Department of Management Studies

Machine Learning LAB [MSC528]
(L-T-P: 0-0-2)

LAB MANUAL

Location: System Lab/Classroom, Students to bring their own laptops loaded with Python

Content

Course Type	Course Code	Name of Course	L	T	P	Credit
DC	MSC528	Machine Learning Lab.	0	0	2	2

Course Objective
In this laboratory course, one will be introduced to some popular machine learning techniques and give insights on how to apply these techniques to solve a new business related problem. The course will be taught with popular software like R, and Python.
Learning Outcomes
• To develop understanding in various types of machine learning algorithm • To develop the skill in application software like Python or R for solving business application problems through machine learning.

Exp. No.	Topics	Lectures	Learning Outcome
 1. 2. 3. 4.	**Supervised Learning:** Linear Regression (with one variable and multiple variables), Gradient Descent; Classification (Logistic Regression, Overfitting, Regularization, Support Vector Machines); Artificial Neural Networks (Perceptron, Multilayer networks, and back-propagation); Decision Trees.	8	Students will learn different types of supervised learning algorithms: classification/regression problems.
 5. 6. 7. 8.	**Unsupervised Learning:** Clustering (K-means, Hierarchical); Dimensionality reduction; Principal Component Analysis; Anomaly Detection.	8	Students will learn to find the structures and patterns in the data.
 9. 10. 11.	**Theory of Generalization:** In-sample and out-of sample error, VC inequality, VC analysis, Bias and Variance Analysis.	6	Students will learn different types of error, and techniques to minimize error in the model.
 12.	**Applications:** Spam Filtering, recommender systems, and others	4	Students will learn the implementation of different types of machine learning algorithms for real-life problems.
Total		26	

Text Books:
1. "Understanding Machine Learning", Shai Shalev-Shwartz and Shai Ben-David. Cambridge University Press. 2017.
2. "Data Analytics using Python", Bharti Motwani, First Edition, Wiley India Pvt. Ltd., 2020.
Reference Books:
1. "Foundation of Data Science", Avrim Blum, John Hopcroft and Ravindran Kannan. January 2017.
2. "Machine Learning", Tom Mitchell, First Edition, McGraw-Hill, 1997.

Experiment 1

Objective: To understand and perform multiple regression analysis

Details: Develop a multiple-regression model using the given data set, do interpretations on the results and perform prediction on some given data sets.

Tool/Software: Python

Procedure:

```
#Import Library
#Import other necessary libraries like pandas,
#numpy...
from sklearn import linear_model
#Load Train and Test datasets
#Identify feature and response variable(s) and
#values must be numeric and numpy arrays
x_train=input_variables_values_training_datasets
y_train=target_variables_values_training_datasets
x_test=input_variables_values_test_datasets
#Create linear regression object
linear = linear_model.LinearRegression()
#Train the model using the training sets and
#check score
linear.fit(x_train, y_train)
linear.score(x_train, y_train)
#Equation coefficient and Intercept
print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)
#Predict Output
predicted= linear.predict(x_test)
```

```
# plotting fitted line
plt.scatter(x, y, color='black')
plt.plot(x, lr.predict(x), color='blue', linewidth=3)
plt.title('Grade vs Hours Studied')
plt.ylabel('Test_Grade')
plt.xlabel('Hours_Studied')
```

Experiment 2

Objective: To perform and understand Classification through Logistic Regression, Overfitting, Regularization and Support Vector Machines on the given data

Details: Develop a classification model using Logistic Regression, SVM, Naïve Bayes and Random Forest algorithms

Tool/Software: Python

Procedure:

(a) Logistic Regression

```
#Import Library
from sklearn.linear_model import LogisticRegression
#Assumed you have, X (predictor) and Y (target)
#for training data set and x_test(predictor)
#of test_dataset
#Create logistic regression object
model = LogisticRegression()
#Train the model using the training sets
#and check score
model.fit(X, y)
model.score(X, y)
#Equation coefficient and Intercept
print('Coefficient: \n', model.coef_)
print('Intercept: \n', model.intercept_)
#Predict Output
predicted= model.predict(x_test)
```

(b) SVM

```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create SVM classification object
model = svm.svc()
#there are various options associated
with it, this is simple for classification.
#Train the model using the training sets and check
#score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

(c) Naïve Bayes

```
#Import Library
from sklearn.naive_bayes import GaussianNB
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create SVM classification object model = GaussianNB()
#there is other distribution for multinomial classes
like Bernoulli Naive Bayes
#Train the model using the training sets and check
#score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

(d) Random forest

```
#Import Library
from sklearn.ensemble import RandomForestClassifier
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create Random Forest object
model= RandomForestClassifier()
#Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Experiment 3

Objective: To understand and perform in-depth of Artificial Neural Networks (Perceptron, Multilayer networks, and back-propagation); on the given data

Details: Building Neural Network Model to understand the key concept of deep learning

Tool/Software: Python

Procedure:

Credit card dataset can be downloaded from https://kaggle.com/mlg-ulb/creditcardfraud

```
# importing libraries
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from kera.layers import Dense, Dropout
import numpy
import pandas

creditcard = pandas.read_csv("creditcard.csv")
print(creditcard.shape)

X = creditcard.iloc[ :, 0:29]
Y creditcard.iloc[ :, 29]

Numpy.random.seed(500)

