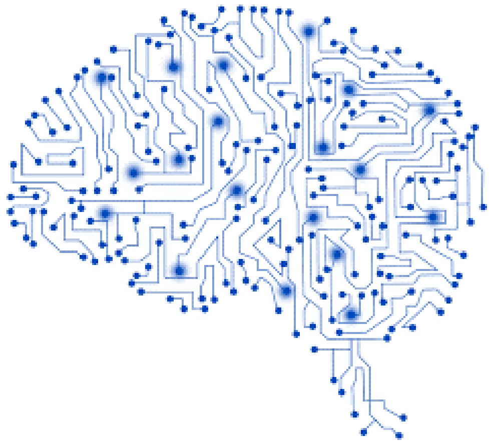
University of Minho
School of Engineering

Intelligent Systems Associate Laboratory



Dados e Aprendizagem Automática

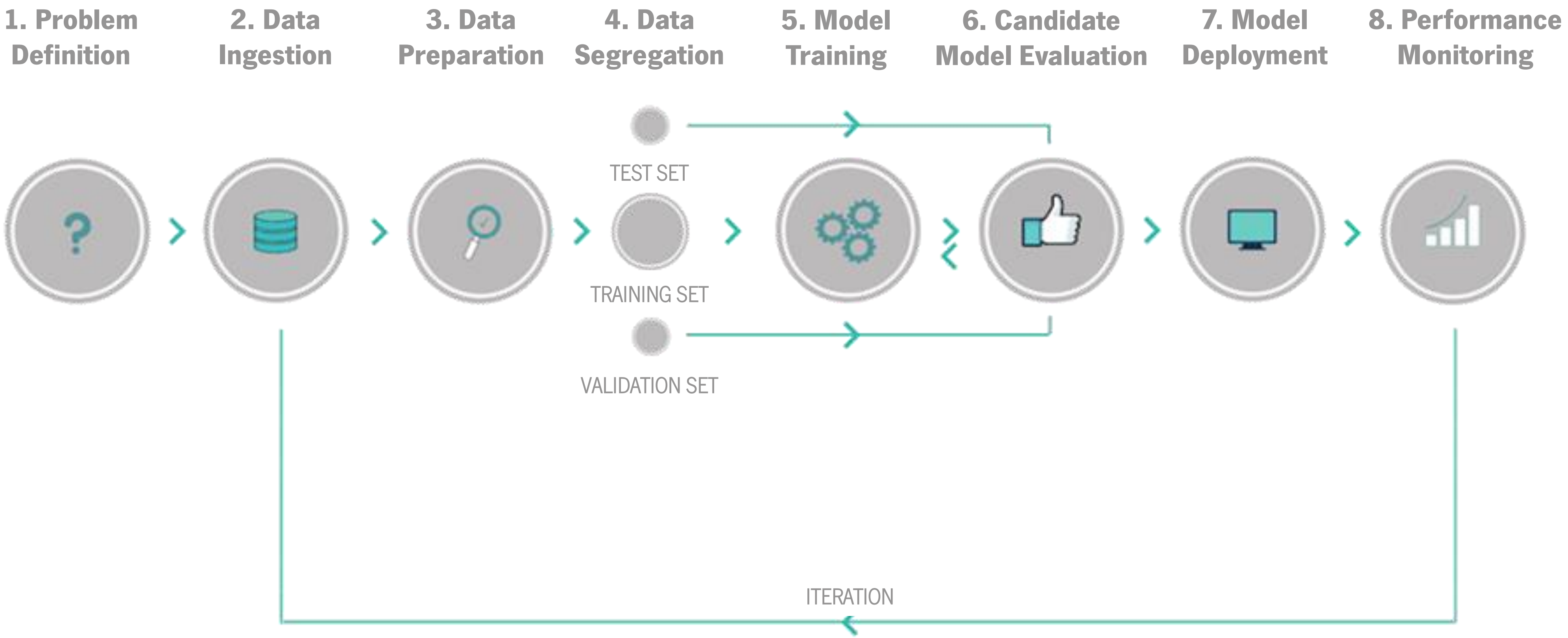
Ensemble Learning

DAA @ MEI-1º/MiEI-4º – 1º Semestre

Bruno Fernandes, Dalila Alves, Filipa Ferraz, Victor Alves

Part VI

Contents


1. Problem Definition
2. Data Ingestion
3. Data Preparation
4. Data Segregation
5. Model Training
6. Candidate Model Evaluation
7. Model Deployment
8. Performance Monitoring
TEST SET
TRAINING SET
VALIDATION SET
ITERATION

