Machine Learning and Deep Learning
Lecture-06

By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Spring 2025

CDSCO2004U/7.5 ECTS

Overview

- Ensembles:
  - Gradient Boosting
- Support Vector Machines (SVM)
- Performance Metrices

Ensembles: Gradient Boosting

Bagging and Boosting (from Lecture-05)

- Bagging builds many independent predictors and combine them using some model averaging techniques. (e.g. weighted average, majority vote)
  - Random Forest
    - Handle overfitting

- In Boosting, predictors are not made independently, but sequentially.
  - Gradient Boosting
    - Can overfit

- Ensemble methods work best when the predictors are as independent from each other.

Boosting

- Subsequent predictors learn from mistakes of previous predictors.
- Observations have an unequal probability of appearing in subsequent models
  - Model with highest error appear most.

- Predictors can be chosen from decision trees, regressors, classifiers.

- Gradient Boosting
  - Faster for prediction.
  - Stopping criteria important to avoid overfitting on training data.

Gradient Boosting

- Gradient Boosting = Gradient Descent + Boosting
- Works by sequentially adding predictors to an ensemble,
  - Each one correcting its predecessor.

- It tries to fit new predictor to residual errors made by previous predictor.
- Sum up predictions from all predictors.

- Variants:

  Adaptive Boosting/AdaBoost

  XGBoost/Extreme gradient boosting

Gradient Boosting

- Steps:
  - Fit an additive model (ensemble) in a forward stage-wise manner.
  - In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.
  - In Gradient Boosting, "shortcomings" are identified by gradients.
  - Gradients tell us how to improve model.

Learning rate shrinks the contribution of each tree by learning_rate

```
from sklearn.ensemble import GradientBoostingClassifier
GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, ... max_depth=1,
random_state=0).fit(X_train, y_train)
```

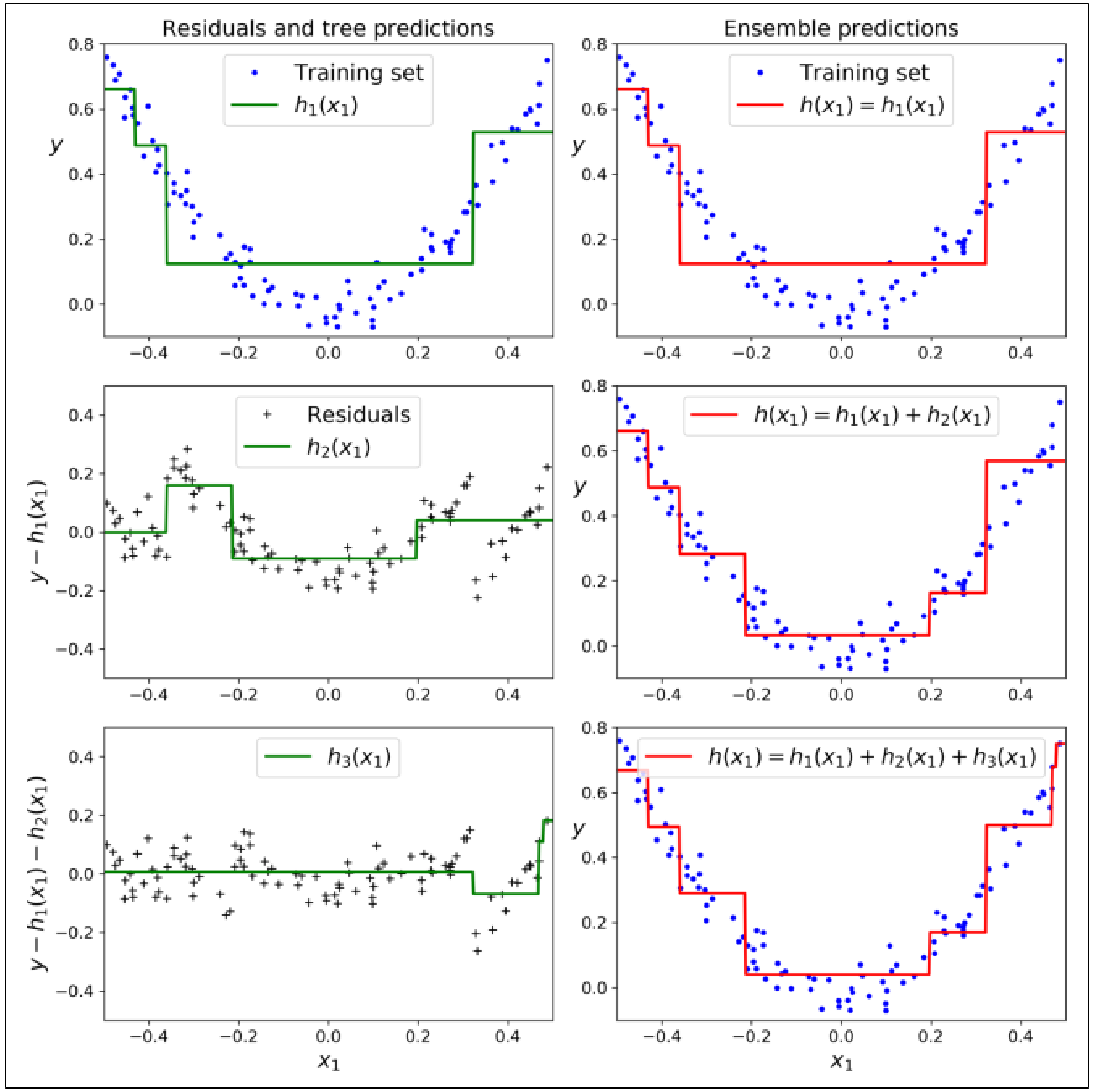
Maximum depth of the individual regression estimators. (limits the no of nodes in tree)

