UNIVERSIDAD DE LIMA
FACULTAD DE CIENCIAS EMPRESARIALES Y ECONÓMICAS
CARRERA DE NEGOCIOS INTERNACIONALES

Predicción de Rendimiento Académico de alumnos usando Machine Learning

Herberth Gutiérrez Villaverde[1]

Manuela Linares Barbero[2]

Angel Agüero Correa[3]

Jhelly Pérez Nuñez[4]

Septiembre 2022

[1] Docente de la Universidad de Lima: Hgutierr@ulima.edu.pe
[2] Docente de la Universidad de Lima: Malinare@ulima.edu.pe
[3] Docente de la Universidad de Lima: AAguero@ulima.edu.pe
[4] Docente de la Universidad de Lima: Jrperez@ulima.edu.pe

Índice

Resumen

En el presente trabajo se hace una evaluación de las técnicas de aprendizaje automáticos: Arboles de decisión, Bosques aleatorios, Redes Neuronales y Máquinas de soporte vectorial para predecir el rendimiento académico de alumnos. Encontramos que el algoritmo de bosques aleatorios presenta una mayor precisión en la predicción del rendimiento académico. Esta investigación es relevante en las instituciones educativas, especialmente para la definición de políticas de seguimiento y apoyo a los alumnos principalmente en riesgo académico. Adicionalmente se hace una estimación del potencial de la aplicación de estos algoritmos de aprendizaje automático en las diferentes carreras de la facultad de Ciencias Empresariales y Económicas de la Universidad de Lima.

Palabras clave: Predicción de rendimiento en educación, Aprendizaje automático, Machine learning, Random forest, Bosques aleatorios, Redes Neuronales, Máquinas de soporte vectorial, SVM, Arboles de decisión.

Introducción

En todos los sistemas educativos básicos y superiores, un componente importante es el sistema de evaluación del aprendizaje, que mide la eficiencia del proceso. Se espera que la mayor cantidad de estudiantes culminen sus planes de estudios en los plazos establecidos. Sin embargo, por diversos factores hay alumnos que no pueden seguir el ritmo del proceso y terminan en una situación de riesgo académico (repitiendo una o varias asignaturas), sin poder alcanzar las expectativas personales, familiares y de la sociedad en su conjunto.

Frente a este problema tenemos la pregunta: ¿es posible predecir con cierta confiabilidad el rendimiento académico de estudiantes, y en especial aquellos que se encuentran en riesgo académico? Igualmente, importante es conocer la respuesta a la pregunta ¿de todas las variables que condicionan el rendimiento académico en los alumnos cuáles son las más determinantes? Las respuestas a estas preguntas ayudarían a las instituciones a diseñar políticas y estrategias preventivas en las metodologías de enseñanza, diseño de las asignaturas, perfil de los docentes y otras medidas encaminadas a prevenir el riesgo académico de los alumnos.

En el presente trabajo se pretende evaluar la factibilidad del uso de técnicas de Aprendizaje Automático (Machine Learning) para predecir el rendimiento académico de los alumnos, sobre la base de información de los resultados académicos anteriores; y un grupo de variables demográficas y sociales que influyen en el desempeño de los estudiantes que normalmente no son registradas por las entidades educativas, pero que se recogen con cuestionarios. Las técnicas de Aprendizaje Automático a considerar son: Árboles de decisión, Bosques Aleatorios, Redes Neuronales y Máquinas de Soporte Vectorial.

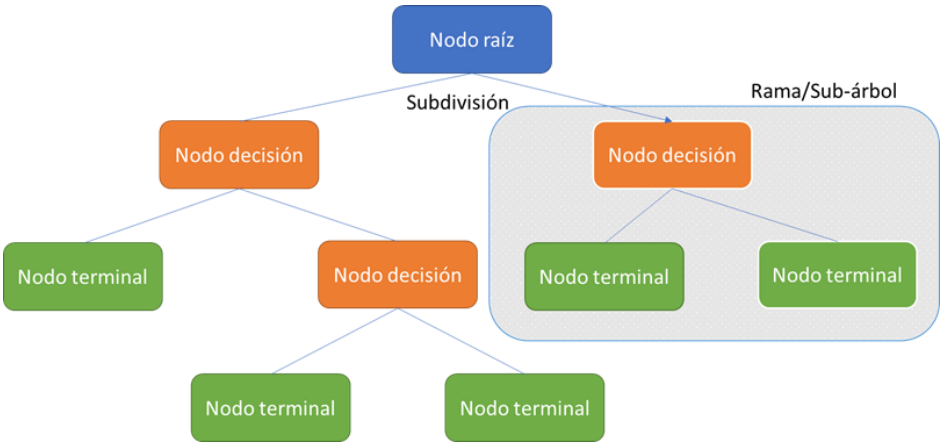
En el desarrollo del trabajo se hará una descripción de cada técnica utilizada, las evaluaciones y resultados alcanzados, la precisión y conveniencia de cada técnica y su posible aplicación en la universidad de Lima.

Técnicas de Aprendizaje Automático (Machine Learning) usadas

Árbol de decisión.

Los Árboles de Decisión son técnicas de aprendizaje automático que dividen sucesivamente un conjunto de datos siguiendo una estructura jerárquica similar a la de un árbol invertido. El primer nodo, llamado "nodo raíz" representa todo el conjunto de datos. El árbol se divide recursivamente en nodos por medio de la selección de los atributos que permiten una división homogénea del conjunto de datos en grupos más pequeños, hasta llegar a los nodos "terminales" con los valores de la clasificación buscada o variable objetivo (Charbuty & Abdulazeez, 2021).

Figura N°1: Árbol de Decisión



Los criterios utilizados para la decisión de dividir los nodos como Gini, Entropía, Chi-cuadrado, entre otros, afectará en gran medida la precisión de un árbol, y su uso dependerá de si los árboles se usan para clasificación o regresión.

