

data_science

June 12, 2018

1 Data science

Florian HASSLER et Dimitri LIZZI
juin 2018

1.1 Partie 1: Titanic

1.1.1 Setup

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

%matplotlib inline
plt.rcParams["figure.figsize"] = (10, 9)
```

1.1.2 Fonctions utilitaires

```
In [2]: def count_bins(arr1, arr2):
    '''Counts the number of bins on each dimensions for a 2d histogram'''
    return len(set(arr1)), len(set(arr2))
```

1.1.3 Import des données depuis un fichier CSV

```
In [3]: titanic_data = np.genfromtxt('titanic.dat', delimiter=',', skip_header=1)

classes = titanic_data[:, 0]
ages = titanic_data[:, 1]
sexes = titanic_data[:, 2]
survived = titanic_data[:, 3]
```

1.1.4 Nettoyage des données

Ages Il y a deux valeurs possibles dans la variable age :

```
In [4]: ages_values = list(set(ages))
ages_count_per_value = {v: sum(ages == v) for v in ages_values}
ages_count_per_value
```

```
Out [4]: {-0.22800000000000001: 2092, 4.379999999999999: 109}
```

Il paraît probable que la première valeur, attribuée à 2092 personnes, désigne les adultes, tandis que l'autre valeur, attribuée à 109 personnes, désigne enfants, car il y a probablement plus d'adultes que d'enfants sur le bateau.

On peut binariser le tableau `ages` pour en faire un tableau `is_adult`, qui contient 1 si la personne est adulte et 0 si la personne est un enfant:

```
In [5]: is_adult = (ages < 0).astype(int)
        is_adult
```

```
Out [5]: array([1, 1, 1, ..., 1, 1, 1])
```

Sexes Il y a deux valeurs possibles pour les sexes:

```
In [6]: sexes_values = list(set(sexes))
        sexes_count_per_value = {v: sum(sexes == v) for v in sexes_values}
        sexes_count_per_value
```

```
Out [6]: {-1.9199999999999999: 470, 0.52100000000000002: 1731}
```

Il paraît raisonnable de penser que sur un bateau de 1912, il y avait plus d'hommes que de femmes, soit 1731 hommes pour 470 femmes.

On peut binariser le tableau `sexes` en `is_male`:

```
In [7]: is_male = (sexes > 0).astype(int)
        is_male
```

```
Out [7]: array([1, 0, 0, ..., 1, 1, 0])
```

Classes Il y a 4 valeurs possibles dans classes:

```
In [8]: classes_values = list(set(classes))
        classes_count_per_value = {v: sum(classes == v) for v in classes_values}
        classes_count_per_value
```

```
Out [8]: {-1.8700000000000001: 325,
          -0.92300000000000004: 285,
           0.021399999999999999: 706,
           0.96499999999999997: 885}
```

En se renseignant un peu sur le titanic, on peut apprendre qu'il y avait 3 classes sur le bateau, plus l'équipage. Selon wikipédia, les effectifs sont les suivants:

- Première classe: 325 personnes
- Seconde classe: 285 personnes
- Troisième classe: 706 personnes
- Equipage: 908 personnes

Notre jeu de données ne semble pas contenir tout l'équipage, mais les autres chiffres correspondent.

On peut simplifier les données en donnant des valeurs entières aux classes:

```
In [9]: def get_cleaned_class(c : int):
        if c > 0.1:
            return 0 # crew
        if c < -1:
            return 1 # first class
        if 0 > c > -1:
            return 2 # second class
        if 0 < c < 0.1:
            return 3 # third class
```

```
classes_clean = np.array([get_cleaned_class(c) for c in classes])
classes_clean
```

```
Out[9]: array([1, 2, 2, ..., 1, 0, 2])
```

Survivants Deux valeurs sont possibles pour les survivants:

```
In [10]: survived_values = set(survived)
        survived_count_per_value = {v: sum(survived == v) for v in survived_values}
        survived_count_per_value
```

```
Out[10]: {-1.0: 1490, 1.0: 711}
```

Selon [wikipédia](#), il y a 710 survivants et 1514 victimes. On peut conclure que la valeur -1 correspond aux victimes, tandis qu la valeur 1 correspond aux survivants.

On peut extraire une feature booléenne `is_survivor` de ces données:

```
In [11]: is_survivor = (survived == 1).astype(int)
        is_survivor
```

```
Out[11]: array([0, 1, 1, ..., 0, 0, 1])
```

1.1.5 Class vs Age vs Sex

```
In [12]: from mpl_toolkits.mplot3d import Axes3D
```

```
fig_3d = plt.figure()
ax = Axes3D(fig_3d)
```

```
points = list(zip(classes_clean, is_adult, is_male))
points_values = set(points)
points_counts = [points.count(point) for point in points_values]
points_survivors = [sum(is_survivor[i]
```

