MACHINE LEARNING AND ITS USES

Isabel Adcock and William Robinson

```
[47]: #try to apply the learning to the new dataset, and see it's accuracy here

[48]: ans = "C:/Users/isabe/Downloads/titanic/gender_submission.csv"
      answers = pd.read_csv(ans)

[49]: answers = answers["Survived"]

[50]: Accuracy_RandomForest_Test = rf.score(test_x, answers)
      Accuracy_RF_int_Test = round(Accuracy_RandomForest_Test * 100)
      print("Accuracy = {}%".format(Accuracy_RF_int_Test))

      Accuracy = 100%

[51]: Accuracy_LogisticRegression_Test = lgr.score(test_x, answers)
      Accuracy_LR_int_Test = round(Accuracy_LogisticRegression_Test * 100)
      print("Accuracy = {}%".format(Accuracy_LR_int_Test))

      Accuracy = 100%

[52]: #I have not done this right!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      #thats quite annoying tbh
```

WHAT IS MACHINE LEARNING?

- Machine learning is a kind of artificial intelligence
- It "gives computers the ability to learn without explicitly being programmed."
- The future of industry?

THE TYPES OF MACHINE LEARNING

Descriptive

- Where the system uses data to explain what has happened
- The machine describes

Predictive

- Where the system uses data to predict what will happen
- The machine predicts

Prescriptive

- Where the system uses data to suggest follow-up actions
- The machine prescribes

How to: Machine Learning

[1]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import os
os.chdir("C:/Users/isabe/Downloads")
tr = "C:/Users/isabe/Downloads/titanic/train.csv"
train = pd.read_csv(tr)
train.head()
```

Import software

Get data

[3]:
```
train_x = train[["Pclass", "Sex", "Age"]]
```

Define variables

[4]:
```
age_mean = round(train["Age"].mean())
train_x = train_x.fillna(value=age_mean)
```

[5]:
```
train_x["Sex"].replace("male", 1, inplace = True)
train_x["Sex"].replace("female", 0, inplace = True)
```

Format data

[7]:
```
train_y = train[["Survived"]]
tr_x, cv_x, tr_y, cv_y = train_test_split(train_x, train_y, test_size = 0.30)
```

Split data into test and train

[8]:
```
rf = RandomForestClassifier()
rf.fit(tr_x, tr_y)
Accuracy_RandomForest = rf.score(cv_x, cv_y)
Accuracy_RF_int = round(Accuracy_RandomForest
print("Accuracy = {}%".format(Accuracy_RF
```

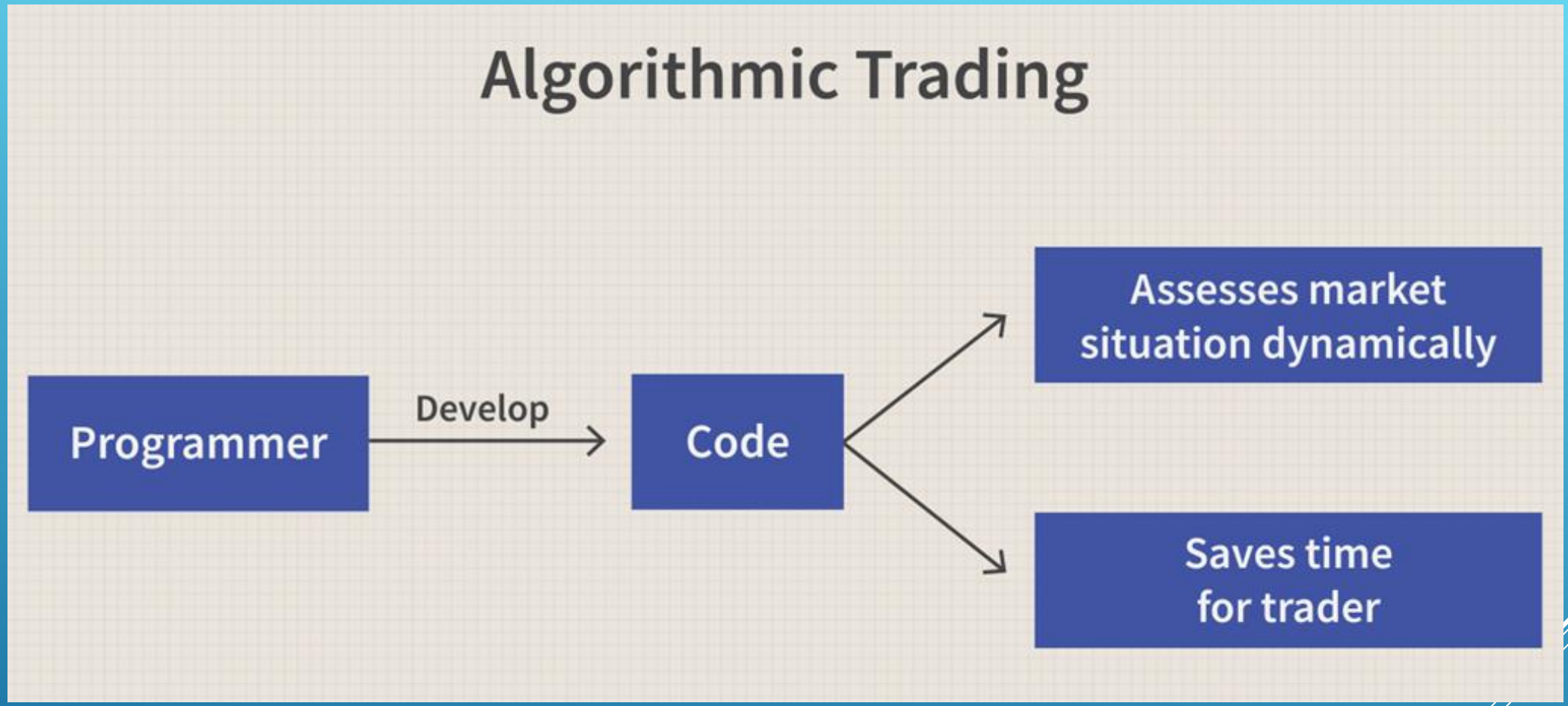
```
Accuracy = 80%
```

Fit the training data into the machine

[9]:
```
lgr = LogisticRegression()
lgr.fit(tr_x, tr_y)
Accuracy_LogisticRegression = lgr.score(cv_x, cv_y)
Accuracy_LR_int = round(Accuracy_LogisticRegression * 100)
print("Accuracy = {}%".format(Accuracy_LR_int))
```