No of boosting stages to perform.

Gradient Boosting

1. Ensemble has one tree, its predictions are same as first tree's predictions.
2. A new tree is trained on residual errors of first tree. On right see that ensemble's predictions are equal to sum of predictions of first two trees.
3. Another tree is trained on residual errors of second tree.

- Ensemble's predictions gets better as trees are added to ensemble.



Support Vector Machine (SVM)

Support Vector Machine (SVM)

- SVM set of related supervised learning methods.

- Use: Classification and Regression
  - Uses ML to maximize predictive accuracy while avoiding over-fitting.

- Can be defined as systems which use hypothesis space of a linear functions in a high dimensional feature space.
  - trained with a learning algorithm from optimization theory
  - implements a learning bias derived from statistical learning theory*.

*provides a framework for studying problem of gaining knowledge, making predictions, making decisions from a set of data.

Classification and Regression

- Classification: Supervised learning-based approach
- Goal: to produce a model which predicts target value of data instances in testing set which are given attributes.

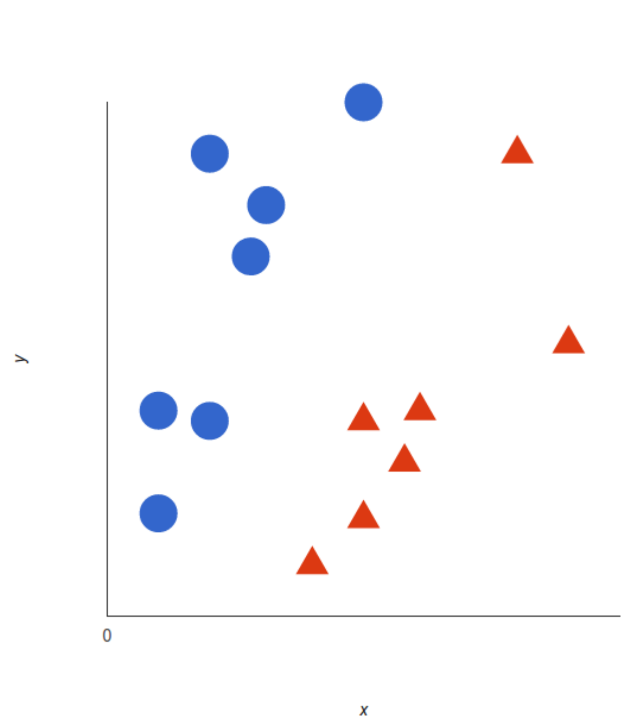
- Regression can be linear and non-linear.
    - need to add alternative loss function.
        - Example: Quadratic and Huber loss function (linear case).
        - loss function must include distance measure.

Classification and Regression

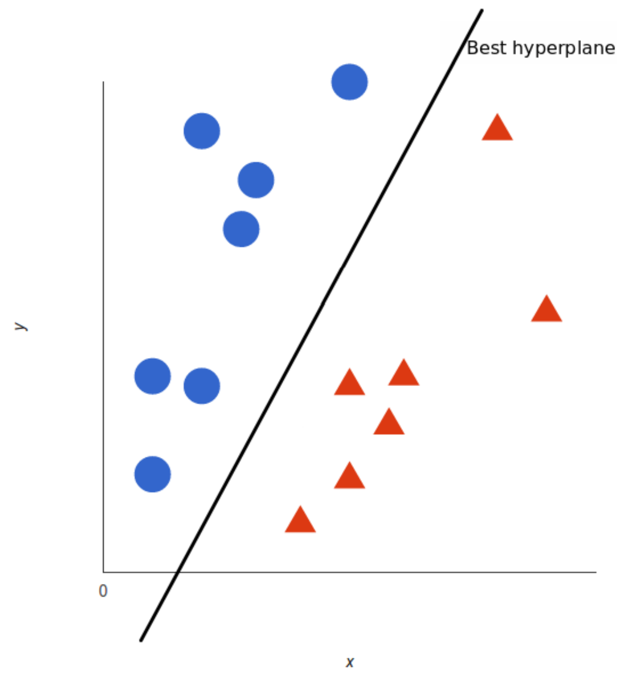
- In non-linear Support Vector Classifier approach, mapping can be used to map data into a high dimensional feature space.
  - Use linear regression.
  - Kernel approach for handling dimensionality problem.

