Employee Retention Prediction

This project explores the application of deep learning to predict employee turnover and improve retention rates. The model utilizes KerasClassifier, a powerful deep learning library, to identify patterns in employee data that correlate with departure. To combat overfitting, a common challenge in machine learning, Dropout regularization is implemented. Dropout randomly drops out a percentage of neurons during training, preventing the model from becoming overly reliant on specific features and enhancing its generalizability to unseen data. This refined model aims to deliver more accurate predictions of employee flight risk, allowing HR departments to proactively implement targeted retention strategies.

In [3]:
```
#importing libraries

import pandas as pd
import numpy as np
df=pd.read_csv('Downloads/hr.csv')
```

In [5]:
```
df.head()
```

Out[5]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	department	salary
0	0.38	0.53	2	157	3	0	1	0	sales	low
1	0.80	0.86	5	262	6	0	1	0	sales	medium
2	0.11	0.88	7	272	4	0	1	0	sales	medium
3	0.72	0.87	5	223	5	0	1	0	sales	low
4	0.37	0.52	2	159	3	0	1	0	sales	low

In [7]:
```
#convert categorical to numeric values
feats=['department','salary']
df_final=pd.get_dummies(df,columns=feats,drop_first=True)
```

In [9]:
```
df_final
```

Out[9]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	department_RandD	department_accounting	department_hr	department_management
0	0.38	0.53	2	157	3	0	1	0	0	0	0	0
1	0.80	0.86	5	262	6	0	1	0	0	0	0	0
2	0.11	0.88	7	272	4	0	1	0	0	0	0	0
3	0.72	0.87	5	223	5	0	1	0	0	0	0	0
4	0.37	0.52	2	159	3	0	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
14994	0.40	0.57	2	151	3	0	1	0	0	0	0	0
14995	0.37	0.48	2	160	3	0	1	0	0	0	0	0
14996	0.37	0.53	2	143	3	0	1	0	0	0	0	0
14997	0.11	0.96	6	280	4	0	1	0	0	0	0	0
14998	0.37	0.52	2	158	3	0	1	0	0	0	0	0

14999 rows × 19 columns

In [12]:
```
#Split data into training and test set
from sklearn.model_selection import train_test_split
X=df_final.drop(['left'],axis=1).values
y=df_final['left'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Transforming the data

In [16]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Build the artificial neural network

In [18]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

In [19]:
```
classifier=Sequential()
```

In [20]:
```
classifier.add(Dense(9, kernel_initializer = "uniform",activation = "relu", input_dim=18))
```

In [21]:
```
classifier.add(Dense(1, kernel_initializer = "uniform",activation = "sigmoid"))
```

In [22]:
```
classifier.compile(optimizer= "adam",loss = "binary_crossentropy",metrics = ["accuracy"])
```

In [24]:
```
classifier.fit(X_train, y_train, batch_size = 10, epochs = 8)
```

```
Epoch 1/8
1125/1125 [==============================] - 2s 1ms/step - loss: 0.2646 - accuracy: 0.8600
Epoch 2/8
1125/1125 [==============================] - 2s 1ms/step - loss: 0.2251 - accuracy: 0.9378
Epoch 3/8
1125/1125 [==============================] - 1s 1ms/step - loss: 0.2053 - accuracy: 0.9477
Epoch 4/8
1125/1125 [==============================] - 2s 1ms/step - loss: 0.1925 - accuracy: 0.9520
Epoch 5/8
1125/1125 [==============================] - 1s 1ms/step - loss: 0.1839 - accuracy: 0.9540
Epoch 6/8
1125/1125 [==============================] - 2s 1ms/step - loss: 0.1780 - accuracy: 0.9553
Epoch 7/8
1125/1125 [==============================] - 1s 1ms/step - loss: 0.1738 - accuracy: 0.9540
Epoch 8/8
1125/1125 [==============================] - 1s 1ms/step - loss: 0.1713 - accuracy: 0.9542
```

Out[24]:
```
<keras.callbacks.History at 0x23e22cbc9d0>
```

Improving the Model Accuracy

In [27]:
```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
```

In [28]:
```
def make_classifier():
    classifier = Sequential()
    classifier.add(Dense(9, kernel_initializer = "uniform", activation = "relu", input_dim=18))
    classifier.add(Dense(1, kernel_initializer = "uniform", activation = "sigmoid"))
    classifier.compile(optimizer= "adam",loss = "binary_crossentropy",metrics = ["accuracy"])
    return classifier
```

In [29]:
```
classifier = KerasClassifier(build_fn = make_classifier, batch_size=10, nb_epoch=2)
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_5444\122607991.py:1: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  classifier = KerasClassifier(build_fn = make_classifier, batch_size=10, nb_epoch=2)
```

In [30]:
```
accuracies = cross_val_score(estimator = classifier,X = X_train,y = y_train,cv = 10,n_jobs = -1)
```

In [31]:
```
mean = accuracies.mean()
mean
```

Out[31]:
```
0.8038005590438843
```

Adding Dropout Regularization to Fight Over-Fitting

In [32]:
```
from keras.layers import Dropout