```
Accuracy = 81%
```

Check the accuracy on the test data



FUTURE OF MACHINE LEARNING IN CREDIT SUISSE

MACHINE LEARNING IN CREDIT SUISSE

More to the point, Credit Suisse is changing the type of people it needs to hire, he said.

"We've changed our thinking from a regular stat-based quant to machine learning, which is a big change, and the challenge of finding [and hiring candidates with] the right skill set is a big issue."

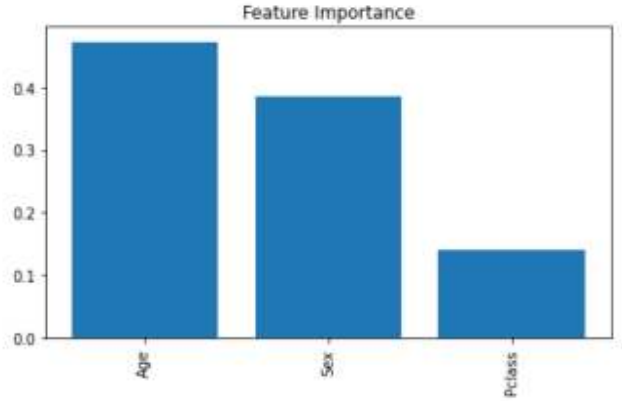
While unstructured data is increasingly in demand, Credit Suisse typically partners with providers who convert data sets from unstructured to structured, rather than bring people in-house to do such tasks.

"From a company's perspective, how do we want to use this data, and what are the use cases? You still need to be able to make sense of it and read where the signals are pointing," Singhvi said. "The capacity to be able to build the model – that's where you need a lot of that data.

PROBLEMS WITH MACHINE LEARNING

- Overfitting
- Oversampling
- Underfitting
- Undersampling
- Bias
- Variance
- Noisy Data
- Incorrect Assumptions