Ensemble Learning

The Problem and the Data

Problem: development of a Machine Learning model capable of **predicting the passenger survived** to the Titanic disaster

Dataset: table with information regarding the **passengers** with 891 entries and 12 features, including:

- *PassengerId*
- *Survived*
- *Pclass*
- *Name*
- *Sex*
- *Age*
- *SibSp*
- *Parch*
- *Ticket*
- *Fare*
- *Cabin*
- *Embarked*

You may need to install `xgboost`. Use one of the following commands:

```
conda install -c conda-forge xgboost
```

```
pip install xgboost
```

EDA


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

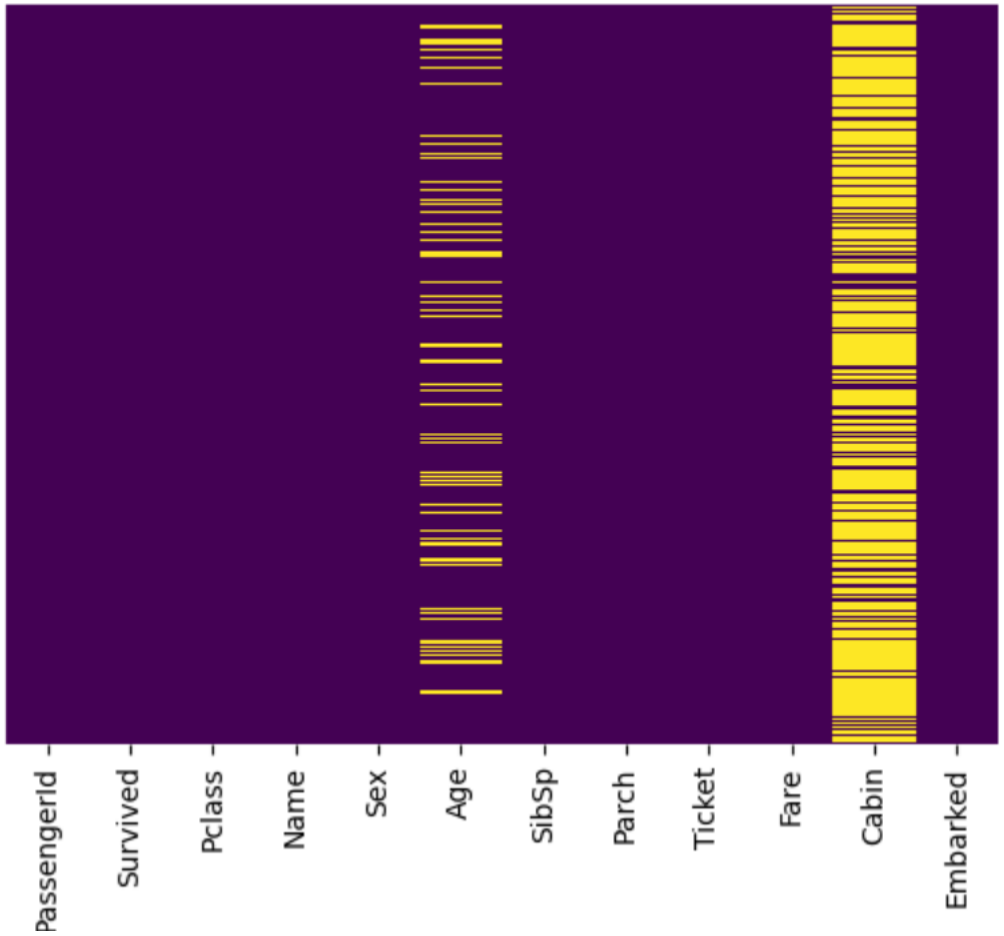
```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S

EDA

Let's check the missing values:

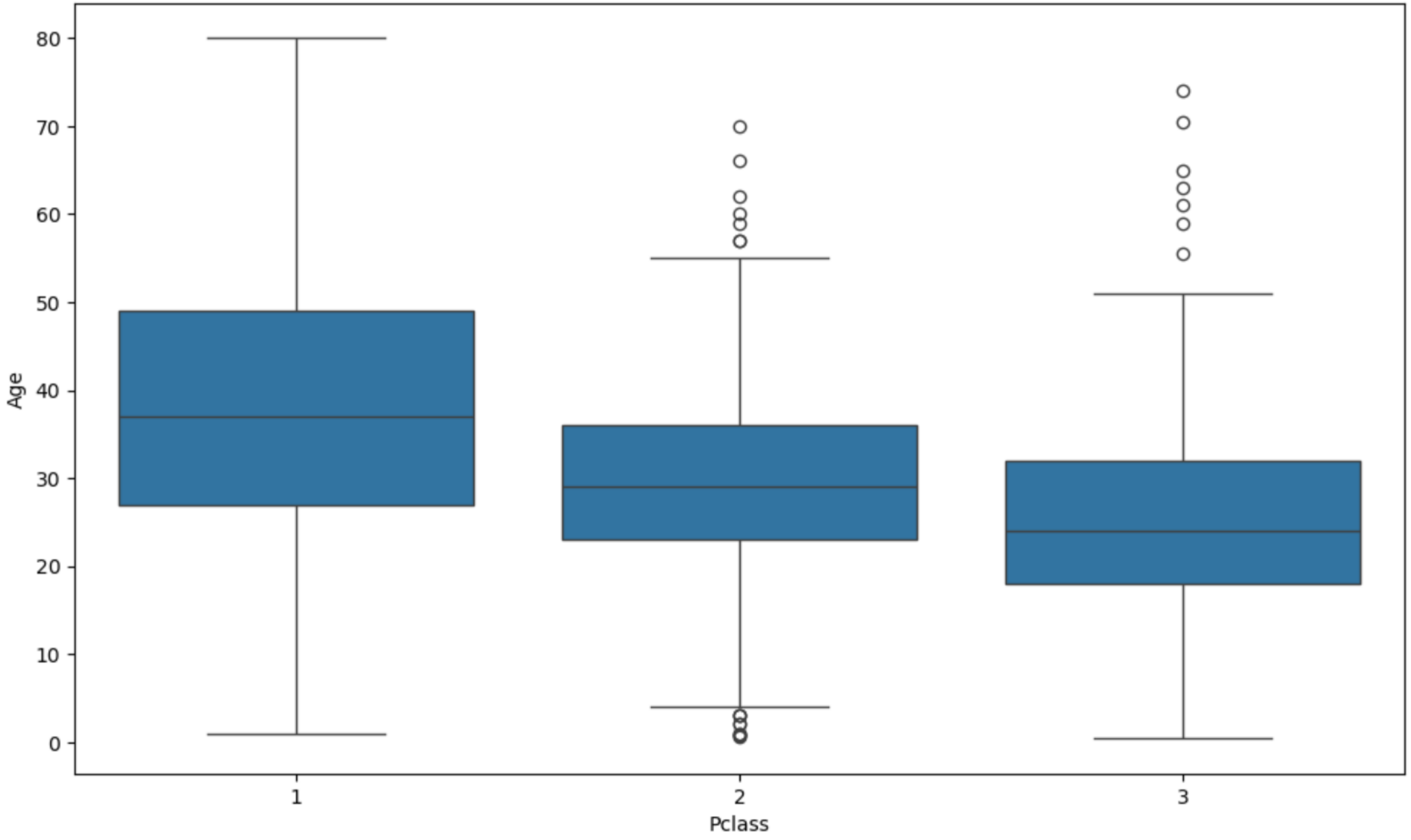
```
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```



