GRADIENT BOOSTED TREES
xgboost library

Outline

- Boosting, Gradient boosting
- Feature Importance and Feature Selection
- Learning Curves
- Early stopping to avoid overfitting
- Tuning parameters

Boosting models

- An ensemble method where new models are added to correct the errors made by existing models
- Models are added sequentially until no further improvement can be made

Boosting models

- An ensemble method where new models are added to correct the errors made by existing models
- Models are added sequentially until no further improvement can be made
- Gradient boosting is an ensemble method where new models are created to predict the residuals (errors) of previous models and then added together to make final prediction

Build xgboost models

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

model = XGBClassifier()
model.fit(X_train,y_train)


pred = model.predict(X_test)
accuracy_score(y_test,pred)
```

Feature Importance

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

model = XGBClassifier()
model.fit(X_train,y_train)
model.feature_importances_

pred = model.predict(X_test)
accuracy_score(y_test,pred)
```

Feature Selection

Select subset of features by importance

SelectFromModel

- takes a first model1 to find features importance
- selects all features with importance greater than or equal to a threshold
- transform the dataset into a new one with the selected features

Feature Selection

Select subset of features by importance

SelectFromModel

- takes a first model1 to find features importance
- selects all features with importance greater than or equal to a threshold
- transform the dataset into a new one with the selected features

Once dataset has the selected features fit a model2 for prediction

Model1 and model2 may be different

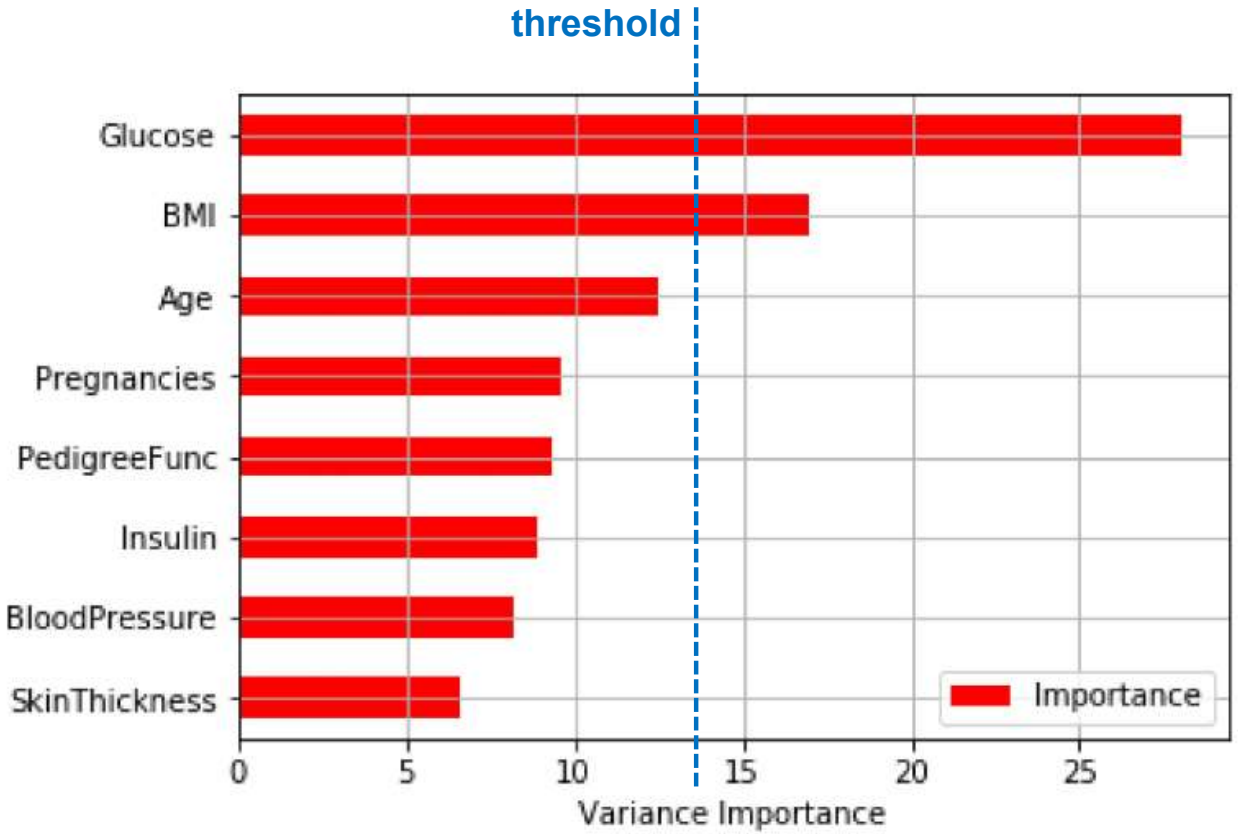
(i.e. model1 may be RF and model 2 may be LogisticRegression)