```
[35]: plt.title("Feature Importance")
      plt.bar(range(test_x.shape[1]), importances[ordered_importances], align = "center")
      plt.xticks(range(test_x.shape[1]),test_x.columns[ordered_importances], rotation = 90)
      plt.tight_layout()
      plt.show()
```

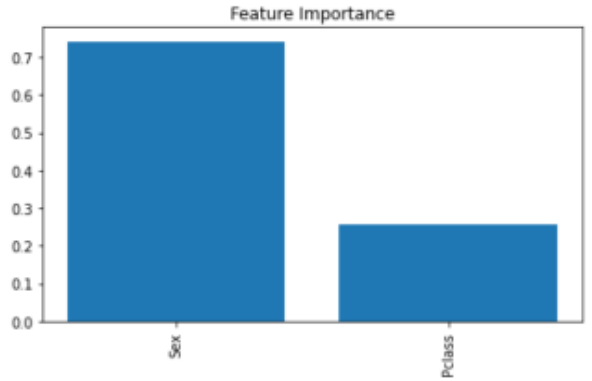



```
[36]: #This is maybe not a good thing, Age is weighted highly, but 177 of the values are the same.
      #Could there be a correlation between missing data for age and survival?
```

```
[21]: ordered_importances
```

```
[21]: array([1, 0], dtype=int64)
```

```
[22]: plt.title("Feature Importance")
      plt.bar(range(test_x.shape[1]), importances[ordered_importances], align = "center")
      plt.xticks(range(test_x.shape[1]),test_x.columns[ordered_importances], rotation = 90)
      plt.tight_layout()
      plt.show()
```




```
[50]: Accuracy_RandomForest_Test = rf.score(test_x, answers)
      Accuracy_RF_int_Test = round(Accuracy_RandomForest_Test * 100)
      print("Accuracy = {}%".format(Accuracy_RF_int_Test))
```

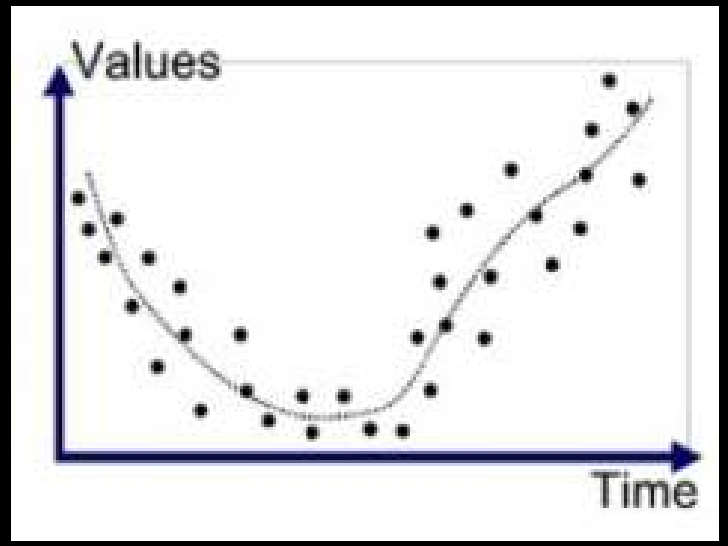
```
Accuracy = 100%
```

```
[51]: Accuracy_LogisticRegression_Test = lgr.score(test_x, answers)
      Accuracy_LR_int_Test = round(Accuracy_LogisticRegression_Test * 100)
      print("Accuracy = {}%".format(Accuracy_LR_int_Test))
```

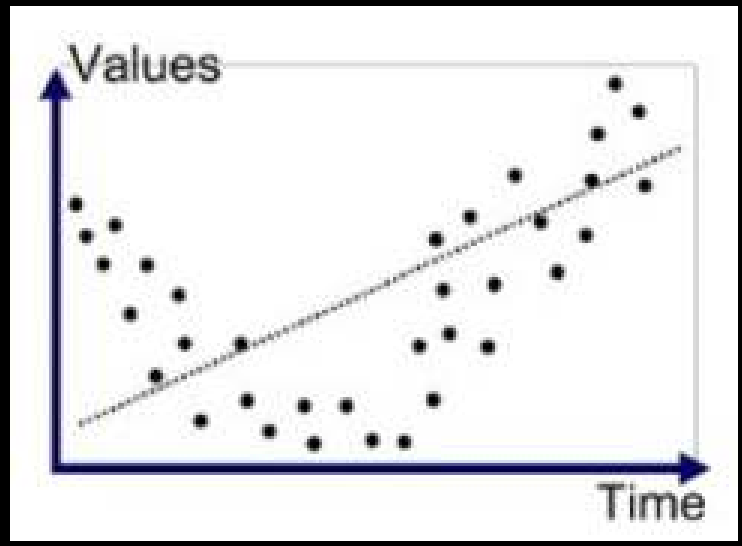
```
Accuracy = 100%
```

