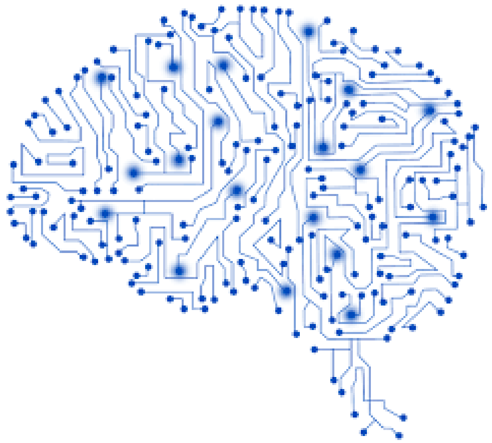
University of Minho
School of Engineering



Dados e Aprendizagem Automática

Ensemble Learning

DAA @ MEI-1º/MiEI-4º – 1º Semestre

Bruno Fernandes, César Analide, Dalila Alves, Filipa Ferraz, Victor Alves

Part X

Contents

Ensemble Learning

Implementing Ensemble Learning Models

For this example, we will use the already known dataset *titanic.csv*. Our goal is to predict which passengers survived the Titanic shipwreck.

We have used Logistic Regression in this context; now, we are going to try a few ensemble learning models.

The Data

It will be used data frame with 891 observations on the following 12 variables:

- **PassengerId**
- **Survival** - 0 = No, 1 = Yes
- **Pclass** - Ticket class; 1 = 1st class, 2 = 2nd class, 3 = 3rd class
- **Sex**
- **Age**
- **SibSp** - Number of siblings/spouses aboard the ship
- **Parch** - Number of parents/children aboard the ship
- **Ticket**
- **Fare**
- **Cabin**
- **Embarked** - Port of embarkation; C = Cherbourg, Q = Queenstown, S = Southampton

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

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

```
pip install xgboost
```

Implementing Ensemble Learning Models

Import libraries

```
import sklearn as skl
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, StratifiedShuffleSplit
from sklearn import tree
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, HistGradientBoostingClassifier
from sklearn.ensemble import GradientBoostingClassifier, VotingClassifier, BaggingClassifier, StackingClassifier
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import classification_report
```

Implementing Ensemble Learning Models

Get the data and inspect it

```
df = pd.read_csv("titanic.csv")
```

```
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
```