Feature Selection

```
from sklearn.feature_selection import SelectFromModel

# model1 to find features importance
model1 = XGBClassifier()
model1.fit(X_train,y_train)
# select features
selection = SelectFromModel(model1,threshold=14, prefit = True)
X_train_selected = selection.transform(X_train)
X_test_selected = selection.transform(X_test)
# fit model2 with selected features
model2 = XGBClassifier()
model2.fit(X_train_selected, y_train)
accuracy_score(y_test, pred = model2.predict(X_test_selected))
```

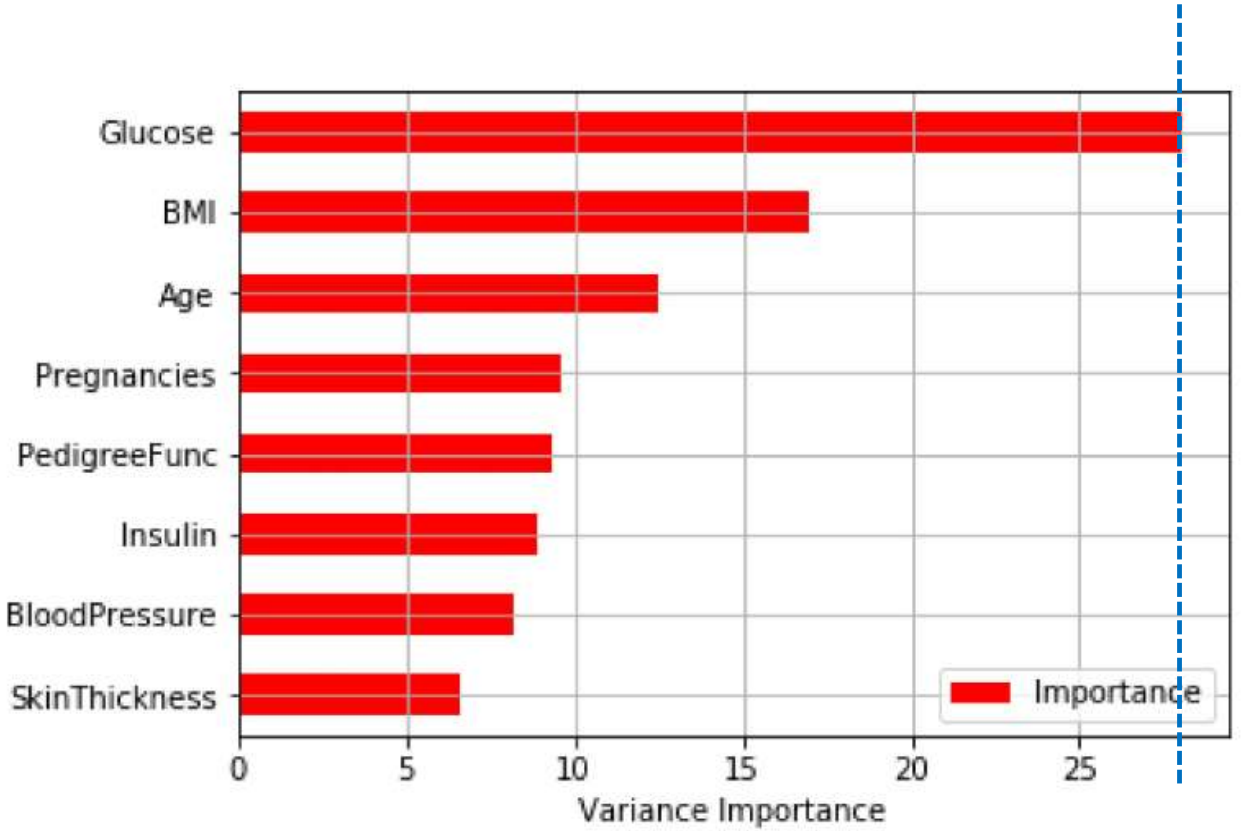
Feature Selection



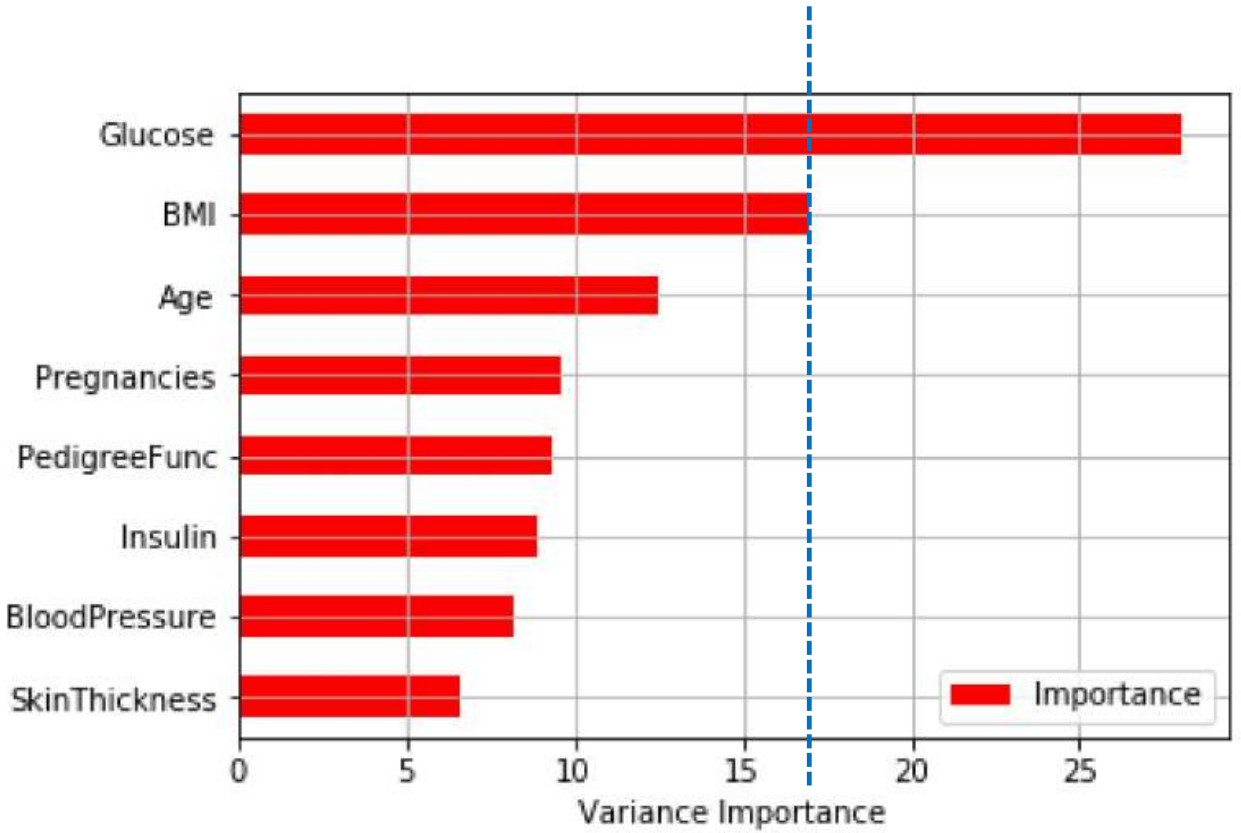
Feature Selection

- Make a loop to find the performance of each set of features
- Sets of features defined by thresholds
- Thresholds are equal to variable importance of each feature