La selección del algoritmo de árbol de decisión también se basa en el tipo de variables objetivo. Por ejemplo, los algoritmos C4.5, CART se usan cuando la variable a predecir es de tipo categórica o continua, mientras que los algoritmos CHAID y QUEST cuando las variables son solo categóricas. (Maimon & Rokach, 2014).

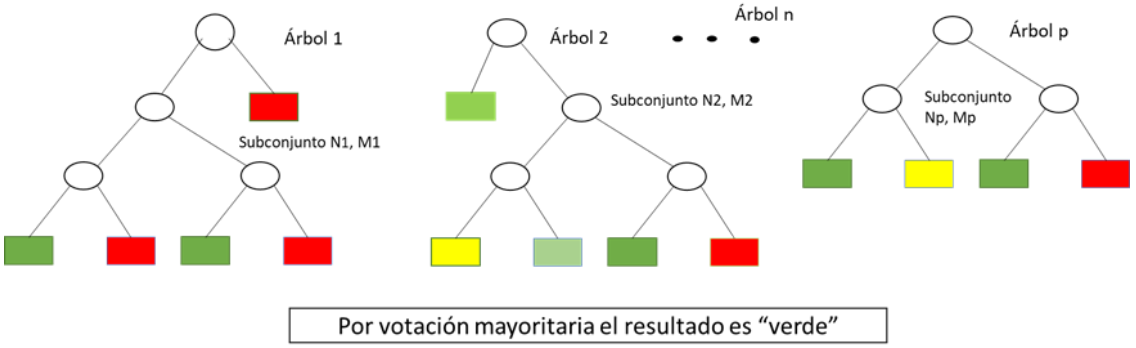
Pese a su gran utilidad los Árboles de decisión no son muy eficientes cuando el conjunto de datos contiene un gran número de atributos, produciendo una gran profundidad en el tamaño del árbol, pudiendo predecir la variable objetivo con 100% de precisión en los datos de entrenamiento y mostrando baja precisión con datos nuevos. El remedio para esto es manejar la profundidad del árbol o usar Bosques Aleatorios (Hartshorn, 2016).

Bosque aleatorio

Los Bosques Aleatorios son algoritmos supervisados de "aprendizaje conjunto" (ensemble learning algorithms) en base al uso de Árboles de Decisión diferentes, construidos a partir de una selección aleatoria de variables (o atributos) del total de variables independientes del conjunto de datos. Cada árbol del bosque evalúa una muestra aleatoria con reemplazo del

conjunto de datos (proceso conocido como bootstrapping) y luego los resultados de todos los árboles de clasificación son tomados en cuenta y usando el principio de "sabiduría de las masas" se consideran las clasificaciones más frecuentes del conjunto de árboles como la solución final (Breiman, 2001; Breiman, 2004)

Figura N°2: Bosque aleatorio



Los bosques aleatorios son una herramienta eficaz para predicciones. Debido a la Ley de los Grandes Números, a diferencia de los Árboles de Decisión, evitan el sobreajuste (overfitting), y en general son más precisos que los Árboles de Decisión. (Biau, 2012)

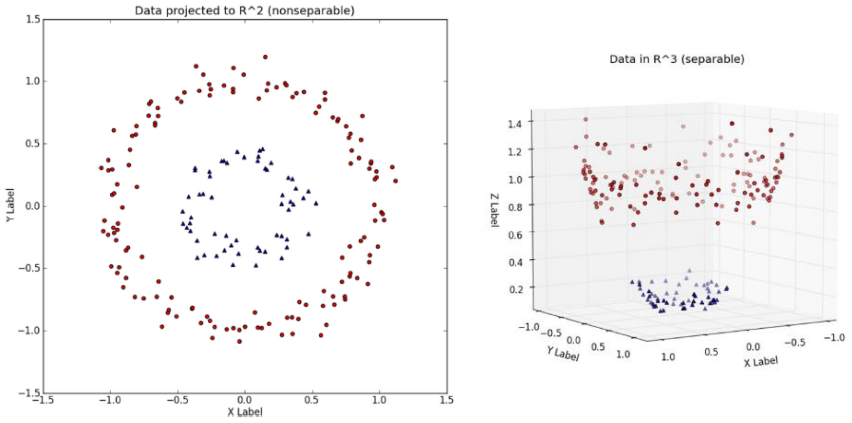
Este es un algoritmo ideal para desarrolladores porque resuelve el problema del sobreajuste de conjuntos de datos. Es una herramienta muy ingeniosa para hacer predicciones precisas necesarias en la toma de decisiones estratégicas en las organizaciones.

Máquinas de Soporte Vectorial(SVM)

SVM es un algoritmo dentro del área de machine learning que nos ayuda a clasificar los datos en grupos similares. Cada grupo pertenecerá a una categoría, siendo este un algoritmo de aprendizaje supervisado, estas categorías son conocidas. El algoritmo SVM es uno de los algoritmos de machine learning más utilizados, gracias a que no es necesario conocer la data en detalle para utilizarlo y su habilidad de trabajar en altas dimensiones (Nti, Quarcoo, Aning, & Fosu, 2022). Para poder identificar a que categoría pertenece cada dato, se mide la distancia entre los mismos intentando identificar la separación entre categoría mediante hiperplanos. Este algoritmo SVM utiliza el concepto de margen máximo, el cual hace mención a la distancia máxima que separa las categorías. Ubicando puntos de soporte cercanos al hiperplano se puede calcular las distancias entre los mismo e ir manipulando hiperplanos hasta ajustarlos para que dividan las categorías. No siempre es factible de realizar esta separación de forma lineal por lo que el algoritmo SVM utilizando motores kernels transforma los datos dándoles una o más dimensiones, las que sean necesarias, para poder ajustar los hiperplanos dividiendo los datos en las categorías pertinentes. (Hosseini, Hosseini, & Ahi, 2021)

En la siguiente figura se puede ver con gran claridad como el algoritmo SVM usando kernel puede modificar los datos incrementando una dimensión de forma que se puede identificar con facilidad donde se encuentra la separación de los datos, mostrando a que categoría pertenece cada uno.