classifier = Sequential()
classifier.add(Dense(9, kernel_initializer = "uniform", activation = "relu", input_dim=18))
classifier.add(Dropout(rate = 0.1))
classifier.add(Dense(1, kernel_initializer = "uniform", activation = "sigmoid"))
classifier.compile(optimizer= "adam",loss = "binary_crossentropy",metrics = ["accuracy"])
```

In [33]:
```
from sklearn.model_selection import GridSearchCV
def make_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(9, kernel_initializer = "uniform", activation = "relu", input_dim=18))
    classifier.add(Dense(1, kernel_initializer = "uniform", activation = "sigmoid"))
    classifier.compile(optimizer= optimizer,loss = "binary_crossentropy",metrics = ["accuracy"])
    return classifier
```

In [34]:
```
classifier = KerasClassifier(build_fn = make_classifier)
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_5444\10115898.py:1: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  classifier = KerasClassifier(build_fn = make_classifier)
```

In [35]:
```
params = {
    'batch_size':[20,35],
    'epochs':[2,3],
    'optimizer':['adam','rmsprop']
}
```

In [36]:
```
grid_search = GridSearchCV(estimator=classifier,
                           param_grid=params,
                           scoring="accuracy",
                           cv=2)
```

In [37]:
```
grid_search = grid_search.fit(X_train,y_train)
```

