Cankaya University

Mechatronics Engineering

MECE-444 PATTERN RECOGNITION

PROJECT

Statlog (Landsat Satellite) Data Set Classification

BATUHAN ILIMAN

201817019

Content Page

Introduction

The Statlog (Landsat Satellite) Data Set serves as a valuable resource for classification studies in the field of remote sensing. This dataset focuses on the classification of pixels derived from Landsat satellite imagery. The satellite images capture the electromagnetic energy values recorded in different spectral bands, which represent various characteristics of the Earth's surface. The Statlog (Landsat Satellite) Data Set consists of a total of 36 features. Each feature represents the values obtained from different spectral bands of the satellite images. These features may include pixel values from regions such as red, green, blue, and infrared. Each data instance corresponds to a single pixel with a 1x1 dimension and is associated with a class label indicating its category. The primary objective of this data set is to accurately predict the class of a pixel using the available features. For instance, the pixel values in the red region may represent a cotton field, while the values in the infrared region may correspond to soil covered with vegetation stubble. Machine learning or statistical classification algorithms can leverage these features to accurately classify the pixels in the data set. The Statlog (Landsat Satellite) Data Set serves as a benchmark widely used by researchers and remote sensing experts. The studies conducted on this data set aim to evaluate the classification performance of satellite imagery and contribute to the development of new classification algorithms.

Data Set Information

The database consists of the multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood. The aim is to predict this classification, given the multi-spectral values. In the sample database, the class of a pixel is coded as a number.

The Landsat satellite data is one of the many sources of information available for a scene. The interpretation of a scene by integrating spatial data of diverse types and resolutions including multispectral and radar data, maps indicating topography, land use etc. is expected to assume significant importance with the onset of an era characterised by integrative approaches to remote sensing (for example, NASA's Earth Observing System commencing this decade). Existing statistical methods are ill-equipped for handling such diverse data types. Note that this is not true for Landsat MSS data considered in isolation (as in this sample database). This data satisfies the important requirements of being numerical and at a single resolution, and standard maximum-likelihood classification performs very well. Consequently, for this data, it should be interesting to compare the performance of other methods against the statistical approach.

One frame of Landsat MSS imagery consists of four digital images of the same scene in different spectral bands. Two of these are in the visible region (corresponding approximately to green and red regions of the visible spectrum) and two are in the (near) infra-red. Each pixel is a 8-bit binary word, with 0 corresponding to black and 255 to white. The spatial resolution of a pixel is about 80m x 80m. Each image contains 2340 x 3380 such pixels.

The database is a (tiny) sub-area of a scene, consisting of 82 x 100 pixels. Each line of data corresponds to a 3x3 square neighbourhood of pixels completely contained within the 82x100 sub-

area. Each line contains the pixel values in the four spectral bands (converted to ASCII) of each of the 9 pixels in the 3x3 neighbourhood and a number indicating the classification label of the central pixel. The number is a code for the following classes:

Number Class

1 red soil
2 cotton crop
3 grey soil
4 damp grey soil
5 soil with vegetation stubble
6 mixture class (all types present)
7 very damp grey soil

NB. There are no examples with class 6 in this dataset.

The data is given in random order and certain lines of data have been removed so we cannot reconstruct the original image from this dataset.

In each line of data the four spectral values for the top-left pixel are given first followed by the four spectral values for the top-middle pixel and then those for the top-right pixel, and so on with the pixels read out in sequence left-to-right and top-to-bottom. Thus, the four spectral values for the central pixel are given by attributes 17,18,19 and 20. If we like we can use only these four attributes, while ignoring the others. This avoids the problem which arises when a 3x3 neighbourhood straddles a boundary.

Attribute Information:

The attributes are numerical, in the range 0 to 255.

Adding related libraries and extraction of datasets prepared in the form of train and test.

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from google.colab import drive
5 drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[2]  1 df = pd.read_csv("/content/drive/MyDrive/satellite/sat.trn", sep=' ', header=None)
     2 print(df.shape)
     3 df_test = pd.read_csv("/content/drive/MyDrive/satellite/sat.tst", sep=' ', header=None)
     4 print(df_test.shape)
```