```
[52]: #I have not done this right!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      #thats quite annoying tbh
```

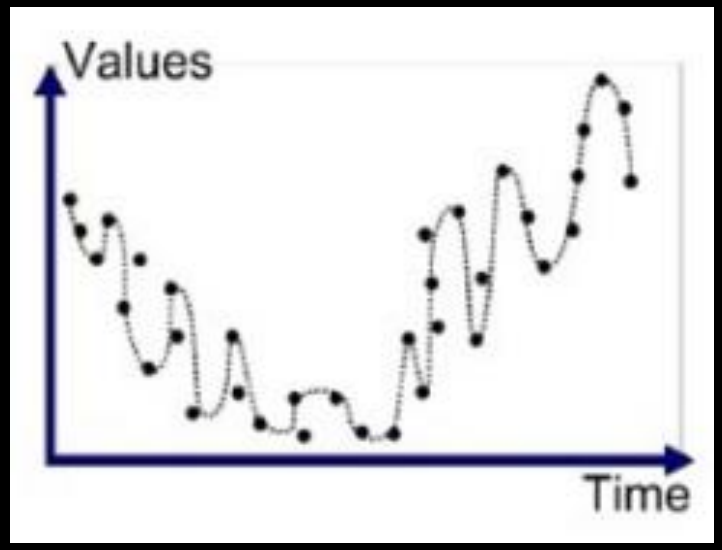
Correct Fitting



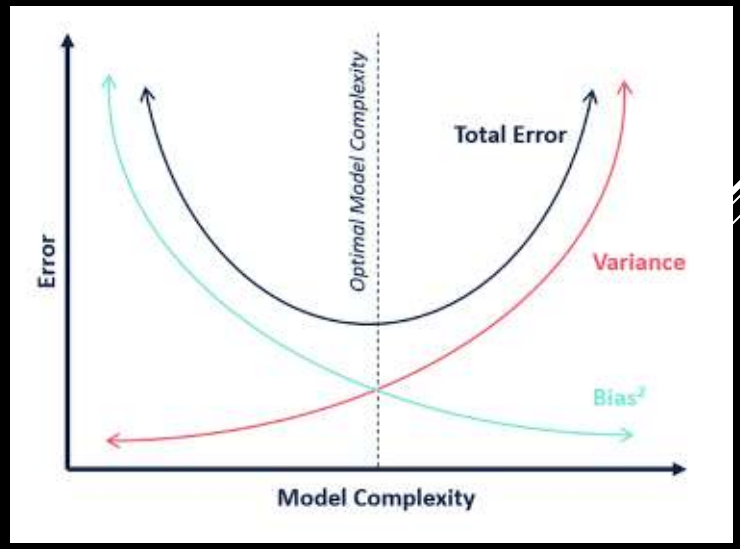
Underfitting



Overfitting



Bias Variance Tradeoff

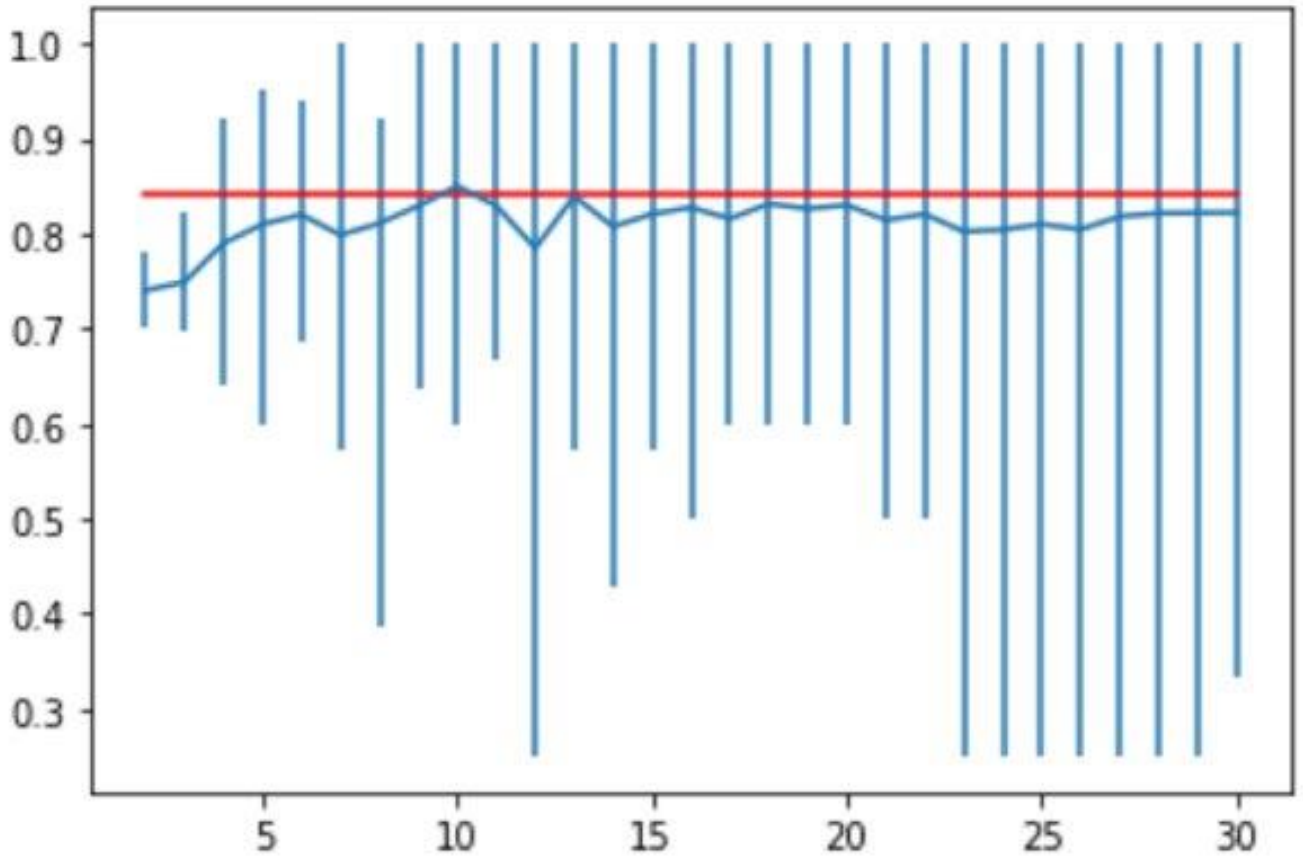


K-FOLD CROSS VALIDATION