```

        for i, point in enumerate(points)
        if point == value)
    for value in points_values]
points_survivors_rates = [point_survivors / float(points_counts[i])
    for i, point_survivors in enumerate(points_survivors)]

x, y, z = zip(*points_values)

classes_ticks = [0, 1, 2, 3]
classes_ticklabels = ['crew', 'first class', 'second class', 'third class']

is_adult_ticks = [0, 1]
is_adult_ticklabels = ['child', 'adult']

is_male_ticks = [0, 1]
is_male_ticklabels = ['female', 'male']

ax.set_xlabel('class')
ax.set_xticks(classes_ticks)
ax.set_xticklabels(classes_ticklabels)

ax.set_ylabel('age')
ax.set_yticks(is_adult_ticks)
ax.set_yticklabels(is_adult_ticklabels)

ax.set_zlabel('sex')
ax.set_zticks(is_male_ticks)
ax.set_zticklabels(is_male_ticklabels)

ax.set_title('class vs age vs sex')
s = ax.scatter(x, y, z, sizes=points_counts, c=points_survivors_rates, alpha=1)
cbar = fig_3d.colorbar(s)
_ = cbar.ax.set_ylabel('survival rate')

```



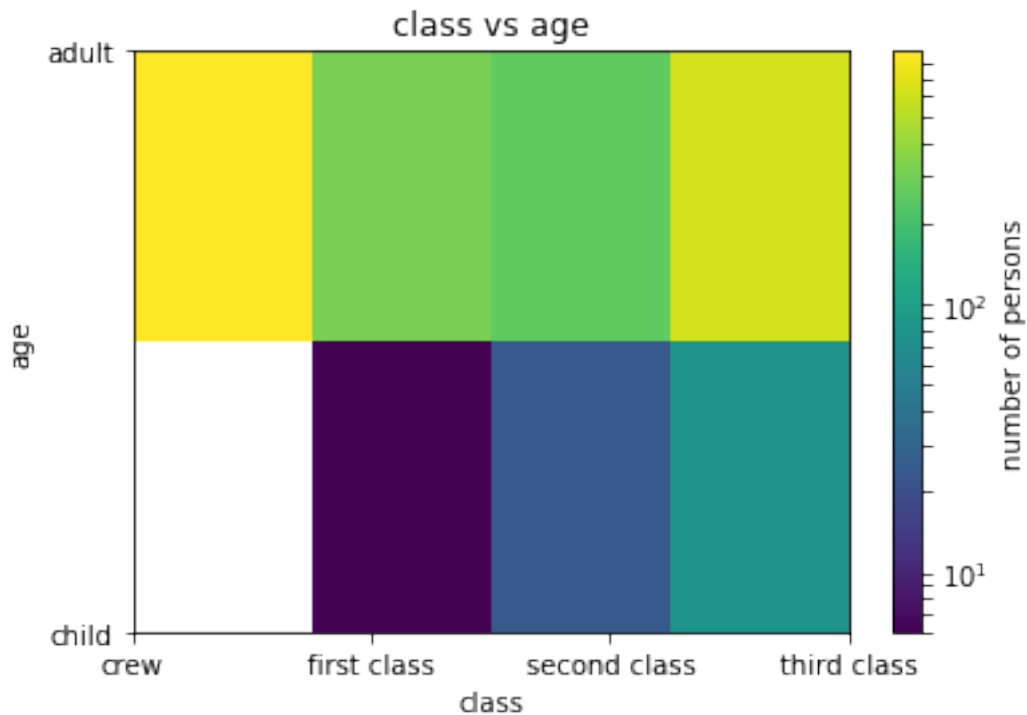
On peut constater dans le graphique ci-dessus que les taux de survie les plus élevés sont chez les enfants de première et seconde classe, ainsi que chez les femmes de première classe. Ceux de troisième classe sont moins chanceux.

Les hommes adultes sont les plus nombreux, mais ils sont aussi ceux avec le taux de décès le plus nombreux.

1.1.6 Classe vs Age

Le graphique ci dessous affiche la nombre de personnes par classe et age. On peut constater qu'il y a plus d'adultes que d'enfants dans chaque classe, et que l'équipage ne comporte pas d'enfants.

```
In [13]: fig_class_age = plt.figure()
plt.xlabel('class')
plt.ylabel('age')
plt.title('class vs age')
_ = plt.hist2d(classes_clean, is_adult, norm=LogNorm(), bins=count_bins(classes_clean))
_ = plt.xticks(classes_ticks, classes_ticklabels)
_ = plt.yticks(is_adult_ticks, is_adult_ticklabels)
cb = plt.colorbar()
_ = cb.ax.set_ylabel("number of persons")
```



1.1.7 Classe vs Sexe

Le graphique ci-dessous montre le nombre de personnes par classe et par sexe. On peut constater que les hommes sont plus nombreux que les femmes dans toutes les classes, et en particulier dans l'équipage.

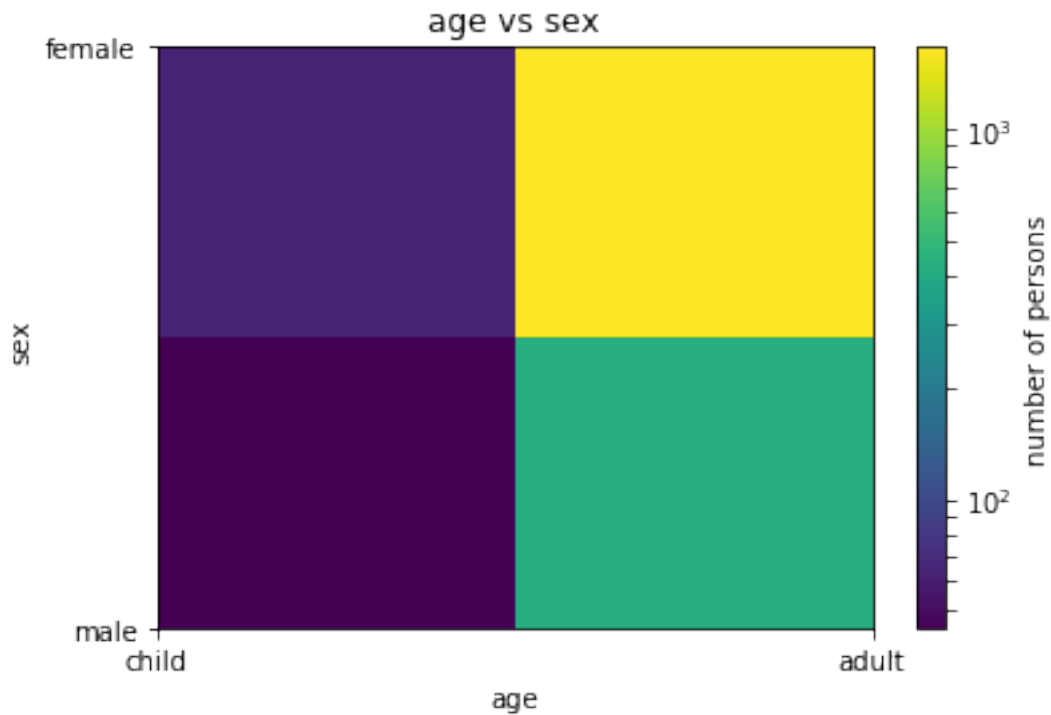
```
In [14]: fig_class_sex = plt.figure()
plt.xlabel('class')
plt.ylabel('sex')
plt.title('class vs sex')
_ = plt.hist2d(classes_clean, is_male, norm=LogNorm(), bins=count_bins(classes_clean,
_ = plt.xticks(classes_ticks, classes_ticklabels)
_ = plt.yticks(is_male_ticks, is_male_ticklabels)
cb = plt.colorbar()
_ = cb.ax.set_ylabel("number of persons")
```



1.1.8 Age vs Sexe

Le graphique ci-dessous montre le nombre de personnes par age et par sexe. On peut constater que les adultes sont bien plus nombreux que les enfants. On peut aussi voir qu'il y a légèrement plus de filles que de garçons chez les enfants.