```
(4435, 37)
(2000, 37)
```

Data pre-processing

Data preprocessing is the process of applying a set of methods and techniques used in data-driven applications such as machine learning or data analysis. It involves organizing, cleaning, transforming, and scaling raw data. These steps are performed to reduce noise in the dataset, eliminate inconsistencies, correct missing or erroneous data, and make the dataset more suitable for analysis. Some of the stages in data preprocessing include:

- Data cleaning: Identifying and removing noise, inconsistencies, and incorrect data in the dataset. For example, filling in missing data or correcting outliers can be done in this stage.

- Data transformation: Changing the format or representation of data in the dataset. For instance, converting categorical data into numerical format or performing feature engineering to create new features can be done in this stage.

- Data scaling: Expressing features in the dataset on the same scale, especially when different features have different scales. This can be achieved through methods such as data normalization or standardization.

Data preprocessing helps improve the performance of machine learning models and ensures that the results are more reliable and meaningful. It is an important step often carried out by data scientists or analysts because the quality and suitability of the dataset can greatly impact the outcomes.

Step1: Check the Null Data

```python
import pandas as pd

def check_missing_values(df_1):
    missing_values = df_1.isnull().sum().sum()
    if missing_values > 0:
        print("Veri kümesi boş değerler içeriyor.")
    else:
        print("Veri kümesi boş değerler içermiyor.")

check_missing_values(df_1)

```

```
Veri kümesi boş değerler içermiyor.
```

Step 2: Check the Scaling Process

```python
from sklearn.preprocessing import StandardScaler

def check_scaling(df_1):
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(df_1)
    if not pd.DataFrame(scaled_data).equals(df_1):
        print("Veri kümesi ölçeklendirilmiş.")
    else:
        print("Veri kümesi ölçeklendirilmemiş.")

check_scaling(df_1)
```

```
Veri kümesi ölçeklendirilmiş.
```

Step 3: Check the Categorical Variables are Encoded?

```python
import pandas as pd

def check_encoding(df_1):
    categorical_cols = df_1.select_dtypes(include=['object']).columns
    if len(categorical_cols) > 0:
        print("Veri kümesi kategorik değişkenlerin kodlandığı bir yapıya sahip.")
    else:
        print("Veri kümesi kategorik değişkenlerin kodlandığı bir yapıya sahip değil.")

check_encoding(df_1)
```

```
Veri kümesi kategorik değişkenlerin kodlandığı bir yapıya sahip değil.
```

After these processes, we can understand that a pre-processing step has already been done in the data set. There is no need to do any extra pre-processing for this dataset.

Editing data according to dataset notifications:

```
[21] 1 df.rename(columns={36: 'Class'}, inplace=True)
```

```
[22] 1 df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	27	28	29	30	31	32	33	34	35	Class
0	92	115	120	94	84	102	106	79	84	102	...	104	88	121	128	100	84	107	113	87	3
1	84	102	106	79	84	102	102	83	80	102	...	100	84	107	113	87	84	99	104	79	3
2	84	102	102	83	80	102	102	79	84	94	...	87	84	99	104	79	84	99	104	79	3
3	80	102	102	79	84	94	102	79	80	94	...	79	84	99	104	79	84	103	104	79	3
4	84	94	102	79	80	94	98	76	80	102	...	79	84	103	104	79	79	107	109	87	3

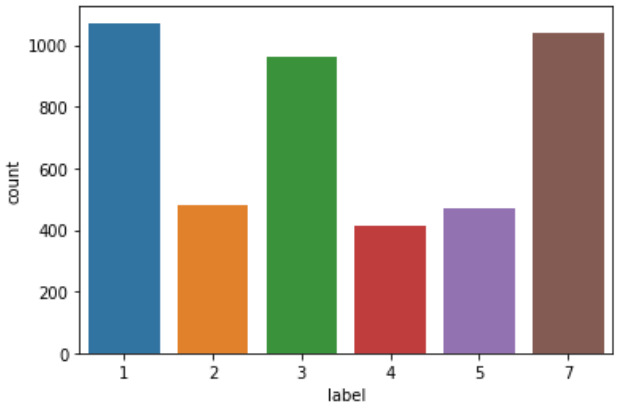
5 rows × 37 columns

- **Check the number of value at each class**

```
1 df['Class'].value_counts()
```

```
1    1072
7    1038
3     961
2     479
5     470
4     415
Name: Class, dtype: int64
```

```
<AxesSubplot:xlabel='label', ylabel='count'>
```



PCA

PCA (Principal Component Analysis) is a data analysis technique used to reduce a large number of variables into a smaller set of meaningful components. This method helps to understand the variability in multidimensional datasets and makes the data more interpretable. PCA identifies the directions, known as "principal components," along which the variance of the data set is maximized. The first principal component captures the largest variability in the data set, and subsequent components complement each other to explain the remaining variance. The main objective of PCA is dimensionality reduction by reducing the excessive variables in the data set. It retains the essential characteristics of the data while making it easier to analyze and visualize. Additionally, PCA can help us understand complex relationships in the data and reduce noise. PCA has a wide range of applications. It is successfully used in financial data analysis, genetic studies, image processing

applications, and many other fields. PCA is not necessary in every algorithm. PCA is used for specific purposes such as reducing the dimensionality of the data, reducing noise, or correcting excessive variability in the dataset. The use of PCA depends on the characteristics, dimensionality, and analysis objectives of the dataset. This dataset is scaled and noise-free from the main dataset. So we don't need to use PCA

