CS582 Machine Learning - Project 1

Professor: Mohamed Abdelrazik

Team:

> Tam Van Vo - 610746

> Quynh Pham - 610716

> Samsher Bahadur Rana - 611060

> Van Vong Tran - 610772

> Yared Geberetsadik Beyene - 110466

Project Colab URL:

https://colab.research.google.com/drive/1iqL0DPrwJ2okp3muu7IZ-pl4RKnbMr-1

Dataset:

https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

Ref:

https://www.dataquest.io/blog/kaggle-getting-started/

Install auto-sklearn

```
# !apt-get install swig -y
# !pip install Cython numpy
# !pip install auto-sklearn
```

Load all libs

```
from random import randrange
import numpy as np
import seaborn as sns
import pandas as pd
import pandas.util.testing as tm
import matplotlib.pyplot as plt
from sklearn import preprocessing
```

Step 1: Data Loading

```
!git clone https://github.com/votamvan/cs582.git
```

```
fatal: destination path 'cs582' already exists and is not an empty directory.
```

```
#from google.colab import files
#uploaded = files.upload()
df_train = pd.read_csv('/content/cs582/data/house-prices/train.csv')
df_test = pd.read_csv('/content/cs582/data/house-prices/test.csv')
df_train.sample(5)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	Lan
1382	1383	70	RM	60.0	7200	Pave	NaN	Reg	Lvl	AllPub	Corner	
585	586	20	RL	88.0	11443	Pave	NaN	Reg	Lvl	AllPub	Inside	
1450	1451	90	RL	60.0	9000	Pave	NaN	Reg	Lvl	AllPub	FR2	
13	14	20	RL	91.0	10652	Pave	NaN	IR1	Lvl	AllPub	Inside	
455	456	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	Inside	

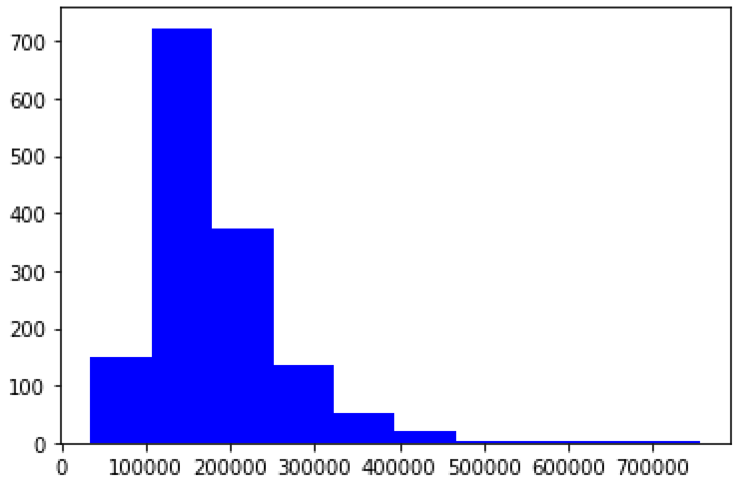
5 rows × 81 columns

Step 2: EDA

SalePrice

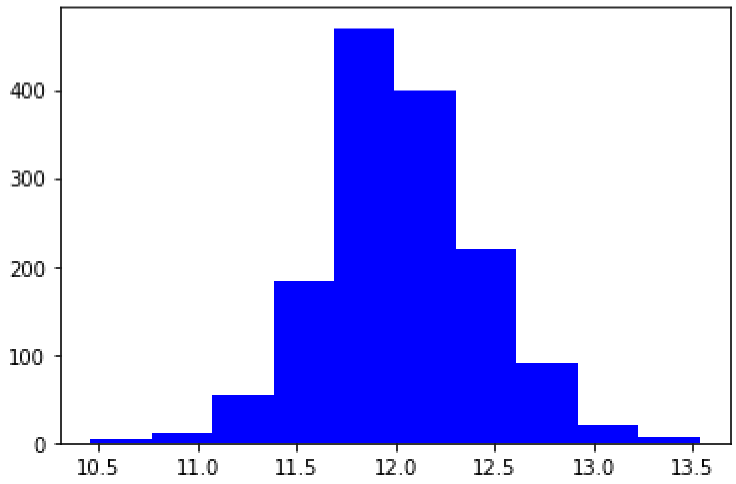
```
print ("SalePrice Skew = ", df_train.SalePrice.skew())
ax = plt.hist(df_train.SalePrice, color='blue')
```

```
SalePrice Skew =  1.8828757597682129
```




```
target = np.log(df_train.SalePrice)
print ("SalePrice Skew is:", target.skew())
ax = plt.hist(target, color='blue')
```