Figura N°4: Incremento de una dimensión por el kernel



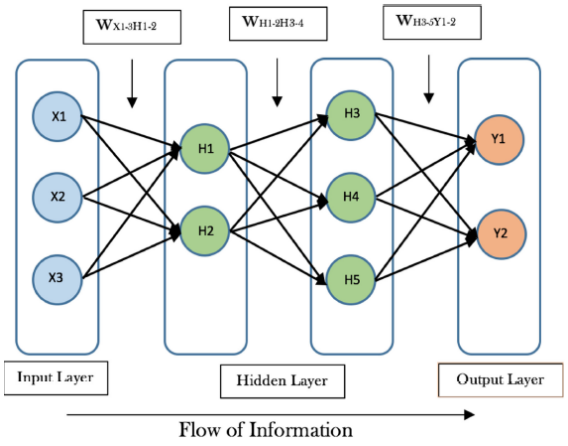
(Hosseini, Hosseini, & Ahi, 2021)

Redes Neuronales

El algoritmo de Redes Neuronales también llamado redes neurales artificiales (Hosseini, Hosseini, & Ahi, 2021), ha sido utilizado en muchos estudios médicos en la predicción de diferentes enfermedades de forma muy confiable (Site, Nurmi, & Lohan, 2021). Las redes neuronales buscan imitar las funciones del cerebro humano al momento de tomar una decisión. Cada vez que tomamos una decisión enumeramos diferentes alternativas de solución, luego identificamos criterios o factores para poder evaluar la mejor solución. Analizamos estos criterios dándole pesos de importancia a cada uno y de esa forma llegamos a una decisión. Las redes neuronales buscan hacer algo similar, al tener varias alternativas de solución o entradas de datos, le otorga pesos a diferentes criterios o nodos para poder evaluar las opciones y finalmente ofrece la salida con la decisión deseable.

A continuación, podemos apreciar en la figura N°2 como se tiene una capa de entrada, una capa de salida y entre ellas las que se denominan capas ocultas en las cuales se realizan los cálculos de los pesos para llevar a una solución.

Figura N°3: Diagrama de una Red Neural



(Hosseini, Hosseini, & Ahi, 2021)

Datos utilizados

Para evaluar las técnicas descritas se usará la base de datos "Student performance in secondary education" de estudiantes de secundaria en Portugal, disponible en el Machine Learning Repository, de la Universidad de California en Irvine. La dirección de la base de datos es: https://archive.ics.uci.edu/ml/datasets/student+performance. Los datos son de dos escuelas: Gabriel Pereira y Mousinho da Silveira y de dos asignaturas: Matemáticas y Lengua Portuguesa. Debido a que no estamos interesados en obtener conclusiones de posibles diferencias entre escuelas o entre asignaturas se decidió trabajar solo con los datos de la escuela Gabriel Pereira y solo con la asignatura de portugués. Por lo tanto, la base de datos a trabajar cuenta con 423 registros (cada registro es un estudiante) y 32 variables. Las características originales de la base de datos y la descripción de las variables se encuentran en el artículo "Using Data Mining to Predict Secondary School Student Performance" (Cortez & Silva, 2008).

Procesamiento de la información y resultados

Resultado con Árbol de decisión

Pasos realizados en la aplicación del algoritmo Árbol de decisión:

1. Subir las librerías

```
import pandas as pd
```

2. Subir la data

```
d = pd.read_csv('C:/Users/Manuela/Documents/ULima/TALLER CIENCIA DE DATOS/Trabajo/student-por.csv, sep=';')

len(d)
```

3. Calcular si el alumno aprueba la asignatura e insertar una columna con este valor en los datos, siendo 0 desaprobado y 1 aprobado

```
d['pass'] = d.apply(lambda row: 1 if (row['G1']+row['G2']+row['G3']) >= 35 else 0, axis=1)
d = d.drop(['G1', 'G2', 'G3'], axis=1)
```

4. Trabajaremos con los datos de una sola escuela: Gabriel Pereira ó GP. Eliminar los datos que no sean de la escuela GP

```
d["school"].unique()
d = d.loc[d['school'] == "GP"]
d.drop("school", inplace = True, axis = 1)
```

5. Convertir los datos cualitativos en datos numéricos (0-1)

```
d = pd.get_dummies(d, columns=['sex', 'school', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
                               'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
                               'nursery', 'higher', 'internet', 'romantic'])
d.head()
```

6. Baraja las filas

```
d = d.sample(frac=1)
```

7. División de la data en entrenamiento y prueba

```
d_train = d[:500]
d_test = d[500:]

d_train_att = d_train.drop(['pass'], axis=1)
d_train_pass = d_train['pass']

d_test_att = d_test.drop(['pass'], axis=1)
d_test_pass = d_test['pass']

d_att = d.drop(['pass'], axis=1)
d_pass = d['pass']
```

8. Número de estudiantes que pasan en todo el dataset:

```
import numpy as np
print("Pasan: %d de %d (%.2f%%)" % (np.sum(d_pass), len(d_pass), 100*float(np.sum(d_pass)) / len(d_pass)))
```

9. Ajustar a un árbol de decisión

```
from sklearn import tree
t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=5)
t = t.fit(d_train_att, d_train_pass)
```

10. Calcular y mostrar el Accuracy así como mostrar la calificación promedio +/- 2 desviaciones estandar (cubriendo el 95% de las calificaciones)

```
t.score(d_test_att, d_test_pass)
from sklearn.model_selection import cross_val_score
scores = cross_val_score(t, d_att, d_pass, cv=5)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

11. Calcular y mostrar el Accuracy para profundidades del 1 al 19.