EDA

Let's analyze the *Age* distribution with the ticket class, *Pclass*:

```
plt.figure(figsize = (12, 7))
sns.boxplot(x = 'Pclass', y = 'Age', data = df)
```



Data Preprocessing

Let's impute the missing values in *Age* with the ticket class, *Pclass*, like we did on a previous class:

```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age
```

```
df['Age'] = df[['Age','Pclass']].apply(impute_age, axis = 1)
```

Data Preprocessing

We will drop features *Cabin*, *Sex*, *Embarked*, *Name* and *Ticket*:

```
df.drop('Cabin', axis = 1, inplace = True)
df.dropna(inplace = True)
```

```
df.drop(['Sex', 'Embarked', 'Name', 'Ticket'], axis = 1,inplace = True)
```

```
df.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
0	1	0	3	22.0	1	0	7.2500
1	2	1	1	38.0	1	0	71.2833
2	3	1	3	26.0	0	0	7.9250
3	4	1	1	35.0	1	0	53.1000
4	5	0	3	35.0	0	0	8.0500

Let's define the X and y and split the data:

```
X = df.drop('Survived', axis = 1)
y = df['Survived']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 2022)
```

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
((622, 6), (622,), (267, 6), (267,))
```

Comparison Models: Decision Tree

Let's try different models. We will implement a Decision Tree and a Support Vector Machine for comparison with the Ensemble Learning Models.