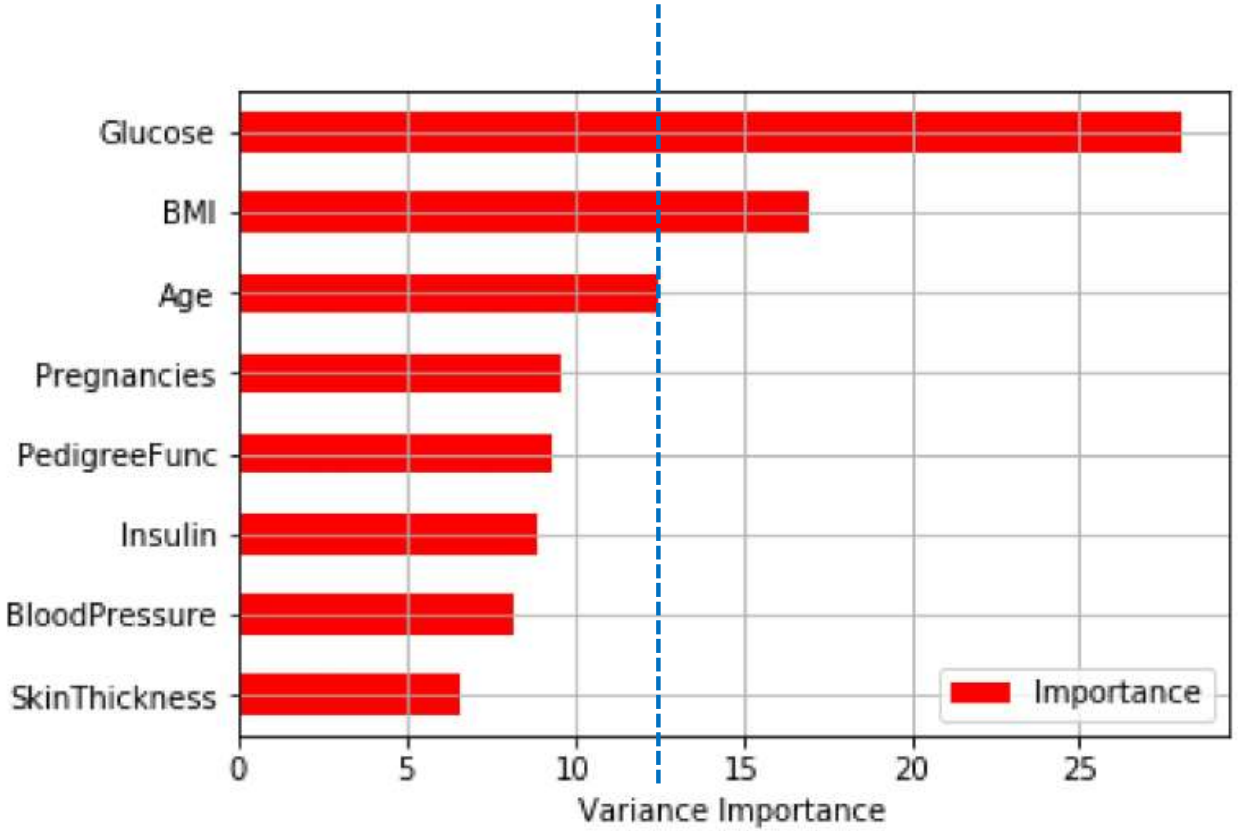
Feature Selection



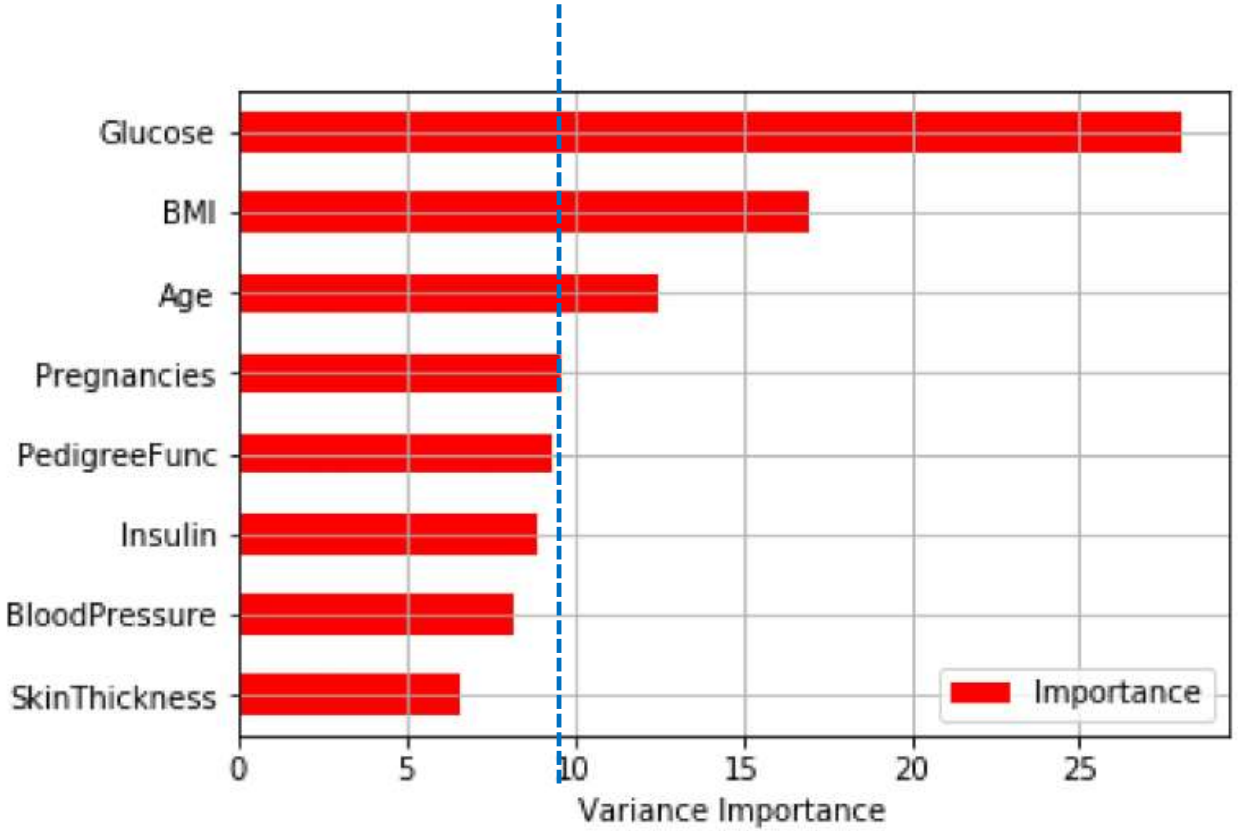
Feature Selection



Feature Selection



Feature Selection



Feature Selection

```
from sklearn.feature_selection import SelectFromModel

# model1 to find features importance
model1 = XGBClassifier()
model1.fit(X_train,y_train)
# loop over selected features
thresholds = sort(model1.feature_importances_)
for thresh in thresholds:
        selection = SelectFromModel(model1, threshold=thresh, prefit=True)
        X_train_selected = selection.transform(X_train)
        X_test_selected  = selection.transform(X_test)
        # model2 with selected features
        model2 = XGBClassifier()
        model2.fit(X_train_selected, y_train)
        accuracy_score(y_test, pred = model2.predict(X_test_selected))
```

Performance step-by-step

```
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y,
                                        test_size=0.33, random_state=1)

# fit model requesting step-by-step test error rates
eval_set = [(X_test, y_test)]
model = XGBClassifier()
model.fit(X_train, y_train, eval_metric="error",
                                      eval_set=eval_set, verbose=True)
```