# splitting the data into training and test set
x_trg, x_test, y_trg, y_test = train_test_split(X, Y, test_size = 0.3)

# Determining number of columns
Input_dim = x_trg.shape[1]
```

 # FIRST MODEL

```
Model1 = Sequential()
Model1.add(Dense(10, input_dim = input_dim, kernel_initializer = 'uniform', activation =
'relu'))
Model1.add(Dropout(0))
Model1.add(Dense(1, kernel_initializer='uniform', activation = 'softmax'))
Model1.compile(loss='binary_crossentropy', optimizer = 'SGD', metrics = ['accuracy'])
Model1.fit(x_trg, y_trg, epochs=5, batch_size=1000)

score=Model1.evaluate(x_test, y_test)
print('Test accuracy of the model is ', %(100*score[1]))
score=Model1.evaluate(x_trg, y_trg)
print('Train accuracy of the model is ', %(100*score[1]))
```

 # SECOND MODEL (By changing units, droupout, epoch and batch size)

```
Model2 = Sequential()
Model2.add(Dense(1000, input_dim = input_dim, kernel_initializer = 'uniform', activation =
'relu'))
Model2.add(Dropout(0.1))
Model2.add(Dense(1, kernel_initializer='uniform', activation = 'softmax'))
Model2.compile(loss='binary_crossentropy', optimizer = 'SGD', metrics = ['accuracy'])
Model2.fit(x_trg, y_trg, epochs=500, batch_size=2000)

score=Model2.evaluate(x_test, y_test)
print('Test accuracy of the model is ', %(100*score[1]))
score=Model2.evaluate(x_trg, y_trg)
print('Train accuracy of the model is ', %(100*score[1]))
```

 # THIRD MODEL (By changing the activation, loss and optimizers)

```
Model3 = Sequential()
Model3.add(Dense(100, input_dim = input_dim, kernel_initializer = 'uniform', activation =
'softmax'))
Model3.add(Dropout(0.1))
Model3.add(Dense(1, kernel_initializer='uniform', activation = 'sigmoid'))
Model3.compile(loss='binary_crossentropy', optimizer = 'RMSprop', metrics = ['accuracy'])
Model3.fit(x_trg, y_trg, epochs=50, batch_size=2000)

score=Model3.evaluate(x_test, y_test)
print('Test accuracy of the model is ', %(100*score[1]))
score=Model3.evaluate(x_trg, y_trg)
print('Train accuracy of the model is ', %(100*score[1]))
```

 # FOURTH MODEL (By changing the optimizer and activation function)

```
Model4 = Sequential()
Model4.add(Dense(100, input_dim = input_dim, kernel_initializer = 'uniform', activation =
'relu'))
Model4.add(Dropout(0.1))
Model4.add(Dense(1, kernel_initializer='uniform', activation = 'sigmoid'))
Model4.compile(loss='binary_crossentropy', optimizer = 'adagrad', metrics = ['accuracy'])
Model4.fit(x_trg, y_trg, epochs=50, batch_size=2000)

score=Model4.evaluate(x_test, y_test)
print('Test accuracy of the model is ', %(100*score[1]))
score=Model4.evaluate(x_trg, y_trg)
print('Train accuracy of the model is ', %(100*score[1]))
```

 # FIFTH MODEL (Grid Approach to determine best value of Epoch and Batch_size)

```
from keras.constraints import maxnorm
weight_constraint = 0

def create_model() :
        model = Sequential()
        model.add(Dense(1000, input_dim = input_dim, kernel_initializer = ‘uniform’,
        activation = ‘relu’, kernel_constraint=maxnorm(weight_constraint)))
        model.add(Dropout(0.1))
        model.add(Dense(1, kernel_initializer=’uniform’, activation = ‘sigmoid’))
        Model.compile(loss=’binary_crossentropy’, optimizer = ‘adagrad’, metrics =
        [‘accuracy’])
        return model

from keras.wrappers.scikit_learn import KerasClassifier
model5 = KerasClassifier(build_fn = create_model)

from sklearn.model_selection import GridSearchCV
epochs = [50, 100]
batch_size = [1500, 2500, 3000]

param_grid = dict(epochs=epochs, batch_size = batch_size)
grid = GridSearchCV(estimator = model5, param_grid=param_grid)
grid_result = grid,fit(x_trg, y_trg)

print(“Results: “, grid_result.cv_results_)
print(“Best Result: “, %(grid_result.best_score_, grid_result.best_params_))

Model4 = Sequential()
Model4.add(Dense(100, input_dim = input_dim, kernel_initializer = ‘uniform’, activation =
‘relu’))
Model4.add(Dropout(0.1))
Model4.add(Dense(1, kernel_initializer=’uniform’, activation = ‘sigmoid’))
Model4.compile(loss=’binary_crossentropy’, optimizer = ‘adagrad’, metrics = [‘accuracy’])
Model4.fit(x_trg, y_trg, epochs=50, batch_size=2000)

score=Model4.evaluate(x_test, y_test)
print(‘Test accuracy of the model is ‘, %(100*score[1]))
score=Model4.evaluate(x_trg, y_trg)
print(‘Train accuracy of the model is ‘, %(100*score[1]))
```

Experiment 4

Objective: To perform Decision Trees

Details: To do classification using Decision Trees

Tool/Software: Python

Procedure:

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import tree
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of
#test_dataset
#Create tree object
model = tree.DecisionTreeClassifier(criterion='gini')
#for classification, here you can change the
#algorithm as gini or entropy (information gain) by
#default it is gini
#model = tree.DecisionTreeRegressor() for
#regression
#Train the model using the training sets and check
#score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Experiment 5

Objective: To perform Clustering (K-means, Hierarchical)

Details: do the hierarchical clustering and visualization for given data with interpretation

Tool/Software: Python

Procedure:

(a) K Means

```
#Import Library
from sklearn.cluster import KMeans
#Assumed you have, X (attributes) for training data set
#and x_test(attributes) of test_dataset
#Create KNeighbors classifier object model
k_means = KMeans(n_clusters=3, random_state=0)
#Train the model using the training sets and check score
model.fit(X)
#Predict Output
predicted= model.predict(x_test)
```

(b) kNN