```
In [15]: plt.xlabel('age')
plt.ylabel('sex')
plt.title('age vs sex')
_ = plt.hist2d(is_adult, is_male, norm=LogNorm(), bins=count_bins(ages, sexes))
_ = plt.xticks(is_adult_ticks, is_adult_ticklabels)
_ = plt.yticks(is_male, is_male_ticklabels)
cb = plt.colorbar()
_ = cb.ax.set_ylabel("number of persons")
```



1.1.9 Recherche de K optimal

Le code suivant classe les personnes du bateau en k groupes à l'aide de la méthode des *K-means*, pour k variant de 1 à 50.

La distance moyenne entre le barycentre de chaque groupe et ses membres est calculée pour chaque valeur de k , afin de pouvoir déterminer quel est le nombre de groupes optimal.

```
In [16]: from scipy.spatial import distance
          from sklearn.cluster import KMeans

          data = np.array(list(zip(
              is_adult,
              is_male,
              classes_clean)))

          K = list(range(1, 50))
          mean_dists = []
          for k in K:
              kmeans = KMeans(n_clusters=k, random_state=0)
              kmeans.fit(data)

              dists = []
              for i, label in enumerate(kmeans.labels_):
                  row = data[i]
```



```

cluster_center = kmeans.cluster_centers_[label]
dist = distance.euclidean(row, cluster_center)
dists.append(dist)

mean_dist = np.mean(dists)
mean_dists.append(mean_dist)

```

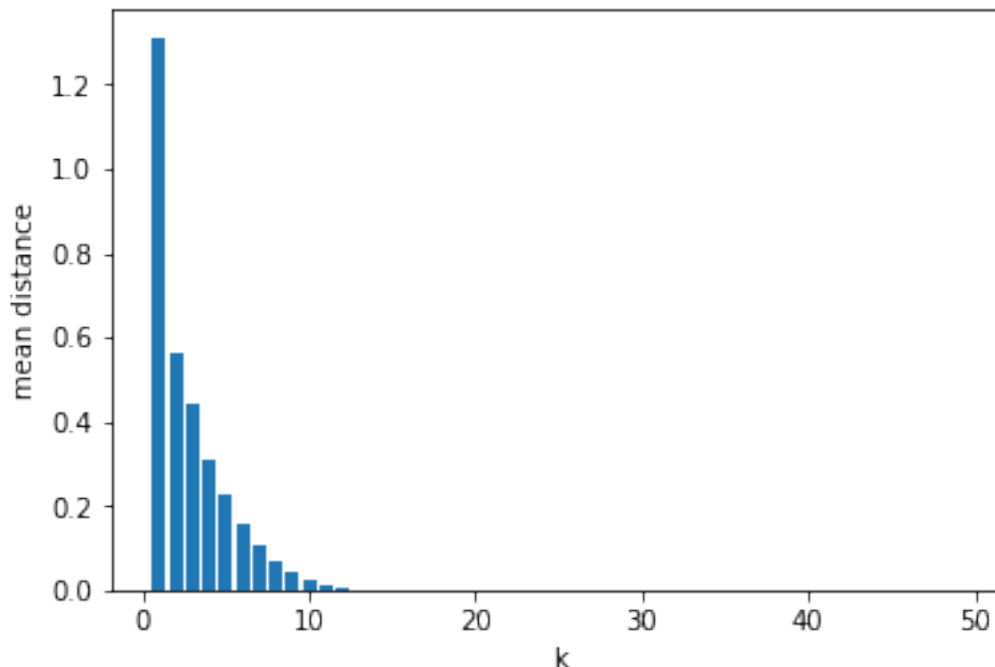
Le graphique suivant montre les distances calculées ci-dessus pour chaque valeur de k . La distance moyenne semble se stabiliser à partir de $k = 13$.