```
for max_depth in range(1, 20):
    t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=max_depth)
    scores = cross_val_score(t, d_att, d_pass, cv=5)
    print("Max depth: %d, Accuracy: %0.2f (+/- %0.2f)" % (max_depth, scores.mean(), scores.std() * 2))

depth_acc = np.empty((19,3), float)
i = 0
for max_depth in range(1, 20):
    t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=max_depth)
    scores = cross_val_score(t, d_att, d_pass, cv=5)
    depth_acc[i,0] = max_depth
    depth_acc[i,1] = scores.mean()
    depth_acc[i,2] = scores.std() * 2
    i += 1

depth_acc
```

12. Mostrar el gráfico

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.errorbar(depth_acc[:,0], depth_acc[:,1], yerr=depth_acc[:,2])
plt.show()
```

13. Calcular y mostrar la matriz de confusión, con la máxima profundidad (2) que tiene el nivel de precisión mayor.

```
from sklearn import tree
t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=2)
t = t.fit(d_train_att, d_train_pass)
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
#t = t.fit(d_train_att, d_train_pass)
y_pred  = t.predict(d_train_att)
cm = confusion_matrix(d_train_pass, y_pred)
print("Matriz de confusión")
print(cm)
```

14. Calcular y mostrar la curva de ROC

```
r_probs = [0 for _ in range(len(d_train_pass))]
nb_probs = t.predict_proba(d_train_att)
nb_probs = nb_probs[:, 1]

r_auc = roc_auc_score(d_train_pass, r_probs)
nb_auc = roc_auc_score(d_train_pass, nb_probs)

print("AUROC = %.3f" % (nb_auc))

nb_fpr, nb_tpr, _ = roc_curve(d_train_pass, nb_probs)
```

```
plt.plot(nb_fpr, nb_tpr, marker=".", label="AUROC = %.3f" % nb_auc)
plt.title("ROC plot")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

Luego de verificar que el mayor nivel de precisión se obtiene con una profundidad máxima de 2 niveles, se ejecuta el algoritmo con esta profundidad y se obtiene la siguiente matriz de confusión.

Tabla 1: Matriz de confusión – Arboles de decisión:

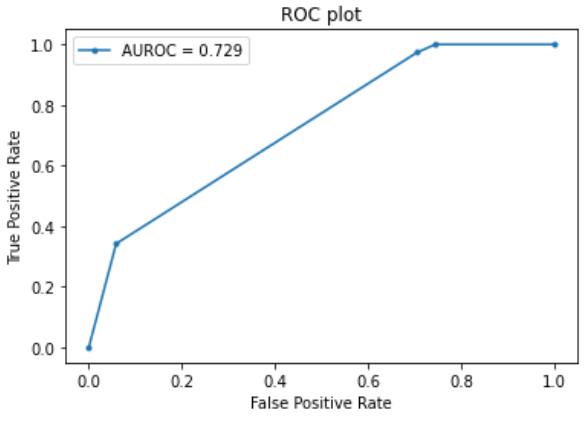
		Predicción	
		0	1
Valor Real	0	15	36
	1	2	74

El modelo ha acertado 74 de los casos de aprobar la asignatura, y 15 de los casos de no aprobar la signatura. Sin embargo, podemos ver 2 falsos positivos (error tipo I) en donde aprobó la asignatura, pero el modelo clasifico que no aprobó y 36 falsos negativos (error tipo II) en donde no aprobó la asignatura, pero el modelo clasifico que si aprobó.

Así mismo podemos observar el nivel de precisión (Accuracy) calculado el cual es de: 0.73

Del mismo modo podemos ver la curva de ROC.

Figura N°5: Curva de ROC – Arboles de decisión:



Resultado con Bosque aleatorio

Pasos realizados en la aplicación del algoritmo de Random Forest:

1. Subir las librerías

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
```

2. Definir la función rf_bch para:
   a. Normalizar la data.
   b. Dividir la data en entrenamiento y testeo.
   c. Crear el modelo de random forest.
   d. Entrenar con la data de entrenamiento.
   e. Calcular y mostrar la matriz de confusión.
   f. Calcular y mostrar la curva de ROC.
   g. Medir el nivel de precisión

```
def rf_bch(d,i,j,nom):
  var_dep = d[nom]
  var_ind = d.drop(nom,axis= 1)
  var_ind_norm = StandardScaler().fit_transform(var_ind)
  var_ind_ent, var_ind_test, var_dep_ent, var_dep_test = train_test_split(var_ind_norm,var_dep,test_size=0.3)
#test_siza=0.3 -- 30% de la data se usa para entrenar
  rf= RandomForestClassifier(n_estimators=10,max_depth=4)
  #activation="relu" -- activar una función para cada nodo. max_iter=3000 -- máximo 3000 iteraciones para mejorar.
  rf.fit(var_ind_ent,var_dep_ent)

  y_pred = rf.predict(var_ind_test)
  cm = confusion_matrix(var_dep_test, y_pred)
  print("Matriz de confusión")
  print(cm)


  r_probs = [0 for _ in range(len(var_dep_test))]
  nb_probs = rf.predict_proba(var_ind_test)
  nb_probs = nb_probs[:, 1]

  r_auc = roc_auc_score(var_dep_test, r_probs)
  nb_auc = roc_auc_score(var_dep_test, nb_probs)

  print("AUROC = %.3f" % (nb_auc))

  nb_fpr, nb_tpr, _ = roc_curve(var_dep_test, nb_probs)

  plt.plot(nb_fpr, nb_tpr, marker=".", label="AUROC = %.3f" % nb_auc)
  plt.title("ROC plot")
  plt.xlabel("False Positive Rate")
  plt.ylabel("True Positive Rate")
  plt.legend()
  plt.show()

  return rf.score(var_ind_test,var_dep_test)
```