```
#Import Library
from sklearn.neighbors import KNeighborsClassifier
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create KNeighbors classifier object model
KNeighborsClassifier(n_neighbors=6)
#default value for n_neighbors is 5
#Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Experiment 6

Objective: To perform Dimensionality reduction using Principal Component Analysis ;

Details: do reduce the number of features using PCA for given data

Tool/Software: Python

Procedure:

```
#Import Library
from sklearn import decomposition
#Assumed you have training and test data set as train and
#test

#Create PCA object pca= decomposition.PCA(n_components=k)
#default value of k =min(n_sample, n_features)
#For Factor analysis
#fa= decomposition.FactorAnalysis()
#Reduced the dimension of training dataset using PCA
train_reduced = pca.fit_transform(train)
#Reduced the dimension of test dataset
test_reduced = pca.transform(test)
```

Experiment 7

Objective: To create Gradient Boosting classifier

Details: do the Gradient Boosting classifier for given data with interpretation

Tool/Software: Python

Procedure:

```
#Import Library
from sklearn.ensemble import GradientBoostingClassifier
#Assumed you have, X (predictor) and Y (target) for
#training data set and x_test(predictor) of test_dataset
#Create Gradient Boosting Classifier object
model= GradientBoostingClassifier(n_estimators=100, \
         learning_rate=1.0, max_depth=1, random_state=0)
#Train the model using the training sets and check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Experiment 8

Objective: To perform Anomaly Detection.

Details: do the Anomaly Detection for given data with interpretation

Tool/Software: Python

Procedure:

Anomaly (or outlier) detection is the data-driven task of identifying these rare occurrences and filtering or modulating them from the analysis pipeline. Such anomalous events can be connected to some fault in the data source, such as financial fraud, equipment fault, or irregularities in time series analysis. One can train machine learning models to detect and report such anomalies retrospectively or in real-time. These anomalous data points can later be either flagged to analyze from a business perspective or removed to maintain the cleanliness of the data before further processing is done.

Top 5 Anomaly Detection Machine Learning Algorithms: DBSCAN, Local Outlier Factor (LOR), Isolation Forest Model, Support Vector Machines (SVM), and Autoencoders.

Python Outlier Detection (PyOD)

PyOD toolkit consists of three major functional groups:

(i) Individual Detection Algorithms :

Type	Abbr	Algorithm	Year	Ref
Probabilistic	ECOD	Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions	2022	[27]
Probabilistic	ABOD	Angle-Based Outlier Detection	2008	[21]
Probabilistic	FastABOD	Fast Angle-Based Outlier Detection using approximation	2008	[21]
Probabilistic	COPOD	COPOD: Copula-Based Outlier Detection	2020	[26]
Probabilistic	MAD	Median Absolute Deviation (MAD)	1993	[18]
Probabilistic	SOS	Stochastic Outlier Selection	2012	[19]
Probabilistic	KDE	Outlier Detection with Kernel Density Functions	2007	[23]
Probabilistic	Sampling	Rapid distance-based outlier detection via sampling	2013	[39]
Probabilistic	GMM	Probabilistic Mixture Modeling for Outlier Analysis		[1] [Ch.2]

Type	Abbr	Algorithm	Year	Ref
Linear Model	PCA	Principal Component Analysis (the sum of weighted projected distances to the eigenvector hyperplanes)	2003	[38]
Linear Model	KPCA	Kernel Principal Component Analysis	2007	[17]
Linear Model	MCD	Minimum Covariance Determinant (use the mahalanobis distances as the outlier scores)	1999	[15] [34]
Linear Model	CD	Use Cook's distance for outlier detection	1977	[10]
Linear Model	OCSVM	One-Class Support Vector Machines	2001	[37]
Linear Model	LMDD	Deviation-based Outlier Detection (LMDD)	1996	[6]
Proximity-Based	LOF	Local Outlier Factor	2000	[8]
Proximity-Based	COF	Connectivity-Based Outlier Factor	2002	[40]
Proximity-Based	(Incremental) COF	Memory Efficient Connectivity-Based Outlier Factor (slower but reduce storage complexity)	2002	[40]
Proximity-Based	CBLOF	Clustering-Based Local Outlier Factor	2003	[16]
Proximity-Based	LOCI	LOCI: Fast outlier detection using the local correlation integral	2003	[30]
Proximity-Based	HBOS	Histogram-based Outlier Score	2012	[11]
Proximity-Based	kNN	k Nearest Neighbors (use the distance to the kth nearest neighbor as the outlier score)	2000	[33]
Proximity-Based	AvgKNN	Average kNN (use the average distance to k nearest neighbors as the outlier score)	2002	[5]
Proximity-Based	MedKNN	Median kNN (use the median distance to k nearest neighbors as the outlier score)	2002	[5]
Proximity-Based	SOD	Subspace Outlier Detection	2009	[22]
Proximity-Based	ROD	Rotation-based Outlier Detection	2020	[4]

Type	Abbr	Algorithm	Year	Ref
Outlier Ensembles	IForest	Isolation Forest	2008	[28]
Outlier Ensembles	INNE	Isolation-based Anomaly Detection Using Nearest-Neighbor Ensembles	2018	[7]
Outlier Ensembles	FB	Feature Bagging	2005	[24]
Outlier Ensembles	LSCP	LSCP: Locally Selective Combination of Parallel Outlier Ensembles	2019	[45]
Outlier Ensembles	XGBOD	Extreme Boosting Based Outlier Detection **(Supervised)**	2018	[44]
Outlier Ensembles	LODA	Lightweight On-line Detector of Anomalies	2016	[31]
Outlier Ensembles	SUOD	SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection **(Acceleration)**	2021	[46]
Neural Networks	AutoEncoder	Fully connected AutoEncoder (use reconstruction error as the outlier score)		[1] [Ch.3]
Neural Networks	VAE	Variational AutoEncoder (use reconstruction error as the outlier score)	2013	[20]
Neural Networks	Beta-VAE	Variational AutoEncoder (all customized loss term by varying gamma and capacity)	2018	[9]
Neural Networks	SO_GAAL	Single-Objective Generative Adversarial Active Learning	2019	[29]
Neural Networks	MO_GAAL	Multiple-Objective Generative Adversarial Active Learning	2019	[29]
Neural Networks	DeepSVDD	Deep One-Class Classification	2018	[35]
Neural Networks	AnoGAN	Anomaly Detection with Generative Adversarial Networks	2017	[36]
Neural Networks	ALAD	Adversarially learned anomaly detection	2018	[43]
Graph-based	R-Graph	Outlier detection by R-graph	2017	[42]
Graph-based	LUNAR	LUNAR: Unifying Local Outlier Detection Methods via Graph Neural Networks	2022	[12]

(ii) Outlier Ensembles & Outlier Detector Combination Frameworks:

Type	Abbr	Algorithm	Year	Ref
Outlier Ensembles	FB	Feature Bagging	2005	[24]
Outlier Ensembles	LSCP	LSCP: Locally Selective Combination of Parallel Outlier Ensembles	2019	[45]
Outlier Ensembles	XGBOD	Extreme Boosting Based Outlier Detection **(Supervised)**	2018	[44]
Outlier Ensembles	LODA	Lightweight On-line Detector of Anomalies	2016	[31]
Outlier Ensembles	SUOD	SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection **(Acceleration)**	2021	[46]
Outlier Ensembles	INNE	Isolation-based Anomaly Detection Using Nearest-Neighbor Ensembles	2018	[7]
Combination	Average	Simple combination by averaging the scores	2015	[2]
Combination	Weighted Average	Simple combination by averaging the scores with detector weights	2015	[2]
Combination	Maximization	Simple combination by taking the maximum scores	2015	[2]
Combination	AOM	Average of Maximum	2015	[2]
Combination	MOA	Maximization of Average	2015	[2]
Combination	Median	Simple combination by taking the median of the scores	2015	[2]
Combination	majority Vote	Simple combination by taking the majority vote of the labels (weights can be used)	2015	[2]

(iii) Utility Functions:

Type	Name	Function	Documentation
Data	generate_data	Synthesized data generation; normal data is generated by a multivariate Gaussian and outliers are generated by a uniform distribution	generate_data
Data	generate_data_clusters	Synthesized data generation in clusters; more complex	generate_data_clusters

Type	Name	Function	Documentation
		data patterns can be created with multiple clusters	
Stat	wpearsonr	Calculate the weighted Pearson correlation of two samples	wpearsonr
Utility	get_label_n	Turn raw outlier scores into binary labels by assign 1 to top n outlier scores	get_label_n
Utility	precision_n_scores	calculate precision @ rank n	precision_n_scores

Experiment 9

Objective: To prepare data for In-sample and out-of sample predictions

Details: To perform In-sample and out-of sample prediction

Tool/Software: Python

Procedure:

Prediction (out of sample)