```

In [17]: _ = plt.bar(K, mean_dists)

_ = plt.xlabel("k")
_ = plt.ylabel("mean distance")

```



Pour plus de précision, le graphique ci-dessous affiche les mêmes données, mais avec une échelle logarithmique pour les distances.

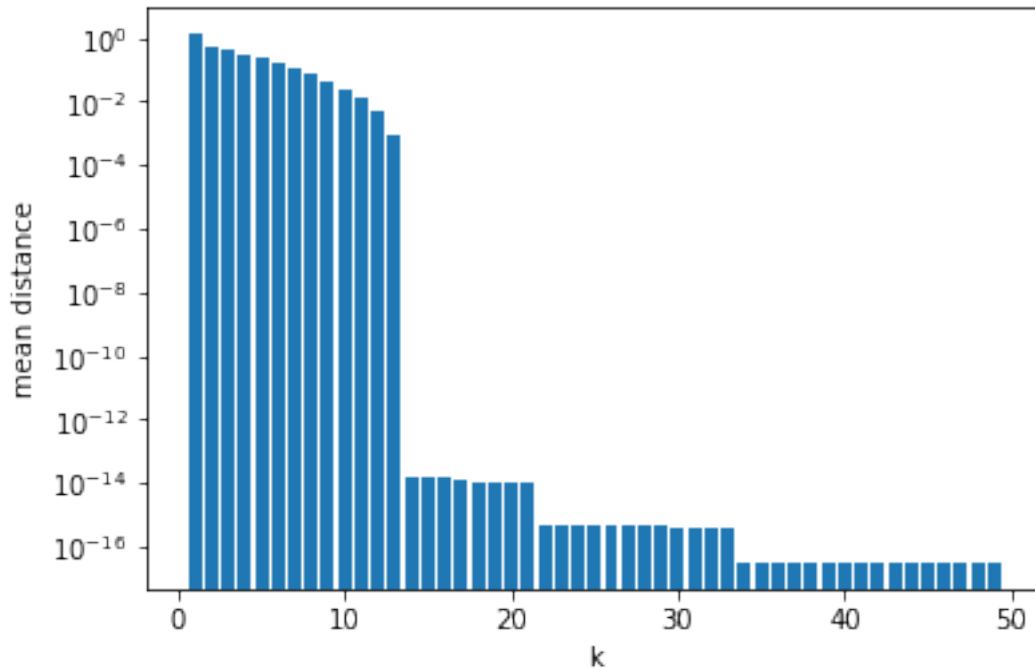
On se rend compte que la distance moyenne se stabilise vraiment à partir de $k = 34$.

```

In [18]: _ = plt.bar(K, mean_dists, log=True)

_ = plt.xlabel("k")
_ = plt.ylabel("mean distance")

```



Ce graphique 3D, permet de se rendre compte que les données(point) sont bien centrées sur les baricentres(croix).

```
In [19]: kmeans = KMeans(n_clusters=34, random_state=0)
          kmeans.fit(data)

          fig_3d = plt.figure()
          ax = Axes3D(fig_3d)

          clusters_centers = kmeans.cluster_centers_
          labels = kmeans.labels_

          ax.set_xlabel('class')
          ax.set_xticks(classes_ticks)
          ax.set_xticklabels(classes_ticklabels)

          ax.set_ylabel('age')
          ax.set_yticks(is_adult_ticks)
          ax.set_yticklabels(is_adult_ticklabels)

          ax.set_zlabel('sex')
          ax.set_zticks(is_male_ticks)
          ax.set_zticklabels(is_male_ticklabels)

          for cluster, cluster_center in enumerate(clusters_centers, 1):
```

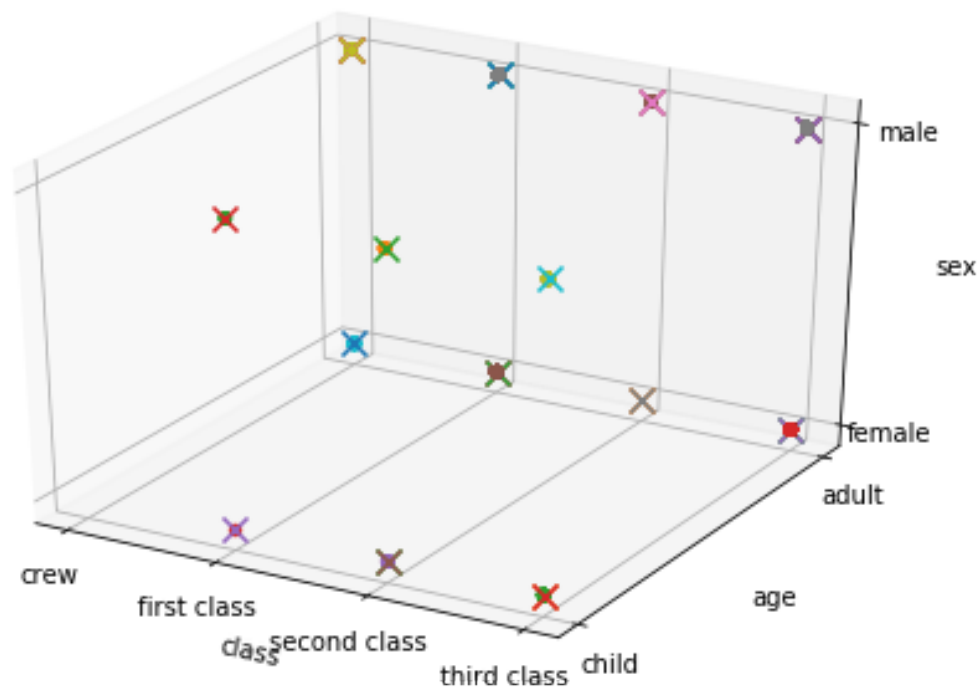
```

color = "C" + str(cluster % 10)

cluster_is_adult = cluster_center[0]
cluster_is_male = cluster_center[1]
cluster_classes = cluster_center[2]

ax.scatter(cluster_classes, cluster_is_adult, cluster_is_male, marker='x', color=
row = [(classes_clean[i], is_adult[i], is_male[i])
        for i, label in enumerate(labels)
        if label == cluster]
if len(row) > 0:
    x, y, z = zip(*row)
    ax.scatter(x, y, z, color=color)

```



1.2 Partie 2

1.2.1 Chargements des datasets breast_cancer et wine