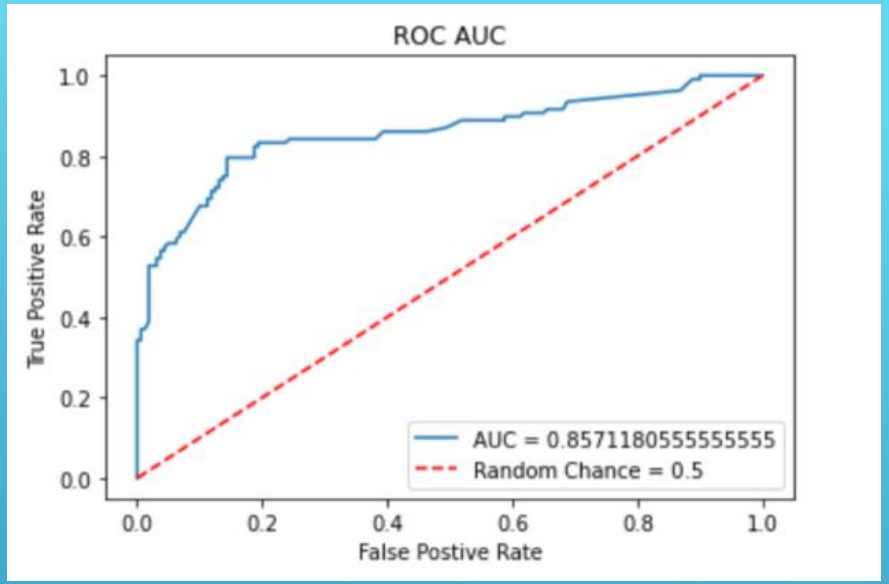
•[37]:
```
for k in folds:
    cv = KFold(n_splits = k, shuffle = True, random_state = 1)
    k_mean, k_min, k_max = evaluate_model(cv)
    print(" > folds = %d,  accuracy = %.3f (%.3f, %.3f)" % (k, k_mean, k_min, k_max))
    means.append(k_mean)
    mins.append(k_mean - k_min)
    maxs.append(k_max - k_mean)