```
[1]:    %matplotlib inline
[2]:    import numpy as np
        import matplotlib.pyplot as plt
        import statsmodels.api as sm
        plt.rc("figure", figsize=(16, 8))
        plt.rc("font", size=14)
```

Artificial data

```
[3]:    nsample = 50
        sig = 0.25
        x1 = np.linspace(0, 20, nsample)
        X = np.column_stack((x1, np.sin(x1), (x1 - 5) ** 2))
        X = sm.add_constant(X)
        beta = [5.0, 0.5, 0.5, -0.02]
        y_true = np.dot(X, beta)
        y = y_true + sig * np.random.normal(size=nsample)
```

Estimation

```
[4]:    olsmod = sm.OLS(y, X)
        olsres = olsmod.fit()
        print(olsres.summary())
```

```
OLS Regression Results
==============================================================================
Dep. Variable:              y        R-squared:              0.983
Model:                      OLS      Adj. R-squared:         0.982
Method:           Least Squares      F-statistic:            883.7
Date:          Wed, 30 Nov 2022      Prob (F-statistic):     1.17e-40
Time:                  22:22:12      Log-Likelihood:         2.0531
No. Observations:            50      AIC:                    3.894
Df Residuals:                46      BIC:                    11.54
Df Model:                     3
Covariance Type:      nonrobust
==============================================================================
```

```
           coef      std err       t         P>|t|     [0.025     0.975]
------------------------------------------------------------------------------
const      5.1442      0.083     62.334      0.000      4.978      5.310
x1         0.4902      0.013     38.512      0.000      0.465      0.516
x2         0.4835      0.050      9.663      0.000      0.383      0.584
x3        -0.0206      0.001    -18.435      0.000     -0.023     -0.018
==============================================================================
Omnibus:                  0.207   Durbin-Watson:                   2.406
Prob(Omnibus):            0.902   Jarque-Bera (JB):                0.404
Skew:                    -0.071   Prob(JB):                        0.817
Kurtosis:                 2.583   Cond. No.                         221.
==============================================================================
```

Notes: Standard Errors assume that the covariance matrix of the errors is correctly specified.

In-sample prediction

[5]:
```
ypred = olsres.predict(X)
print(ypred)
```