```
SalePrice Skew is: 0.12133506220520406
```



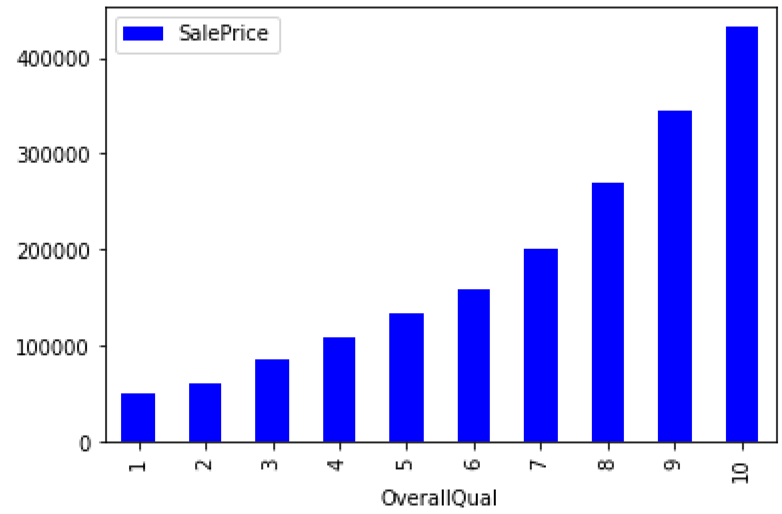
Handle Numerical Data

```
numeric_features = df_train.select_dtypes(include=[np.number])
corr = numeric_features.corr()
print("Correlation with SalePrice")
print (corr['SalePrice'].sort_values(ascending=False)[:5])
print (corr['SalePrice'].sort_values(ascending=False)[-5:])
```

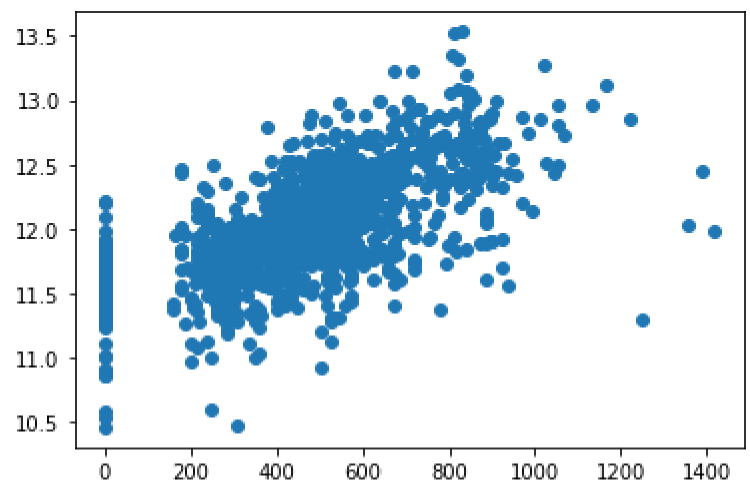
```
Correlation with SalePrice
SalePrice      1.000000
OverallQual    0.790982
GrLivArea      0.708624
GarageCars     0.640409
GarageArea     0.623431
Name: SalePrice, dtype: float64
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
Name: SalePrice, dtype: float64
```

```
quality_pivot = df_train.pivot_table(index='OverallQual',
                                     values='SalePrice', aggfunc=np.median)
ax = quality_pivot.plot(kind='bar', color='blue')
print("The median sales price strictly increases as Overall Quality increases.")
```

The median sales price strictly increases as Overall Quality increases.

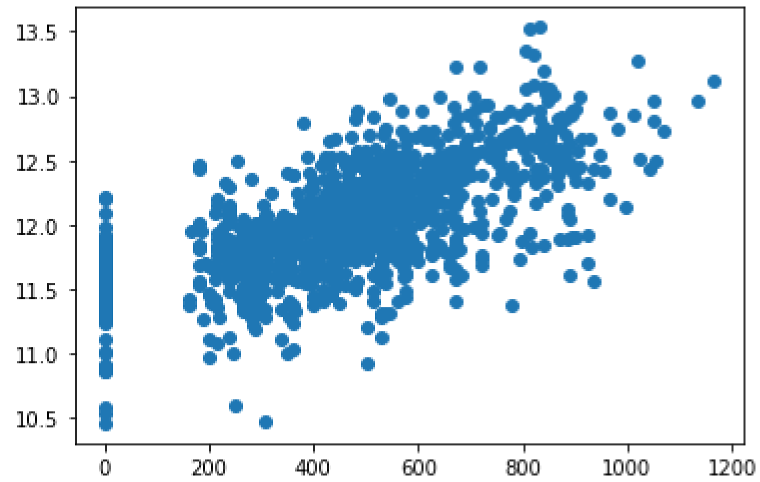



```
ax = plt.scatter(x=df_train['GarageArea'], y=target)
```




```
print('Remove outliers')
df_train = df_train[df_train['GarageArea'] < 1200]
target = np.log(df_train.SalePrice)
ax = plt.scatter(x=df_train['GarageArea'], y=target)
```

```
Remove outliers
```



Handle non-numerial data

```
'Street','Alley'
'Grvl':0, 'Pave':1,'NA':-1

'LotShape'
'Reg':3,'IR1':2,'IR2':1,'IR3':0

'LandContour'
'Lvl':1,'Bnk':0,'Low':0,'HLS':0

'Utilities'
'AllPub': 3,'NoSewr': 2,'NoSeWa': 1,'ELO':0

'BsmtFinType1','BsmtFinType2','GarageFinish'
'GLQ':5,'ALQ':4,'BLQ':3,'Rec':2,'LwQ':1,'Unf':0, 'RFn':1,'Fin':2