```
In [20]: from sklearn import datasets
```

```

breast_cancer = datasets.load_breast_cancer()
wine = datasets.load_wine()

```

1.2.2 Validation croisée à 5 segments

Une validation croisée à 5 segments est effectuée 10 fois sur chaque jeu de données, avec plusieurs paramétrages des classifieurs suivants:

- k-neighbors
 - k = 15, 30
- decision tree
 - min_samples_leaf=15, 30
- multi-layer perceptron
 - couche 1: 5000 neurones
 - couche 2: 0, 1000 neurones
 - fonctions d'activation: tanh, relu

```
In [21]: from sklearn.model_selection import RepeatedKFold, cross_val_score
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.preprocessing import StandardScaler
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.neural_network import MLPClassifier
         from collections import defaultdict

         kneighbors_classifiers = {'KNeighborsClassifier(%s)' % i:
                                   KNeighborsClassifier(i)
                                   for i in [15, 30]}

         decisiontree_classifiers = {'DecisionTreeClassifier(min_samples_leaf=%s)' % i:
                                     DecisionTreeClassifier(min_samples_leaf=i)
                                     for i in [15, 30]}

         mlpc_classifiers = {
             'MLPClassifier(hidden_layer_sizes=(%s,%s), activation=%s, early_stopping=True)'
             % (l1, l2, activation):
                 MLPClassifier(hidden_layer_sizes=(l1,l2) if l2 > 0 else (l1,),
                               activation=activation,
                               early_stopping=True)
             for l1 in {5000}
             for l2 in {0, 1000}
             for activation in {'tanh', 'relu'}
         }

         classifiers = {
             "KNeighborsClassifier":kneighbors_classifiers,
             "DecisionTreeClassifier":decisiontree_classifiers,
             "MLPClassifier": mlpc_classifiers
         }
```

```

datasets = {
    'breast_cancer': (breast_cancer['data'], breast_cancer['target']),
    'wine': (wine['data'], wine['target'])
}

```

```

rkf = RepeatedKfold(n_splits=5, n_repeats=10)

```

```

mean_scores = {
    dataset_name: {
        classifier_type: defaultdict(dict)
        for classifier_type in classifiers.keys()
    }
    for dataset_name in datasets.keys()
}

```

```

std_scores = {
    dataset_name: {
        classifier_type: defaultdict(dict)
        for classifier_type in classifiers.keys()
    }
    for dataset_name in datasets.keys()
}

```

```

In [22]: for dataset_name, dataset in datasets.items():
        data, target = dataset
        scaler = StandardScaler().fit_transform(data)
        for classifier_type, classifier_instances in classifiers.items():
            for classifier_options, classifier in classifier_instances.items():
                scores = cross_val_score(classifier, data, target, cv=rkf)
                mean_scores[dataset_name][classifier_type][classifier_options] = scores.mean()
                std_scores[dataset_name][classifier_type][classifier_options] = scores.std()

```

Le graphique ci dessous montre les scores moyens avec chaque classifieur pour le dataset breast cancer.

On peut constater que certains résultats sont plutôt bons : - KNeighbors avec k=15 - MLP avec 5000 neurones dans la première couche cachée et la méthode d'activation tanh - decision tree avec min_sample_leafs=15

```

In [23]: plt.rcParams["figure.figsize"] = (10, 27)

```

```

fig, axes = plt.subplots(len(classifiers), 1)
plt.subplots_adjust(hspace=0.237)
for classifier_index, classifier_type in enumerate(classifiers.keys()):
    scores = list(mean_scores['breast_cancer'][classifier_type].items())
    scores.sort(key=lambda s: s[1])
    classifier_names, values = zip(*scores)
    stds = [std_scores['breast_cancer'][classifier_type][classifier_name] for classifier_name, classifier in classifier_instances.items()]
    indices = list(range(len(scores)))