```
[ 4.62918513  5.10181392  5.53604867  5.90553962  6.19344652  6.39520541
  6.51927844  6.58576362  6.62309273  6.66336008  6.73704949  6.8680259
  7.0696144   7.34241101  7.6741847   8.04188694  8.41543849  8.76267138
  9.05461395  9.27025086  9.39997585  9.44716998  9.42764619  9.36705121
  9.29665195  9.24819838  9.24870712  9.31602268  9.45588568  9.66098963
  9.91218143 10.18160813 10.43729308 10.64838966 10.79024923 10.84847086
 10.82126737 10.71975894 10.5661461  10.39006226 10.223705   10.09654614
 10.03048955 10.03627243 10.11170125 10.24201209 10.40229633 10.56159308
 10.68797594 10.75379845]
```

Create a new sample of explanatory variables Xnew, predict and plot

[6]:
```
x1n = np.linspace(20.5, 25, 10)
Xnew = np.column_stack((x1n, np.sin(x1n), (x1n - 5) ** 2))
Xnew = sm.add_constant(Xnew)
ynewpred = olsres.predict(Xnew)  # predict out of sample
print(ynewpred)
```

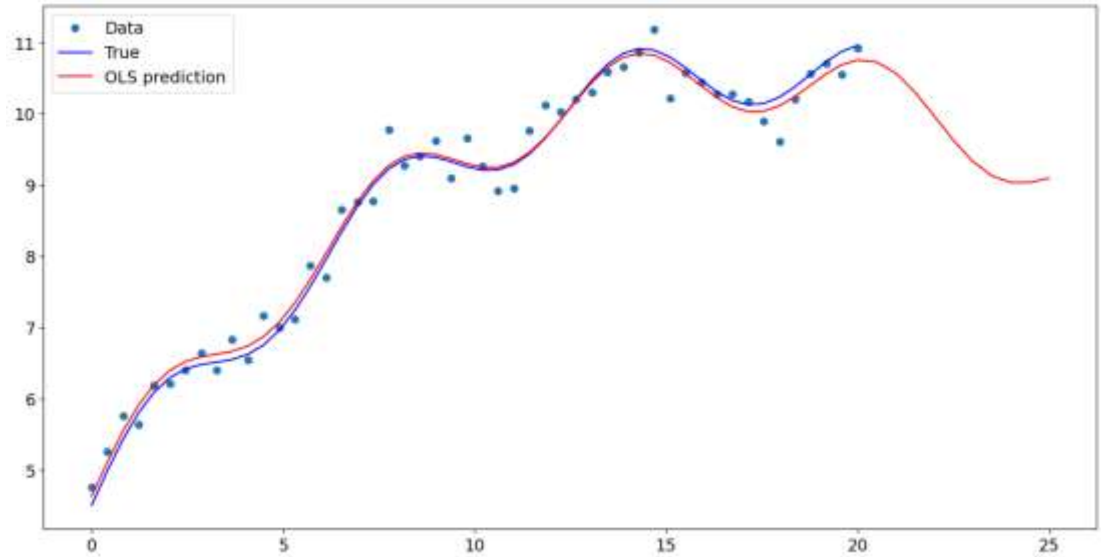
```
[10.72527983 10.56846113 10.30230297  9.9700142   9.62847291  9.33430062
  9.12999942  9.03354592  9.03399001  9.09413578]
```

Plot comparison

[7]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(x1, y, "o", label="Data")
ax.plot(x1, y_true, "b-", label="True")
```

```
ax.plot(np.hstack((x1, x1n)), np.hstack((ypred, ynewpred)), "r", label="OLS
prediction")
ax.legend(loc="best")
```

[7]: <matplotlib.legend.Legend at 0x7f90fead27a0>



Predicting with Formulas

Using formulas can make both estimation and prediction a lot easier

[8]:
```
from statsmodels.formula.api import ols
data = {"x1": x1, "y": y}
res = ols("y ~ x1 + np.sin(x1) + I((x1-5)**2)", data=data).fit()
```

We use the I to indicate use of the Identity transform. ie., we do not want any expansion magic from using **2

[9]:
```
res.params
```

[9]:
```
Intercept           5.144200
x1                  0.490167
np.sin(x1)          0.483489
I((x1 - 5) ** 2)   -0.020601
dtype: float64
```

Now we only have to pass the single variable and we get the transformed right-hand side variables automatically

[10]:
```
res.predict(exog=dict(x1=x1n))
```

[10]:
```
0    10.725280
1    10.568461
2    10.302303
```

```
3    9.970014
4    9.628473
5    9.334301
6    9.129999
7    9.033546
8    9.033990
9    9.094136
dtype: float64
```

Experiment 10

Objective: To perform Bias and Variance Analysis.

Details: to carry out Bias and Variance analysis with interpretation

Tool/Software: Python

Procedure:

Bias is the difference between predicted values and expected results. A machine learning model with a low bias is a perfect model and a model with a high bias is expected with a high error rate on the training and test sets.

Variance is the variability of your model's predictions over different sets of data. A machine learning model with high variance indicates that the model may work well on the data it was trained on, but it will not generalize well on the dataset it has never seen before.

pip install mlxtend

Now let's train a machine learning model and then we will see how we can calculate its bias and variance using Python:

```
from mlxtend.evaluate import bias_variance_decomp
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error


data = pd.read_csv("https://raw.githubusercontent.com/amankharwal/Website-
data/master/student-mat.csv")
data = data[["G1", "G2", "G3", "studytime", "failures", "absences"]]


predict = "G3"
x = np.array(data.drop([predict], 1))
y = np.array(data[predict])


from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)


linear_regression = LinearRegression()
linear_regression.fit(xtrain, ytrain)
y_pred = linear_regression.predict(xtest)
```

So till now, we have trained a machine learning model by using the linear regression algorithm, below is how we can calculate its bias and variance using Python:

```
mse, bias, variance = bias_variance_decomp(linear_regression, xtrain, ytrain, xtest, ytest,
                                loss='mse', num_rounds=200, random_seed=123)
print("Average Bias : ", bias)
print("Average Variance : ", variance)
```

Average Bias : 3.909459558063484
Average Variance : 0.07349200663859749

Experiment 11

Objective: To create recommender systems.