Binary Classification

Binary classification, also known as binomial classification, is a fundamental task in machine learning where the goal is to categorize or predict a given input into one of two distinct classes. It involves separating data points into two mutually exclusive groups based on their features or characteristics. The two classes are typically referred to as positive and negative, true and false, or 0 and 1. The process of binary classification involves training a model using a labeled dataset, where each data point is associated with its correct class label. The model then learns to make predictions on new, unseen data by assigning it to one of the two classes. The output of binary classification is a decision boundary that divides the input space into the two classes, enabling the model to classify new instances. This type of classification is widely used in various applications such as spam detection, disease diagnosis, sentiment analysis, and fraud detection, among others.

One frame of Landsat MSS imagery consists of four digital images of the same scene in different spectral bands. Two of these are in the visible region (corresponding approximately to green and red regions of the visible spectrum) and two are in the (near) infra-red. Each pixel is a 8-bit binary word, with 0 corresponding to black and 255 to white. The spatial resolution of a pixel is about 80m x 80m. Each image contains 2340 x 3380 such pixels.

The database is a (tiny) sub-area of a scene, consisting of 82 x 100 pixels. Each line of data corresponds to a 3x3 square neighbourhood of pixels completely contained within the 82x100 sub-area. Each line contains the pixel values in the four spectral bands (converted to ASCII) of each of the 9 pixels in the 3x3 neighbourhood and a number indicating the classification label of the central pixel. The number is a code for the following classes:

Remember that Number Class:

1 red soil

2 cotton crop

3 grey soil

4 damp grey soil

5 soil with vegetation stubble

6 mixture class (all types present)

7 very damp grey soil

```
[26] txt = ['top_left1', 'top_left2', 'top_left3', 'top_left4',
             'top_center1', 'top_center2', 'top_center3', 'top_center4',
             'top_right1', 'top_right2', 'top_right3', 'top_right4',
             'middle_left1', 'middle_left2', 'middle_left3', 'middle_left4',
             'middle_center1', 'middle_center2', 'middle_center3', 'middle_center4',
             'middle_right1', 'middle_right2', 'middle_right3', 'middle_right4',
             'bottom_left1', 'bottom_left2', 'bottom_left3', 'bottom_left4',
             'bottom_center1', 'bottom_center2', 'bottom_center3', 'bottom_center4',
             'bottom_right1', 'bottom_right2', 'bottom_right3', 'bottom_right4', 'Class']
     df = pd.read_csv("/content/drive/MyDrive/satellite/sat.trn", sep=' ', dtype=str, names=txt)
     df
```

	top_left1	top_left2	top_left3	top_left4	top_center1	top_center2	top_center3	top_center4	top_right1	top_right2	...	bottom_left4	b
0	92	115	120	94	84	102	106	79	84	102	...	104	
1	84	102	106	79	84	102	102	83	80	102	...	100	
2	84	102	102	83	80	102	102	79	84	94	...	87	
3	80	102	102	79	84	94	102	79	80	94	...	79	
4	84	94	102	79	80	94	98	76	80	102	...	79	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4430	56	64	108	96	64	71	108	96	68	75	...	92	
4431	64	71	108	96	68	75	108	96	71	87	...	96	
4432	68	75	108	96	71	87	108	88	71	91	...	89	
4433	71	87	108	88	71	91	100	81	76	95	...	89	
4434	71	91	100	81	76	95	108	88	80	95	...	85	

4435 rows × 37 columns

In each line of data the four spectral values for the top-left pixel are given first followed by the four spectral values for the top-middle pixel and then those for the top-right pixel, and so on with the pixels read out in sequence left-to-right and top-to-bottom.

