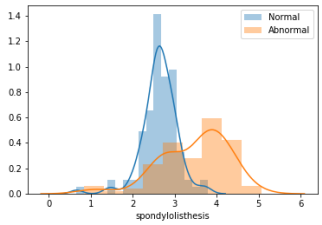
The dataset contains 6 biomechanical features related to 310 patient's pelvis and lumbar spine, 210 of which are classified as "abnormal" and 100 as

[1]Information about this dataset is available in https://archive.ics.uci.edu/ml/datasets/Vertebral+Column

Figure 1: Spondylolisthesis distribution across normal or abnormal



(a) Original data



(b) log(Original Data + 13)

"normal". To avoid biases induced due an unbalanced the dataset, I am resampling it into a 200 rows dataset by choosing all the normal patients and a random sample of 100 abnormal patients.

On the new undersampled dataset, every feature seems to follow a normal-like distribution, with the exception of "spondylolisthesis", shown on figure 1a, which seem to follow a much more log-normal distribution (see figure 1b).

Finally, as some clustering algorithms that I am going to use are based on distance metrics, it is important that every feature has the same scale so they do not get unfair influence. For this, I am using a standard Gaussian scaler, given by:

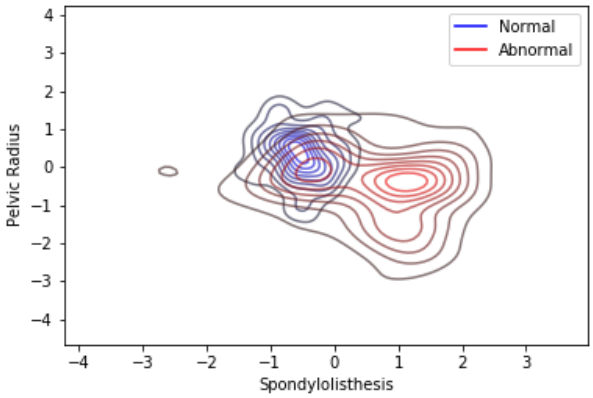
$$\hat{x} = \frac{x - \bar{x}}{std(x)}$$

3 Supervised Learning Algorithms

The fact that this dataset has been already labelled by Dr Henrique da Mota allows me to apply supervised learning algorithms (which are techniques to recognize patterns across groups in data by using information of pre-existing labels) and to test their performance. Using labels means that I already know which are the clusters in the data, thus these are complete (all possible groups are present), which is also the case with this biomedical data.

There is another advantage in having group labels: one may perform model tuning. Not every feature has to be important in predicting an out-

Figure 2: KDE on Pelvic Radius Vs Spondylolisthesis



come and thus including all of them may turn into an over-complex model that doesn't perform as well as it could. To manage this risk, each of the considered supervised learning models below is tuned by performing "best subset 5 fold cross validation selection" and scoring via its mean accuracy. Also, there are hyper-parameters in each model which change its behaviour; these are tuned as well by performing a cross-validation grid search on the best subset model[2].

To better understand the behaviour of each algorithm and allow easy visualization, I will also fit each of the models on only "Pelvic Radius" and "Spondylolisthesis", which is the combination of two features that allows better class separation[3] (see figure 2).

Regarding the metrics to be used to assess each algorithms success, I am going to compare them according to:

- $Accuracy = \dfrac{tp + tn}{tp + fp + tn + fn}$. (ratio of correctly predicted samples).

[2]This procedure is not guaranteed to output the best possible model. To do so, one should test every combination of hyper-parameters in each subset and cross-validate on them with a leave-one-out procedure, but that would be too computationally intensive and there's not much to gain from it.

[3]An alternative would be to use a dimensionality reduction technique like Principal Component Analysis, but they all seem to converge in that I will be loosing too much information unless I allow 4 components to be left, which is not good for visualization.

- $Precision = \dfrac{tp}{tp + fp}$ (ratio of positive predictions that are in fact positive)
- $Recall = \dfrac{tp}{tp + fn}$ (ratio of positive samples classified as so).
- $F1 = 2 * \dfrac{Precision * Recall}{Precision + Recall}$ (harmonic average of precision and recall).

Where tp is a true positive, tn is a true negative, fp is a false positive, fn is a false negative, a positive is the case in which a lumbar spine is normal and a negative is when it's abnormal.

3.1 Logistic Regression

Logistic Regression may stand out as the classical supervised approach used to classify data. It has proven to be a simple but powerful technique and thus will be useful to create a threshold to help me decide whether another classifier is performing well on this dataset or not.