Decision Tree

```
dt_model = DecisionTreeClassifier(max_depth = 2, random_state = 2022)
dt_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=2, random_state=2022)
```

```
dt_score = dt_model.score(X_test, y_test)
print("Accuracy: %.2f%%" % (dt_score * 100))
```

```
Accuracy: 69.29%
```

```
dt_predictions = dt_model.predict(X_test)
print(classification_report(y_test, dt_predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.76      0.76       173
           1       0.56      0.56      0.56        94

    accuracy                           0.69       267
   macro avg       0.66      0.66      0.66       267
weighted avg       0.69      0.69      0.69       267
```

Saving the model's accuracy for latter comparison:

```
results = {'DT': dt_score}
```

Comparison Models: Support Vector Machine

Support Vector Machine

```
svm_model = SVC(random_state = 2022)
svm_model.fit(X_train, y_train)
```

```
SVC
SVC(random_state=2022)
```

```
svm_score = svm_model.score(X_test, y_test)
```

```
print("Accuracy: %.2f%%" % (svm_score * 100))
```

```
Accuracy: 68.16%
```

```
svm_predictions = svm_model.predict(X_test)
```

```
print(classification_report(y_test, svm_predictions))
```

	precision	recall	f1-score	support
0	0.68	0.97	0.80	173
1	0.71	0.16	0.26	94
accuracy			0.68	267
macro avg	0.70	0.56	0.53	267
weighted avg	0.69	0.68	0.61	267

```
results['SVM'] = svm_score
```

```
sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale',
        coef0=0.0, shrinking=True, probability=False, tol=0.001,
        cache_size=200, class_weight=None, verbose=False,
        max_iter=-1, decision_function_shape='ovr',
        break_ties=False, random_state=None)
```

Ensemble Learning Models: BAGGing *(Bootstrap AGGregating)*

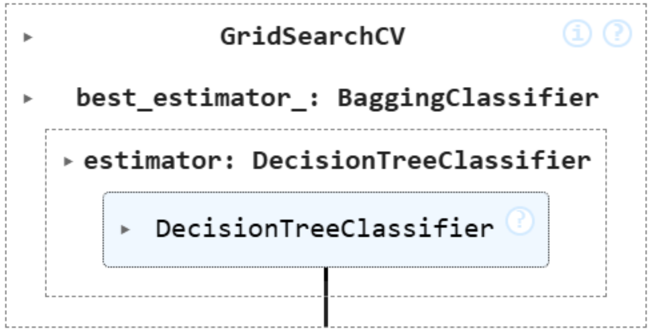
We will use *StratifiedShuffleSplit* which is another split technique:

```
sss = StratifiedShuffleSplit(n_splits = 10, test_size = 20, random_state = 2022)
```

```
sklearn.model_selection.StratifiedShuffleSplit(n_splits=10, *,
                test_size=None, train_size=None, random_state=None)
```

Let's implement the *Bagging Classifier* from Sklearn:

```
bg_model = BaggingClassifier(estimator = dt_model, bootstrap = True)
n_estimators = [10, 40, 60, 80, 100, 160]
parameters = {'n_estimators': n_estimators}
grid_bg = GridSearchCV(estimator = bg_model, param_grid = parameters, cv = sss)
grid_bg.fit(X_train, y_train)
```




```
sklearn.ensemble.BaggingClassifier(estimator=None,
                n_estimators=10, *, max_samples=1.0,
                max_features=1.0, bootstrap=True,
                bootstrap_features=False, oob_score=False,
                warm_start=False, n_jobs=None, random_state=None,
                verbose=0)
```

BAGGing involves adjusting decision trees to different samples from the same dataset, evaluating by calculating the average of these predictions.

Ensemble Learning Models: Bagging

Best estimator

```
bst_bg_model = grid_bg.best_estimator_
print(bst_bg_model)
```

```
BaggingClassifier(estimator=DecisionTreeClassifier(max_depth=2,
                                                   random_state=2022),
                  n_estimators=100)
```

```
bst_bg_model.fit(X_train, y_train)
```

```
▸ BaggingClassifier
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

```
bst_bg_score = bst_bg_model.score(X_test, y_test)
print("Accuracy: %.2f%%" % (bst_bg_score*100))
```

```
Accuracy: 70.41%
```

```
bg_predictions = bst_bg_model.predict(X_test)
print(classification_report(y_test, bg_predictions))
```

	precision	recall	f1-score	support
0	0.77	0.78	0.77	173
1	0.58	0.56	0.57	94
accuracy			0.70	267
macro avg	0.67	0.67	0.67	267
weighted avg	0.70	0.70	0.70	267