Details: Implement a Movie Recommender System in Python

Tool/Software: Python

Procedure:

We will develop a very simple movie recommender system in Python that uses the correlation between the ratings assigned to different movies. Thus, we will find the similarity between the movies.
The dataset that we are going to use for this problem is the MovieLens Dataset.

Let's import the basic libraries and import the data.

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt # data visualization
import seaborn as sns        # statistical data visualization
%matplotlib inline
plt.style.use('fivethirtyeight')

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
        for filename in filenames:
                print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

/kaggle/input/movielens-dataset/ratings.csv
/kaggle/input/movielens-dataset/movies.csv

We can see that there are 2 files in the dataset - ratings and movies. Let's explore them.
Let's explore ratings file

In [2]:
```
ratings_file = '/kaggle/input/movielens-dataset/ratings.csv'
df_ratings = pd.read_csv(ratings_file)
df_ratings.head()
```

Out[2]:

	userId	movieId	rating	timestamp
0	1	16	4.0	1217897793
1	1	24	1.5	1217895807

	userId	movieId	rating	timestamp
2	1	32	4.0	1217896246
3	1	47	4.0	1217896556
4	1	50	4.0	1217896523

We can see from the output that the ratings.csv file contains the userId, movieId, ratings and timestamp attributes. Each row in the dataset corresponds to one rating. The userId column contains the ID of the user who left the rating. The movieId column contains the Id of the movie, the rating column contains the rating left by the user. Ratings can have values between 1 and 5. Finally, the timestamp refers to the time at which the user left the rating. There is one problem with this dataset. It contains the IDs of the movies but not their titles. We will need movie names of the movies to recommend. The movie names are stored in the movies.csv file. So, let's explore that file.

Let's explore movies file

In [3]:
```
movies_file = '/kaggle/input/movielens-dataset/movies.csv'
df_movies = pd.read_csv(movies_file)
df_movies.head()
```

Out[3]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

We can see that, this dataset contains movieId, the title of the movie, and its genre. We need a dataset that contains the userId, movie title and its ratings. We have this information in two different files: ratings.csv and movies.csv. To get our desired information in a single dataframe, we can merge the two dataframes objects on the movieId column since it is common between the two dataframes. We can do this using the merge() function from the Pandas library, as shown below.

In [4]:
```
movie_data = pd.merge(df_ratings, df_movies, on='movieId')
movie_data.head()
```

Out[4]:

	userId	movieId	rating	timestamp	title	genres
0	1	16	4.0	1217897793	Casino (1995)	Crime|Drama
1	9	16	4.0	842686699	Casino (1995)	Crime|Drama
2	12	16	1.5	1144396284	Casino (1995)	Crime|Drama
3	24	16	4.0	963468757	Casino (1995)	Crime|Drama
4	29	16	3.0	836820223	Casino (1995)	Crime|Drama

We can see our newly created dataframe contains userId, title and rating of the movie as required. Now let's take a look at the average rating of each movie. To do so, we can group the dataset by the title of the movie and then calculate the mean of the rating for each movie. We will then display the first five movies along with their average rating using the head() method as follows:

In [5]:

```
movie_data.groupby('title')['rating'].mean().head()
```

Out[5]:

```
title
'71 (2014)                                  3.500
'Hellboy': The Seeds of Creation (2004)     3.000
'Round Midnight (1986)                      2.500
'Til There Was You (1997)                   4.000
'burbs, The (1989)                          3.125
Name: rating, dtype: float64
```

We can see that the average ratings are not sorted. Let's sort the ratings in the descending order of their average ratings:

In [6]:

```
movie_data.groupby('title')['rating'].mean().sort_values(ascending=False).head()
```

Out[6]:

```
title
Being Human (1993)                  5.0
Three Ages (1923)                   5.0
The Liberator (2013)                5.0
October Baby (2011)                 5.0
Resident Evil: Retribution (2012)   5.0
Name: rating, dtype: float64
```

The movies have now been sorted according to the ascending order of their ratings. However, there is a problem. A movie can make it to the top of the above list even if only a single user has given it five stars. Therefore, the above stats can be misleading. Normally, a movie which is really a good one gets a higher rating by a large number of users. Let's now plot the total number of ratings for a movie:

In [7]:

```
movie_data.groupby('title')['rating'].count().sort_values(ascending=False).head()
```

Out[7]:

```
title
Pulp Fiction (1994)                     325
Forrest Gump (1994)                     311
Shawshank Redemption, The (1994)        308
Jurassic Park (1993)                    294
Silence of the Lambs, The (1991)        290
Name: rating, dtype: int64
```

Now, we can see some great movies at the top. The above list supports our point that good movies normally receive higher ratings. Now we know that both the average rating per movie and the number of ratings per movie are important attributes. So, let's create a new dataframe that contains both of these attributes. We will create a new dataframe called ratings_mean_count and first add the average rating of each movie to this dataframe as follows-

In [8]:

```
ratings_mean_count = pd.DataFrame(movie_data.groupby('title')['rating'].mean())
```

Next up, we will add the number of ratings for a movie to the ratings_mean_count dataframe as follows-

In [9]:

```
ratings_mean_count['rating_counts'] =
pd.DataFrame(movie_data.groupby('title')['rating'].count())
```

Now, let's preview our new dataframe.