As a remainder from Coursework 3 on last semester, logistic regression makes use of the logit function to linearly estimate the logarithmic odds on an event:

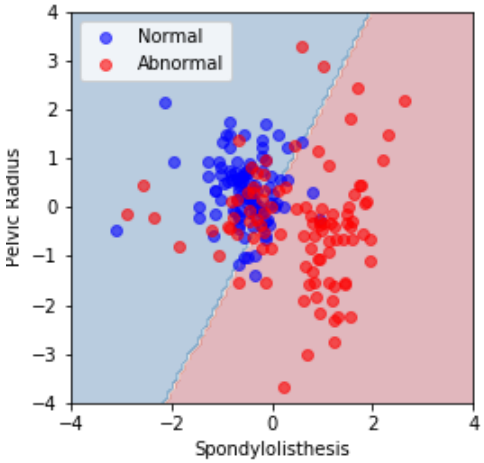
$$logit(\rho) = \log\left(\frac{\rho}{1-\rho}\right) \quad for\, 0 \leq \rho \leq 1$$

$$logit(\rho) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$$

While logistic regression is not directly a method to classify events, it does estimate the probability of an observation belonging to a class and thus, by setting a threshold (usually at 50%), one may perform classification with it.

After optimizing it for this dataset, the best subset is selected to be: 'pelvic incidence', 'sacral slope', 'pelvic radius' and 'spondylolisthesis' and it allows the model to obtain a cross-validated mean accuracy score of 79%. Its decision boundary is shown in figure 3. A nice straight line is drawn which makes it easy to interpret.

Figure 3: Logistic Regression in 2D



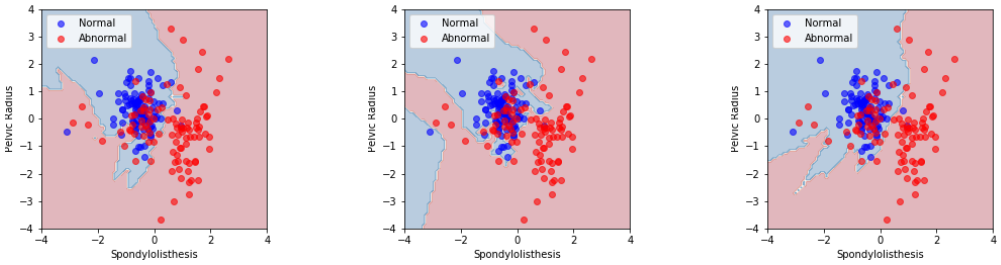
3.2 K-Nearest Neighbours

Instead of trying to find the underlying mathematical generative model of a dataset, K-Nearest Neighbours (KNN) is memory-based, which means that it finds a pre-determined number (k) of training points which are closest to the target point and then uses the majority vote to classify them [3]. If k is even, then ties are broken at random.

As k is chosen by hand, this is going to be a very important hyper-parameter to optimize. Usually, low values of k leads to over-fitting, while high values will result in under-fitting. Also, as this algorithm relays on a distance metric, results may also be influenced by it. Some of these metrics are:

- Euclidean: $\sqrt{\sum(x-y)^2}$
- Manhattan: $\sum(\mid x-y \mid)$
- Minkowski: $\sqrt[p]{\sum(x-y)^p}$
- Chebyshev: $max(\mid x-y \mid)$

Figure 4: K-Nearest Neighbours 2D results



(a) Selected fit, $k = 5$ (b) Over-fitted, $k = 1$ (c) Under-fitted,$k = 19$

After optimizing KNN, Minkowsky distance with k = 5 and three features are selected: 'lumbar lordosis angle', 'pelvic radius' and 'spondylolisthesis'. This leads to a cross-validated accuracy score of 82% (higher than Logistic Regression). For its 2D representation see figure 4a. We may also see the effects of different k's on the performance of the algorithm: figure 4b shows the decision boundaries for an over-fitted KNN with $k = 1$, while figure 4c does so for an under-fitted KNN with $k = 19$. The main takeaway from these plots is how an incorrect value for k makes the region for a normal lumbar spine wider.

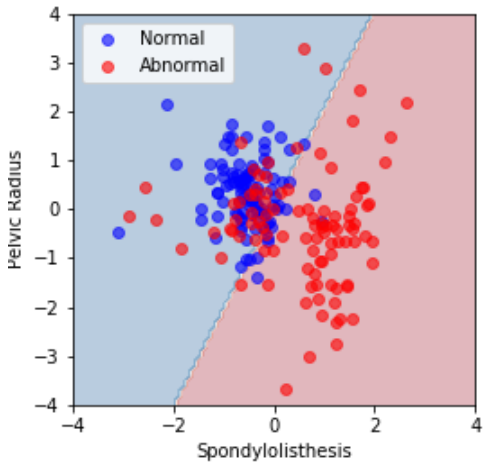
3.3 Linear Discriminant Analysis

By assuming that our explanatory features are composed by Gaussian densities and each of the classes have a common covariance matrix, we may obtain best possible class separation intersecting their densities enclosing 95% of the data in each case. This is an informal explanation of the process a Linear Discriminant Analysis (LDA) model does[4].

After looking at figure 2, it is easy to see that this dataset doesn't have common covariance matrices and thus this method is not going to be optimal. Nevertheless, I still fitted it for the sake of comparison. Best subset selected is comprised by three variables: 'pelvic tilt', 'pelvic radius' and 'spondylolisthesis' and its cross-validated accuracy score is 80%, which is still better than

[4]For a mathematical demonstration on LDA see Trevor Hastie, Robert Tibshirani & Jerome Friedman. The Elements of Statistical Learning. 106-112.