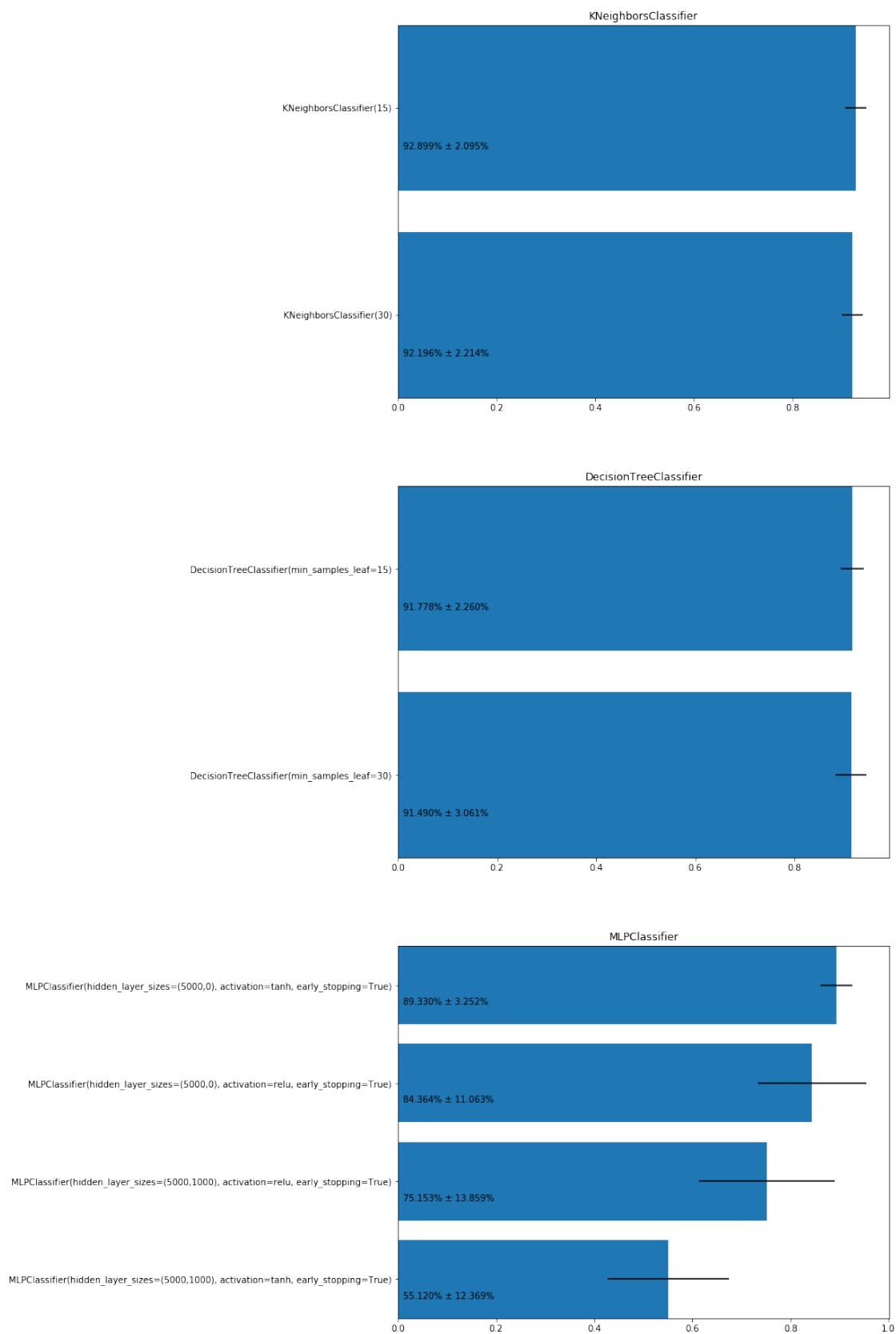
```

```

ax = axes[classifier_index]
ax.set_title(classifier_type.replace('_', ' '))
ax.margins(y=0)
ax.barh(indices, values, tick_label=classifier_names, xerr=stds)

for j in indices:
    ax.text(.01, j - 0.2, '%.3f%% ± %.3f%%' % (100. * values[j], 100 * stds[j]))

```