- News: SVM receive pixel maps as input gives accuracy better than neural networks with elaborated features in a handwriting recognition task.

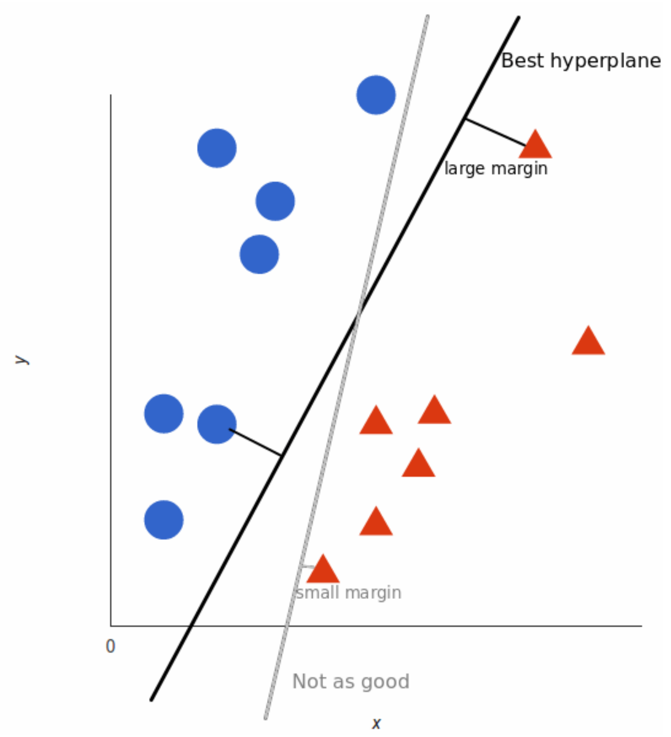
Support Vector Machine (SVM)



- Data: *red* and *blue*, data has two features: x and *y*.
- We want a classifier that, given a pair of *(x,y)* coordinates, outputs if it's either *red* or *blue*.



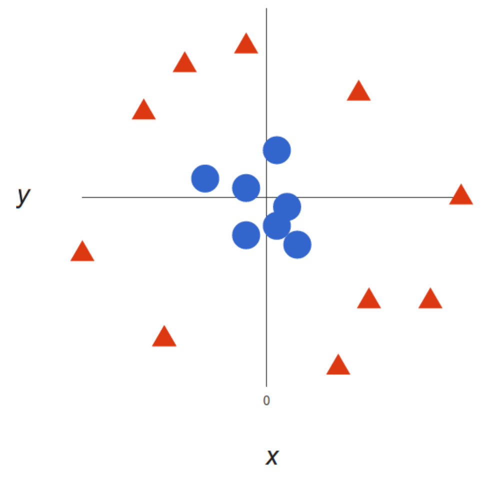
- SVM takes data points and outputs hyperplane (In 2D it's simply a line (decision boundary)) that separates tags.

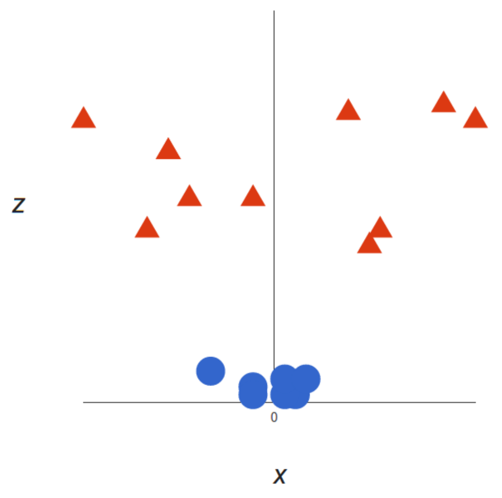


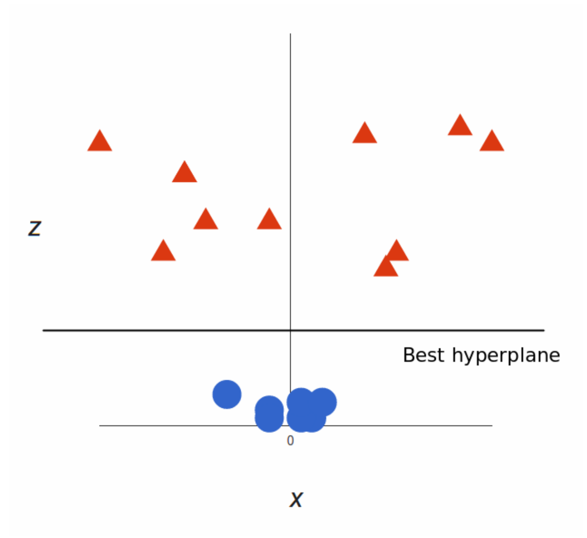
- SVM maximizes margins from both tags.
- Not all hyperplanes are created equal.