'PavedDrive'
'N':0,'Y':1,'P':0.5

'ExterQual','ExterCond','BsmtQual','BsmtCond','BsmtExposure','HeatingQC','KitchenQual','FireplaceQu','GarageQual','GarageCond','PoolQC'
'Gd':3, 'TA':2, 'Ex':4, 'Fa':1, 'NA':-1, 'Po':0, 'No':0,'Mn':1,'Av':2
```

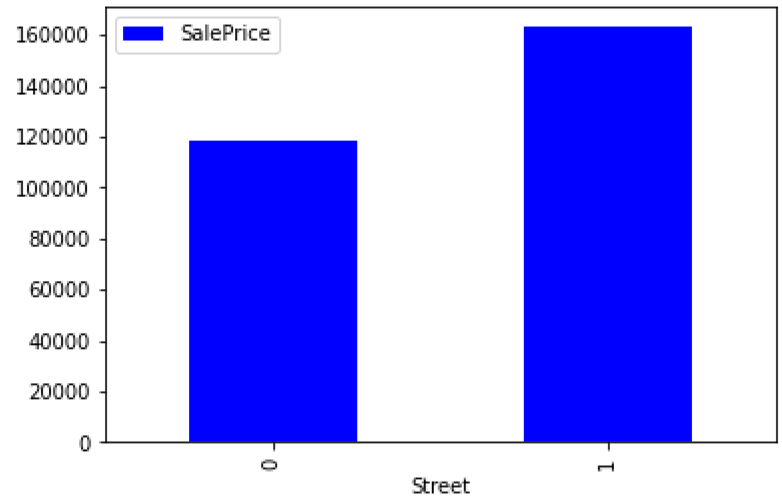
```
'CentralAir'
'N':0,'Y':1
```

```
def ranking_features(df,feature,di):
  for feature in features:
      df[feature].replace(di, inplace=True)
      df[feature]=df[feature].fillna(-1)
  return df
di = {'Grvl':0, 'Pave':1,'NA':-1,'Reg':3,'IR1':2,'IR2':1,
      'IR3':0,'Lvl':1,'Bnk':0,'Low':0,'HLS':0,'AllPub': 3,
      'NoSewr': 2,'NoSeWa': 1,'ELO':0,'GLQ':5,'ALQ':4,'BLQ':3,
      'Rec':2,'LwQ':1,'Unf':0, 'RFn':1,'Fin':2,'N':0,'Y':1,
      'P':0.5,'Gd':3, 'TA':2, 'Ex':4, 'Fa':1, 'Po':0, 'No':0,'Mn':1,'Av':2}
features = ['Street','Alley','LotShape','LandContour','Utilities','BsmtFinType1',
            'BsmtFinType2','GarageFinish','PavedDrive','ExterQual','ExterCond',
            'BsmtQual','BsmtCond','BsmtExposure','HeatingQC','KitchenQual','FireplaceQu',
            'GarageQual','GarageCond','PoolQC','CentralAir']
df_train = ranking_features(df_train,features,di)
df_test = ranking_features(df_test,features,di)
```

Encode Street data

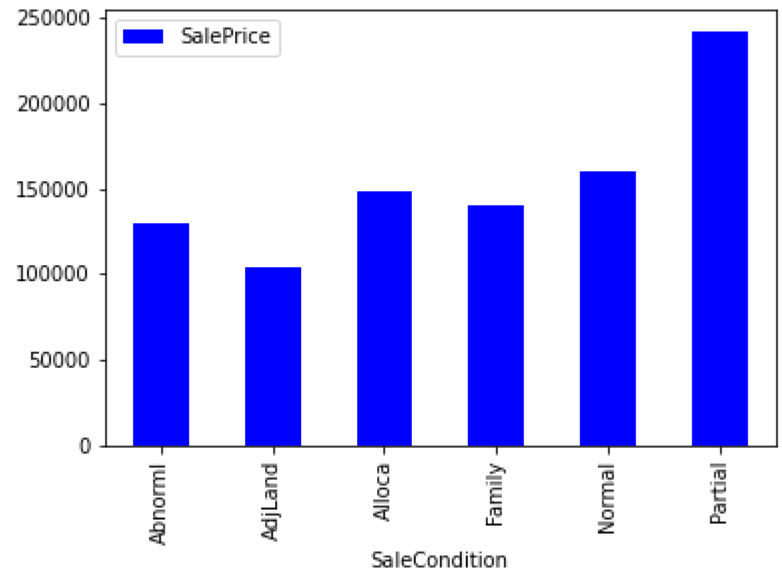
```
street_pivot = df_train.pivot_table(index='Street',
                                    values='SalePrice', aggfunc=np.median)
ax = street_pivot.plot(kind='bar', color='blue')
df_train['enc_street'] = pd.get_dummies(df_train.Street, drop_first=True)
df_test['enc_street'] = pd.get_dummies(df_test.Street, drop_first=True)
print (df_train.enc_street.value_counts())
```