In [10]: ratings_mean_count.head()

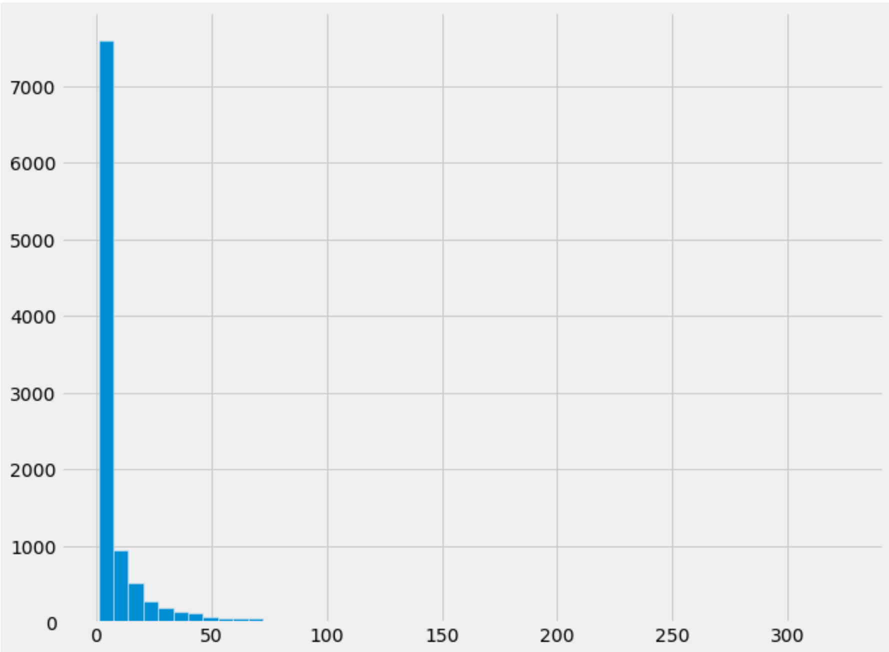
Out[10]:

title	rating	rating_counts
'71 (2014)	3.500	1
'Hellboy': The Seeds of Creation (2004)	3.000	1
'Round Midnight (1986)	2.500	1
'Til There Was You (1997)	4.000	3
'burbs, The (1989)	3.125	20

We can see movie title, along with the average rating and number of ratings for the movies. Now, let's plot a histogram for the number of ratings represented by the rating_counts column in the above dataframe.

In [11]:
```
plt.figure(figsize=(10,8))
plt.rcParams['patch.force_edgecolor'] = True
ratings_mean_count['rating_counts'].hist(bins=50)
```

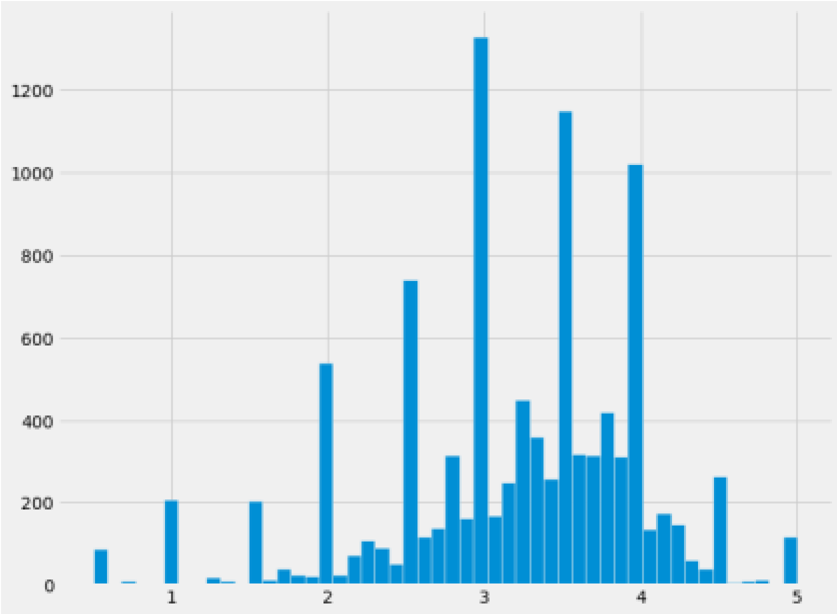
Out[11]:



From the above plot, we can see that most of the movies have received less than 50 ratings and there are no movies having more than 100 ratings. Now, we will plot a histogram for average ratings.

In [12]:
```
plt.figure(figsize=(10,8))
plt.rcParams['patch.force_edgecolor'] = True
ratings_mean_count['rating'].hist(bins=50)
```

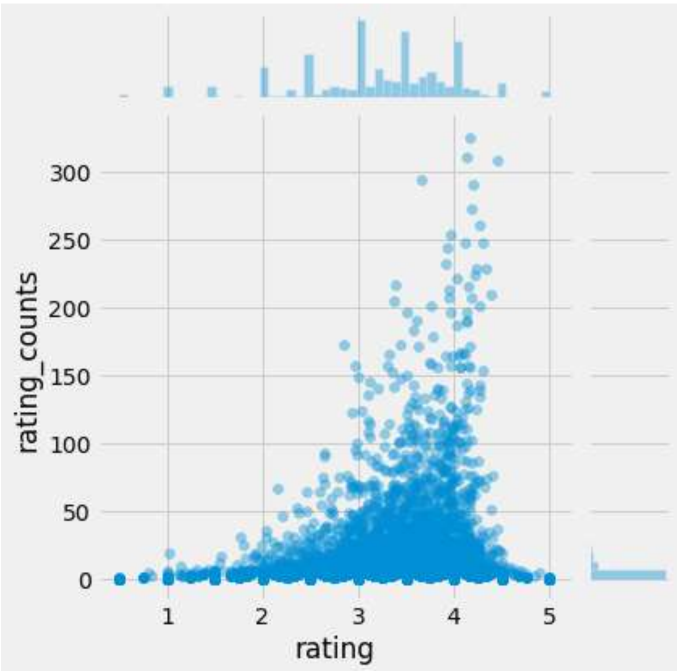
Out[12]:



We can see that the integer values have taller bars than the floating values since most of the users assign rating as integer value i.e. 1, 2, 3, 4 or 5. Furthermore, it is evident that the data has a weak normal distribution with the mean of around 3.5. There are a few outliers in the data as well. Movies with a higher number of ratings usually have a high average rating as well since a good movie is normally well-known and a well-known movie is watched by a large number of people, and thus usually has a higher rating. Let's see if this is also the case with the movies in our dataset. We will plot average ratings against the number of ratings.