Thus, the four spectral values for the central pixel are given by attributes 17,18,19 and 20 (which would be 'middle_center1', 'middle_center2', 'middle_center3', 'middle_center4'). If we want we can use only these four attributes, while ignoring the others. This avoids the problem which arises when a 3x3 neighbourhood straddles a boundary.

```
df_band1 = df[['top_left1', 'top_center1', 'top_right1',
               'middle_left1', 'middle_center1', 'middle_right1',
               'bottom_left1', 'bottom_center1', 'bottom_right1']].astype('int')
df_band2 = df[['top_left2', 'top_center2', 'top_right2',
               'middle_left2', 'middle_center2', 'middle_right2',
               'bottom_left2', 'bottom_center2', 'bottom_right2']].astype('int')
df_band3 = df[['top_left3', 'top_center3', 'top_right3',
               'middle_left3', 'middle_center3', 'middle_right3',
               'bottom_left3', 'bottom_center3', 'bottom_right3']].astype('int')
df_band4 = df[['top_left4', 'top_center4', 'top_right4',
               'middle_left4', 'middle_center4', 'middle_right4',
               'bottom_left4', 'bottom_center4', 'bottom_right4']].astype('int')
```

```
[30] df_band1.values
```

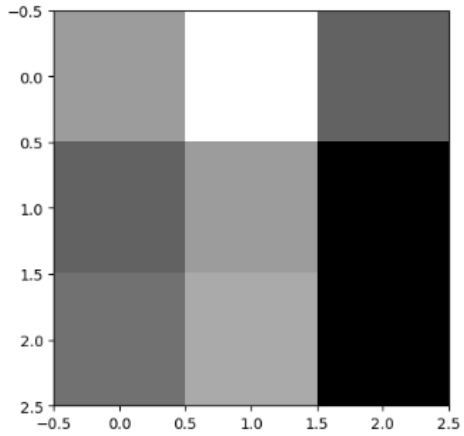
```
array([[ 92,  84,  84, ..., 102,  88,  84],
       [ 84,  84,  80, ...,  88,  84,  84],
       [ 84,  80,  84, ...,  84,  84,  84],
       ...,
       [ 68,  71,  71, ...,  66,  63,  70],
       [ 71,  71,  76, ...,  63,  70,  70],
       [ 71,  76,  80, ...,  70,  70,  63]])
```

When we try to create a cmap map by taking an example from a class

```
import matplotlib

plt.imshow(dfs, cmap = matplotlib.cm.binary, interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x7f8e438d0af0>
```



However, this effort is somewhat futile because, as mentioned in the documentation, "the data is given in random order and certain lines of data have been removed, so we cannot reconstruct the original image from this dataset." Therefore, the machine learning algorithm would need to operate without the image, as it is not possible to arrange the entire image in any consistent manner.

As an example, let's make a classification for class 5 (soil with vegetation stubble).

```
[34]  1 band1, band2, band3, band4 = [], [], [], []
      2
      3 for i in range(0,len(df_band1)):
      4   array = df_band1[i]
      5   array = np.reshape(array, (3,3))
      6   band1.append(array)
      7
      8 for i in range(0,len(df_band2)):
      9   array = df_band2[i]
     10   array = np.reshape(array, (3,3))
     11   band2.append(array)
     12
     13 for i in range(0,len(df_band3)):
     14   array = df_band3[i]
     15   array = np.reshape(array, (3,3))
     16   band3.append(array)
     17
     18 for i in range(0,len(df_band4)):
     19   array = df_band4[i]
     20   array = np.reshape(array, (3,3))
     21   band4.append(array)
```

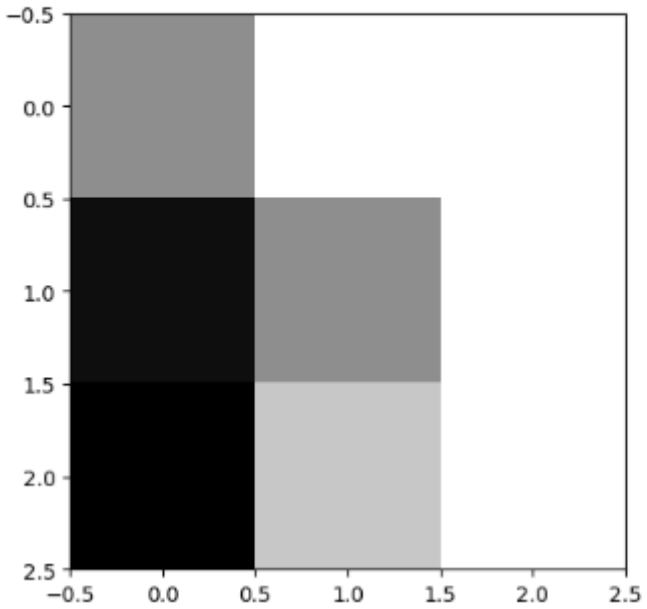
```
[35]  1 band1[0]

      array([[ 92,  84,  84],
             [101,  92,  84],
             [102,  88,  84]])
```

Let's convert the bands we divided into 3x3 arrays and take cmap.

```
1 import matplotlib
2
3 plt.imshow(band1[0], cmap = matplotlib.cm.binary, interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x7fcb9accaec0>
```



Values are more meaningful and interpretable after binary classification.