Figure 5: Linear Discriminant Analysis in 2D



Logistic Regression. Figure 5 shows the classification boundaries drawn on a 2D setting.

3.4 Support Vector Classifier

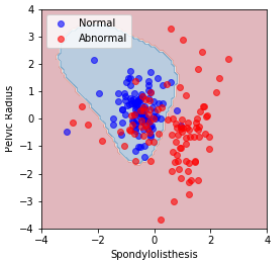
Support Vector Machines and specifically a Support Vector Classifier (SVC) maps points in space by using a kernel function while trying to separate categories by maximizing the gap between them. They do so by drawing hyper-planes with support bands; the distances from band to band is called margin (M) and represents the maximization problem.

When classes do not overlap, it is possible to find a hyperplane that creates the biggest margin between the training points for two classes. Otherwise, the margin maximization problem is still applicable if some points are allowed to fall on the wrong side of the margin by some K proportion[3].

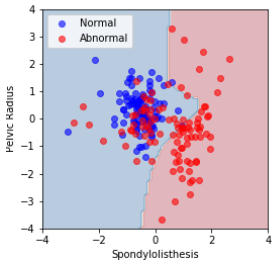
Some of the kernels commonly used to map features into higher dimensions to allow non-linear classification are[5]

- Linear: given by $k(x, y) = x^T y + c$. This is equivalent to using no kernel.

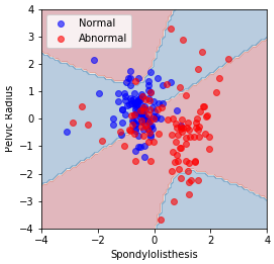
Figure 6: Support Vector Classifier in 2D



(a) Selected fit, RBF kernel



(b) Polynomial Kernel



(c) Sigmoid Kernel

- Polynomial: given by $k(x, y) = (\alpha x^T y + c)^d$. Non-stationary kernel useful for problems in which data is normalized.
- Hyperbolic Tangent: given by $k(x, y) = tanh(\alpha x^T y + c)$. Also known as the Sigmoid Kernel.
- Radial Basis Function (RBF): given by $k(x, y) = exp(-\gamma \mid\mid x - y \mid\mid^2$. This is actually a family of kernels, from which we have the Gaussian Kernel in which $\gamma = \frac{1}{2\sigma^2}$

After optimizing SVC to the lumbar spine data, the best subset is found to be the same as KNN: 'lumbar lordosis angle', 'pelvic radius' and 'spondylolisthesis' and RBF seems to be the best performing kernel with an accuracy score of 84%. Figure 6a shows the 2D version of the chosen kernel, it shows a nice circumference around the normal lumbar spines which seems to make sense with what one should expect from body measurement data (a "normal" lumbar spine cannot have neither a very low nor a very high measure in any sense, thus a circle-like shape is necessary). For comparison purposes, figure 6b shows the same SVC fit but using a Polynomial kernel and figure 6c does the same for a Hyperbolic Tangent kernel.

3.5 Random Forest

Decision trees are a low-bias high-variance procedure that benefits from bagging (bootstrap aggregation) because it helps to reduce their variance. Ran-

dom forest builds a large collection of de-correlated trees just to average them. This makes of Random Forest a highly computational intensive technique. The basic algorithm runs as follows:

1. Get a bootstrap sample from training data.
2. Select m variables, pick the best a split among it.
3. Split the node into two nodes.
4. Repeat steps 2 and three until the size of the minimum node is reached.
5. Repeat steps 1-4 to build B trees.

Then, to classify, each new sample will be classified through the trees and its class will be selected by majority vote.

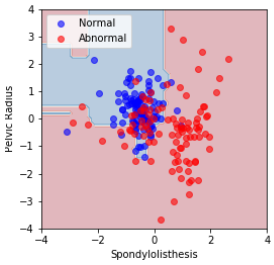
While training a random forest the maximum depth it is allowed to reach will determine its likelihood to succeed. Although there are no specific values for this hyper-parameter, a low value usually leads to under-fit and a high value will lead to over-fit.

After tunning, final model subset is: 'pelvic tilt', 'sacral slope', 'pelvic radius' and 'spondylolisthesis', with a maximum depth of 4. An accuracy score of 79% is achieved by cross-validation.

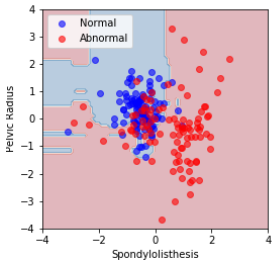
To better understand how it classifies samples, figure 7a presents the 2D map for this algorithm. It is easy to see how node splitting into leaves results in square-like zones. Figure 7b shows the same classifier but with no limit on its depth, so it get over-fitted and we see it by those tiny blue regions mixed up in the abnormal zone. Similarly, figure 7c shown an under-fitted random forest which fails to recognize extreme upper left corner values as abnormal.