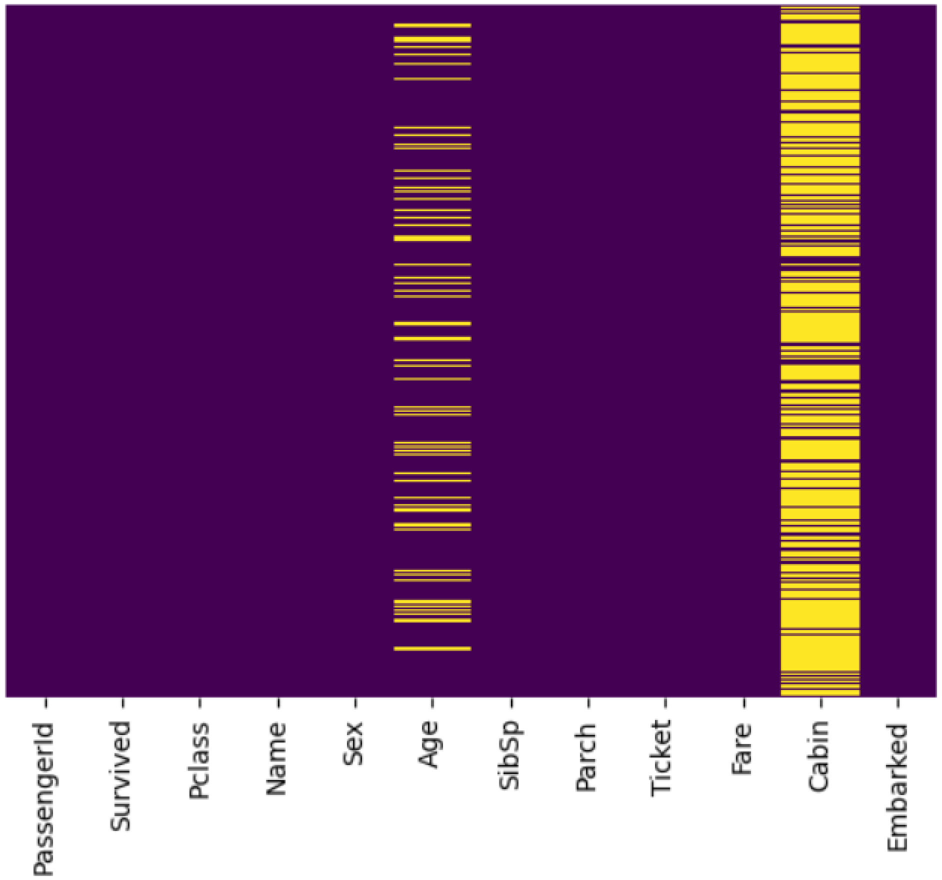
```
df.head()
```

	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

Implementing Ensemble Learning Models

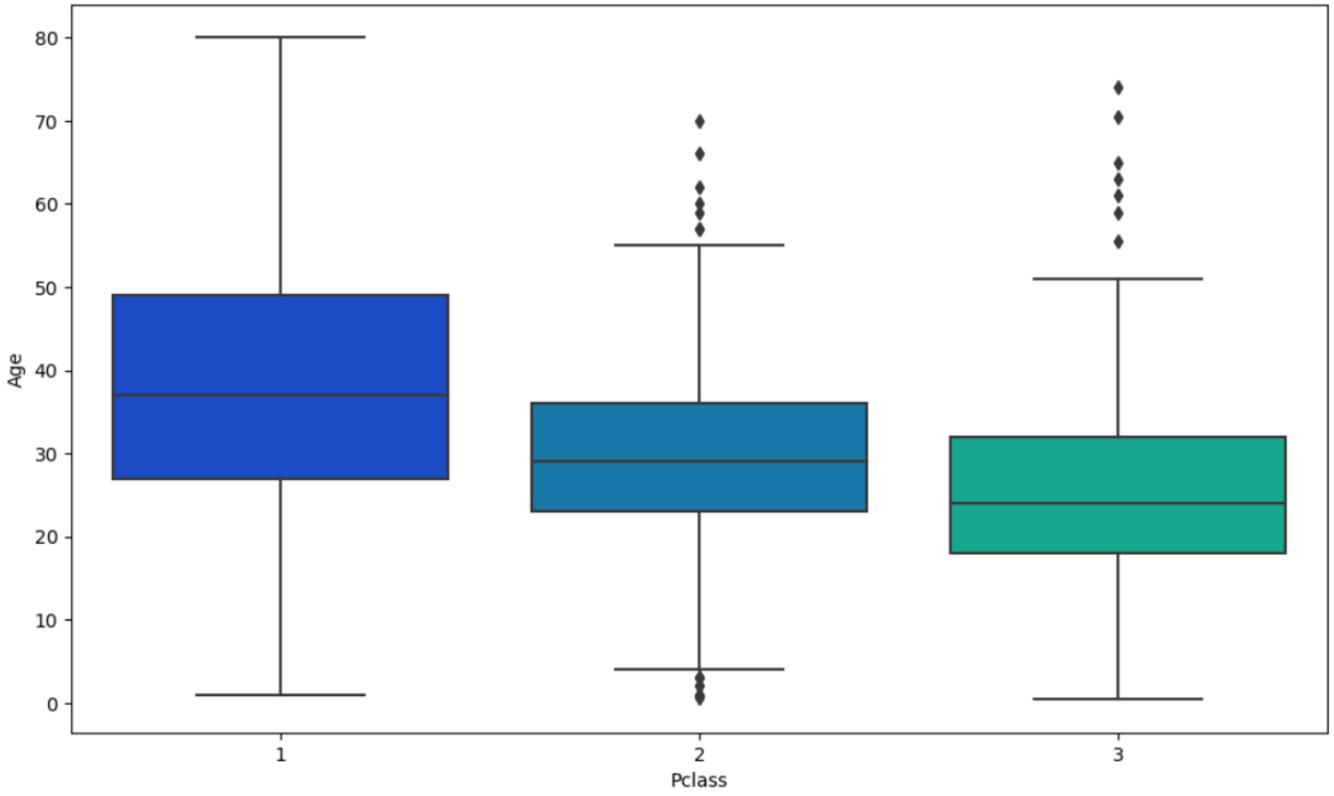
Let's check the missing values:

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



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

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



Implementing Ensemble Learning Models

Let's impute the missing values in *Age* with the ticket class, *Pclass*:

```
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)
```

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

Implementing Ensemble Learning Models

Prepare the data frames

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

Train Test Split

```
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,))
```

Implementing Ensemble Learning Models

Models

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

Implement a *DT Classifier* with *max_depth* of 2 and a *random_state* of 2022. Then fit the created model:

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

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

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

Obtain the accuracy of the model:

```
dt_score = dt_model.score(X_test, y_test)
```

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

```
Accuracy: 69.29%
```

Implementing Ensemble Learning Models

Obtain the predictions and print the *classification_report*:

```
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
```

Save the model's accuracy in a dictionary:

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

Implementing Ensemble Learning Models

Support Vector Machine

Implement a *SVM* with a *random_state* of 2022. Then fit the created model:

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

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

```
▾ SVC
SVC(random_state=2022)
```

Obtain the accuracy of the model:

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

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

```
Accuracy: 68.16%
```

Obtain the predictions and print the *classification_report*:

```
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
```

Save the model's accuracy in the dictionary:

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

Let's try some *Ensemble Learning* models.

Implementing Ensemble Learning Models

Bagging *(Bootstrap Aggregating)*

Implement a *StratifiedShuffleSplit* with *n_splits* of 10, *test_size* of 20 and a *random_state* of 2022:

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

Implement a *BaggingClassifier* with the previous *DT* model and *bootstrap* set *True*:

```
bg_model = BaggingClassifier(estimator = dt_model, bootstrap = True)
```

Implement a *GridSearchCV* with the *BaggingClassifier* model, *n_estimators* of [10, 40, 60, 80, 100, 160] as *parameters* and the *StratifiedShuffleSplit* as *cv*. Then fit the created model:

```
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)
```

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

Obtain the accuracy of the best model:

```
bst_bg_score = bst_bg_model.score(X_test, y_test)
```

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

```
Accuracy: 70.79%
```

Obtain the predictions and print the *classification_report*:

```
bg_predictions = bst_bg_model.predict(X_test)
```

```
print(classification_report(y_test, bg_predictions))
```

```
              precision    recall  f1-score   support

           0       0.76      0.80      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       0.70      0.71      0.70       267
```

Save the model's accuracy in the dictionary:

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

Implementing Ensemble Learning Models

Random Forest

Implement a *RandomForestClassifier* with *bootstrap* set *False*, *max_depth* of 2 and *verbose* of 1. Then fit the created model:

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

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

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

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

Obtain the accuracy of the model:

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

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

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

```
Accuracy: 73.41%
```

Obtain the predictions and print the *classification_report*:

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

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

```
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
```

Save the model's accuracy in the dictionary:

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

Implementing Ensemble Learning Models

Boosting

Gradient Boosting

Implement a *GradientBoostingClassifier* with *n_estimators* of 100, *learning_rate* of 1.0, *max_depth* of 1 and *random_state* of 2022. Then fit the created model:

```
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=2
022)
```

Obtain the accuracy of the model:

```
gbc_score = gbc_model.score(X_test, y_test)
```

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

```
Accuracy: 71.16%
```

Obtain the predictions and print the *classification_report*:

```
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
```

Save the model's accuracy in the dictionary:

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

Implementing Ensemble Learning Models

XGBoost

Implement a *XGBClassifier* with *max_depth* of 1 and *objective* of *reg:squarederror*. Then fit the created model:

```
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_rou
nds=None,
              enable_categorical=False, eval_metric=None, feature_ty
pes=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,
```

Obtain the accuracy of the model:

```
xgb_score = xgb_model.score(X_test, y_test)
```

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

```
Accuracy: 73.03%
```

Obtain the predictions and print the *classification_report*:

```
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
```

Save the model's accuracy in the dictionary:

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

Implementing Ensemble Learning Models

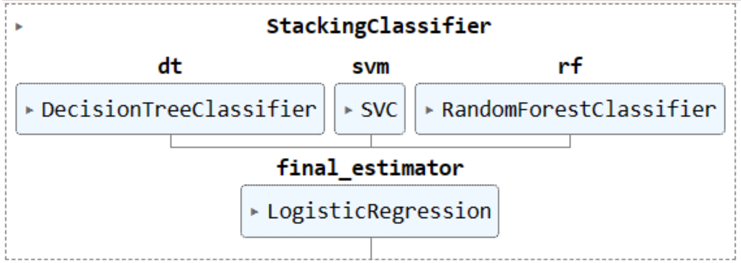
Stacking

Implement a *StackingClassifier* with 3 *estimators* (DT, SVM and RF models) and *LogisticRegression* as *final_estimator*. Then fit the created model:

```
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)
```



Obtain the accuracy of the model:

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

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

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

```
Accuracy: 71.91%
```

Obtain the predictions and print the *classification_report*:

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

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

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

```
              precision    recall  f1-score   support

           0       0.76      0.83      0.79       173
           1       0.62      0.51      0.56        94

    accuracy                           0.72       267
   macro avg       0.69      0.67      0.68       267
weighted avg       0.71      0.72      0.71       267
```

Save the model's accuracy in the dictionary:

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

Implementing Ensemble Learning Models

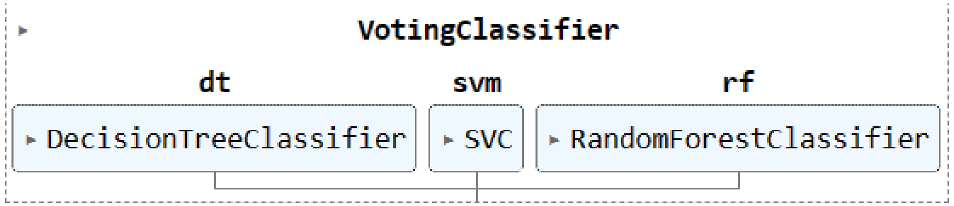
Max Voting

Majority Class Labels (Majority/Hard Voting)

Implement a *VotingClassifier* with 3 *estimators* (DT, SVM and RF models), *voting* of *hard* and *weights* of [2, 1, 2]. Then fit the created model:

```
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)
```



Obtain the accuracy for each model using *cross_val_score*:

```
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))
```

```
Accuracy: 0.69 (+/- 0.05) [dt]
Accuracy: 0.66 (+/- 0.04) [svm]
Accuracy: 0.71 (+/- 0.05) [rf]
Accuracy: 0.69 (+/- 0.07) [Ensemble]
```

Obtain the accuracy of the max voted model:

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

```
Accuracy: 71.91%
[Parallel(n_jobs=1)]: Using backend SequentialBackend wi
workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:
```