Stochastic Gradient Descent Classifier(SGD):

The Stochastic Gradient Descent (SGD) is a commonly used classification algorithm in machine learning. SGD is known for its ability to efficiently train models on large datasets. The algorithm considers each data example individually and performs gradient calculations accordingly. Therefore, it is called "stochastic" as it uses a random example at each step. This method is used to make rapid and efficient progress during the training process. SGD is a general optimization algorithm that can be applied to various machine learning models. It is particularly preferred for training models on large datasets. The algorithm selects a specific example, computes the gradient, and updates the model parameters at each step. Instead of calculating the gradient over the entire dataset, it rapidly progresses by computing on random examples. This allows for faster training and reduced memory usage. SGD works effectively for many machine learning models. For instance, it can be used in linear and non-linear classification models such as logistic regression, support vector machines, and multi-layer perceptrons. Variants of SGD are also utilized in complex structures like deep learning models. By using gradient computations to update model parameters, SGD helps the model better fit the data. SGDClassifier is a versatile algorithm for classification tasks that offers efficient optimization and the ability to handle large datasets. Its incremental updates and various loss functions make it a flexible choice for a wide range of classification problems.

```python
from sklearn.linear_model import SGDClassifier # Stochastic Gradient Descent Classifier
sgd_clf1 = SGDClassifier(random_state=42, eta0=1, learning_rate='adaptive')
sgd_clf2 = SGDClassifier(random_state=42, eta0=1, learning_rate='adaptive')
sgd_clf3 = SGDClassifier(random_state=42, eta0=1, learning_rate='adaptive')
sgd_clf4 = SGDClassifier(random_state=42, eta0=1, learning_rate='adaptive')
sgd_clf1.fit(band_01, yband1_5)
sgd_clf2.fit(band_02, yband2_5)
sgd_clf3.fit(band_03, yband3_5)
sgd_clf4.fit(band_04, yband4_5)
```

```
SGDClassifier
SGDClassifier(eta0=1, learning_rate='adaptive', random_state=42)
```

```
detect1, detect2, detect3, detect4 = [],[],[],[]

for i in range(0,len(band_01)):
  pred1 = sgd_clf1.predict([band_01[i]])
  pred2 = sgd_clf2.predict([band_02[i]])
  pred3 = sgd_clf3.predict([band_03[i]])
  pred4 = sgd_clf4.predict([band_04[i]])
  detect1.append(pred1)
  detect2.append(pred2)
  detect3.append(pred3)
  detect4.append(pred4)
```

```
unique, counts = np.unique(detect1, return_counts=True)
dict(zip(unique, counts))
```

```
{False: 3369, True: 1066}
```

```
df['Class'].value_counts() # True positives
```

```
1    1072
7    1038
3     961
2     479
5     470
4     415
Name: Class, dtype: int64
```

[65]
```
from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf1, band_01, yband1_5, cv=3, scoring="accuracy")
```

```
array([0.96822177, 0.94857916, 0.95196211])
```

[66]
```
cross_val_score(sgd_clf2, band_02, yband2_5, cv=3, scoring="accuracy")
```

```
array([0.90872211, 0.90054127, 0.91069012])
```

[67]
```
cross_val_score(sgd_clf3, band_03, yband3_5, cv=3, scoring="accuracy")
```

```
array([0.95605139, 0.99052774, 0.96549391])
```

[68]
```
cross_val_score(sgd_clf4, band_04, yband4_5, cv=3, scoring="accuracy")
```

```
array([0.95740365, 0.98849797, 0.95737483])
```

KNeighborsClassifier (KNN):

The KNeighborsClassifier (KNN) is a machine learning algorithm used for classification problems. It classifies data points based on their nearest neighbors in a given space. KNN is an instance-based learning method. During the training phase, each data point and its corresponding target class label are recorded. When we want to classify a data point, the algorithm uses a distance metric to find its nearest neighbors. The most commonly used distance metric is the Euclidean distance, but different metrics can also be preferred. KNN has a parameter called the number of neighbors (K) used to determine the classification. The K value determines how many neighbors will be considered when making a classification. As the K value increases, the algorithm's decisions become smoother but can lead to overfitting and increased computational costs. If K is small, the algorithm can make more precise decisions close to the data, but noisy data points can have a stronger influence. The KNN algorithm classifies based on the majority class label of the neighbors. For example, if most of the nearest neighbors of a data point belong to the "blue" class, the data point will be classified as "blue". The KNN algorithm has a simple and understandable structure. However, it can be computationally expensive when working with large datasets. Additionally, it performs better when the dataset is well-distributed and there is less confusion between classes.