A final important point to talk about this classifier is that it takes considerably more computational resources than the others: while Logistic Regression and KNN took 1.6 and 1 seconds, respectively, to be optimized in my laptop, Random Forests took 26 seconds.

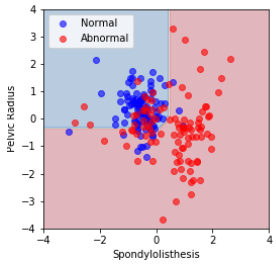
Figure 7: Random Forests Classifier in 2D



(a) Selected fit, max depth is 4



(b) Overfitted, there is no max depth



(c) Underfitted, max depth is 1

3.6 Gradient Boosting

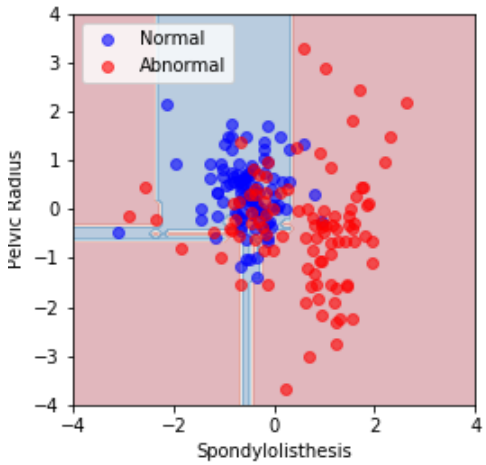
Just like Random Forests, Gradient Boosting combines so-called weak learners to produce a single strong learner by iteration. They way it does it is to iteratively ensemble the residual of a weak model to produce a stronger one until a maximum number of iterations, convergence or stopping criteria is met.

Gradient boosting got its name because of being a Gradient Descent algorithm, in which iterative optimization is used to find the (sometimes local) minimum of a function. The steps taken in each iteration are proportional to the negative of the gradient used [6].

The algorithm may be described as:

1. Initialize model with a initial learner F_1 (as decision trees).
2. Compute residuals (h).
3. Fit residuals to a base learner.
4. Compute a multiplier γ.
5. Update model $F_{i+1} = F_i + \gamma h$
6. Repeat steps 2 - 5 until convergence is achieved.

Figure 8: Gradient Boosting in 2D



By using decision trees as the base learner, the cross-validated best subset in lumbar spine dataset is 'sacral slope', 'pelvic radius' and 'spondylolisthesis', which allows the classifier to achieve an accuracy score of 82.5%, making it the second best performing method so far. A 2D representation of it is shown in figure 8. As it uses the same base learner than Random Forests, it shows a very similar boundary shape and is computationally expensive, taking 13.7 seconds to be optimized.

4 Unsupervised Learning Algorithms

On all the previous algorithms class labels were used to train the models and again to assess their success. Unfortunately, sometimes datasets have not been labelled yet wither because of their size, lack of human resources or there is not known to cluster. In the context of the lumbar spine dataset, little domain information is needed to assume that there could be 2 groups in the data: abnormal spines and normal spines, but without doctors to classify them, it is necessary to relay on unsupervised learning algorithms to identify them. A straightforward downside is that there is no clear measure of success, which makes method comparison harder.

The goal of unsupervised learning is to infer properties about the data (X); and although the dimension of X is usually high, one is not required to change the set of variables [3]. In fact, this is a direct consequence of having no labels: it would be unrealistic to try to perform feature selection or hyper-parameter tuning. Therefore I am implementing the algorithms with their default values and only choosing the number of clusters I want to have.

4.1 K-means

K-means is a distance based metric that requires all features to be quantitative. Depending on the distance metric and kernel (if so) used, K-means takes a different set of assumptions on the structure of the dataset. Most commonly used metric is the squared Euclidean distance, defined in section 3.2, which assumes that each cluster is Gaussian and has a common variance.

Mathematically, K-mean minimizes the total within-cluster sum of squares, which is given by:

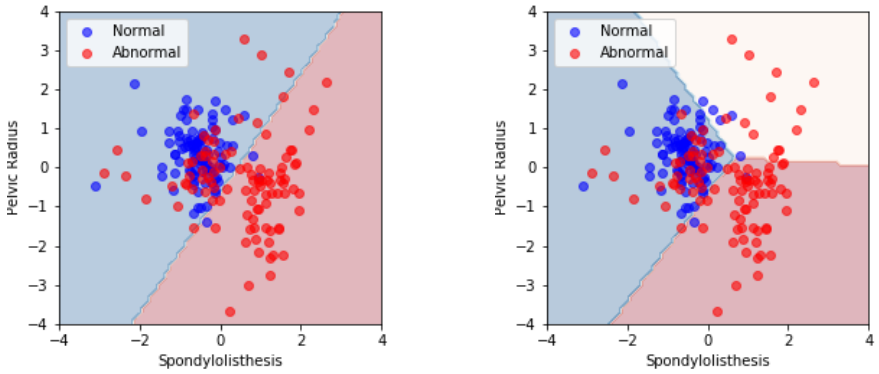
$$argmin_C \sum_{i=1}^{k} \sum_{x \in C_i} || \ x - \mu_i \ ||_2$$

where μ_i are each of the centres, k is the number of clusters and C_i are each of the observations that lie within the same cluster. This equation cannot be optimized in one setting because the location of the centres and the clusters change each time it is run, therefore an iterative process is needed. The basic steps used to obtain clusterings are:

1. Initialize k cluster centres.
2. Calculate distances from each data point to each centre and assign each point to a cluster C_i.
3. Calculate the new centres as the average of each cluster.
4. Repeat until centres do not change any more.