```

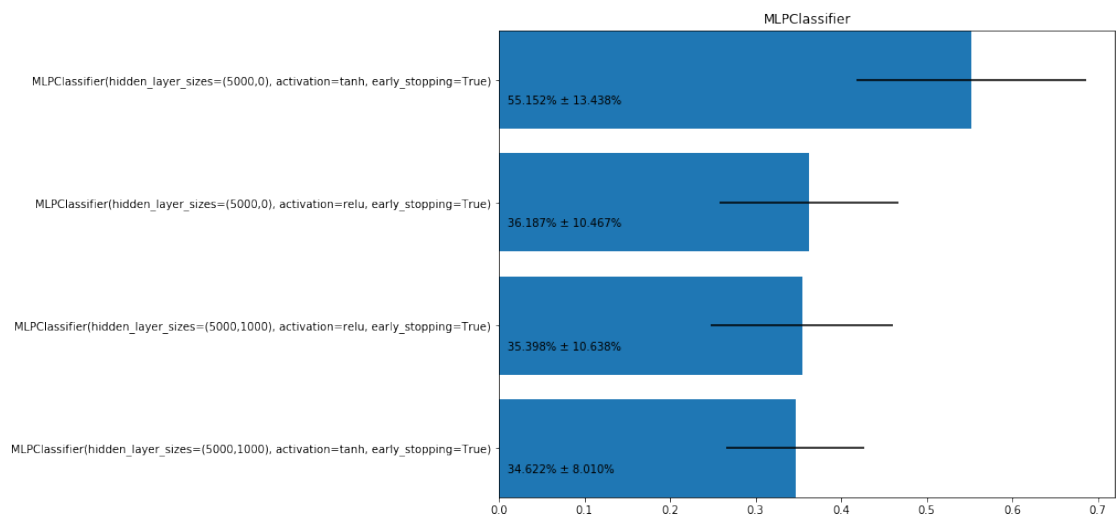
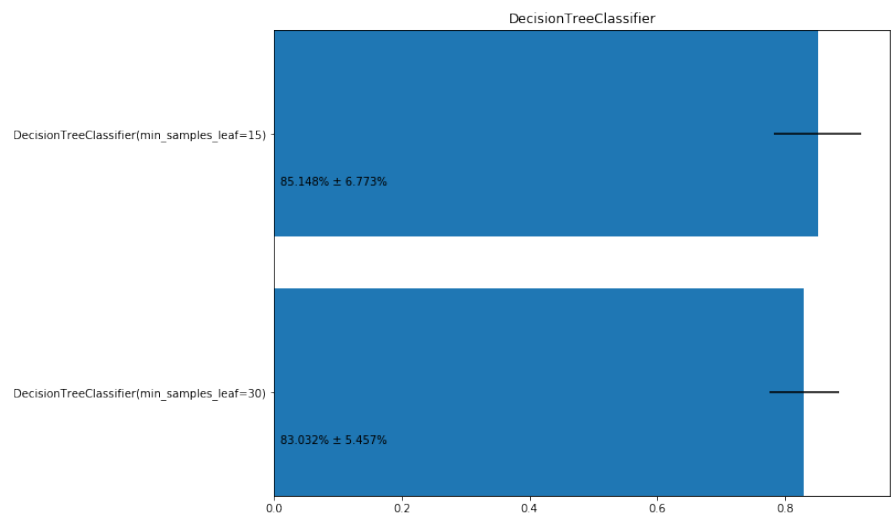
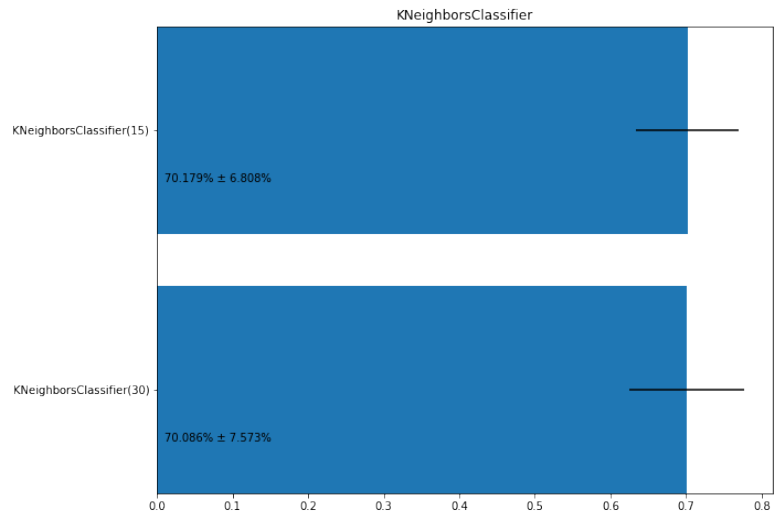
In [24]: plt.rcParams["figure.figsize"] = (10, 27)

fig, axes = plt.subplots(len(classifiers), 1)
plt.subplots_adjust(hspace=0.237)
for classifier_index, classifier_type in enumerate(classifiers.keys()):
    scores = list(mean_scores['wine'][classifier_type].items())
    scores.sort(key=lambda s: s[1])
    classifier_names, values = zip(*scores)
    stds = [std_scores['wine'][classifier_type][classifier_name] for classifier_name in classifier_names]
    indices = list(range(len(scores)))

    ax = axes[classifier_index]
    ax.set_title(classifier_type.replace('_', ' '))
    ax.margins(y=0)
    ax.barh(indices, values, tick_label=classifier_names, xerr=stds)

    for j in indices:
        ax.text(.01, j - 0.2, '%.3f%% ± %.3f%%' % (100. * values[j], 100 * stds[j]))

```

1.3 Partie 3

Pour cette dernière partie, nous avons choisi d'utiliser le dataset `onlineNewsPopularity` qui contient 39'644 entrées et 61 attributs. Les données sont des métadonnées récoltées pendant 2 ans concernant des articles publiés sur Mashable. Le but de ces données est de prédire si un article va faire le buzz en essayant d'estimer le nombre de partages.

```
In [34]: # data from https://archive.ics.uci.edu/ml/datasets/online+news+popularity
news_data = np.genfromtxt('onlineNewsPopularity.csv', delimiter=',', skip_header=1)
nb_columns = np.shape(news_data)[1]
(
    url, # URL of the article (non-predictive)
    timedelta, # Days between the article publication and the dataset acquisition (n
    n_tokens_title, # Number of words in the title
    n_tokens_content, # Number of words in the content
    n_unique_tokens, # Rate of unique words in the content
    n_non_stop_words, # Rate of non-stop words in the content
    n_non_stop_unique_tokens, # Rate of unique non-stop words in the content
    num_hrefs, # Number of links
    num_self_hrefs, # Number of links to other articles published by Mashable
    num_imgs, # Number of images
    num_videos, # Number of videos
    average_token_length, # Average length of the words in the content
    num_keywords, # Number of keywords in the metadata
    data_channel_is_lifestyle, # Is data channel 'Lifestyle'?
    data_channel_is_entertainment, # Is data channel 'Entertainment'?
    data_channel_is_bus, # Is data channel 'Business'?
    data_channel_is_socmed, # Is data channel 'Social Media'?
    data_channel_is_tech, # Is data channel 'Tech'?
    data_channel_is_world, # Is data channel 'World'?
    kw_min_min, # Worst keyword (min. shares)
    kw_max_min, # Worst keyword (max. shares)
    kw_avg_min, # Worst keyword (avg. shares)
    kw_min_max, # Best keyword (min. shares)
    kw_max_max, # Best keyword (max. shares)
    kw_avg_max, # Best keyword (avg. shares)
    kw_min_avg, # Avg. keyword (min. shares)
    kw_max_avg, # Avg. keyword (max. shares)
    kw_avg_avg, # Avg. keyword (avg. shares)
    self_reference_min_shares, # Min. shares of referenced articles in Mashable
    self_reference_max_shares, # Max. shares of referenced articles in Mashable
    self_reference_avg_shares, # Avg. shares of referenced articles in Mashable
    weekday_is_monday, # Was the article published on a Monday?
    weekday_is_tuesday, # Was the article published on a Tuesday?
    weekday_is_wednesday, # Was the article published on a Wednesday?
    weekday_is_thursday, # Was the article published on a Thursday?
```