Obtain the predictions and print the *classification_report*:

```
hvt_predictions = hvt_model.predict(X_test)
```

```
print(classification_report(y_test, hvt_predictions))
```

```
              precision    recall  f1-score   support

           0       0.77      0.80      0.79       173
           1       0.61      0.56      0.59        94

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

Save the model's accuracy in the dictionary:

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

Implementing Ensemble Learning Models

Models accuracy comparison

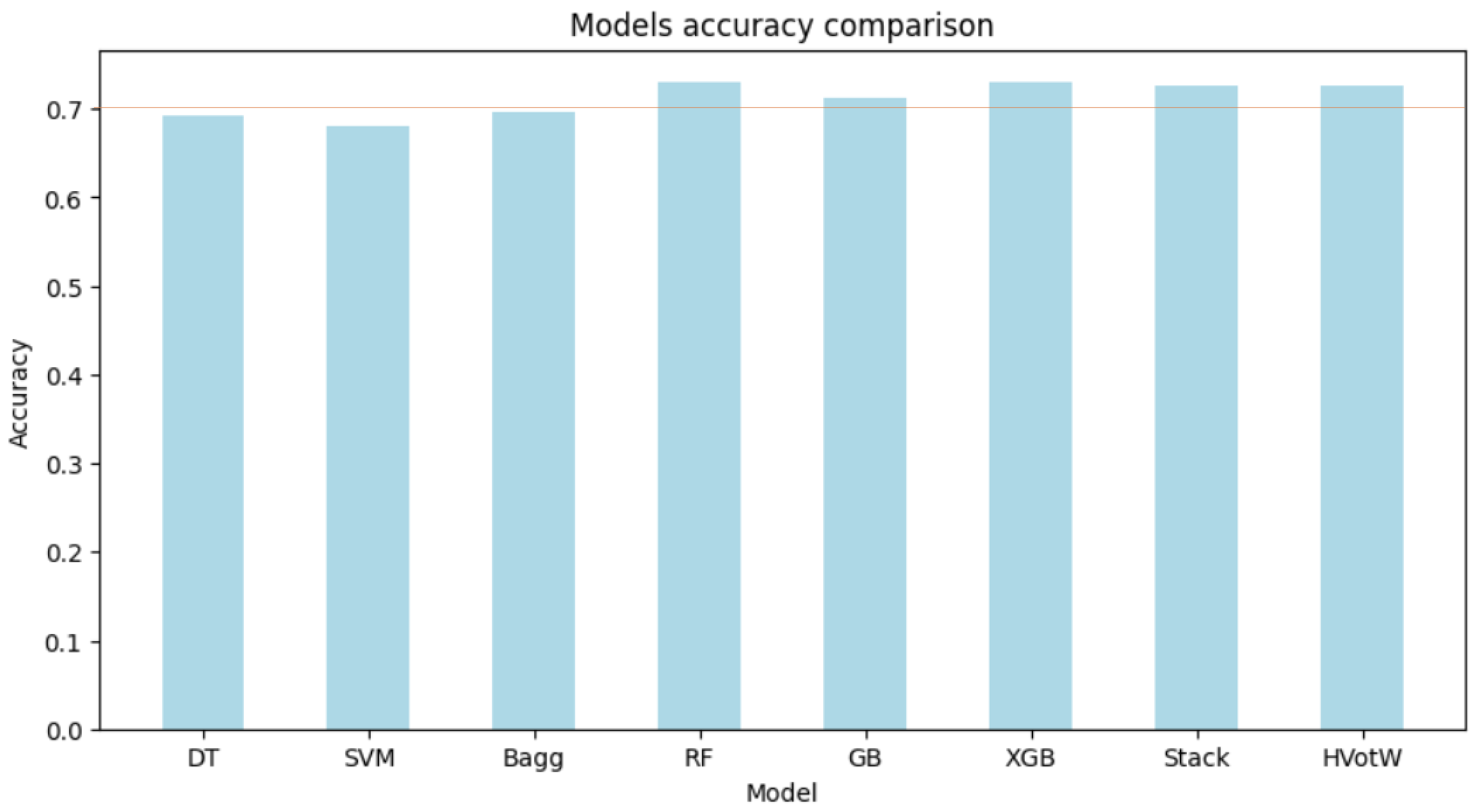
Let's plot a bar chart with the accuracy of each created model:

```
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()
```



Implementing Ensemble Learning Models

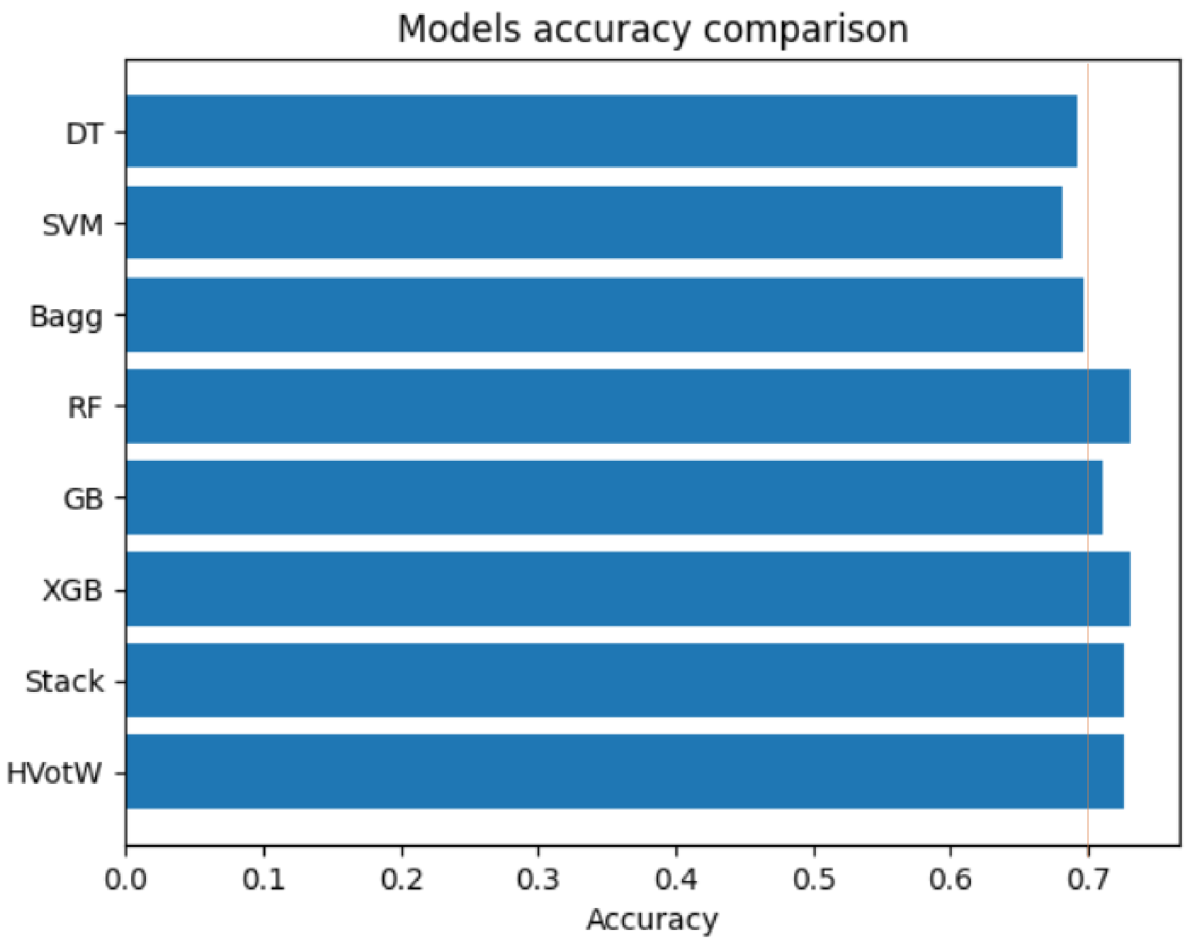
We can visualize the bars horizontally:

```
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()
```



Implementing Ensemble Learning Models

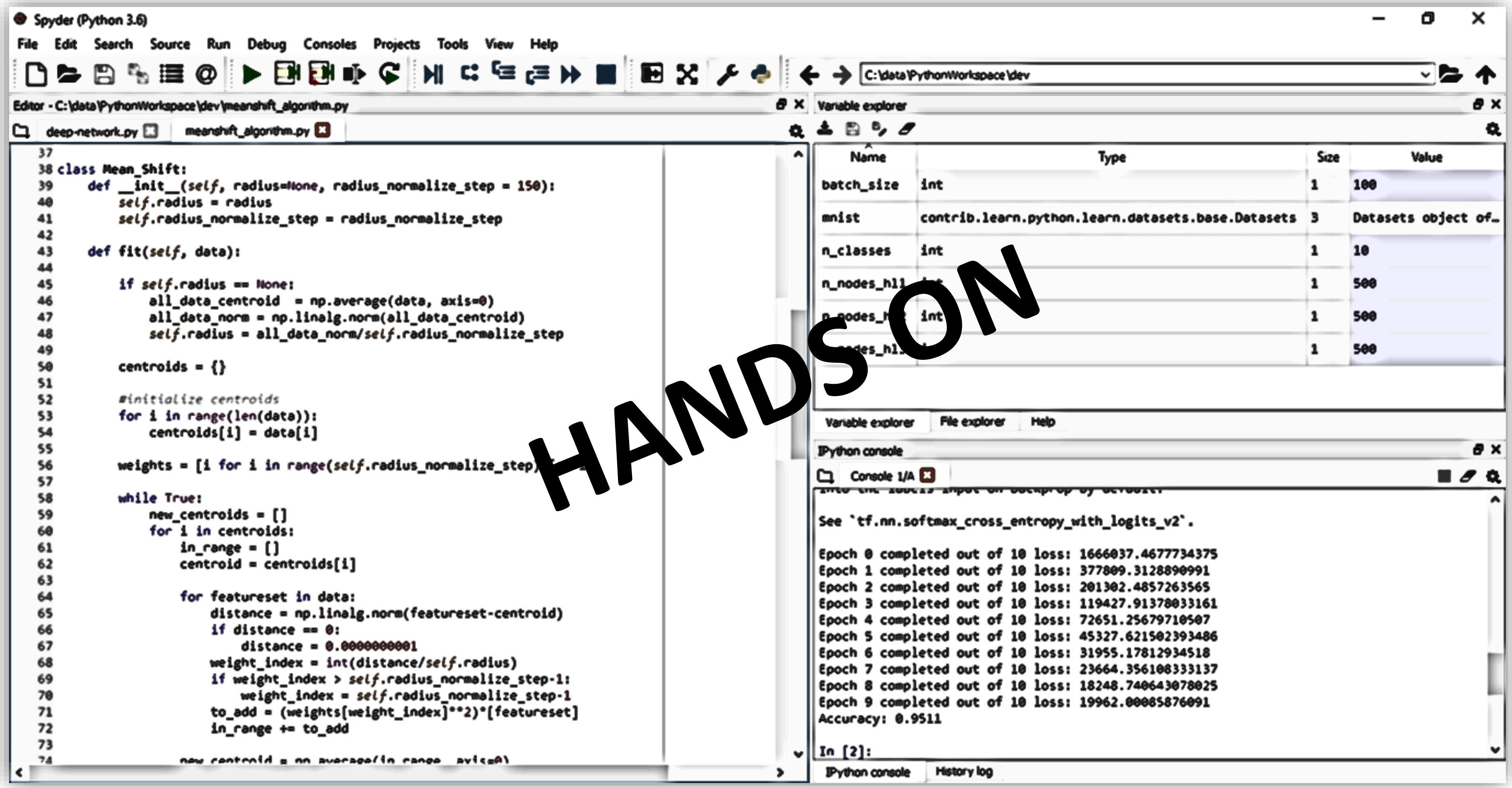
Or simply print the dictionary created with the results:

```
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   0.73
HVotW   0.73
```

Which model perfomed better?
Which models are worth tunning?

Hands On


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\data\PythonWorkspace\dev
Editor - C:\data\PythonWorkspace\dev\meanshift_algorithm.py
deep-network.py
meanshift_algorithm.py
37
38 class Mean_Shift:
39 def __init__(self, radius=None, radius_normalize_step = 150):
40 self.radius = radius
41 self.radius_normalize_step = radius_normalize_step
42
43 def fit(self, data):
44
45 if self.radius == None:
46 all_data_centroid = np.average(data, axis=0)
47 all_data_norm = np.linalg.norm(all_data_centroid)
48 self.radius = all_data_norm/self.radius_normalize_step
49
50 centroids = {}
51
52 #initialize centroids
53 for i in range(len(data)):
54 centroids[i] = data[i]
55
56 weights = [i for i in range(self.radius_normalize_step)
57
58 while True:
59 new_centroids = []
60 for i in centroids:
61 in_range = []
62 centroid = centroids[i]
63
64 for featureset in data:
65 distance = np.linalg.norm(featureset-centroid)
66 if distance == 0:
67 distance = 0.0000000001
68 weight_index = int(distance/self.radius)
69 if weight_index > self.radius_normalize_step-1:
70 weight_index = self.radius_normalize_step-1
71 to_add = (weights[weight_index]**2)*[featureset]
72 in_range += to_add
73
Variable explorer
Name
Type
Size
Value
batch_size
int
1
100
mnist
contrib.learn.python.learn.datasets.base.Datasets
3
Datasets object of…
n_classes
int
1
10
n_nodes_hl1
1
500
1
500
1
500
Variable explorer
File explorer
Help
IPython console
Console 1/A
See `tf.nn.softmax_cross_entropy_with_logits_v2`.
Epoch 0 completed out of 10 loss: 1666037.4677734375
Epoch 1 completed out of 10 loss: 377809.3128890991
Epoch 2 completed out of 10 loss: 201302.4857263565
Epoch 3 completed out of 10 loss: 119427.91378033161
Epoch 4 completed out of 10 loss: 72651.25679710507
Epoch 5 completed out of 10 loss: 45327.621502393486
Epoch 6 completed out of 10 loss: 31955.17812934518
Epoch 7 completed out of 10 loss: 23664.356108333137
Epoch 8 completed out of 10 loss: 18248.740643078025
Epoch 9 completed out of 10 loss: 19962.00085876091
Accuracy: 0.9511
In [2]:
IPython console
History log
HANDS ON