After implementing above algorithm to lumbar spine data, and comparing it to the known labels, K-Means achieved 70.5% accuracy, much lower than any supervised algorithm, but still much better than random (i.e. 50%). On

Figure 9: K-Means in 2D



(a) $k = 2$ (b) $k = 3$

a 2D graph (see figure 9a), taking only into account "Pelvic Radius" and "Spondylolisthesis", it seems like KMeans perform very similar to Logistic Regression and Linear Discriminant Analysis. As KMeans always works, and the outcome is very sensitive to the initialized number of components, it is easy to get it wrong without having prior information, although figure 9b still doesn't look very convincing.

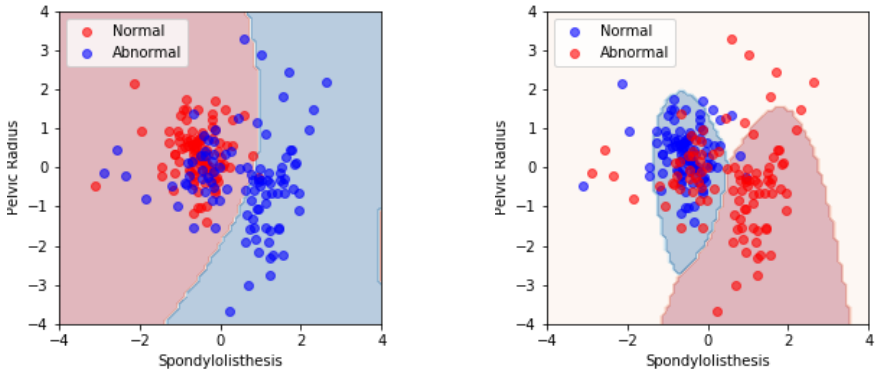
4.2 Gaussian Mixtures

Gaussian Mixtures might be considered as a "soft" version of K-means clustering because its main difference relays on it making a probabilistic, instead of deterministic, assignments of points to clusters [3]. In fact, KMeans can be seen as a special case of Gaussian Mixtures with equal covariance per component [2].

This method implements an Expectation-Maximization (EM) algorithm, thus it has two main steps [3]:

- E-step: each observation is given a weight based on the likelihood of the corresponding Gaussians.
- M-step: clusters are averaged via weighted means.

Figure 10: Gaussian Mixtures in 2D



(a) $k = 2$ (b) $k = 3$

This method achieves a 74% accuracy on the lumbar spine dataset, its greater flexibility allows it to better account for the structure in this data and outperforms K-Means. Figure 10a shows its decision boundaries in a 2D space. Just as K-means, the result is highly dependent on the number of desired clusters; figure 10b shows an example in which $k = 3$ was used. Surprisingly, it does find a good circle in which most of the Normal lumbar spines are, the groups together the abnormal while leaving just some outliers on a third sparse cluster.

4.3 Spectral Clustering

In a general sense, Spectral Clustering is a unique approach to clustering in which first a dimensionality reduction technique (like Principal Component Analysis) is used before applying a clustering algorithm like K-Means on the lower dimension. I also need the number of clusters to be specified beforehand and, because of its dimensionality reduction procedure, it is not possible to categorize a new point (predict) after it has been trained.

As I mentioned earlier, it is not good to perform dimensionality reduction on the lumbar spine data, and a direct consequence of that is this method performing very poorly, with an accuracy score of 53% (just above random).

4.4 Agglomerative Clustering

Following the same logic as a dendrogram, agglomerative clustering is a bottom-up approach in which each observation starts at its own cluster and then they start being merged together according to linkage criteria until the desired number of clusters is achieved [2]. There are a number of linkage criteria, these are:

- Ward: minimizes the sum of squared differences within all clusters.
- Maximum linkage: minimizes the maximum distance between two clusters.
- Average linkage: minimizes the average distance between two clusters.
- Single linkage: minimizes the minimum distance between two clusters.

Similarly to Spectral Clustering, as this method computes clusters on a hierarchical manner, it is not possible to predict the cluster of one additional observation without training the whole model again; and, if one does train it, the structure might be affected by the new point.

Surprisingly, with 74.5%, this rather simple approach achieves the best cross-validated accuracy score of all unsupervised learning algorithms I tested with the lumbar spine dataset. It actually gets a score very close to the one Logistic Regression achieved.

5 Results summary

Ten models have been analysed and tested to try to identify those factors doctors take into account to categorize a lumbar spine as normal or abnormal. No method has managed to get a perfect score on this manner as classes seem to overlap even on a 5-dimensional space. Figure 11 and table 1 show a summary of all results sorted by accuracy.