```
1    1450
0       5
Name: enc_street, dtype: int64
```



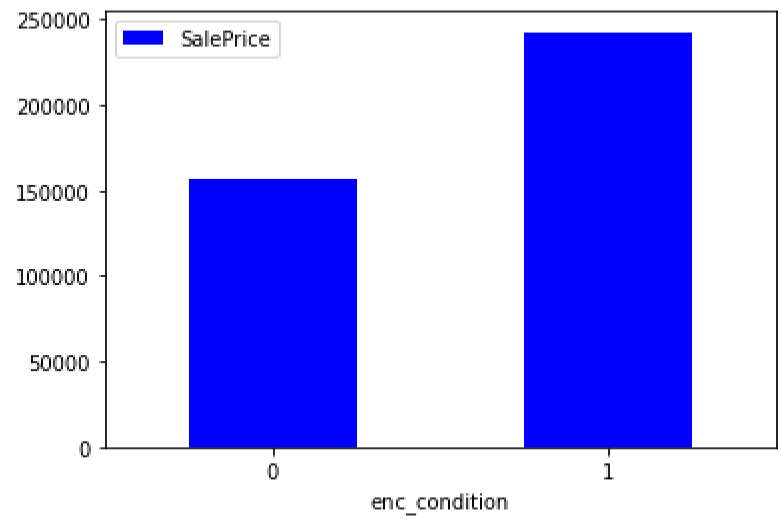
Encode SaleCondition

```
condition_pivot = df_train.pivot_table(index='SaleCondition',
                                       values='SalePrice', aggfunc=np.median)
ax = condition_pivot.plot(kind='bar', color='blue')
```




```
def encode(x):
 return 1 if x == 'Partial' else 0
df_train['enc_condition'] = df_train.SaleCondition.apply(encode)
df_test['enc_condition'] = df_test.SaleCondition.apply(encode)
condition_pivot = df_train.pivot_table(index='enc_condition',
                                       values='SalePrice', aggfunc=np.median)
condition_pivot.plot(kind='bar', color='blue')
ax = plt.xticks(rotation=0)
```

```
ax = plt.xticks(rotation=0)
```



Handle null data

```
def unique_nullcount(df):
  rows = []
  for (i, j) in df.iteritems():
      rows.append([i, df_train[i].nunique(), df_train[i].isna().sum()])
  df = pd.DataFrame(rows, columns=["Feature", "Unique value","Count null"])
  .sort_values(["Unique value","Count null"], ascending = (True,True))
  pd.set_option('display.max_rows', df.shape[0]+1)
  return df

def unique_value_list(df,feature, ct):
  features = df[df[feature] < ct]['Feature']
  for i in features:
      print(i, df_train[i].unique())
```

```
df = unique_nullcount(df_train.select_dtypes(include=[np.number]))
print(df[df['Count null']>0])
```

```
          Feature  Unique value  Count null
41  GarageYrBlt            97          81
2   LotFrontage           110         258
13   MasVnrArea           325           8
```

```
df_train.LotFrontage=df_train.LotFrontage.fillna(0.0)
df_train.MasVnrArea=df_train.MasVnrArea.fillna(0.0)
df_train.GarageYrBlt=df_train.GarageYrBlt.fillna(df_train.GarageYrBlt.mode()[0])

df_test.LotFrontage=df_test.LotFrontage.fillna(0.0)
df_test.MasVnrArea=df_test.MasVnrArea.fillna(0.0)
df_test.GarageYrBlt=df_test.GarageYrBlt.fillna(df_test.GarageYrBlt.mode()[0])
```

```
df = unique_nullcount(df_train.select_dtypes(include=[np.number]))
print(df[df['Count null']>0])
```

```
Empty DataFrame
Columns: [Feature, Unique value, Count null]
Index: []
```

Step 3: Data Preparation

Finalize dataset

We categorize

```
data = df_train.select_dtypes(include=[np.number]).interpolate().dropna()
X_all = data.drop(['SalePrice', 'Id'], axis=1)
y_all = np.log(df_train.SalePrice)
mean_y = np.mean(y_all)
y_all[y_all <= mean_y] = 0
y_all[y_all > mean_y] = 1
y_all = y_all.astype(int)
# Standardization of datasets
#X, X_test, y, y_test = train_test_split(X, y,test_size=0.33, random_state=42)
X,y=X_all, y_all
scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
```

Step 4: Choose Model

1. Get best estimator
2. Training Curve and Validation Curve

General Functions