KNeighborsClassifier is a straightforward classification algorithm that relies on the concept of nearest neighbors for making predictions. It is widely used in various domains and provides a simple yet effective approach for classification tasks.

```
[ ]  1 from sklearn.neighbors import KNeighborsClassifier
     2 y_train_large = (y_train >= 7)
     3 y_train_odd = (y_train % 2 == 1)
     4 y_multilabel = np.c_[y_train_large, y_train_odd]
     5 knn_clf = KNeighborsClassifier()
     6 knn_clf.fit(df_band1, y_multilabel)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

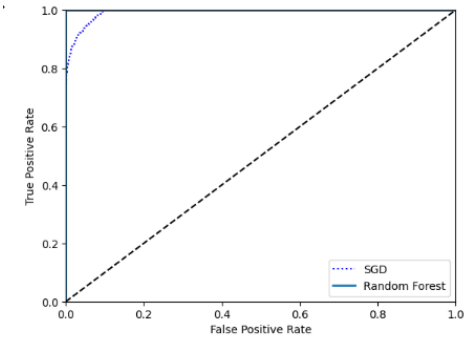
```
     1 knn_clf.predict(dfs1[0]) # Indeed, 3 is not large (False) but it is odd (True)
```

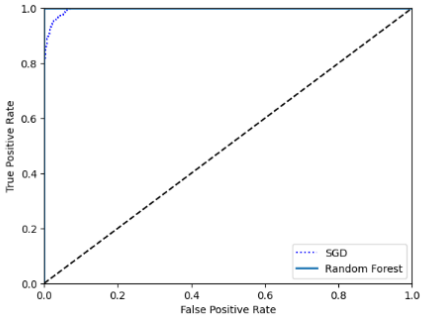
```
[ ]  1 y_train_knn_pred = cross_val_predict(knn_clf, df_band1, y_train, cv=3)
     2 f1_score(y_train, y_train_knn_pred, average="micro")
```

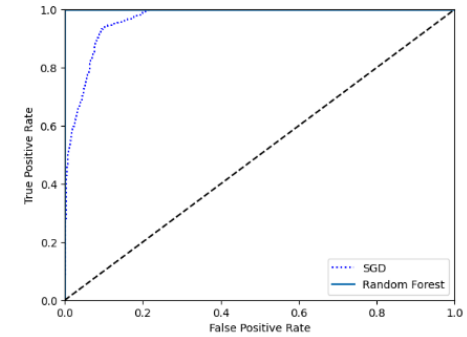
```
0.5580608793686584
```

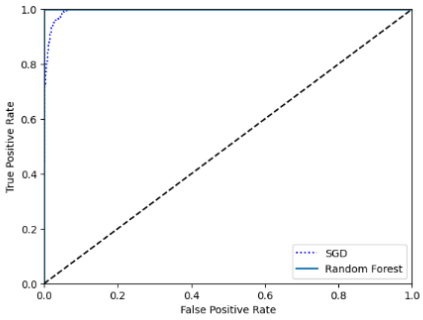
AUC-ROC (KNN vs SGD True Positive – False Positive Rate)

AUC-ROC (Area Under the Receiver Operating Characteristic Curve) is a widely used performance metric to evaluate the effectiveness of classification models, such as K-Nearest Neighbors (KNN) and Stochastic Gradient Descent (SGD). It measures the trade-off between the true positive rate and the false positive rate. The true positive rate, also known as sensitivity or recall, represents the proportion of actual positive instances correctly identified by the model. On the other hand, the false positive rate indicates the proportion of negative instances incorrectly classified as positive. The AUC-ROC curve plots the true positive rate on the y-axis and the false positive rate on the x-axis, creating a graphical representation of the model's performance. The curve summarizes the model's ability to distinguish between positive and negative instances across various classification thresholds. When comparing KNN and SGD using the AUC-ROC metric, we analyze their respective curves and assess which model performs better in terms of true positive rate and false positive rate. A higher AUC-ROC value indicates a model with better discriminatory power and overall performance. ROC curves and the AUC-ROC metric are valuable tools for evaluating and comparing the performance of binary classification models. They help to assess the classifier's trade-off between true positive and false positive rates and provide a comprehensive overview of its discriminatory power.