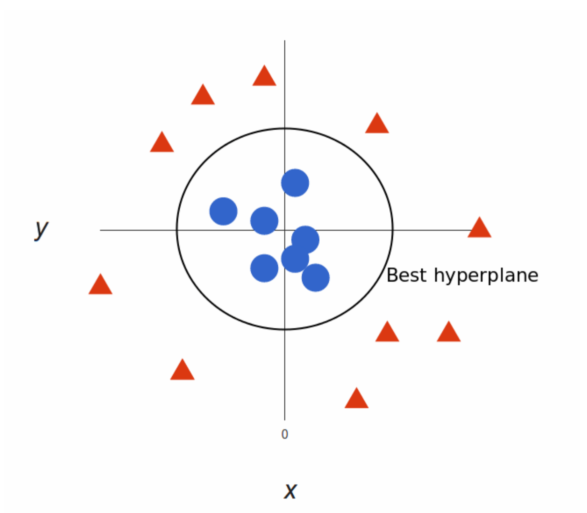
Img src: SVM By Bruno Stecanella

SVM: Nonlinear Data









- Add z dimension
- Rule: $z = x^2 + y^2$

- Data in two linearly separated groups

- In 3D, hyperplane is a plane parallel to x axis at a certain z (z = 1).

- Decision boundary is a circumference of radius 1 (separates both tags using SVM)

Img src: SVM By Bruno Stecanella

SVM: Parameter C

- C is trade-off between training error and flatness of solution.
    - Tells how much outliers are considered in calculation.
    - Aim: Keep training error small but need to generalize as well.

- Larger C means less training error but risks losing generalization.
- Smaller C means classifier is flat.

- Grid search can be used to estimate C.
- RBF-SVM: two parameters (C and gamma (the radius of radial basis function (RBF)))

SVM: Kernel

- SVM allows to classify linearly separable data (text classification).
- If not linearly separable, use kernel.
- Kernel is not part of SVM.
- SVM only finds decision boundary.

- SVM doesn't need vectors, dot products are enough.
  - Helps to avoid expensive calculations
- SVM do its job using dot product (kernel function)
  - kernel is linear ==> linear classifier.
  - nonlinear kernel ==> nonlinear classifier (without data transformation).

https://www.youtube.com/watch?v=_PwhiWxHK8o

Support Vector Machine (SVM)