In [13]:
```
plt.figure(figsize=(10,8))
plt.rcParams['patch.force_edgecolor'] = True
sns.jointplot(x='rating', y='rating_counts', data=ratings_mean_count, alpha=0.4)
```

Out[13]:



The graph shows that, in general, movies with higher average ratings actually have more number of ratings, compared with movies that have lower average ratings.

Finding Similarities Between Movies

Now, it is the time to find the similarity between the movies. We will use the correlation between the ratings of a movie as the similarity metric. To find the correlation between the ratings of the movie, we need to create a matrix where each column is a movie name and each row contains the rating assigned by a specific user to that movie. This matrix will have a lot of null values since every movie is not rated by every user. We will create the matrix of movie titles and corresponding user ratings.

In [14]:
```
user_movie_rating = movie_data.pivot_table(index='userId', columns='title', values='rating')
user_movie_rating.head()
```

Out[14]:

title	'71 (2014)	'Hellboy': The Seeds of Creation (2004)	'Round Midnight (1986)	'Til There Was You (1997)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	*batteries not included (1987)	...And Justice for All (1979)	10 (1979)	...	[REC] (2007)	[REC]² (2009)	[REC]³ 3 Génesis (2012)	a/k/a Tommy Chong (2005)	eXistenZ (1999)	loudQUIETloud: A Film About the Pixies (2006)	xXx (2002)	xXx: State of the Union (2005)	¡Three Amigos! (1986)	À nous la liberté (Freedom for Us) (1931)
userId																					
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 10323 columns

We know that each column contains all the user ratings for a particular movie. Now, let's find all the user ratings for the movie Forrest Gump (1994) and find the movies similar to it. We chose this movie since it has the highest number of ratings and we want to find the correlation between movies that have a higher number of ratings. We will find the user ratings for Forrest Gump (1994) as follows-

In [15]:
```
forrest_gump_ratings = user_movie_rating['Forrest Gump (1994)']
forrest_gump_ratings.head()
```

Out[15]:
```
userId
1      3.0
2      NaN
3      3.0
4      NaN
5      NaN
Name: Forrest Gump (1994), dtype: float64
```

We can see from the above output, that it will return a Pandas series. Now, we will retrieve all the movies that are similar to Forrest Gump (1994). We can find the correlation between the user ratings for the Forest Gump (1994) and all the other movies using corrwith() function as shown below:

In [16]:
```
movies_like_forest_gump = user_movie_rating.corrwith(forrest_gump_ratings)
```

```
/opt/conda/lib/python3.7/site-packages/numpy/lib/function_base.py:2526: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar)
/opt/conda/lib/python3.7/site-packages/numpy/lib/function_base.py:2455: RuntimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
```

In [17]:
```
corr_forrest_gump = pd.DataFrame(movies_like_forest_gump, columns=['Correlation'])
corr_forrest_gump.dropna(inplace=True)
corr_forrest_gump.head()
```

Out[17]:

title	Correlation
'burbs, The (1989)	0.056266
(500) Days of Summer (2009)	0.144325
*batteries not included (1987)	0.000000
...And Justice for All (1979)	0.089924
10 (1979)	0.693375

Now, let's sort the movies in descending order of correlation to see highly correlated movies at the top.

In [18]:
```
corr_forrest_gump.sort_values('Correlation', ascending=False).head(10)
```

Out[18]:

title	Correlation
Martian Child (2007)	1.0
Save the Tiger (1973)	1.0
Underworld (1996)	1.0
Shortbus (2006)	1.0
Court Jester, The (1956)	1.0
Bottle Shock (2008)	1.0
Anna Karenina (2012)	1.0
Elegy (2008)	1.0
Half Light (2006)	1.0
Unvanquished, The (Aparajito) (1957)	1.0

From the above output, we can see that the movies that have high correlation with Forrest Gump (1994) are not very well known. This shows that correlation alone is not a good metric for similarity because there can be a user who watched 'Forest Gump (1994) and only one other movie and rated both of them as 5. A solution to this problem is to retrieve only those correlated movies that have at least more than 50 ratings. To do so, we will add the rating_counts column from the rating_mean_count dataframe to our corr_forrest_gump dataframe.

In [19]:
```
corr_forrest_gump = corr_forrest_gump.join(ratings_mean_count['rating_counts'])
corr_forrest_gump.head()
```

Out[19]:

title	Correlation	rating_counts
'burbs, The (1989)	0.056266	20
(500) Days of Summer (2009)	0.144325	37
*batteries not included (1987)	0.000000	11
...And Justice for All (1979)	0.089924	10
10 (1979)	0.693375	3

We can see that the movie 10, which has the highest correlation has only three ratings. This means that only three users gave same ratings to Forest Gump (1994). However, we can deduce that a movie cannot be declared similar to the another movie based on just 3 ratings. This is why we added rating_counts column. Now, let's now filter movies correlated to Forest Gump (1994), that have more than 50 ratings. The following code snippet will do that-

In [20]:
```
corr_forrest_gump[corr_forrest_gump ['rating_counts']>50].sort_values('Correlation', ascending=False).head()
```

Out[20]:

	Correlation	rating_counts
title		
Forrest Gump (1994)	1.000000	311
Happy Gilmore (1996)	0.715602	79
12 Angry Men (1957)	0.545139	63
As Good as It Gets (1997)	0.521448	98
First Knight (1995)	0.520438	52

Now, we can see from the above output the movies that are highly correlated with Forrest Gump (1994). The movies in the list are some of the most famous movies Hollywood movies, and since Forest Gump (1994) is also a very famous movie, there is a high chance that these movies are highly correlated. Thus, we created a simple recommender system.