3. Subir la data

```
d = pd.read_csv("C:/Users/Manuela/Documents/ULima/TALLER CIENCIA DE DATOS/Trabajo/student-por.csv", sep=';')
```

4. Calcular si el alumno aprueba la asignatura e insertar una columna con este valor en los datos, siendo 0 desaprobado y 1 aprobado

```
d['pass'] = d.apply(lambda row: 1 if (row['G1']+row['G2']+row['G3']) >= 35 else 0, axis=1)
d = d.drop(['G1', 'G2', 'G3'], axis=1)
d.head()
```

5. Trabajaremos con los datos de una sola escuela: Gabriel Pereira ó GP. Eliminar los datos que no sean de la escuela GP

```
d["school"].unique()
d = d.loc[d['school'] == "GP"]
d.drop("school", inplace = True, axis = 1)
```

6. Convertir los datos cualitativos en datos numéricos (0-1)

```
d = pd.get_dummies(d, columns=['sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
                               'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
                               'nursery', 'higher', 'internet', 'romantic'])
d.head()
```

7. Definir la función rf_arq para:
   a. Normalizar la data.
   b. Dividir la data en entrenamiento y testeo.
   c. Crear el modelo de random forest.
   d. Entrenar con la data de entrenamiento.
   e. Medir el nivel de precisión

```
def rf_arq(d,i,j,nom):
  var_dep = d[nom]
  var_ind = d.drop(nom,axis= 1)
  var_ind_norm = StandardScaler().fit_transform(var_ind)
  var_ind_ent, var_ind_test, var_dep_ent, var_dep_test = train_test_split(var_ind_norm,var_dep,test_size=0.3)
#test_siza=0.3 -- 30% de la data se usa para entrenar
  rf= RandomForestClassifier(n_estimators=i,max_depth=j)
  rf.fit(var_ind_ent,var_dep_ent)

  return rf.score(var_ind_test,var_dep_test)
```

8. Llamamos a la función rf_arq dentro de un loop doble de 10 a 101 en pasos de 10 cada uno, viendo diferentes arquitecturas del random forest y mostramos el resultado del mayor nivel de precisión y su arquitectura

```
mayor = 0
for i in range(10,101,10):
  for j in range(10,101,10):
```

```
    A = rf_arq(d,i,j,'pass')
    if A > mayor:
       mayor = A
       n1 = i
       n2 = j
print("Arquitectura: %dx%d, Accuracy: %0.4f" % (n1, n2, mayor ))
```

9. Usando la arquitectura que nos dio en mejor nivel de precisión llamamos a la función rf_bch y mostramos el resultado del nivel de precisión, matriz de confusión y curva de ROC

```
A = rf_bch(d,n1,n2,'pass')
print("Arquitectura: %dx%d, Accuracy: %0.4f" % (n1, n2,  A ))
```

Luego de ejecutar el algoritmo con los datos se puede apreciar el nivel de precisión por arquitectura del random forest. Al seleccionar la arquitectura que mejor se ajusta: n_estimator= 40, max_depth = 20, Accuracy: 0.8031 podemos ver la siguiente matriz de confusión.

Tabla 2: Matriz de confusión – Random Forest:

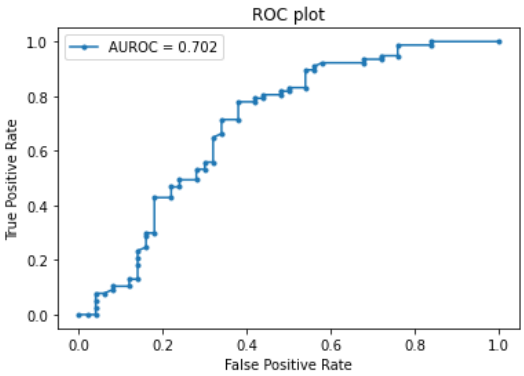
		Predicción	
		0	1
Valor Real	0	16	34
	1	6	71

El modelo ha acertado 71 de los casos de aprobar la asignatura, y 16 de los casos de no aprobar la signatura. Sin embargo, podemos ver 6 falsos positivos (error tipo I) en donde aprobó la asignatura, pero el modelo clasifico que no aprobó y 34 falsos negativos (error tipo II) en donde no aprobó la asignatura, pero el modelo clasifico que si aprobó.

Así mismo podemos observar el nivel de precisión (Accuracy) calculado el cual es de: 0.6850

Del mismo modo podemos ver la curva de ROC.

Figura N°6: Curva de ROC – Random Forest:



Resultado con Máquinas de Soporte Vectorial(SVM)

Pasos realizados en la aplicación del algoritmo SVM:

1. Subir las librerías

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import svm
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
```

2. Definir la función para:
   a. Normalizar la data.
   b. Dividir la data en entrenamiento y testeo.
   c. Crear el modelo SVM.
   d. Entrenar con la data de entrenamiento.
   e. Calcular y mostrar la matriz de confusión.
   f. Medir el nivel de precisión

```
def svm_bch(d,nom):
  var_dep = d[nom]
  var_ind = d.drop(nom,axis= 1)
  var_ind_norm = StandardScaler().fit_transform(var_ind)
  var_ind_ent, var_ind_test, var_dep_ent, var_dep_test = train_test_split(var_ind_norm,var_dep,test_size=0.3)
  mi_svm = svm.SVC()
  mi_svm.fit(var_ind_ent,var_dep_ent)
  y_pred  = mi_svm.predict(var_ind_test)
  cm = confusion_matrix(var_dep_test, y_pred)
  print("Matriz de confusión")
  print(cm)

  return mi_svm.score(var_ind_test,var_dep_test)
```

3. Subir la data

```
d = pd.read_csv("C:/Users/Manuela/Documents/ULima/TALLER CIENCIA DE DATOS/Trabajo/student-por.csv", sep=';')
```

4. Calcular si el alumno aprueba la asignatura e insertar una columna con este valor en los datos, siendo 0 desaprobado y 1 aprobado