```
results['Bagg']= bst_bg_score
```

Ensemble Learning Models: Bagging

Now, bagging with **Random Forest**:

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, *,
          criterion='gini', max_depth=None, min_samples_split=2,
          min_samples_leaf=1, min_weight_fraction_leaf=0.0,
          max_features='sqrt', max_leaf_nodes=None,
          min_impurity_decrease=0.0, bootstrap=True, oob_score=False,
          n_jobs=None, random_state=None, verbose=0, warm_start=False,
          class_weight=None, ccp_alpha=0.0, max_samples=None,
          monotonic_cst=None)
```

```
rf_model = RandomForestClassifier(bootstrap = False, max_depth = 2, verbose = 1)
rf_model.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
RandomForestClassifier
RandomForestClassifier(bootstrap=False, max_depth=2, verbose=1)
```

```
rf_score = rf_model.score(X_test, y_test)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
print("Accuracy: %.2f%%" % (rf_score * 100))
```

```
Accuracy: 73.41%
```

Ensemble Learning Models: Bagging


```
rf_predictions = rf_model.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
rf_predictions = rf_model.predict(X_test)
```

```
print(classification_report(y_test, rf_predictions))
```

```
              precision    recall  f1-score   support

           0       0.77      0.83      0.80       173
           1       0.64      0.55      0.59        94

    accuracy                           0.73       267
   macro avg       0.71      0.69      0.70       267
weighted avg       0.73      0.73      0.73       267
```

```
results['RF'] = rf_score
```

Ensemble Learning Models: Boosting

Gradient Boosting

```
gbc_model = GradientBoostingClassifier(n_estimators = 100, learning_rate = 1.0,
                                       max_depth = 1, random_state = 2022)
gbc_model.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=2022)
```

```
gbc_score = gbc_model.score(X_test, y_test)
print("Accuracy: %.2f%%" % (gbc_score*100))
```

```
Accuracy: 71.16%
```

```
gbc_predictions = gbc_model.predict(X_test)
print(classification_report(y_test, gbc_predictions))
```

	precision	recall	f1-score	support
0	0.76	0.81	0.78	173
1	0.60	0.53	0.56	94
accuracy			0.71	267
macro avg	0.68	0.67	0.67	267
weighted avg	0.71	0.71	0.71	267

```
results['GB'] = gbc_score
```

```
sklearn.ensemble.GradientBoostingClassifier(*, loss='log_loss',
        learning_rate=0.1, n_estimators=100, subsample=1.0,
        criterion='friedman_mse', min_samples_split=2,
        min_samples_leaf=1, min_weight_fraction_leaf=0.0,
        max_depth=3, min_impurity_decrease=0.0, init=None,
        random_state=None, max_features=None, verbose=0,
        max_leaf_nodes=None, warm_start=False,
        validation_fraction=0.1, n_iter_no_change=None,
        tol=0.0001, ccp_alpha=0.0)
```

Boosting involves the sequential addition of ensemble members that correct the predictions made by previous models and produce a weighted average of the predictions.

Ensemble Learning Models: Boosting

XGBoost

```
xgb_model = XGBClassifier(max_depth = 1, objective = 'reg:squarederror')
xgb_model.fit(X_train, y_train)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=1, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
```

```
xgb_score = xgb_model.score(X_test, y_test)
print("Accuracy: %.2f%%" % (xgb_score * 100))
```

```
Accuracy: 73.03%
```

```
xgboost.sklearn.XGBClassifier(base_score=None, booster=None,
                callbacks=None, colsample_bylevel=None,
                colsample_bynode=None, colsample_bytree=None,
                device=None, early_stopping_rounds=None,
                enable_categorical=False, eval_metric=None,
                feature_types=None, gamma=None,
                grow_policy=None, importance_type=None,
                interaction_constraints=None,
                learning_rate=None, max_bin=None,
                max_cat_threshold=None, max_cat_to_onehot=None,
                max_delta_step=None, max_depth=None,
                max_leaves=None, min_child_weight=None,
                missing=nan, monotone_constraints=None,
                multi_strategy=None, n_estimators=None,
                n_jobs=None, num_parallel_tree=None,
                random_state=None, ...)
```

