ML Prac Doc

## Simple Linear Regression

# Simple Linear Regression

"""

from google.colab import drive

drive.mount('/content/drive')

"""## Importing the libraries"""

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

"""## Importing the dataset"""

dataset = pd.read\_csv('/content/drive/MyDrive/Msc/Pracs/Sem3/Salary\_Data.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

"""## Splitting the dataset into the Training set and Test set"""

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

"""## Training the Simple Linear Regression model on the Training set"""

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

"""## Predicting the Test set results"""

y\_pred = regressor.predict(X\_test)

"""## Visualising the Training set results"""

plt.scatter(X\_train, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

"""## Visualising the Test set results"""

plt.scatter(X\_test, y\_test, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

## Multiple Linear Regression

# Multiple Linear Regression

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('50\_Startups.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

print(X)

# Encoding categorical data

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])], remainder='passthrough')

X = np.array(ct.fit\_transform(X))

print(X)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

# Training the Multiple Linear Regression model on the Training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = regressor.predict(X\_test)

np.set\_printoptions(precision=2)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

## Support Vector Machine

# Support Vector Machine (SVM)

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

print(X)

print(y)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X = sc.fit\_transform(X)

print(X)

# Training the SVM model on the Training set

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

classifier.fit(X, y)

# Predicting a new result(Optional)

print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Data set results

y\_pred = classifier.predict(X)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y.reshape(len(y),1)),1))

# Making the Confusion Matrix(Optinal)

from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y, y\_pred)

print(cm)

accuracy\_score(y, y\_pred)

# Visualising the Data set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X), y

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

## K-means Clustering

# K-Means Clustering

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

print(X)

# Training the K-Means model on the dataset

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X)

print(y\_kmeans)

# Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

## Hierarchical Clustering

# Hierarchical Clustering

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

# Using the dendrogram to find the optimal number of clusters

import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))

plt.title('Dendrogram')

plt.xlabel('Customers')

plt.ylabel('Euclidean distances')

plt.show()

# Training the Hierarchical Clustering model on the dataset

from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters = 5, affinity = 'euclidean', linkage = 'ward')

y\_hc = hc.fit\_predict(X)

# Visualising the clusters

plt.scatter(X[y\_hc == 0, 0], X[y\_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_hc == 1, 0], X[y\_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_hc == 2, 0], X[y\_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_hc == 3, 0], X[y\_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_hc == 4, 0], X[y\_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

## Naive Bayes

# Naive Bayes

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

print(X)

print(y)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X = sc.fit\_transform(X)

print(X)

# Training the Naive Bayes model on the Training set

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X, y)

# Predicting a new result

print(classifier.predict(sc.transform([[30,87000]])))

# Predicting the Test set results

y\_pred = classifier.predict(X)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y.reshape(len(y\_test),1)),1))

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y, y\_pred)

print(cm)

accuracy\_score(y, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X), y

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

## K-nearest Neighbors

# K-Nearest Neighbors (K-NN)

## Importing the libraries

"""

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

"""## Importing the dataset"""

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

"""## Splitting the dataset into the Training set and Test set"""

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

print(X\_train)

print(y\_train)

print(X\_test)

print(y\_test)

"""## Feature Scaling"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

print(X\_train)

print(X\_test)

"""## Training the K-NN model on the Training set"""

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

classifier.fit(X\_train, y\_train)

"""## Predicting a new result"""

print(classifier.predict(sc.transform([[30,87000]])))

"""## Predicting the Test set results"""

y\_pred = classifier.predict(X\_test)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

"""## Making the Confusion Matrix"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

"""## Visualising the Training set results"""

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 1),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 1))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

"""## Visualising the Test set results"""

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 1),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 1))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('K-NN (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

## Artificial Neural Networks

# Artificial Neural Network

### Importing the libraries

"""

import numpy as np

import pandas as pd

import tensorflow as tf

tf.\_\_version\_\_

"""## Part 1 - Data Preprocessing

### Importing the dataset

"""

dataset = pd.read\_csv('Churn\_Modelling.csv')

X = dataset.iloc[:, 3:-1].values

y = dataset.iloc[:, -1].values

dataset.head()

print(X)

print(y)

"""### Encoding categorical data

Label Encoding the "Gender" column

"""

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X[:, 2] = le.fit\_transform(X[:, 2])

print(X)

"""One Hot Encoding the "Geography" column"""

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')

X = np.array(ct.fit\_transform(X))

print(X)

"""### Splitting the dataset into the Training set and Test set"""

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

"""### Feature Scaling"""

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

"""## Part 2 - Building the ANN

### Initializing the ANN

"""

ann = tf.keras.models.Sequential()

"""### Adding the input layer and the first hidden layer"""

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

"""### Adding the second hidden layer"""

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

"""### Adding the output layer"""

ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

"""## Part 3 - Training the ANN

### Compiling the ANN

"""

ann.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

"""### Training the ANN on the Training set"""

ann.fit(X\_train, y\_train, batch\_size = 32, epochs = 100)

"""## Part 4 - Making the predictions and evaluating the model

### Predicting the result of a single observation

\*\*Homework\*\*

Use our ANN model to predict if the customer with the following informations will leave the bank:

Geography: France

Credit Score: 600

Gender: Male

Age: 40 years old

Tenure: 3 years

Balance: \$ 60000

Number of Products: 2

Does this customer have a credit card ? Yes

Is this customer an Active Member: Yes

Estimated Salary: \$ 50000

So, should we say goodbye to that customer ?

\*\*Solution\*\*

"""

print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5)

"""Therefore, our ANN model predicts that this customer stays in the bank!

\*\*Important note 1:\*\* Notice that the values of the features were all input in a double pair of square brackets. That's because the "predict" method always expects a 2D array as the format of its inputs. And putting our values into a double pair of square brackets makes the input exactly a 2D array.

\*\*Important note 2:\*\* Notice also that the "France" country was not input as a string in the last column but as "1, 0, 0" in the first three columns. That's because of course the predict method expects the one-hot-encoded values of the state, and as we see in the first row of the matrix of features X, "France" was encoded as "1, 0, 0". And be careful to include these values in the first three columns, because the dummy variables are always created in the first columns.

### Predicting the Test set results

"""

y\_pred = ann.predict(X\_test)

y\_pred = (y\_pred > 0.5)

print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

"""### Making the Confusion Matrix"""

from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)