```
d['pass'] = d.apply(lambda row: 1 if (row['G1']+row['G2']+row['G3']) >= 35 else 0, axis=1)
d = d.drop(['G1', 'G2', 'G3'], axis=1)
d.head()
```

5. Trabajaremos con los datos de una sola escuela: Gabriel Pereira ó GP. Eliminar los datos que no sean de la escuela GP

```
d["school"].unique()
d = d.loc[d['school'] == "GP"]
d.drop("school", inplace = True, axis = 1)
```

6. Convertir los datos cualitativos en datos numéricos (0-1)

```
d = pd.get_dummies(d, columns=['sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
                               'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
                               'nursery', 'higher', 'internet', 'romantic'])
d.head()
```

7. Llamamos a la función y mostramos el resultado del nivel de precisión y la matriz de confusión

```
A = svm_bch(d,'pass')
print("Accuracy: %0.4f" % ( A ))
```

Luego de ejecutar el algoritmo con los datos se puede apreciar la siguiente matriz de confusión.

Tabla 3: Matriz de confusión – SVM:

		Predicción	
		0	1
Valor Real	0	25	28
	1	7	67

El modelo ha acertado 67 de los casos de aprobar la asignatura, y 25 de los casos de no aprobar la signatura. Sin embargo, podemos ver 7 falsos positivos (error tipo I) en donde aprobó la asignatura, pero el modelo clasifico que no aprobó y 28 falsos negativos (error tipo II) en donde no aprobó la asignatura, pero el modelo clasifico que si aprobó.

Así mismo podemos observar el nivel de precisión (Accuracy) calculado el cual es de: 0.7244

Resultado con Redes Neuronales

Pasos realizados en la aplicación del algoritmo de Redes Neuronales:

1. Subir las librerías

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
```

2. Definir la función rn_bch para:
    h. Normalizar la data.
    i. Dividir la data en entrenamiento y testeo.
    j. Crear el modelo de red neuronal.
    k. Entrenar con la data de entrenamiento.
    l. Calcular y mostrar la matriz de confusión.
    m. Calcular y mostrar la curva de ROC.
    n. Medir el nivel de precisión

```
def rn_bch(d,i,j,nom):
  var_dep = d[nom]
  var_ind = d.drop(nom,axis= 1)
  var_ind_norm = StandardScaler().fit_transform(var_ind)
  var_ind_ent, var_ind_test, var_dep_ent, var_dep_test = train_test_split(var_ind_norm,var_dep,test_size=0.3)
#test_siza=0.3 -- 30% de la data se usa para entrenar
  rn= MLPClassifier(hidden_layer_sizes=(i,j),activation='relu',max_iter=3000) #hidden_layer_sizes=(i,j) --- i nodos en
la 1era capa y j nodos en la 2da capa, arquitectura de 4x4.
  #activation="relu" -- activar una función para cada nodo. max_iter=3000 -- máximo 3000 iteraciones para mejorar.
  rn.fit(var_ind_ent,var_dep_ent)
  y_pred = rn.predict(var_ind_test)
  cm = confusion_matrix(var_dep_test, y_pred)
  print("Matriz de confusión")
  print(cm)


  r_probs = [0 for _ in range(len(var_dep_test))]
  nb_probs = rn.predict_proba(var_ind_test)
  nb_probs = nb_probs[:, 1]

  r_auc = roc_auc_score(var_dep_test, r_probs)
  nb_auc = roc_auc_score(var_dep_test, nb_probs)

  print("AUROC = %.3f" % (nb_auc))

  nb_fpr, nb_tpr, _ = roc_curve(var_dep_test, nb_probs)

  plt.plot(nb_fpr, nb_tpr, marker=".", label="AUROC = %.3f" % nb_auc)
  plt.title("ROC plot")
  plt.xlabel("False Positive Rate")
  plt.ylabel("True Positive Rate")
  plt.legend()
```

```
    plt.show()

    return rn.score(var_ind_test,var_dep_test)
```

3. Subir la data

```
d = pd.read_csv("C:/Users/Manuela/Documents/ULima/TALLER CIENCIA DE DATOS/Trabajo/student-por.csv", sep=';')
```

4. Calcular si el alumno aprueba la asignatura e insertar una columna con este valor en los datos, siendo 0 desaprobado y 1 aprobado

```
d['pass'] = d.apply(lambda row: 1 if (row['G1']+row['G2']+row['G3']) >= 35 else 0, axis=1)
d = d.drop(['G1', 'G2', 'G3'], axis=1)
d.head()
```

5. Trabajaremos con los datos de una sola escuela: Gabriel Pereira ó GP. Eliminar los datos que no sean de la escuela GP

```
d["school"].unique()
d = d.loc[d['school'] == "GP"]
d.drop("school", inplace = True, axis = 1)
```

6. Convertir los datos cualitativos en datos numéricos (0-1)

```
d = pd.get_dummies(d, columns=['sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
                    'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
                    'nursery', 'higher', 'internet', 'romantic'])
d.head()
```

