TENSORFLOW (KERAS)

Activate TF 2.0 env – from anaconda prompt type “conda activate tfenv”

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

1. Feature Engineering
   1. Visualize
      1. df.isnull().sum()
      2. df.describe().transpose()
      3. plt.figure(figsize=(12, 8)
      4. sns.distplot(df['price'])
   2. Check Feature Correlation
      1. Date
         1. df['date'] = pd.to\_datetime(df['date'])
         2. df['day'] = df['date'].apply(lambda date:date.day)
         3. df['month'] = df['date'].apply(lambda date:date.month)
         4. df['year'] = df['date'].apply(lambda date:date.year)
         5. df.groupby('?').mean()['price'].plot()
         6. df.groupby('?').mean()['price'].plot()
      2. Other Features
         1. corr = df.corr()[*label]*[:-1].sort\_values().plot(kind='bar')
         2. sns.heatmap(corr)
2. Create Regression Model
   1. Separate features from label
      1. X = df.drop(‘price’, axis=1).values
      2. y = df[‘price’].values
   2. Train, test split
      1. from sklearn.model\_selection import train\_test\_split
      2. X = df[['feature1', 'feature2']].values - *features*
      3. y = df['price'].values - *label*
      4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=42
      5. X\_train.shape
      6. Y\_train
   3. Scale
      1. from sklearn.preprocessing import MinMaxScaler
      2. scaler = MinMaxScaler()
      3. scaler.fit(X\_train)
      4. X\_train = scaler.transform(X\_train)
      5. X\_test = scaler.transform(X\_test)
   4. Model
      1. Import tensorflow as tf
      2. from tensorflow.keras.models import Sequential
      3. from tensorflow.keras.layers import Dense, Activation
      4. from tensorflow.keras.optimizers import Adam
      5. model = Sequential()
      6. model.add(Dense(25,activation='relu')), …,
      7. model.add(Dense(1))
      8. model.compile(optimizer='adam',loss='mse')
   5. Save Model
      1. from tensorflow.models import load\_model
      2. model.save(‘?.h5’)
   6. Train
      1. model.fit(x=X\_train,y=y\_train.values, validation\_data=(X\_test,y\_test.values), batch\_size=128,epochs=400)
      2. losses = pd.DataFrame(model.history.history)
      3. losses.plot()
   7. Evaluate
      1. from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error,explained\_variance\_score
      2. predictions = model.predict(X\_test)
      3. df['price'].mean()
      4. plt.scatter(y\_test,predictions), plt.plot(y\_test,y\_test,'r')
      5. errors = y\_test.values.reshape(6480, 1) – predictions
      6. sns.distplot(errors)
   8. New Prediction
      1. single\_house = df.drop('price',axis=1).iloc[0]
      2. single\_house = scaler.transform(single\_house.values.reshape(-1, 19))
      3. model.predict(single\_house)
      4. print(df.iloc[0])
   9. Overfit Fit Test
      1. model\_loss = pd.DataFrame(model.history.history)
      2. model\_loss.plot()
   10. Early Stopping
       1. from tensorflow.keras.callbacks import EarlyStopping
       2. Create new model (i.e. copy -> past)
       3. early\_stop = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=25)
       4. model.fit(x=X\_train,y=y\_train.values, validation\_data=(X\_test,y\_test.values), batch\_size=128,epochs=400, callbacks=[early\_stop])
       5. model\_loss = pd.DataFrame(model.history.history)
       6. model\_loss.plot()
   11. Adding DropOut Layers
       1. from tensorflow.keras.layers import Dropout
       2. Create new model (i.e. copy -> past)
       3. early\_stop = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=25)
       4. model.fit(x=X\_train,y=y\_train.values, validation\_data=(X\_test,y\_test.values), batch\_size=128,epochs=400, callbacks=[early\_stop])
       5. model\_loss = pd.DataFrame(model.history.history)
       6. model\_loss.plot()
3. Create Classification Model
   1. Feature Engineering
      1. df.info()
      2. df.describe().transpose()
         1. Handle empty
            1. df.isnull().sum()
         2. Investigate String values
            1. df.select\_dtypes(['object']).columns
         3. Dummies
            1. dummies = pd.get\_dummies(df[['verification\_status', 'application\_type','initial\_list\_status','purpose' ]],drop\_first=True)
            2. df = df.drop(['verification\_status', 'application\_type','initial\_list\_status','purpose'],axis=1)
            3. df = pd.concat([df,dummies],axis=1)
   2. Visualize Data:
      1. sns.countplot(x=*label*, data=df)
      2. sns.heatmap(df.corr())
      3. df.corr()[*label*.sort\_values()
      4. df.corr()[*label*].sort\_values().plot(kind='bar')
      5. df.corr()['benign\_0\_\_mal\_1'][:-1].sort\_values().plot(kind='bar')
   3. Train Test Split
      1. X = df.drop('benign\_0\_\_mal\_1',axis=1).values
      2. y = df['benign\_0\_\_mal\_1'].values
      3. from sklearn.model\_selection import train\_test\_split
      4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25,random\_state=101)
   4. Scale
      1. from sklearn.preprocessing import MinMaxScaler
      2. scaler = MinMaxScaler()
      3. scaler.fit(X\_train)
      4. X\_train = scaler.transform(X\_train)
      5. X\_test = scaler.transform(X\_test)
   5. Model
      1. model = Sequential()
      2. model.add(Dense(units=30,activation='relu'))…,
      3. model.add(Dense(units=1,activation='sigmoid')) – BINARY CLASSIFICATION
      4. model.compile(loss='binary\_crossentropy', optimizer='adam')
      5. model.fit(x=X\_train, y=y\_train, epochs=600, validation\_data=(X\_test, y\_test), verbose=1)
   6. Add Early Stopping (2.i)
   7. Add DropOut Layers (2.j)
4. Model Evaluation
   1. predictions = model.predict\_classes(X\_test)
   2. from sklearn.metrics import classification\_report,confusion\_matrix
   3. print(classification\_report(y\_test,predictions)) - <https://en.wikipedia.org/wiki/Precision_and_recall>
   4. print(confusion\_matrix(y\_test,predictions))