#iterates through values of k in the range of folds
#cv prepares the cross validation procedure according the the current value of k
#uses the evaluate model function
#stores the value of the
```

```
 > folds = 2,  accuracy = 0.740 (0.700, 0.780)
 > folds = 3,  accuracy = 0.749 (0.697, 0.824)
 > folds = 4,  accuracy = 0.790 (0.640, 0.920)
 > folds = 5,  accuracy = 0.810 (0.600, 0.950)
 > folds = 6,  accuracy = 0.820 (0.688, 0.941)
 > folds = 7,  accuracy = 0.799 (0.571, 1.000)
 > folds = 8,  accuracy = 0.811 (0.385, 0.923)
 > folds = 9,  accuracy = 0.829 (0.636, 1.000)
 > folds = 10,  accuracy = 0.850 (0.600, 1.000)
 > folds = 11,  accuracy = 0.829 (0.667, 1.000)
 > folds = 12,  accuracy = 0.785 (0.250, 1.000)
 > folds = 13,  accuracy = 0.839 (0.571, 1.000)
 > folds = 14,  accuracy = 0.807 (0.429, 1.000)
 > folds = 15,  accuracy = 0.821 (0.571, 1.000)
```

[15]:
```
plt.errorbar(folds, means, yerr = [mins, maxs])
plt.plot(folds, [ideal for _ in range(len(folds))], color = "r")
plt.show()

#plots no. of folds against means on a wiggly line
#plots folds against the difference between the mins and maxs (which we made above)
#plots a red line representing the ideal we found at the beginning y = ideal basically
```



```
[13]: print("Accuracy Score: {}%".format(round(accuracy_score(actual, predicted)*100)))

      Accuracy Score: 80%

[14]: print("Classification Report: {}%".format(classification_report(actual, predicted)))

      Classification Report:              precision    recall  f1-score   support

                   0       0.81      0.86      0.84       161
                   1       0.77      0.70      0.74       107

            accuracy                           0.80       268
           macro avg       0.79      0.78      0.79       268
        weighted avg       0.80      0.80      0.80       268
        %
```



CONFUSION MATRICES

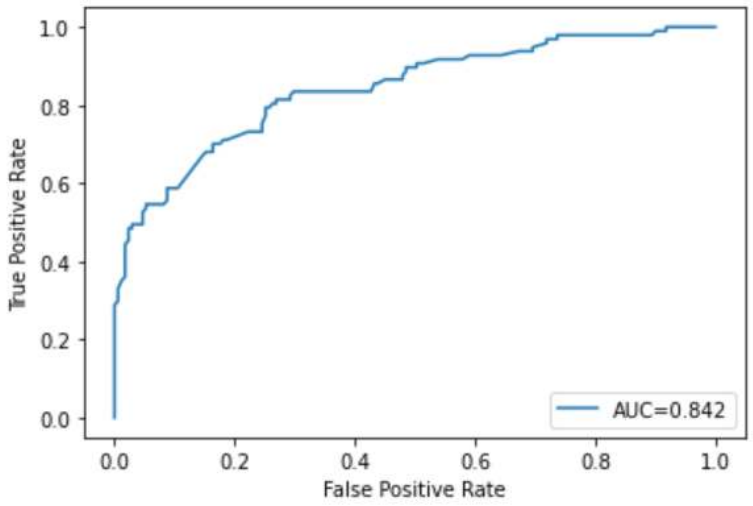
Analysing and evaluating predictions

	Predicted Positive	Predicted Negative
True Positive	True Positive	False Negative
True Negative	False Positive	True Negative

ROC AND AUC

In [55]:
```
#metrics
y_pred_proba = clf.predict_proba(crossValidation_x)[::, 1]
fpr, tpr, _ = metrics.roc_curve(crossValidation_y,  y_pred_proba)
auc = metrics.roc_auc_score(crossValidation_y, y_pred_proba)
print("AUC:", auc)
auc = round(auc, 3)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