7. Definir la función rn_arq para:
    f. Normalizar la data.
    g. Dividir la data en entrenamiento y testeo.
    h. Crear el modelo de red neuronal.
    i. Entrenar con la data de entrenamiento.
    j. Medir el nivel de precisión

```
def rn_arq(d,i,j,nom):
    var_dep = d[nom]
    var_ind = d.drop(nom,axis= 1)
    var_ind_norm = StandardScaler().fit_transform(var_ind)
    var_ind_ent, var_ind_test, var_dep_ent, var_dep_test = train_test_split(var_ind_norm,var_dep,test_size=0.3) #test_siza=0.3 -- 30% de la data se usa para entrenar
    rn= MLPClassifier(hidden_layer_sizes=(i,j),activation='relu',max_iter=3000)  #hidden_layer_sizes=(i,j) --- i nodos en la 1era capa y j nodos en la 2da capa, arquitectura de 4x4.
    #activation="relu" -- activar una función para cada nodo. max_iter=3000  -- máximo 3000 iteraciones para mejorar.
    rn.fit(var_ind_ent,var_dep_ent)

    return rn.score(var_ind_test,var_dep_test)
```

8. Llamamos a la función rn_arq dentro de un loop doble de 3 a 50 cada uno, viendo diferentes arquitecturas de la red neuronal y mostramos el resultado del mayor nivel de precisión y su arquitectura

```
mayor = 0
for i in range(3,50):
   for j in range(3,50):
      A = rn_arq(d,i,j,'pass')
      if A > mayor:
         mayor = A
         n1 = i
         n2 = j
print("Arquitectura: %dx%d, Accuracy: %0.4f" % (n1, n2, mayor ))
```

9. Usando la arquitectura que nos dio en mejor nivel de precisión llamamos a la función rn_bch y mostramos el resultado del nivel de precisión, matriz de confusión y curva de ROC

```
A = rn_bch(d,n1,n2,'pass')
print("Arquitectura: %dx%d, Accuracy: %0.4f" % ( n1, n2, A ))
```

Luego de ejecutar el algoritmo con los datos se puede apreciar el nivel de precisión por arquitectura de la red neuronal (Ver anexos 1, 2 y 3). Al seleccionar la arquitectura que mejor se ajusta: Arquitectura: 23x45, Accuracy: 0.7874 podemos ver la siguiente matriz de confusión.

Tabla 4: Matriz de confusión – Red Neuronal:

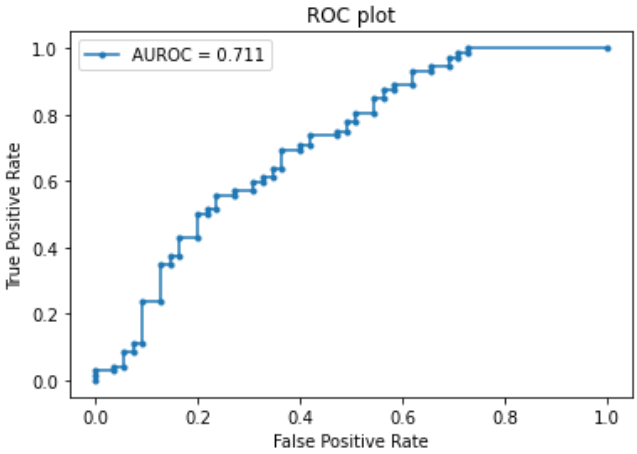
		Predicción	
		0	1
Valor Real	0	26	29
	1	14	58

El modelo ha acertado 58 de los casos de aprobar la asignatura, y 26 de los casos de no aprobar la signatura. Sin embargo, podemos ver 14 falsos positivos (error tipo I) en donde aprobó la asignatura, pero el modelo clasifico que no aprobó y 29 falsos negativos (error tipo II) en donde no aprobó la asignatura, pero el modelo clasifico que si aprobó.

Así mismo podemos observar el nivel de precisión (Accuracy) calculado el cual es de: 0.6614

Del mismo modo podemos ver la curva de ROC.

Figura N°7: Curva de ROC – Red Neuronal:



Discusión de los resultados

Para poder comprar los 4 algoritmos de machine learning utilizados usaremos las métricas más comunes dentro del aprendizaje supervisado que son la Matriz de Confusión, el nivel de precisión y la curva de ROC (Receiver Operating Characteristic) (Irsoy, Yildiz , & Alpaydin, 2012; Tariq , Hamzah , Foo NG, Wang, & Ibrahim, 2021) para evidenciar nuestra selección de algoritmo. Podemos ver en la siguiente tabla el nivel de precisión y el indicador AUROC que proviene de la curva de ROC.

Tabla N°5:

Algoritmo	Accuracy	AUROC
Árbol de decisiones	0.7300	0.729
Bosque aleatorio	0.8031	0.702
Máquinas de Soporte Vectorial	0.7244	----------
Redes Neuronales	0.7874	0.711

Podemos apreciar que con el algoritmo de bosques aleatorios se logra un nivel de precisión mayor de 0.8031, utilizando como hiperparámetros 40 árboles y una profundidad máxima de 20 niveles.

Si bien el algoritmo de árboles de decisiones como el de redes neuronales tienen un valor mayor en el indicador AUROC de la curva de ROC de 0.729 y 0.711 respectivamente. Podemos apreciar que el algoritmo de bosques aleatorios al ser de 0.702, es mayor al umbral de 0.7 dándonos una estimación aceptable.

Aplicaciones de ML & IA a las carreras de la FCEE

Consideramos que el potencial de uso de Ciencia de Datos puede darse en todos los campos de acción de las carreras de la FCEE. Muchas aplicaciones son comunes a todas las carreras, de las que podemos destacar, por ejemplo:

1. Análisis de sentimientos. El objetivo fundamental de esta aplicación es detectar las emociones y opiniones de los clientes o usuarios sobre un bien, servicio, candidato político o cualquier objeto sujeto de emoción u opinión, a través del análisis de textos publicados en redes sociales u otros medios con técnicas de Procesamiento del Lenguaje Natural (NLP). (Appiahene, Afrifa, Kyei, & Nimbe, 2022).
2. Sistemas de recomendación. tienen como objetivo predecir las preferencias de los usuarios por un determinado bien o servicio y en base a ello proporcionarle servicios personalizados. Hay varios tipos de sistemas de recomendación, como: en función de la similitud entre diferentes productos, en función del comportamiento de otros usuarios, según el conocimiento básico de los usuarios, los elementos y las relaciones entre los elementos, y en función de más de un enfoque de filtrado. (Peng, 2022; Huang & Rust, 2021).