```
from sklearn.model_selection import GridSearchCV
def best_estimator(estimator,tuned_parameters,X,y,cv):
  grid = GridSearchCV(estimator, tuned_parameters,
                      cv = cv, scoring = 'accuracy', n_jobs=-1)
  grid.fit(X,y)
  best_estimator = grid.best_estimator_
  print(f'Best score: {grid.best_score_}')
  print(grid.best_estimator_)
  return best_estimator
```

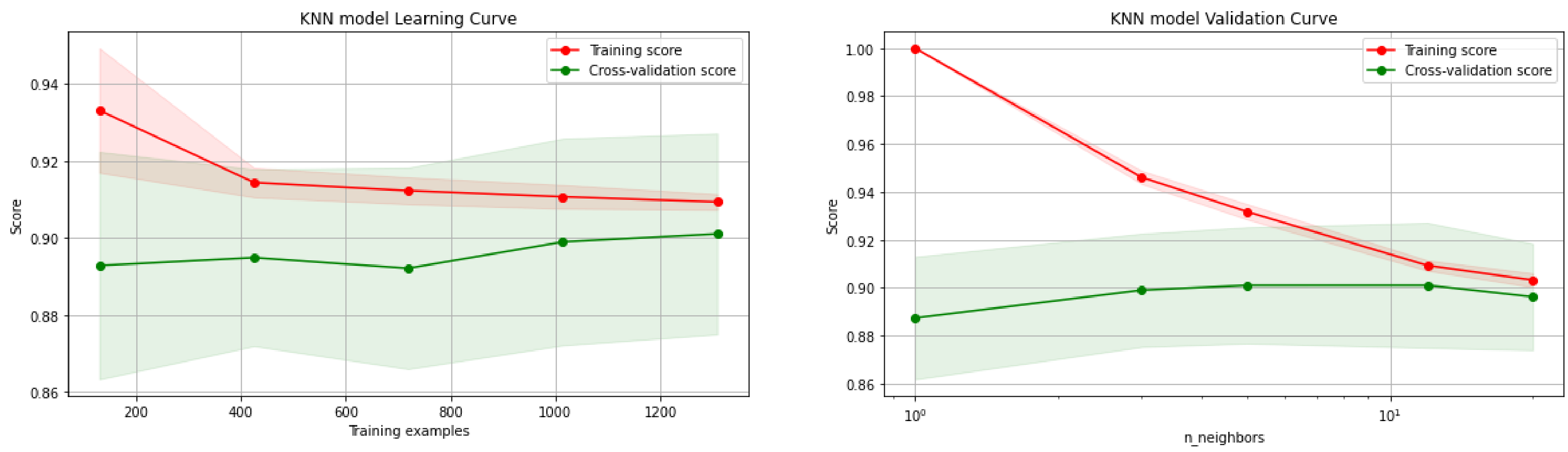
```
#show complexity curve (validation curve) and learning curve
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import validation_curve
from sklearn.model_selection import KFold
from sklearn.model_selection import ShuffleSplit
from sklearn.neighbors import KNeighborsClassifier
def plot_learning_curve(estimator, title, X, y, ax=None,
                        cv=None, train_sizes=np.linspace(.1, 1.0, 5)):
    if ax is None: _, ax = plt.subplots(figsize=(20, 5))
    ax.set_title(title+' Learning Curve')
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
                                                            cv=cv, n_jobs=-1, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    # Plot learning curve
    ax.grid()
    ax.fill_between(train_sizes, train_scores_mean - train_scores_std,
                      train_scores_mean + train_scores_std, alpha=0.1, color="r")
    ax.fill_between(train_sizes, test_scores_mean - test_scores_std,
                      test_scores_mean + test_scores_std, alpha=0.1, color="g")
    ax.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
    ax.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")
    ax.legend(loc="best")
    ax.set_xlabel("Training examples")
    ax.set_ylabel("Score")
    return plt
def plot_validation_curve(estimator, title, X, y, ax=None, cv=None, param_name=None, param_range=None):
    if ax is None: _, ax = plt.subplots(figsize=(20, 5))
    ax.set_title(title+' Validation Curve')
    train_scores, test_scores = validation_curve(estimator, X, y, param_name=param_name,
                                                 param_range=param_range,
                                                 scoring="accuracy", cv=cv, n_jobs=-1)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    # Plot learning curve
    ax.grid()
    ax.semilogx(param_range, train_scores_mean, 'o-', color="r", label="Training score")
    ax.fill_between(param_range, train_scores_mean - train_scores_std,
                      train_scores_mean + train_scores_std, alpha=0.1, color="r")
    ax.semilogx(param_range, test_scores_mean, 'o-', color="g", label="Cross-validation score")
    ax.fill_between(param_range, test_scores_mean - test_scores_std,
                      test_scores_mean + test_scores_std, alpha=0.1, color="g")
    ax.legend(loc="best")
    ax.set_xlabel(param_name)
    ax.set_ylabel("Score")
    return plt


def show_TV_curve(estimator, title, X,y, param_name, param_range, cv) :
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
  plot_learning_curve(estimator, title, X, y, ax=ax1, cv=cv)
  plot_validation_curve(estimator, title, X, y, ax=ax2, cv=cv,
                        param_name=param_name, param_range=param_range)
  plt.show()
```

1/ KNN

```
from sklearn.neighbors import KNeighborsClassifier
weight_options = ["uniform", "distance"]
k_range = np.arange(1,30)
tuned_parameters = dict(n_neighbors = k_range, weights = weight_options)
knn_estimator = best_estimator(KNeighborsClassifier(), tuned_parameters,X,y,10)
```

```
Best score: 0.9127147766323024
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=12, p=2,
                     weights='distance')
```