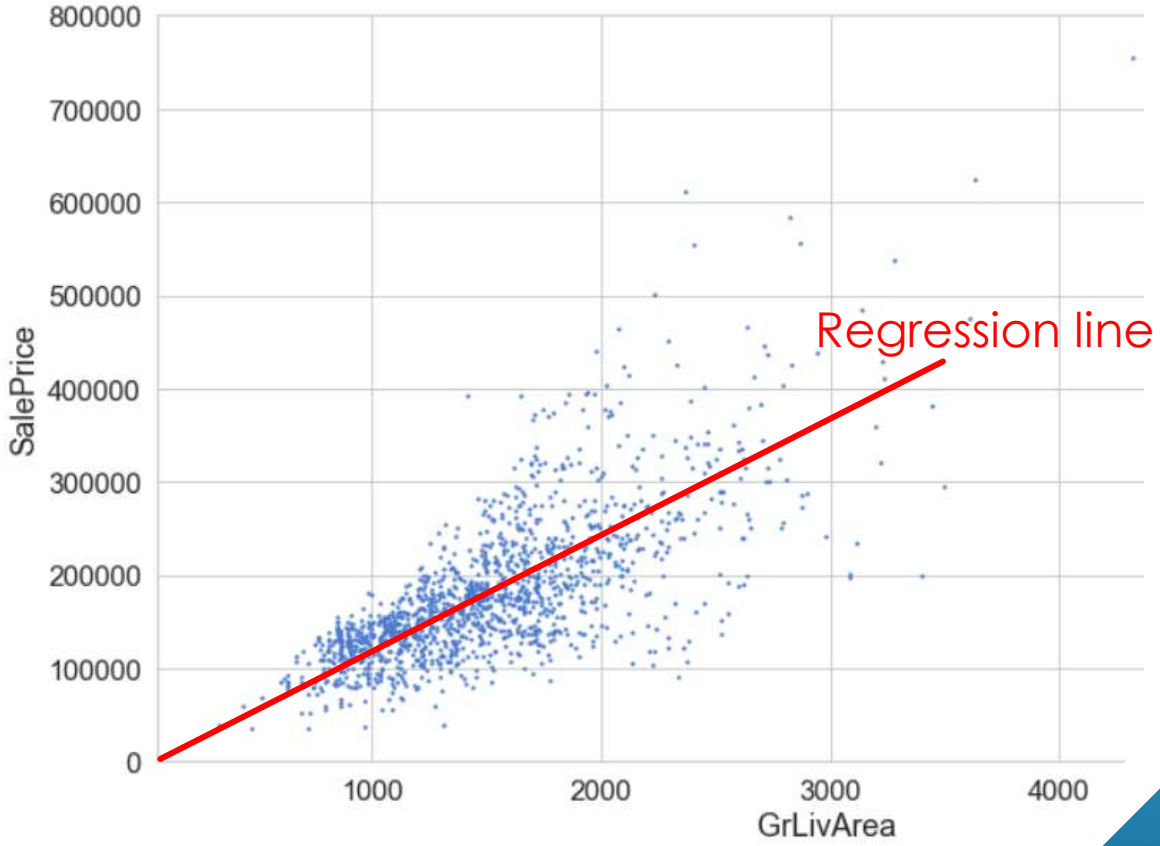
```
AUC: 0.8424669922228251
```



HOUSE PRICES

```
[61]: var = 'GrLivArea'
data = pd.concat([dataFrameTrain["SalePrice"], dataFrameTrain[var]], axis=1)
data.plot.scatter(x=var, y="SalePrice", ylim=(0,800000), s=2)
```

```
Out[61]: <AxesSubplot:xlabel='GrLivArea', ylabel='SalePrice'>
```



THANK YOU FOR LISTENING