En las tablas que siguen se proponen aplicaciones y técnicas con alto potencial para las carreras de la FCEE:

Tabla N°6: Aplicaciones potenciales y técnicas de Aprendizaje Automático para las carreras de la FCEE

		Carreras					Técnicas								
N°	Aplicación	Adm.	Cont.	Eco.	Mkt.	NNII	Algoritmos genéticos	Árboles de decisión	Boosting	Clustering	Deep learning	Lógica difusa	Ontologías	Random forest	Redes neuronales
1	Análisis contable y financiero		X					X						X	X
2	Análisis de costos		X					X						X	X
3	Análisis demográfico			X						X		X	X	X	
4	Análisis de riesgo crediticio		X	X							X				X
5	Análisis de riesgo en seguros		X	X							X				
6	Análisis de riesgo hipotecario		X	X							X				
7	Análisis de segmentación	X		X	X	X				X	X			X	
8	Análisis de sentimientos	X		X	X	X	X				X	X	X		
9	Análisis de transparencia y rendición de cuentas		X	X				X						X	X
10	Análisis exploratorio de la dinámica del mercado de valores			X			X	X		X	X	X	X	X	X
11	Detección de fraudes en seguros		X	X			X	X			X			X	X
12	Determinación de precios de mercado en línea	X			X	X		X	X		X			X	X
13	Diseño y optimización de aprovisionamiento y subastas	X				X	X	X		X	X			X	X
14	Estrategias y administración de portafolios de inversión	X		X		X	X	X		X		X		X	X
15	Gestión de inventarios	X					X	X		X	X			X	X
16	Gestión de proyectos	X	X	X	X	X		X		X				X	X
17	Loan pricing (determinación de tasas de interés para préstamos)			X			X		X					X	X
18	Optimización de promociones (advertising)	X			X	X	X		X					X	X
19	Predicción de indicadores económicos			X					X		X				
20	Predicción de inflación			X					X		X				
21	Predicción de pedidos pendientes	X				X		X	X		X			X	X
22	Predicción de ventas	X			X	X		X	X		X			X	X
23	Predicción del valor de las acciones en la bolsa de valores			X				X	X		X			X	X
24	Pronóstico de la demanda en la cadena de suministro	X				X	X		X		X	X	X		X
25	Recomendación de noticias en línea	X			X	X		X			X			X	
26	Simulación de procesos	X	X	X	X	X	X	X						X	X
27	Sistemas de recomendación	X	X	X	X	X		X			X			X	X
28	Valorización de opciones			X				X	X		X			X	X

Referencias

Appiahene, P., Afrifa, S., Kyei, E. A., & Nimbe, P. (2022). Understanding the Uses, Approaches and Applications of Sentiment Analysis. *Research Square*.

Biau, G. (2012). Analysis of a random forests model. *The Journal of Machine Learning Research.*, 1063-1095.

Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5-32.

Breiman, L. (2004). *Consistency for a simple model of random forests.* Statistics Department. . Berkeley: University of California at Berkeley.

Charbuty, B., & Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends, 2*(1), 20-28.

Cortez, P., & Silva, A. (2008). Using data mining to predict secondary school student performance. *Dep. Information Systems/Algoritmi R&D Centre University of Minho*.

Hartshorn, S. (2016). *Machine learning with random forests and decision trees: A Visual guide for beginners.*

Hosseini, M.-P., Hosseini, A., & Ahi, K. (2021). A Review on Machine Learning for EEG Signal Processing in Bioengineering. *IEEE Reviews in Biomedical Engineering , 14*, 204-218. doi:10.1109/RBME.2020.2969915

Huang, M., & Rust, R. (2021). A strategic framework for artificial intelligence in marketing. *J. of the Acad. Mark. Sci., 49*, 30–50. doi:10.1007/s11747-020-00749-9

Irsoy, O., Yildiz , O. T., & Alpaydin, E. (2012). Design and Analysis of Classifier Learning Experiments in Bioinformatics: Survey and Case Studies. *IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, 9*(6), 1663-1675. doi:10.1109/TCBB.2012.117

Maimon, O. Z., & Rokach, L. (2014). Data mining with decision trees: theory and applications. *World scientific, 81*.

Nti, I. K., Quarcoo, J. A., Aning, J., & Fosu, G. K. (2022). A Mini-Review of Machine Learning in Big Data Analytics: Applications, Challenges, and Prospects. *Big Data Minning and Analytics, 5*(2), 81–97. doi:10.26599/BDMA.2021.9020028

Peng, Y. (2022). A Survey on Modern Recommendation System based on Big Data. *arXiv preprint arXiv:2206.02631.*

Site, A., Nurmi, J., & Lohan, E. S. (2021). Systematic Review on Machine-Learning Algorithms Used in Wearable-Based eHealth Data Analysis. *IEEE Access, 9*, 112221-112235. doi:10.1109/ACCESS.2021.3103268

Tariq , N., Hamzah , R. A., Foo NG, T., Wang, S. L., & Ibrahim, H. (2021). Quality Assessment Methods to Evaluate the Performance of Edge Detection Algorithms for Digital Image: A Systematic Literature Review. *IEEE Access, 9*, 87763-87776. doi:10.1109/ACCESS.2021.3089210