Ensemble Learning Models: Boosting


```
xgb_predictions = xgb_model.predict(X_test)
print(classification_report(y_test, xgb_predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.85      0.80       173
           1       0.65      0.51      0.57        94

    accuracy                           0.73       267
   macro avg       0.71      0.68      0.69       267
weighted avg       0.72      0.73      0.72       267
```

```
results['XGB'] = xgb_score
```

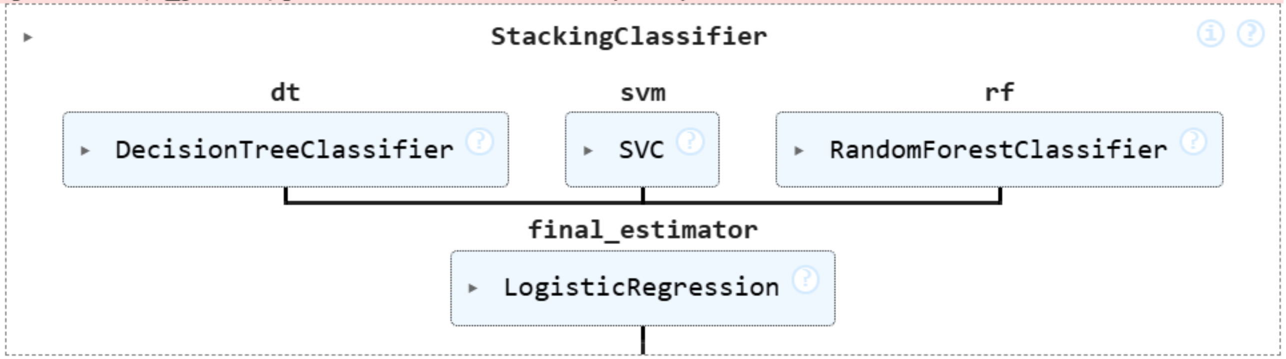
Ensemble Learning Models: Stacking

Stacking Classifier

```
estimators = [("dt", dt_model), ("svm", svm_model), ("rf", rf_model)]
st_model = StackingClassifier(estimators = estimators,
                              final_estimator = LogisticRegression())
```

```
st_model.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```




```
sklearn.ensemble.StackingClassifier(estimators,
                final_estimator=None, *, cv=None,
                stack_method='auto', n_jobs=None,
                passthrough=False, verbose=0)
```

Stacking involves fitting different types of models to the same dataset, using another model to learn the best way to combine the predictions.

Ensemble Learning Models: Stacking


```
st_score = st_model.score(X_test, y_test)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
print("Accuracy: %.2f%%" % (st_score*100))
```

```
Accuracy: 72.28%
```

```
st_predictions = st_model.predict(X_test)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
print(classification_report(y_test, st_predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.84      0.80       173
           1       0.63      0.51      0.56        94

    accuracy                           0.72       267
   macro avg       0.70      0.67      0.68       267
weighted avg       0.71      0.72      0.72       267
```

```
results['Stack']= st_score
```

Ensemble Learning Models: Max Voting

Majority Class Labels (Majority/Hard Voting)

```
hvt_model = VotingClassifier(estimators = [("dt", dt_model), ("svm", svm_model), ("rf", rf_model)],
                             voting = 'hard', weights = [2, 1, 2])
```

```
hvt_model.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
VotingClassifier
dt                          svm        rf
DecisionTreeClassifier      SVC        RandomForestClassifier
```

```
for model, label in zip([dt_model, svm_model, rf_model, hvt_model], ['dt', 'svm', 'rf', 'Ensemble']):
    hvt_score = cross_val_score(model, X_test, y_test, scoring = 'accuracy', cv = 5)
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (hvt_score.mean(), hvt_score.std(), label))
```

```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
Accuracy: 0.69 (+/- 0.05) [dt]
Accuracy: 0.66 (+/- 0.04) [svm]
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

```
sklearn.ensemble.VotingClassifier(estimators,
        *, voting='hard', weights=None,
        n_jobs=None,
        flatten_transform=True,
        verbose=False)
```

Max voting involves fitting several models, each making a prediction and voting on each sample. Only the class with the highest votes is included in the final prediction class.