```
param_range = [1,3,5,12,20]
show_TV_curve(KNeighborsClassifier(12),'KNN model',X,y,'n_neighbors',param_range,10)
```



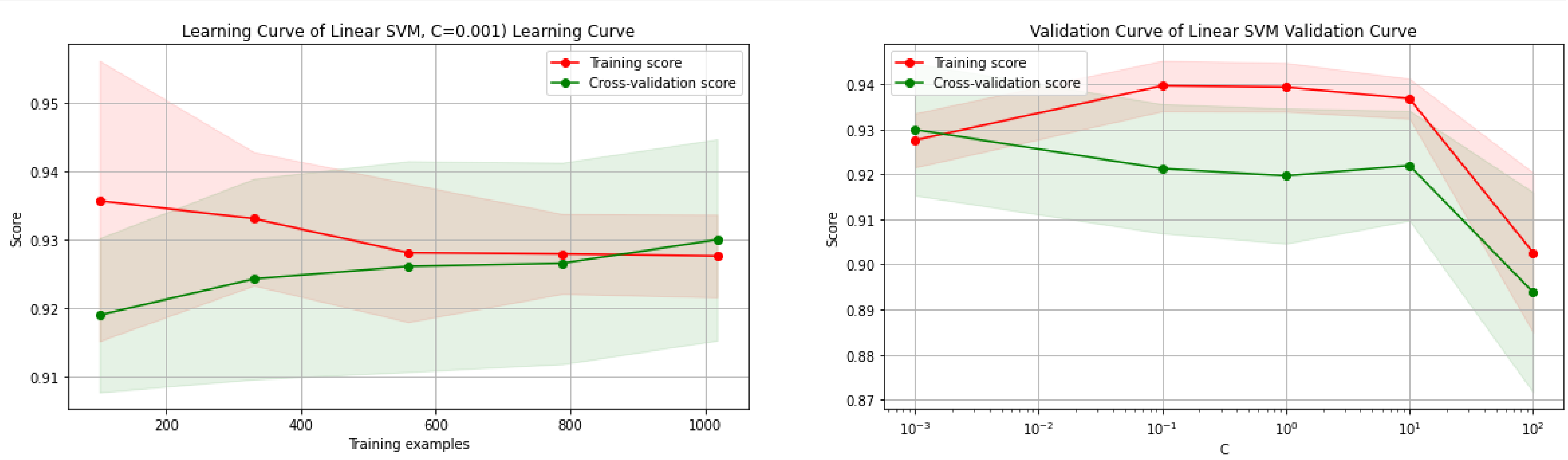
From the Validation Curve, by increasing the n_neighbors value we getting better training score and CV score. But after n_neighbors> 12, the curves going down. So after this point if we increasing the k doesn't improve the model

2/ SVC

```
from sklearn.svm import LinearSVC
tuned_parameters = {
    'C': [0.001, 0.1, 1, 10, 100]
}
svc_estimator = best_estimator(LinearSVC(), tuned_parameters,X,y,5)
```

```
Best score: 0.9257731958762887
LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

```
cv = ShuffleSplit(n_splits=10, test_size=0.3, random_state=0)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
title = f"Learning Curve of Linear SVM, C={svc_estimator.C})"
plot_learning_curve(svc_estimator, title, X, y, ax=ax1, cv=cv)
plot_validation_curve(svc_estimator, "Validation Curve of Linear SVM", X, y, ax=ax2, cv=cv,
                      param_name="C", param_range=[0.001, 0.1, 1, 10, 100])
plt.show()
```

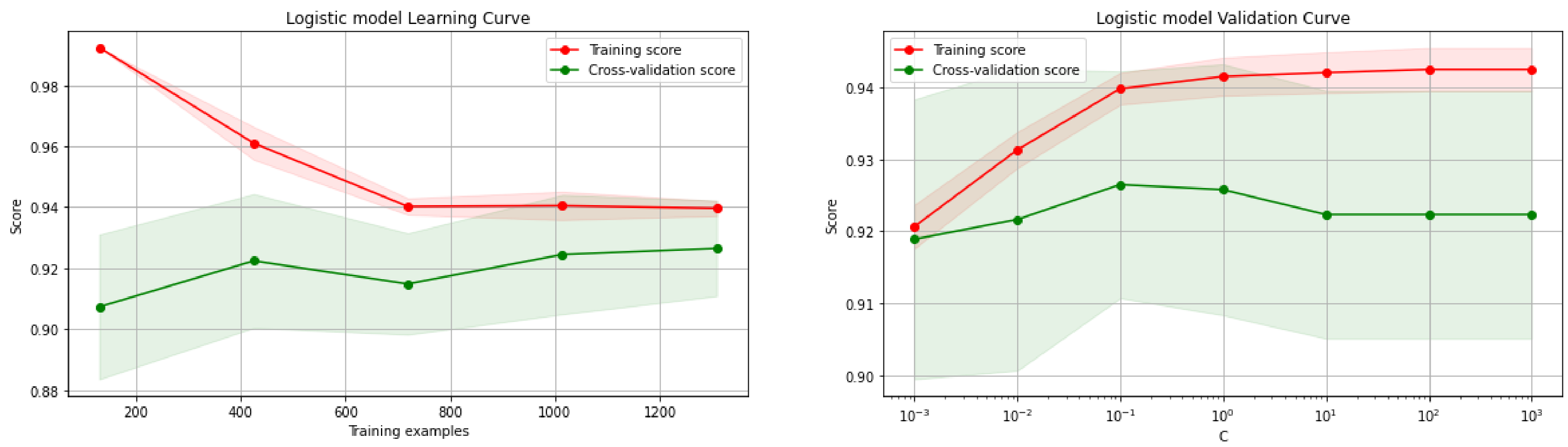


3/ Logistic

```
from sklearn.linear_model import LogisticRegression
tuned_parameters = {
    'C': np.logspace(-3,3,7),
    'penalty':['l1','l2']# l1 lasso l2 ridge
}
log_estimator = best_estimator(LogisticRegression(), tuned_parameters,X,y,10)
```

```
Best score: 0.9264604810996564
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be chan
  FutureWarning)
```

```
param_range = np.logspace(-3,3,7)
show_TV_curve(LogisticRegression(C=0.1),'Logistic model',X,y,'C',param_range,10)
```