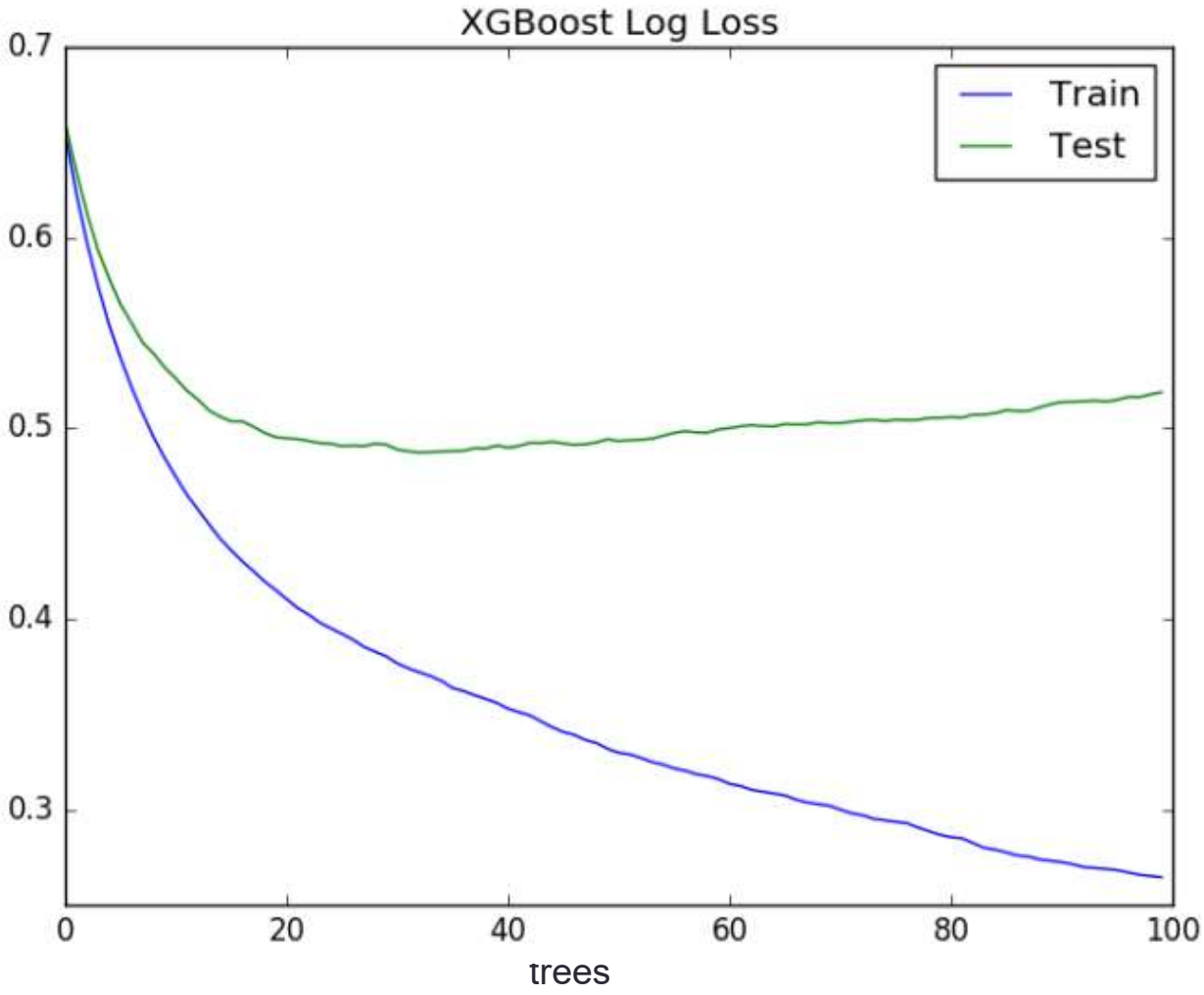
Learning Curves

```
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y,
                                    test_size=0.33, random_state=1)

# fit model requesting train and test error rates
eval_set = [(X_train,y_train),(X_test, y_test)]
model = XGBClassifier()
model.fit(X_train, y_train, eval_metric="error",
                                    eval_set=eval_set, verbose=True)

# store train and test error rates (then plot them)
model.evals_result()
```

Learning Curves



Early Stopping

define a number of trees

over which no improvement is observed

```
model = XGBClassifier()
eval_set = [(X_test, y_test)]
model.fit(X_train, y_train, early_stopping_rounds = 10,
        eval_metric="error",eval_set=eval_set, verbose=True)
```

Tuning parameter –Kfold cross validation

```
from sklearn.model_selection import  GridSearchCV, StratifiedKFold
n_trees = range(20,80,10)
model = XGBClassifier()
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
param_grid = dict(n_estimators=n_trees)
grid_search = GridSearchCV(model, param_grid,scoring="neg_log_loss",
                                                              cv=kfold)
grid_result = grid_search.fit(X,Y)
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
n_trees = list(n_estimators)
```

Tuning parameter –Kfold cross validation

.

	params	mean_test_score	std_test_score
0	20	-0.493858	0.039632
1	30	-0.483538	0.046256
2	40	-0.480428	0.049159
3	50	-0.480157	0.051132
4	60	-0.482531	0.053802
5	70	-0.484747	0.054087

Tuning parameters –Kfold cross validation

```
n_trees = range(30,90,10)
depth_values = [1,2,3]
model = XGBClassifier()
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
param_grid = dict(max_depth=depth_values, n_estimators=n_trees)
grid_search = GridSearchCV(model, param_grid,scoring="neg_log_loss",
                                                        cv=kfold)
grid_result = grid_search.fit(X,Y)
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
```

Tuning parameters –Kfold cross validation