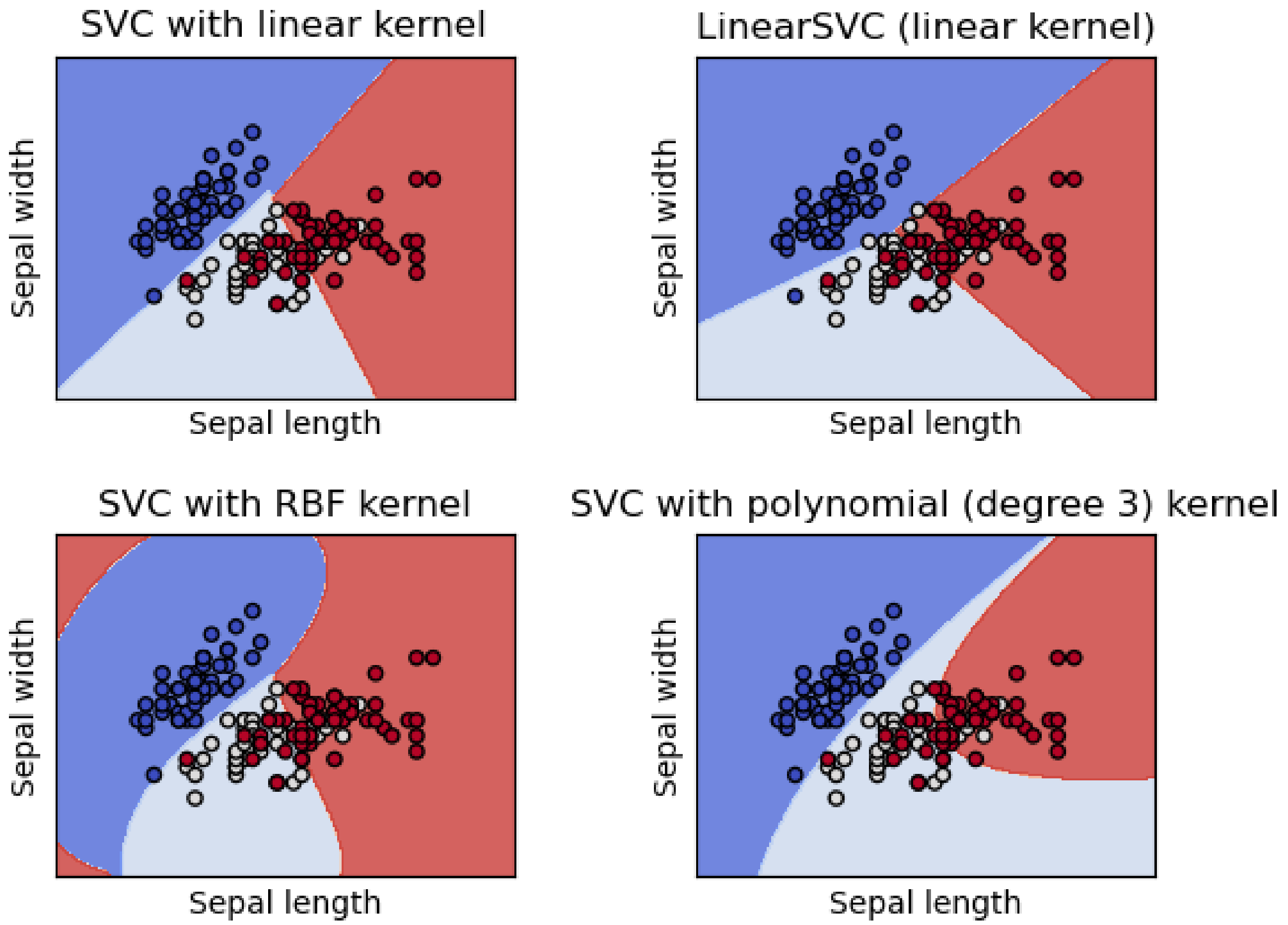
- Application: hand-writing analysis, face analysis (pattern classification and regression-based applications).

- Strengths: Easy training. No local optimal.
    - Scales well.
    - Trade-off between classifier complexity and error can be controlled explicitly.

- Weakness: Efficiency depends on choosing kernel function.

SVM and IRIS Data set

```
# import data
iris = datasets.load_iris()

# Take the first two features.
X = iris.data[:, :2]
y = iris.target

# SVM
C = 1.0  # SVM regularization parameter
models = (
    svm.SVC(kernel="linear", C=C),
    svm.LinearSVC(C=C, max_iter=10000),
    svm.SVC(kernel="rbf", gamma=0.7, C=C),
    svm.SVC(kernel="poly", degree=3,
gamma="auto", C=C),
)
```



Performance Metrics in Machine Learning

Classification Metrics

- Classification Metrics evaluate a model’s performance and tell how good or bad the classification is.
- Classification models have discrete output.
  - Accuracy, Recall and Precision
  - F1 Score
  - Confusion Matrix (not a metric)
  - AU-ROC