It is worth noticing that the best performing model were those who were able to draw a circle-like shape around the normal class, meaning that any extreme value in any measure will likely categorize your lumbar spine as

Figure 11: Results summary

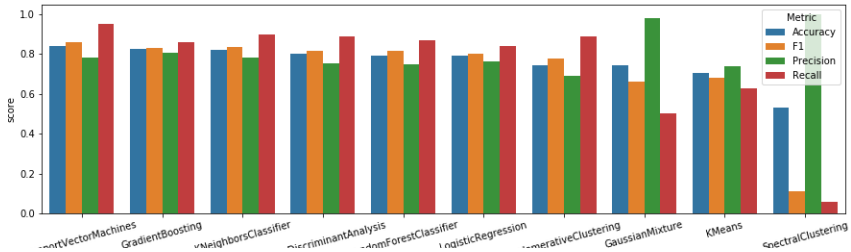


Table 1: Results summary

Model	Accuracy	F1	Precision	Recall
SupportVectorMachines	0.840	0.857	0.785	0.95
GradientBoosting	0.825	0.831	0.806	0.86
KNeighborsClassifier	0.820	0.835	0.784	0.90
LinearDiscriminantAnalysis	0.800	0.816	0.756	0.89
RandomForestClassifier	0.790	0.816	0.750	0.87
LogisticRegression	0.790	0.780	0.765	0.84
AgglomerativeClustering	0.745	0.777	0.690	0.89
GaussianMixture	0.745	0.662	0.980	0.50
KMeans	0.705	0.681	0.741	0.63
SpectralClustering	0.530	0.113	1.000	0.06

abnormal. Equivalently, the worst performing model were those that failed to recognize this pattern a simply draw a line around the centre of the points, allowing extreme values to be classified as normal.

For supervised learning models 'pelvic radius' and 'spondylolisthesis' were consistently identified as important features to determine the class of each observation, while there was some volatility among the others.

Finally, supervised learning consistently achieves better results than unsupervised, thus it is crucial to have some sense on the number of clusters and shape of the data before implementing unsupervised learning models in order to achieve high-quality results.

References

[1] Daniel Kahneman & Amos Tversky. On the Reality of Cognitive Illusions. 1996 Psychological Review.

[2] Scikit-learn developers. User Guide. URL: https://scikit-learn.org/stable/user_guide.html (accessed 25/02/2019).

[3] Trevor Hastie, Robert Tibshirani & Jerome Friedman. The Elements of Statistical Learning. 2008 Springer.

[4] Christopher M. Bishop. Pattern Recognition and Machine Learning. 2006 Springer Science+Business Media.

[5] Cesar Souza. Kernel Functions for Machine Learning Applications. URL: http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications (Accessed: 27/02/2019).

[6] Wikipedia. Gradient descent. URL: https://en.wikipedia.org/wiki/Gradient_descent (Accessed 27/02/2019)

Appendix: Python Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score, accuracy_score, precision_score, re

# Supervised Algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC # https://scikit-learn.org/stable/modules/svm
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis # ht
from sklearn.ensemble import RandomForestClassifier # https://scikit-lear
from sklearn.ensemble import GradientBoostingClassifier # https://scikit-

# Unsupervised Algorithms https://scikit-learn.org/stable/modules/cluster
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClus
from sklearn.mixture import GaussianMixture

# My own package to optimize models
# Available on https://github.com/luisds95
import sys
sys.path.insert(0, 'D:\\OneDrive\\Git\\scikit-learn-helpers')
import sklearnHelpers as skh


# # Functions
def cv(model, X, y):
    return np.mean(cross_val_score(model, X, y, cv=5, scoring='f1'))

def scaler(x):
    x = np.array(x)
    return (x - x.mean())/x.std()

def sup_scores(model, X, y):
    scores = {'supervised':True}
    scores['f1'] = np.mean(cross_val_score(model, X, y, scoring='f1', cv=
    scores['accuracy'] = np.mean(cross_val_score(model, X, y, scoring='ac
    scores['recall'] = np.mean(cross_val_score(model, X, y, scoring='reca
```

```
    scores['precision'] = np.mean(cross_val_score(model, X, y, scoring='p
    return scores

def unsup_scores(y, y_pred):
    scores = {'supervised':False}
    scores['f1'] = f1_score(y, y_pred)
    scores['accuracy'] = accuracy_score(y, y_pred)
    scores['recall'] = recall_score(y, y_pred)
    scores['precision'] = precision_score(y, y_pred)
    return scores

def plot_contour(model, params, subx, x1n, x2n, x1a, x2a, supervised=True
    xg, yg = np.meshgrid(np.arange(-4.1, 4.1, 0.1),np.arange(-4.1, 4.1, 0

    l = model(**params)
    if supervised == True:
        l.fit(subx, y)
    else:
        l.fit(subx)
    zz = l.predict(np.c_[xg.ravel(), yg.ravel()])
    zz = zz.reshape(xg.shape)