Ensemble Learning Models: Max Voting


```
hvt_score = hvt_model.score(X_test, y_test)
print("Accuracy: %.2f%%" % (hvt_score*100))
```

```
Accuracy: 70.79%
```

```
hvt_predictions = hvt_model.predict(X_test)
print(classification_report(y_test, hvt_predictions))
```

```
              precision    recall  f1-score   support

           0       0.77      0.79      0.78       173
           1       0.59      0.56      0.58        94

    accuracy                           0.71       267
   macro avg       0.68      0.67      0.68       267
weighted avg       0.71      0.71      0.71       267
```

```
results['HVotW']= hvt_score
```

Models Accuracy Comparison

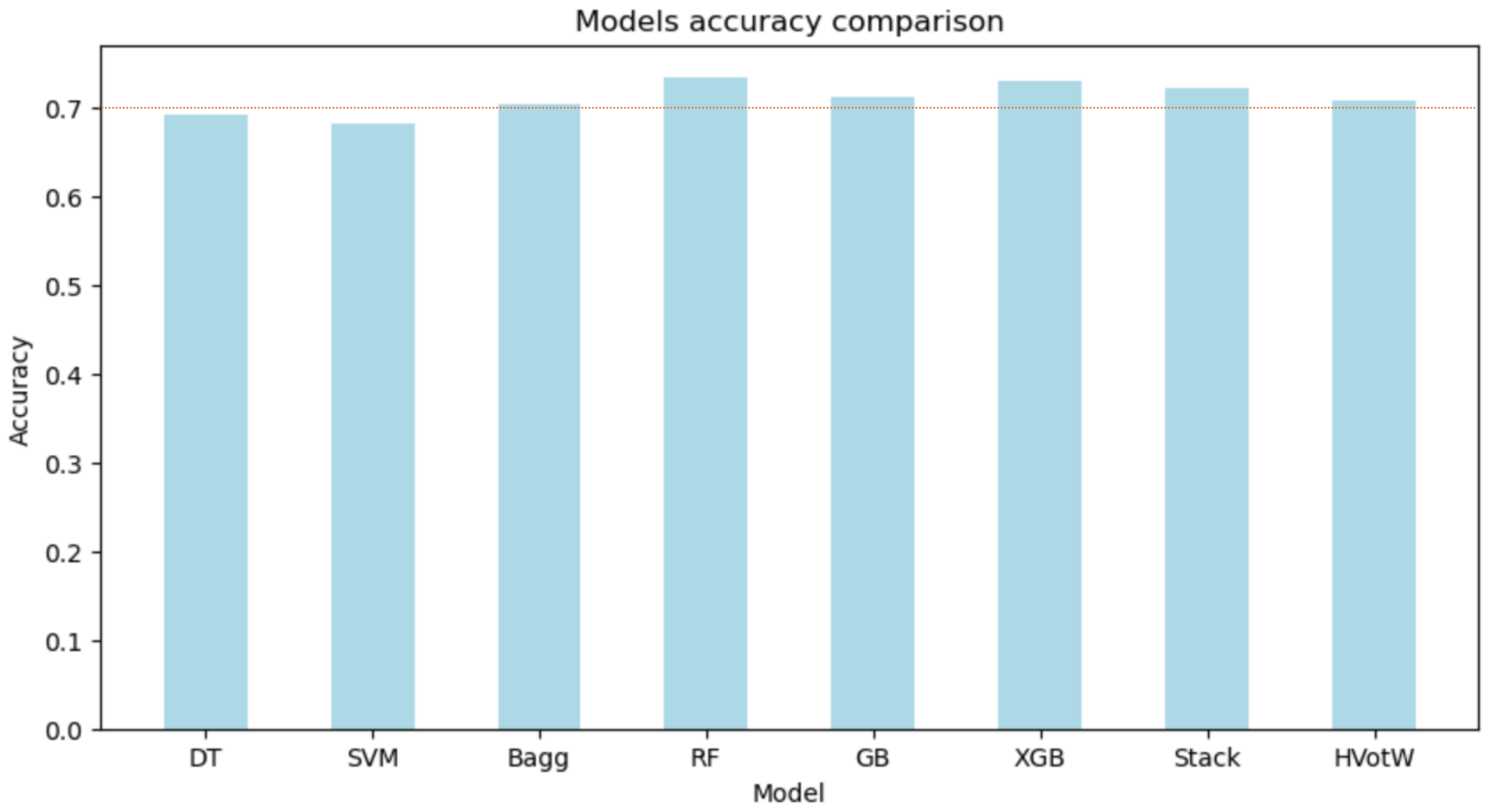
Let's create a bar chart with the accuracy of each model created:

```
fig = plt.figure(figsize = (10, 5))

mod = list(results.keys())
acc =list(results.values())

plt.bar(mod, acc, color ='lightblue',
        width = 0.5)

plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.title("Models accuracy comparison")
plt.show()
```