$\}$ [Lecture-02] from slide #35 to 37

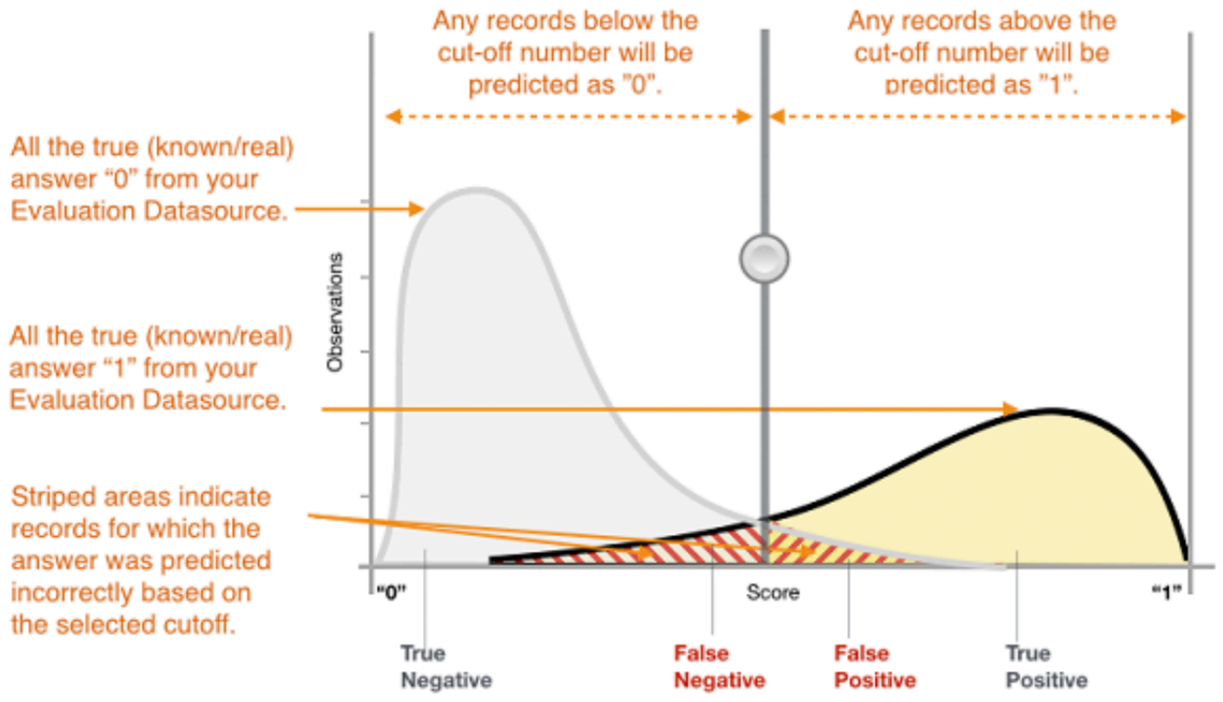
AUC - ROC Curve

- **AUC**, area under the receiver operating characteristic (**ROC**) curve.
- The Reciever operating characteristic curve plots the true positive (**TP**) rate versus the false positive (**FP**) rate at different classification thresholds.
- The thresholds are different probability cutoffs that separate the two classes in binary classification.
- It uses probability to tell us how well a model separates the classes.

- Measures the ability of a binary ML model to predict a higher score for positive examples as compared to negative examples.

Binary Classification

- Actual output of many binary classification algorithms is a prediction score which indicates system's certainty that given observation belongs to positive class.
- Binary classification accuracy metrics quantify the two types of correct predictions and two types of errors.
- Typical metrics are accuracy (ACC), precision, recall, false positive rate, F1-measure.


Any records below the cut-off number will be predicted as "0".
Any records above the cut-off number will be predicted as "1".
All the true (known/real) answer "0" from your Evaluation Datasource.
All the true (known/real) answer "1" from your Evaluation Datasource.
Striped areas indicate records for which the answer was predicted incorrectly based on the selected cutoff.
Observations
"0"
Score
"1"
True Negative
False Negative
False Positive
True Positive

AUC - ROC Curve

```python
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve

n = 10000
ratio = .95
n_0 = int((1-ratio) * n)
n_1 = int(ratio * n)

y = np.array([0] * n_0 + [1] * n_1)
# below are the probabilities obtained from a hypothetical model that always predicts the majority class
# probability of predicting class 1 is going to be 100%
y_proba = np.array([1]*n)
y_pred = y_proba > .5
```

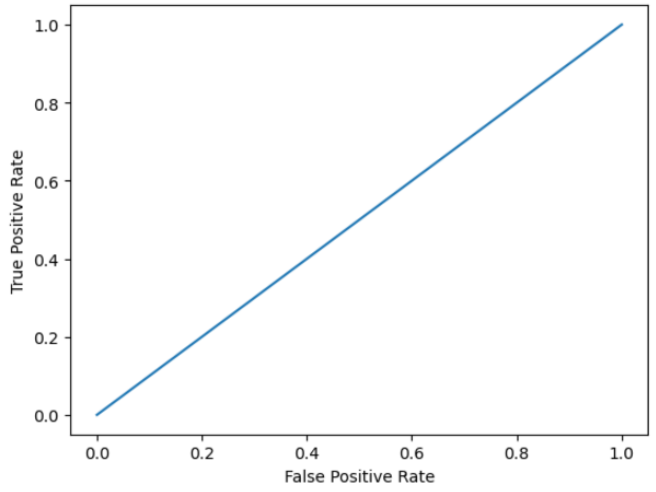
```python
# below are the probabilities obtained from a hypothetical model that doesn't always predict the mode
y_proba_2 = np.array(
    np.random.uniform(0, .7, n_0).tolist() +
    np.random.uniform(.3, 1, n_1).tolist()
)
y_pred_2 = y_proba_2 > .5
```

```python
import matplotlib.pyplot as plt

def plot_roc_curve(true_y, y_prob):
    """
    plots the roc curve based of the probabilities
    """

    fpr, tpr, thresholds = roc_curve(true_y, y_prob)
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
```