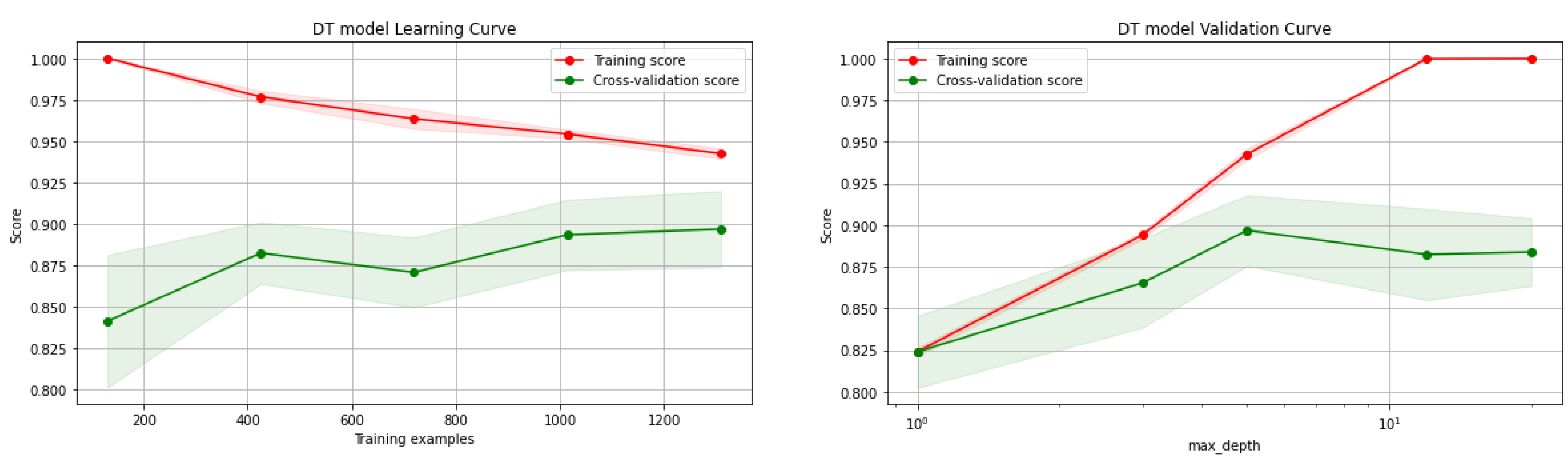


4/ Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
tuned_parameters = {
     'max_depth':np.arange(1,30)
}
dt_estimator = best_estimator(DecisionTreeClassifier(), tuned_parameters,X,y,5)
```

```
Best score: 0.902405498281787
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=6,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

```
param_range = [1,3,5,12,20]
show_TV_curve(DecisionTreeClassifier(max_depth=5),'DT model',X,y,'max_depth',param_range,10)
```



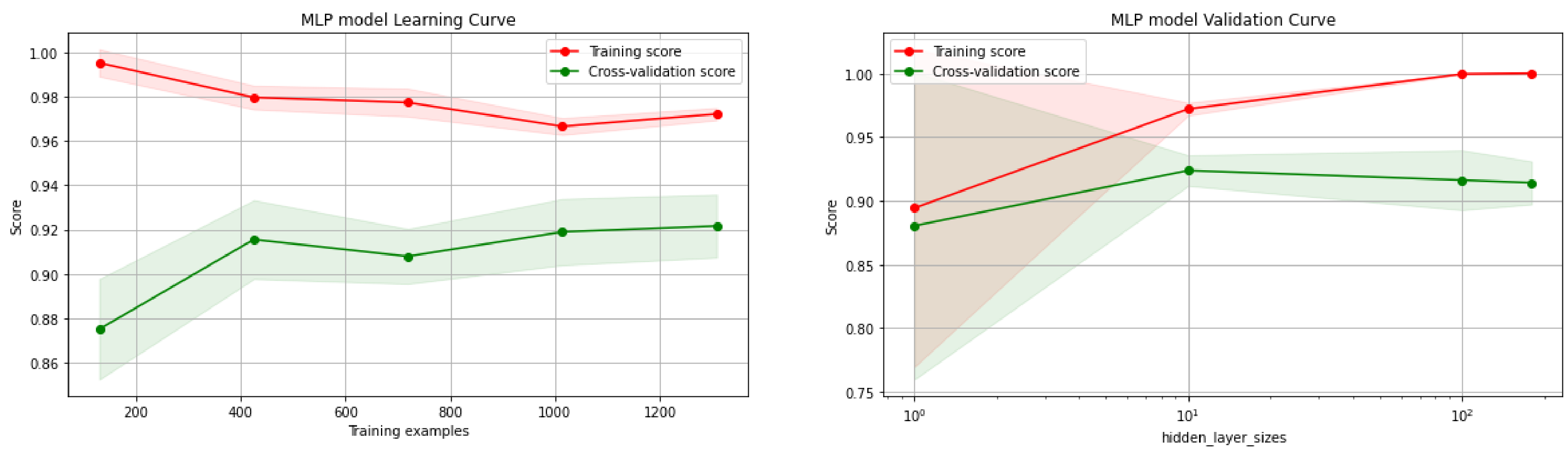
From the validation curve we can see with max_depth in between 5-7, model has pretty good performance but by increasing the depth, the model becomes overfitting