    c1='blue'
    c2='red'
    if reverse:
        c1='red'
        c2='blue'

    plt.figure(figsize=(4,4))
    plt.contourf(xg, yg, zz, cmap=plt.cm.RdBu, alpha=.3)
    plt.scatter(x1n, x2n, c=c1, alpha=0.6, label='Normal')
    plt.scatter(x1a, x2a, c=c2, alpha=0.6, label='Abnormal')
    plt.xlabel('Spondylolisthesis')
    plt.ylabel('Pelvic␣Radius')
    plt.xlim((-4, 4))
    plt.ylim((-4, 4))
    #plt.title(type(l).__name__)
    plt.legend()
    plt.savefig('Graphs/{}_contour{}.png'.format(type(l).__name__, suffix
    plt.show()
```

```
# # Read in data

df = pd.read_csv('vertebral_column_data.txt', sep=' ', header=None,
                 names=['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordos
                 'pelvic_radius', 'spondylolisthesis', 'normal'])
df['normal'] = df['normal'] == 'NO'
df.head()

df.shape

df.describe()

df.groupby('normal').count()

# As data is slightly unbalanced, let's balance it:
np.random.seed = 542
normals = df[df['normal']]
unnormals = df[~df['normal']]
unnormals = unnormals.iloc[np.random.choice(len(unnormals), len(normals),
ndf = normals.append(unnormals)

sns.pairplot(ndf, hue='normal')
plt.legend()
plt.savefig('Graphs/pairplot.png')
plt.show()

sns.distplot(ndf['spondylolisthesis'][ndf['normal']], label='Normal')
sns.distplot(ndf['spondylolisthesis'][~ndf['normal']], label='Abnormal')
plt.legend()
plt.savefig('Graphs/spondylolisthesis_dist.png')
plt.show()

# As spondylolisthesis seems exponential, let's loggit
ndf['spondylolisthesis'] = np.log(ndf['spondylolisthesis']+13)
sns.distplot(ndf['spondylolisthesis'][ndf['normal']], label='Normal')
sns.distplot(ndf['spondylolisthesis'][~ndf['normal']], label='Abnormal')
plt.legend()
plt.savefig('Graphs/log_spondylolisthesis_dist.png')
```

```
plt.show()

# Also, as some methods rely on distance, let's scale the variables
X = ndf.drop('normal', axis=1)
for x in X:
    X[x] = scaler(X[x])
y = ndf['normal']

sns.heatmap(ndf.corr(), cmap='bwr', center=0)
plt.show()

subx = X[['spondylolisthesis', 'pelvic_radius']]
x1n = X[y]['spondylolisthesis']
x2n = X[y]['pelvic_radius']
x1a = X[~y]['spondylolisthesis']
x2a = X[~y]['pelvic_radius']
plt.figure(figsize=(6,4))
sns.kdeplot(x1n, x2n, color='blue', alpha=0.6, label='Normal')
sns.kdeplot(x1a, x2a, color='red', alpha=0.6, label='Abnormal')
plt.xlabel('Spondylolisthesis')
plt.ylabel('Pelvic Radius')
plt.legend()
plt.savefig('Graphs/two_features_KDE.png')
plt.show()


# # Using PCA to try to visualize
# It doesn't work

pca = PCA()
X_trans = pca.fit_transform(X)
pca.explained_variance_ratio_

sns.barplot(x=np.arange(6), y=pca.explained_variance_ratio_, color='blue'
plt.show()


# # Supervised Algorithms

all_scores = {}
```

```
# Logistic Regression
logic = LogisticRegression(solver='lbfgs')
param_grid = {'C': np.linspace(0.1, 1, 10), 'solver':['lbfgs']}
logic = skh.tune_fit_model(X, y, logic, scoring='accuracy',
best_subset=True, param_grid=param_grid)
all_scores['LogisticRegression'] = sup_scores(logic['model'],
X[logic['subset']], y)
print(all_scores['LogisticRegression'])
plot_contour(LogisticRegression, logic['parameters'], subx, x1n,
 x2n, x1a, x2a)

# KNN
knn = KNeighborsClassifier()
param_grid = {'n_neighbors':np.arange(1, 9, 2), 'metric':['minkowski', 'e
knn = skh.tune_fit_model(X, y, knn, scoring='accuracy', best_subset=True,
all_scores['KNeighborsClassifier'] = sup_scores(knn['model'],
 X[knn['subset']], y)
print(all_scores['KNeighborsClassifier'])
plot_contour(KNeighborsClassifier, knn['parameters'], subx, x1n,
x2n, x1a, x2a)

plot_contour(KNeighborsClassifier, {'n_neighbors':1}, subx, x1n,
x2n, x1a, x2a, suffix='_overfitted')

plot_contour(KNeighborsClassifier, {'n_neighbors':19}, subx, x1n,
x2n, x1a, x2a, suffix='_underfitted')