```

weekday_is_friday, # Was the article published on a Friday?
weekday_is_saturday, # Was the article published on a Saturday?
weekday_is_sunday, # Was the article published on a Sunday?
is_weekend, # Was the article published on the weekend?
LDA_00, # Closeness to LDA topic 0
LDA_01, # Closeness to LDA topic 1
LDA_02, # Closeness to LDA topic 2
LDA_03, # Closeness to LDA topic 3
LDA_04, # Closeness to LDA topic 4
global_subjectivity, # Text subjectivity
global_sentiment_polarity, # Text sentiment polarity
global_rate_positive_words, # Rate of positive words in the content
global_rate_negative_words, # Rate of negative words in the content
rate_positive_words, # Rate of positive words among non-neutral tokens
rate_negative_words, # Rate of negative words among non-neutral tokens
avg_positive_polarity, # Avg. polarity of positive words
min_positive_polarity, # Min. polarity of positive words
max_positive_polarity, # Max. polarity of positive words
avg_negative_polarity, # Avg. polarity of negative words
min_negative_polarity, # Min. polarity of negative words
max_negative_polarity, # Max. polarity of negative words
title_subjectivity, # Title subjectivity
title_sentiment_polarity, # Title polarity
abs_title_subjectivity, # Absolute subjectivity level
abs_title_sentiment_polarity, # Absolute polarity level
shares, # Number of shares (target)
) = (news_data[:, i] for i in range(nb_columns))

limit = 8000

X = np.array(list(zip(n_tokens_title, n_tokens_content, n_unique_tokens, n_non_stop_w
    num_hrefs, num_imgs, num_videos, average_token_length, num_keywords, data_channel
    data_channel_is_entertainment, data_channel_is_bus, data_channel_is_socmed, data
    data_channel_is_world, weekday_is_monday, weekday_is_tuesday, weekday_is_wednesd
    weekday_is_friday, weekday_is_saturday, weekday_is_sunday, is_weekend))))[:limit]

X = StandardScaler().fit_transform(X)

y = shares[:limit]

```

Nous avons choisis un certains nombre de champ qui nous paraissaient pertinents pour la prédiction souhaitée. On effectue ici une validation croisée à 5 segments, sans répétitions. Trois classifieurs sont utilisés:

- K-neighbors avec k=30
- decision tree avec min_samples_leaf = 30
- MLP avec 5000 neurones à la première couche et 1000 neurones sur la seconde

```
In [35]: kneighbors_classifiers = {'KNeighborsClassifier(%s)' % i:
```

```

        KNeighborsClassifier(i)
    for i in [30]}

decisiontree_classifiers = {'DecisionTreeClassifier(min_samples_leaf=%s)' % i:
    DecisionTreeClassifier(min_samples_leaf=i)
    for i in [30]}

mlpc_classifiers = {
    'MLPClassifier(hidden_layer_sizes=(%s,%s), activation=%s, early_stopping=True)'
    % (l1, l2, activation):
        MLPClassifier(hidden_layer_sizes=(l1,l2) if l2 > 0 else (l1,),
            activation=activation,
            early_stopping=True)
    for l1 in {5000}
    for l2 in {1000}
    for activation in {'tanh'}}
}

classifiers = {
    "KNeighborsClassifier":kneighbors_classifiers,
    "DecisionTreeClassifier":decisiontree_classifiers,
    "MLPClassifier": mlpc_classifiers
}

rkf = RepeatedKFold(n_splits=5, n_repeats=1)

mean_scores = {
    classifier_type: defaultdict(dict)
    for classifier_type in classifiers.keys()
}

std_scores = {
    classifier_type: defaultdict(dict)
    for classifier_type in classifiers.keys()
}

for classifier_type, classifier_instances in classifiers.items():
    for classifier_options, classifier in classifier_instances.items():
        scores = cross_val_score(classifier, X, y, cv=rkf)
        mean_scores[classifier_type][classifier_options] = scores.mean()
        std_scores[classifier_type][classifier_options] = scores.std()

```

Nous pouvons constater que les résultats sont très mauvais, ceci est probablement dû au fait que les features choisies n'aident pas beaucoup à deviner le nombre de partages d'un article.

```
In [37]: plt.rcParams["figure.figsize"] = (10, 27)
```

```
fig, axes = plt.subplots(len(classifiers), 1)
```

```

plt.subplots_adjust(hspace=0.237)
for classifier_index, classifier_type in enumerate(classifiers.keys()):
    scores = list(mean_scores[classifier_type].items())
    scores.sort(key=lambda s: s[1])
    classifier_names, values = zip(*scores)
    stds = [std_scores[classifier_type][classifier_name] for classifier_name in classifier_names]
    indices = list(range(len(scores)))

    ax = axes[classifier_index]
    ax.set_title(classifier_type.replace('_', ' '))
    ax.margins(y=0)
    ax.barh(indices, values, tick_label=classifier_names, xerr=stds)

    for j in indices:
        ax.text(.01, j - 0.2, '%.3f%% ± %.3f%%' % (100. * values[j], 100 * stds[j]))

```