```
[ ]  1 from sklearn.metrics import roc_auc_score
     2 roc_auc_score(yband1_5, y_scores1)
```

```
0.992794276713614
```

```
[ ]  1 roc_auc_score(yband2_5, y_scores2)
```

```
0.9675251823126265
```

```
[ ]  1 roc_auc_score(yband3_5, y_scores3)
```

```
0.9962794441201944
```

```
[ ]  1 roc_auc_score(yband4_5, y_scores4)
```

```
0.9949649554775465
```

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

```
tp = arr_data[1][1]
fp = arr_data[0][1]
tn = arr_data[0][0]
fn = arr_data[1][0]


sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);
```

MCC measures the performance of a classification model by comparing its predictions with the actual classes.

Specificity is a performance measure used in machine learning and statistical classification models. Specificity represents the rate at which true negatives are correctly identified.

Specificity is calculated using the following formula:

Specificity = True Negatives / (True Negatives + False Positives)

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a machine learning algorithm used for solving classification and regression problems. SVM aims to separate examples in a dataset into two or more classes. The main objective of SVM is to find a hyperplane that separates the data points. This hyperplane is selected to

maximize the distance between data points of different classes, and it is determined using support vectors. Support vectors are the data points that are closest to the hyperplane and are used to make classification decisions. One of the major advantages of SVM is its ability to perform well in high-dimensional datasets. SVM creates a classification function that precisely determines the class to which new data points belong. This function can be used for classifying new instances. SVM can handle both linear and non-linear problems. Linear SVM can separate data points using a linear hyperplane, while non-linear SVM uses kernel functions to map the data points into higher-dimensional spaces where linear separation is possible. In SVMs, the goal is to find an optimal hyperplane that separates the data points of different classes in a way that maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class. The data points that lie closest to the hyperplane are known as support vectors.

```
Print Class: 0
TP=459, FP=10, TN=1529, FN=2
Sensitivity: 0.996
Specificity: 0.994
Accuracy: 0.994
Balanced Accuracy: 0.995
Precision: 0.979
F1-Score: 0.987
MCC: 0.983

Print Class: 1
TP=218, FP=8, TN=1768, FN=6
Sensitivity: 0.973
Specificity: 0.995
Accuracy: 0.993
Balanced Accuracy: 0.984
Precision: 0.965
F1-Score: 0.969
MCC: 0.965

Print Class: 2
TP=383, FP=57, TN=1546, FN=14
Sensitivity: 0.965
Specificity: 0.964
Accuracy: 0.964
Balanced Accuracy: 0.964
Precision: 0.87
F1-Score: 0.915
MCC: 0.895
```

```
Print Class: 3
TP=122, FP=48, TN=1741, FN=89
Sensitivity: 0.578
Specificity: 0.973
Accuracy: 0.932
Balanced Accuracy: 0.776
Precision: 0.718
F1-Score: 0.64
MCC: 0.607

Print Class: 4
TP=204, FP=16, TN=1747, FN=33
Sensitivity: 0.861
Specificity: 0.991
Accuracy: 0.976
Balanced Accuracy: 0.926
Precision: 0.927
F1-Score: 0.893
MCC: 0.88