# SVM
svm = SVC(gamma='auto')
param_grid = {'C': np.linspace(0.2, 2, 10), 'gamma':['auto', 'scale'],
              'kernel':['linear', 'poly', 'rbf', 'sigmoid']}
svm = skh.tune_fit_model(X, y, svm, scoring='accuracy', best_subset=True,
all_scores['SupportVectorMachines'] = sup_scores(svm['model'],
 X[svm['subset']], y)
print(all_scores['SupportVectorMachines'])
plot_contour(SVC, svm['parameters'], subx, x1n, x2n, x1a, x2a)

plot_contour(SVC, {'C': 1.0, 'gamma': 'auto', 'kernel': 'poly'},
```

```
 subx, x1n, x2n, x1a, x2a, suffix='_poly')


plot_contour(SVC, {'C': 1.0, 'gamma': 'auto', 'kernel': 'sigmoid'},
 subx, x1n, x2n, x1a, x2a, suffix='_sigmoid')

# LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=2)
lda = skh.tune_fit_model(X, y, lda, scoring='accuracy', best_subset=True)
all_scores['LinearDiscriminantAnalysis'] = sup_scores(lda['model'], X[lda
print(all_scores['LinearDiscriminantAnalysis'])
plot_contour(LinearDiscriminantAnalysis, lda['parameters'], subx, x1n,
 x2n, x1a, x2a)

# Random Forests
rf = RandomForestClassifier(n_estimators=100)
param_grid = {'max_depth':np.arange(1, 5), 'max_features':['sqrt'], 'n_es
rf = skh.tune_fit_model(X, y, rf, scoring='accuracy', best_subset=True, p
all_scores['RandomForestClassifier'] = sup_scores(rf['model'],
 X[rf['subset']], y)
print(all_scores['RandomForestClassifier'])
plot_contour(RandomForestClassifier, rf['parameters'], subx, x1n, x2n,
 x1a, x2a)

plot_contour(RandomForestClassifier, {'max_depth': None,
'n_estimators': 150}, subx, x1n, x2n, x1a, x2a, suffix='_overfitted')

plot_contour(RandomForestClassifier, {'max_depth': 1,
 'n_estimators': 150}, subx, x1n, x2n, x1a, x2a, suffix='_underfitted')

# Gradient Boosting
gb = GradientBoostingClassifier()
param_grid = {'learning_rate':np.arange(0.001, 0.02, 0.001), 'n_estimator
gb = skh.tune_fit_model(X, y, gb, scoring='accuracy', best_subset=True, p
all_scores['GradientBoosting'] = sup_scores(gb['model'], X[gb['subset']],
print(all_scores['GradientBoosting'])
plot_contour(GradientBoostingClassifier, gb['parameters'],
subx, x1n, x2n, x1a, x2a)

plot_contour(GradientBoostingClassifier, {'learning_rate': 0.01,
```

```
'n_estimators': 10}, subx, x1n, x2n, x1a, x2a, suffix='_underfitted')

plot_contour(GradientBoostingClassifier, {'learning_rate': 0.5,
'n_estimators': 10}, subx, x1n, x2n, x1a, x2a, suffix='_overfitted')


# # Unsupervised Algorithms

kmeans = KMeans(n_clusters=2).fit(X)
all_scores['KMeans'] = unsup_scores(~y, kmeans.labels_)
print(all_scores['KMeans'])
plot_contour(KMeans, kmeans.get_params(), subx, x1n, x2n, x1a,
x2a, False)

plot_contour(KMeans, {'n_clusters':3}, subx, x1n, x2n, x1a,
x2a, False, suffix='_3clusters')

gm = GaussianMixture(n_components=2).fit(X)
all_scores['GaussianMixture'] = unsup_scores(~y, gm.predict(X))
print(all_scores['GaussianMixture'])
plot_contour(GaussianMixture, gm.get_params(), subx, x1n, x2n,
x1a, x2a, False, True)

plot_contour(GaussianMixture, {'n_components':3}, subx, x1n, x2n,
x1a, x2a, False, suffix='_3clusters')

sc = SpectralClustering(n_clusters=2).fit(X)
all_scores['SpectralClustering'] = unsup_scores(~y, sc.labels_,)
all_scores['SpectralClustering']


agloc = AgglomerativeClustering(n_clusters=2).fit(X)
all_scores['AgglomerativeClustering'] = unsup_scores(y, agloc.labels_)
all_scores['AgglomerativeClustering']


# # Plotting

scodf = []
for name, score in all_scores.items():
```

```
    scodf.append({'model':name, 'Accuracy':score['accuracy'], 'Precision'
                  'Recall':score['recall'], 'F1':score['f1'],})
scodf = pd.DataFrame(scodf)
scodf.sort_values(['Accuracy', 'F1'], ascending=False, inplace=True)
scodf.reset_index(inplace=True, drop=True)

melted = pd.melt(scodf, id_vars="model", var_name="Metric",
value_name="score")

plt.figure(figsize=(15,4))
sns.barplot(x='model', y='score', hue='Metric', data=melted)
plt.xticks(rotation=15)
plt.savefig('Graphs/All_scores.png')
plt.show()

scodf[['model'] + list(scodf.columns[:-1])]
```