5/ MLP

```
from sklearn.neural_network import MLPClassifier
tuned_parameters = {
    'hidden_layer_sizes': [(10,),(50,),(100,)]
}
mlp_estimator = best_estimator(MLPClassifier(), tuned_parameters,X,y,12)
```

```
Best score: 0.9230240549828179
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(10,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:566: ConvergenceWarning: Stochas
  % self.max_iter, ConvergenceWarning)
```

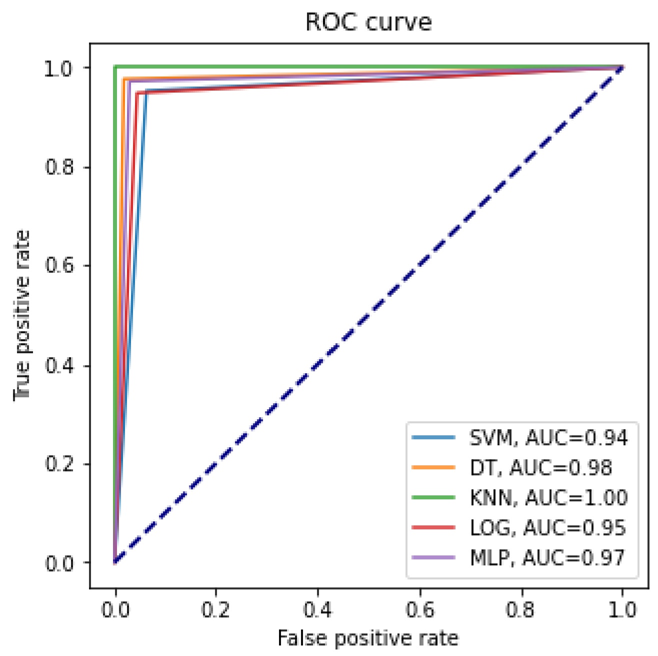
```
param_range = [1, 10, 100,180]
show_TV_curve(MLPClassifier(hidden_layer_sizes=10),'MLP model',X,y,'hidden_layer_sizes',param_range,10)
```



From the validation curve we can see with hidden_layer_sizes = 10, model has pretty good performance but by increasing the size, the model become overfitting

Step 5: AUC curve

```
from sklearn.metrics import roc_curve
from sklearn import metrics
def plot_roc_curve(estimators, titles, X, y, ax=None):
    if ax is None: _, ax = plt.subplots(figsize=(5, 5))

    for i, estimator in enumerate(estimators):
      y_pred = estimator.predict(X)
      fpr, tpr, _ = roc_curve(y, y_pred)
      ax.plot(fpr, tpr, label=f"{titles[i]}, AUC=" + "{:.2f}".format(metrics.auc(fpr, tpr)))

    ax.set_title('ROC curve')
    ax.legend(loc='best')
    ax.set_xlabel('False positive rate')
    ax.set_ylabel('True positive rate')
    ax.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    return plt
estimators = [svc_estimator,dt_estimator,knn_estimator,log_estimator,mlp_estimator]
titles = ['SVM','DT','KNN','LOG','MLP']
plot_roc_curve(estimators, titles, X_test, y_test)
plt.show()
```



KNN has highest score but it seems to be overfitting.

Step 6: Ensemble

By Voting

```
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import warnings

eclf1 = VotingClassifier(voting='hard',
 estimators=[('knn', knn_estimator), ('svc', svc_estimator),
 ('log', log_estimator), ('dt', dt_estimator), ('mlp', mlp_estimator)] )

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.33, random_state=42)
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    eclf1.fit(X_train, y_train)
    y_pred = eclf1.predict(X_test)
    print(f"Voting Score = {accuracy_score(y_test, y_pred)}")
```

```
Voting Score = 0.9521829521829522
```

Step 7: Apply AutoML

```
import sklearn
import autosklearn.classification

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.33, random_state=42)
automl = autosklearn.classification.AutoSklearnClassifier()
automl.fit(X_train, y_train)
y_hat = automl.predict(X_test)
print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))
print(sklearn.metrics.classification_report(y_test, y_hat))
```

```
Accuracy score 0.9376299376299376
              precision    recall  f1-score   support

           0       0.94      0.95      0.94       270
           1       0.93      0.92      0.93       211

    accuracy                           0.94       481
   macro avg       0.94      0.94      0.94       481
weighted avg       0.94      0.94      0.94       481
```