Print Class: 5
TP=405, FP=70, TN=1460, FN=65
Sensitivity: 0.862
Specificity: 0.954
Accuracy: 0.932
Balanced Accuracy: 0.908
Precision: 0.853
F1-Score: 0.857
MCC: 0.813
```

Conclusion

Machine learning models are artificial intelligence systems that can make predictions by extracting patterns from data. However, different machine learning models can make predictions at different rates. This variation arises from the design of the models, the algorithms used, and the training data. Machine learning models are generally classified as supervised or unsupervised. Supervised learning utilizes both input data and their corresponding correct outputs during the training process. On the other hand, unsupervised learning is a method where models learn without knowing the outputs. Supervised models tend to make more accurate predictions because they leverage actual outputs during training, allowing them to better learn the relationships with input data. Another factor is the architecture and complexity of the model. Machine learning models employ various algorithms and structures. For instance, simple models like linear regression or logistic regression may have limited learning capacity and perform better on simpler datasets. However, complex models such as deep learning models based on neural networks can excel in more intricate datasets, although they may require more computational power. Training data also plays a crucial role in prediction capability. Models make predictions based on the patterns they learn from training data. If the training data is not diverse or representative enough, the model may produce biased or incomplete results. Training the model with a large, high-quality, and diverse dataset can enhance its ability to make accurate predictions. Lastly, the nature of the feature being predicted affects the model's performance. Some features are easier to predict than others. Moreover, the distribution of the target feature can impact the prediction capability. Features with balanced distributions can help the model make more accurate predictions, while imbalanced distributions may pose challenges.

To summarize, machine learning models can make predictions at different rates. Supervised learning models tend to be more accurate, and the architecture, complexity, training data, and nature of the feature being predicted all influence the prediction capability. Choosing the right algorithm, utilizing high-quality and diverse data, and understanding the characteristics of the target feature can improve the model's performance.

Papers That Cite This Data Set[1]:

Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin. Linear dimensionalityreduction using relevance weighted LDA. School of Electrical and Electronic Engineering Nanyang Technological University. 2005. [View Context].

Giorgio Valentini. Random Aggregated and Bagged Ensembles of SVMs: An Empirical Bias?Variance Analysis. Multiple Classifier Systems. 2004. [View Context].

Xiaoli Z. Fern and Carla Brodley. Cluster Ensembles for High Dimensional Clustering: An Empirical Study. Journal of Machine Learning Research n, a. 2004. [View Context].

Jaakko Peltonen and Samuel Kaski. Discriminative Components of Data. IEEE. 2004. [View Context].

S. Augustine Su and Jennifer G. Dy. Automated hierarchical mixtures of probabilistic principal component analyzers. ICML. 2004. [View Context].

Jaakko Peltonen and Arto Klami and Samuel Kaski. Improved Learning of Riemannian Metrics for Exploratory Analysis. Improved Learning of Riemannian Metrics for Exploratory Analysis. Neural Networks. 2004. [View Context].

Fabian Hoti and Lasse Holmström. A semiparametric density estimation approach to pattern classification. Pattern Recognition, 37. 2004. [View Context].

Giorgio Valentini and Thomas G. Dietterich. Low Bias Bagged Support Vector Machines. ICML. 2003. [View Context].

Zoubin Ghahramani and Hyun-Chul Kim. Bayesian Classifier Combination. Gatsby Computational Neuroscience Unit University College London. 2003. [View Context].

Giorgio Valentini. Ensemble methods based on bias--variance analysis Theses Series DISI-TH-2003. Dipartimento di Informatica e Scienze dell'Informazione . 2003. [View Context].

Igor V. Tetko. Associative Neural Network. Neural Processing Letters, 16. 2002. [View Context].

Jaakko Peltonen and Arto Klami and Samuel Kaski. Learning More Accurate Metrics for Self-Organizing Maps. ICANN. 2002. [View Context].

Peter Sykacek and Stephen J. Roberts. Adaptive Classification by Variational Kalman Filtering. NIPS. 2002. [View Context].

Stephen D. Bay. Multivariate Discretization for Set Mining. Knowl. Inf. Syst, 3. 2001. [View Context].

Kagan Tumer and Joydeep Ghosh. Robust Combining of Disparate Classifiers through Order Statistics. CoRR, csLG/9905013. 1999. [View Context].

Kagan Tumer and Nikunj C. Oza. Decimated Input Ensembles for Improved Generalization. NASA Ames Research Center. 1999. [View Context].

Xavier Giannakopoulos and Juha Karhunen and Erkki Oja. An Experimental Comparison of Neural Algorithms for Independent Component Analysis and Blind Separation. Int. J. Neural Syst, 9. 1999. [View Context].

Cesar Guerra-Salcedo and L. Darrell Whitley. Genetic Approach to Feature Selection for Ensemble Creation. GECCO. 1999. [View Context].

Robert E. Schapire and Yoav Freund and Peter Bartlett and Wee Sun Lee. The Annals of Statistics, to appear. Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. AT&T Labs. 1998. [View Context].

Grigorios Tsoumakas and Ioannis P. Vlahavas. Fuzzy Meta-Learning: Preliminary Results. Greek Secretariat for Research and Technology. [View Context].

Xavier Giannakopoulos and Juha Karhunen and Erkki Oja. A COMPARISON OF NEURAL ICA ALGORITHMS USING REAL-WORLD DATA. IDSIA. [View Context].

Adil M. Bagirov and Julien Ugon. An algorithm for computation of piecewise linear function separating two sets. CIAO, School of Information Technology and Mathematical Sciences, The University of Ballarat. [View Context].

Giorgio Valentini. An experimental bias--variance analysis of SVM ensembles based on resampling

techniques. [View Context].

Cesar Guerra-Salcedo and Stephen Chen and Darrell Whitley and Sarah Smith. Fast and Accurate Feature Selection Using Hybrid Genetic Strategies. Department of Computer Science Colorado State University. [View Context].

Je Scott and Mahesan Niranjan and Richard W. Prager. Realisable Classifiers: Improving Operating Performance on Variable Cost Problems. Cambridge University Department of Engineering. [View Context].

Vikas Sindhwani and P. Bhattacharya and Subrata Rakshit. Information Theoretic Feature Crediting in Multiclass Support Vector Machines. [View Context].

Jaakko Peltonen and Arto Klami and Samuel Kaski. Learning Metrics for Information Visualization. Neural Networks Research Centre Helsinki University of Technology. [View Context].

C. esar and Cesar Guerra-Salcedo and Darrell Whitley. Feature Selection Mechanisms for Ensemble Creation : A Genetic Search Perspective. Department of Computer Science Colorado State University. [View Context].