```
Epoch 1/2
282/282 [==============================] - 1s 1ms/step - loss: 0.5718 - accuracy: 0.7600
Epoch 2/2
282/282 [==============================] - 0s 2ms/step - loss: 0.3980 - accuracy: 0.8010
176/176 [==============================] - 0s 976us/step
Epoch 1/2
282/282 [==============================] - 1s 1ms/step - loss: 0.5681 - accuracy: 0.7662
Epoch 2/2
282/282 [==============================] - 0s 2ms/step - loss: 0.3923 - accuracy: 0.8032
176/176 [==============================] - 0s 1ms/step
Epoch 1/2
282/282 [==============================] - 1s 2ms/step - loss: 0.5768 - accuracy: 0.7587
Epoch 2/2
282/282 [==============================] - 0s 1ms/step - loss: 0.4247 - accuracy: 0.7601
176/176 [==============================] - 0s 1ms/step
Epoch 1/2
282/282 [==============================] - 1s 1ms/step - loss: 0.5836 - accuracy: 0.7625
Epoch 2/2
282/282 [==============================] - 0s 1ms/step - loss: 0.4163 - accuracy: 0.7995
176/176 [==============================] - 0s 1ms/step
Epoch 1/3
282/282 [==============================] - 1s 1ms/step - loss: 0.5940 - accuracy: 0.7553
Epoch 2/3
282/282 [==============================] - 0s 2ms/step - loss: 0.3944 - accuracy: 0.8425
Epoch 3/3
282/282 [==============================] - 0s 1ms/step - loss: 0.3001 - accuracy: 0.8931
176/176 [==============================] - 0s 1ms/step
Epoch 1/3
282/282 [==============================] - 1s 1ms/step - loss: 0.5916 - accuracy: 0.7639
Epoch 2/3
282/282 [==============================] - 0s 2ms/step - loss: 0.3762 - accuracy: 0.8613
Epoch 3/3
282/282 [==============================] - 0s 1ms/step - loss: 0.2814 - accuracy: 0.9024
176/176 [==============================] - 0s 1ms/step
Epoch 1/3
282/282 [==============================] - 1s 1ms/step - loss: 0.5812 - accuracy: 0.7594
Epoch 2/3
282/282 [==============================] - 0s 1ms/step - loss: 0.4317 - accuracy: 0.7601
Epoch 3/3
282/282 [==============================] - 0s 1ms/step - loss: 0.3823 - accuracy: 0.7601
176/176 [==============================] - 0s 1ms/step
Epoch 1/3
282/282 [==============================] - 1s 1ms/step - loss: 0.5759 - accuracy: 0.7612
Epoch 2/3
282/282 [==============================] - 0s 2ms/step - loss: 0.4176 - accuracy: 0.7897
Epoch 3/3
282/282 [==============================] - 0s 1ms/step - loss: 0.3638 - accuracy: 0.8027
176/176 [==============================] - 0s 995us/step
Epoch 1/2
161/161 [==============================] - 1s 2ms/step - loss: 0.6382 - accuracy: 0.7536
Epoch 2/2
161/161 [==============================] - 0s 1ms/step - loss: 0.4789 - accuracy: 0.8019
176/176 [==============================] - 0s 1ms/step
Epoch 1/2
161/161 [==============================] - 1s 1ms/step - loss: 0.6221 - accuracy: 0.7611
Epoch 2/2
161/161 [==============================] - 0s 1ms/step - loss: 0.4500 - accuracy: 0.7831
176/176 [==============================] - 0s 1ms/step
Epoch 1/2
161/161 [==============================] - 1s 2ms/step - loss: 0.6258 - accuracy: 0.7580
Epoch 2/2
161/161 [==============================] - 0s 1ms/step - loss: 0.4867 - accuracy: 0.7827
176/176 [==============================] - 0s 987us/step
Epoch 1/2
161/161 [==============================] - 1s 1ms/step - loss: 0.6215 - accuracy: 0.7637
Epoch 2/2
161/161 [==============================] - 0s 2ms/step - loss: 0.4840 - accuracy: 0.7634
176/176 [==============================] - 0s 1ms/step
Epoch 1/3
161/161 [==============================] - 1s 2ms/step - loss: 0.6260 - accuracy: 0.7566
Epoch 2/3
161/161 [==============================] - 0s 1ms/step - loss: 0.4663 - accuracy: 0.7601
Epoch 3/3
161/161 [==============================] - 0s 1ms/step - loss: 0.4157 - accuracy: 0.7601
176/176 [==============================] - 0s 970us/step
Epoch 1/3
161/161 [==============================] - 1s 1ms/step - loss: 0.6251 - accuracy: 0.7625
Epoch 2/3
161/161 [==============================] - 0s 1ms/step - loss: 0.4599 - accuracy: 0.7623
Epoch 3/3
161/161 [==============================] - 0s 1ms/step - loss: 0.3971 - accuracy: 0.8062
176/176 [==============================] - 0s 960us/step
Epoch 1/3
161/161 [==============================] - 1s 1ms/step - loss: 0.6327 - accuracy: 0.7569
Epoch 2/3
161/161 [==============================] - 0s 1ms/step - loss: 0.4998 - accuracy: 0.7911
Epoch 3/3
161/161 [==============================] - 0s 2ms/step - loss: 0.4118 - accuracy: 0.7996
176/176 [==============================] - 0s 1ms/step
Epoch 1/3
161/161 [==============================] - 1s 1ms/step - loss: 0.6220 - accuracy: 0.7612
Epoch 2/3
161/161 [==============================] - 0s 1ms/step - loss: 0.4754 - accuracy: 0.7687
Epoch 3/3
161/161 [==============================] - 0s 1ms/step - loss: 0.4073 - accuracy: 0.8021
176/176 [==============================] - 0s 1ms/step
Epoch 1/3
563/563 [==============================] - 1s 1ms/step - loss: 0.4924 - accuracy: 0.7766
Epoch 2/3
563/563 [==============================] - 1s 1ms/step - loss: 0.3309 - accuracy: 0.8138
Epoch 3/3
563/563 [==============================] - 1s 2ms/step - loss: 0.2718 - accuracy: 0.8451
```

In [38]:
```
best_param = grid_search.best_params_
best_accuracy = grid_search.best_score_
```

In [39]:
```
best_param
```

Out[39]:
```
{'batch_size': 20, 'epochs': 3, 'optimizer': 'adam'}
```

In [40]:
```
best_accuracy
```

Out[40]:
```
0.9092360360360361
```

In [ ]:
```

```