AUC - ROC Curve

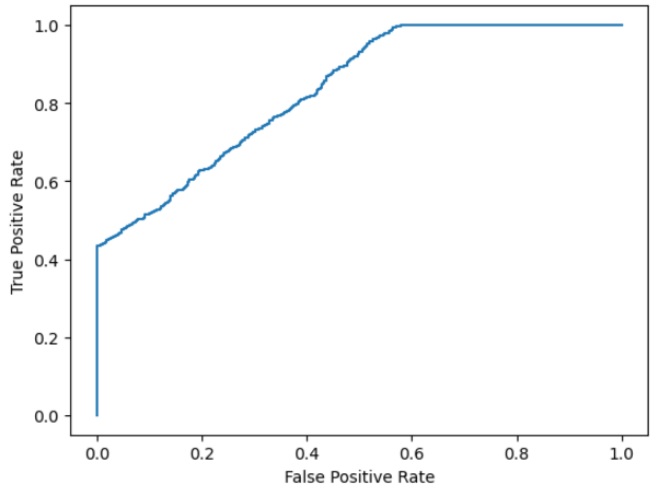
```
plot_roc_curve(y, y_proba)
print(f'model 1 AUC score: {roc_auc_score(y, y_proba)}')
```

```
model 1 AUC score: 0.5
```



```
plot_roc_curve(y, y_proba_2)
print(f'model 2 AUC score: {roc_auc_score(y, y_proba_2)}')
```

```
model 2 AUC score: 0.8334048421052631
```



Regression Metrics

- Regression models have continuous output.
- Calculate some sort of distance between predicted and ground truth.
- Regression models:
  - Mean Absolute Error (MAE),
  - Mean Squared Error (MSE),
  - Root Mean Squared Error (RMSE),
  - R^2 (R-Squared).

Regression Metrics

- **Mean Absolute Error (MAE)**: is the average of the difference between the ground truth and the predicted values.
    - Robust towards outliers
    - Doesn’t give an idea of the error direction (under/over prediction)
    - MAE is non-differentiable

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - \check{y}_j|$$

Where:
y_j: ground-truth value
y_hat: predicted value
N: number of datums

```
mae = np.abs(y-y_hat)
print(f"MAE: {mae.mean():0.2f} (+/- {mae.std():0.2f})")
```

- **Mean Squared Error (MSE)**: finds average of squared difference between target value and value predicted by regression model.
- Can be optimized better but more prone to outliers
- Penalizes small errors by squaring them, leads to an overestimation.

```
mse = (y-y_hat)**2
print(f"MSE: {mse.mean():0.2f} (+/- {mse.std():0.2f})")
```

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2$$

Regression Metrics

- Root Mean Squared Error (RMSE): sqrt(MSE): is square root of average of squared difference between target value and value predicted by regression model.
  - Penalizes small errors done by MSE by square rooting it.
  - Less prone outliers.

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2}$$

```
mse = (y-y_hat)**2
rmse = np.sqrt(mse.mean())
print(f"RMSE: {rmse:0.2f}")
```

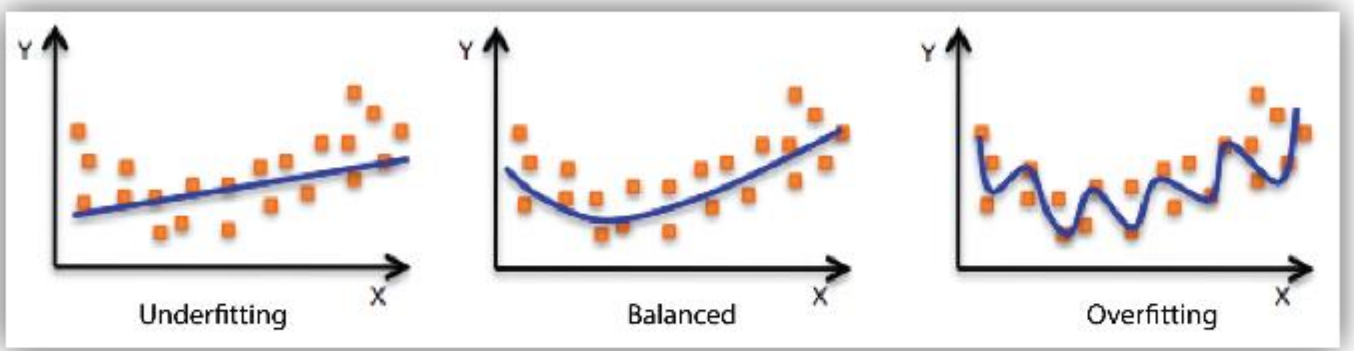
R^2

- R^2 (R-Squared): Range $(-\infty,1)$
- R^2 Coefficient of determination:
  - "How much the total variation in target is explained by the variation in regression line?
  - This is calculated using the sum of squared errors.
  - If the sum of Squared Error is small: Regression has captured 100% of the variance in the target variable.
  - If high then wasn't able to capture any variance.
    - If model overfits then variance explained will be 100% [Issue]
    - To solve it, we use Adjusted R^2; is lower than R^2
      - Only shows the real improvement

Good to know!!