Models Accuracy Comparison

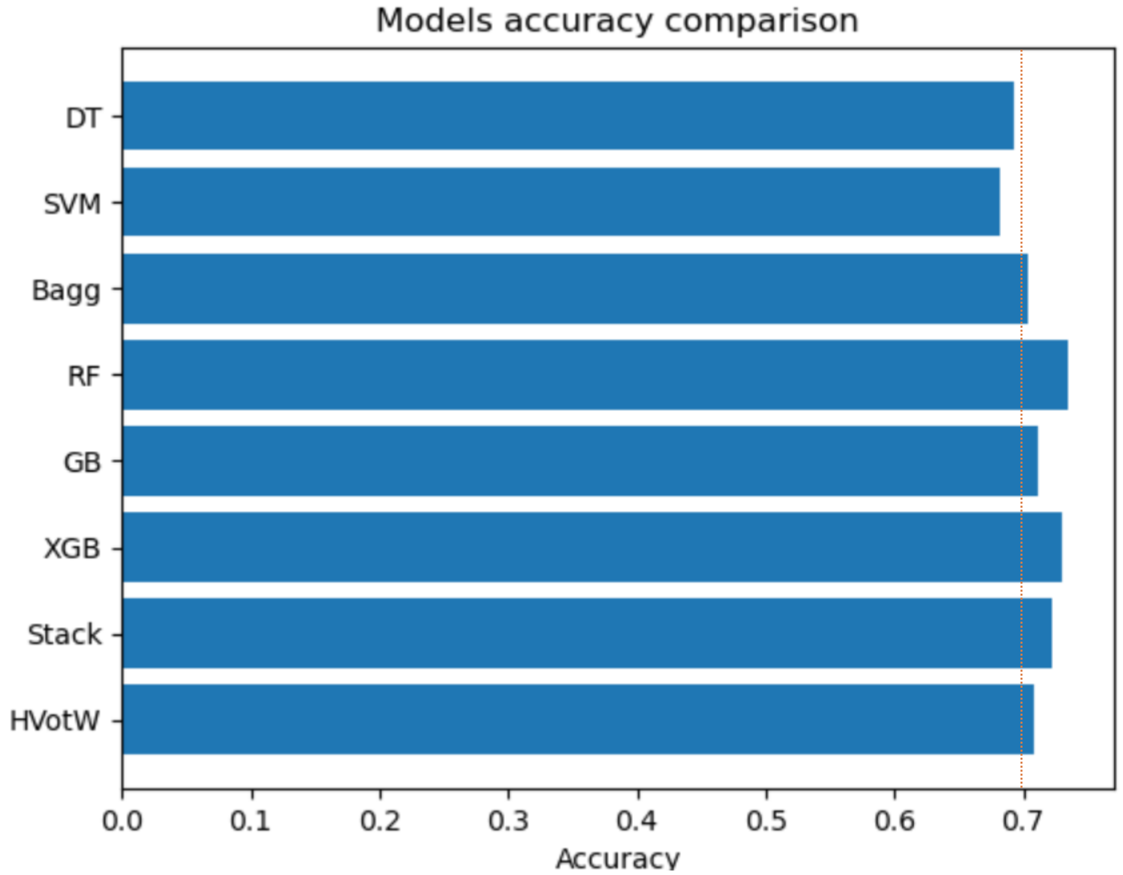
We can visualize it with horizontal bars:

```
fig, ax = plt.subplots()

y_values = np.arange(len(mod))

ax.barh(y_values, acc, align='center')
ax.set_yticks(y_values, labels = mod)
ax.invert_yaxis()
ax.set_xlabel('Accuracy')
ax.set_title('Models accuracy comparison')

plt.show()
```



Models Accuracy Comparison

Or we can print the results:

```
print("Models accuracy comparison")
for key, value in results.items():
    print("%s \t %.2f" % (key, value))
```

```
Models accuracy comparison
DT       0.69
SVM      0.68
Bagg     0.70
RF       0.73
GB       0.71
XGB      0.73
Stack    0.72
HVotW    0.71
```

Which model performed better?
Which models are worth tuning?

Models Accuracy Comparison

Or we can print the results:

```
print("Models accuracy comparison")
for key, value in results.items():
    print("%s \t %.2f" % (key, value))
```

```
Models accuracy comparison
DT      0.69
SVM     0.68
Bagg    0.70
RF      0.73
GB      0.71
XGB     0.73
Stack   0.72
HVotW   0.71
```

Which model perfomed better?
Which models are worth tunning?

Hands On