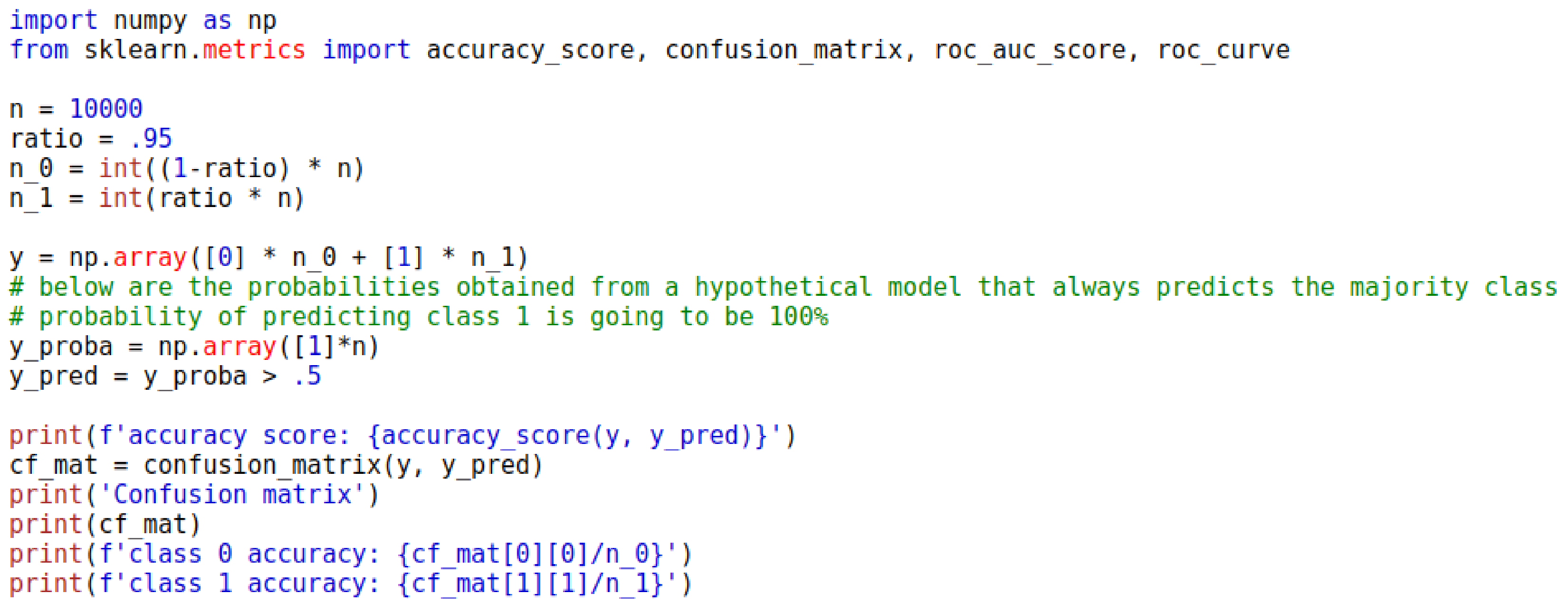
Model Fit

- Improving Model Accuracy:
  1. Collect more training data
  2. Feature processing: Add more variables and better feature processing
  3. Model parameter tuning:

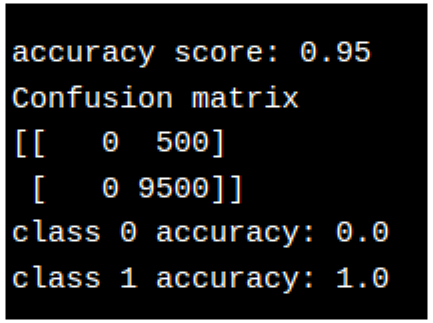


- Underfitting: Model performs poorly on training data because the model is unable to capture the relationship between the input examples (X) and target values (y).
- Overfitting: Model performs well on training data but does not perform well on test data. Model unable to generalize to unseen examples.

Case: High Accuracy in Imbalanced Data



```python
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve

n = 10000
ratio = .95
n_0 = int((1-ratio) * n)
n_1 = int(ratio * n)

y = np.array([0] * n_0 + [1] * n_1)
# below are the probabilities obtained from a hypothetical model that always predicts the majority class
# probability of predicting class 1 is going to be 100%
y_proba = np.array([1]*n)
y_pred = y_proba > .5

print(f'accuracy score: {accuracy_score(y, y_pred)}')
cf_mat = confusion_matrix(y, y_pred)
print('Confusion matrix')
print(cf_mat)
print(f'class 0 accuracy: {cf_mat[0][0]/n_0}')
print(f'class 1 accuracy: {cf_mat[1][1]/n_1}')
```



```
accuracy score: 0.95
Confusion matrix
[[   0  500]
 [   0 9500]]
class 0 accuracy: 0.0
class 1 accuracy: 1.0
```

Imbalance: Majority of values are one