Assignment 1

March 3, 2018

```
In [61]: from IPython.display import HTML
         HTML('''<style>html, body{overflow-y: visible !important} .CodeMirror{min-
```

```
Out[61]: <IPython.core.display.HTML object>
```

1 Foundations of Data Mining: Assignment 1

Please complete all assignments in this notebook. You should submit this notebook, as well as a PDF version (See File > Download as).

```
In [ ]: # Please fill in your names here
        NAME_STUDENT_1 = ""
        NAME_STUDENT_2 = ""
```

```
In [1]: %matplotlib inline
        from preamble import *
        plt.rcParams['savefig.dpi'] = 100 # This controls the size of your figures
        # Comment out and restart notebook if you only want the last output of each
        InteractiveShell.ast_node_interactivity = "all"
```

1.1 MoneyBall (5 points, 1+2+1+1)

In the early 2000s, 2 baseball scouts completely changed the game of baseball by analysing the available data about baseball players and hiring the best ones. The MoneyBall dataset contains this data (click the link for more details). The goal is to accurately predict the number of 'runs' each player can score.

```
In [55]: moneyball = oml.datasets.get_dataset(41021) # Download MoneyBall data
         # Get the predictors X and the target y
         X, y, attribute_names = moneyball.get_data(target=moneyball.default_target
         # Describe the data with pandas, just to get an overview
         ballframe = pd.DataFrame(X, columns=attribute_names)
         ballframe.describe()
```

```
Out[55]:            Team  League     Year       RA   ...    RankPlayoffs        G  \
         count  1232.00  1232.0  1232.00  1232.00   ...          244.00  1232.00
         mean     15.67     0.5  1988.96   715.08   ...            1.72     3.92
```

```
std       9.72     0.5    14.82    93.08   ...             1.10     0.62
min       0.00     0.0  1962.00   472.00   ...             0.00     0.00
25%       7.00     0.0  1976.75   649.75   ...             1.00     4.00
50%      16.00     0.5  1989.00   709.00   ...             2.00     4.00
75%      23.00     1.0  2002.00   774.25   ...             3.00     4.00
max      38.00     1.0  2012.00  1103.00   ...             4.00     7.00

          OOBP    OSLG
count  420.00  420.00
mean     0.33    0.42
std      0.02    0.03
min      0.29    0.35
25%      0.32    0.40
50%      0.33    0.42
75%      0.34    0.44
max      0.38    0.50

[8 rows x 14 columns]
```

1 . Visually explore the data. Plot the distribution of each feature (e.g. histograms), as well as the target. Visualize the dependency of the target on each feature (use a 2d scatter plot). Is there anything that stands out? Is there something that you think might require special treatment? - Feel free to create additional plots that help you understand the data - Only visualize the data, you don't need to change it (yet)

2 . Compare all linear regression algorithms that we covered in class (Linear Regression, Ridge, Lasso and ElasticNet), as well as kNN. Evaluate using cross-validation and the R^2 score, with the default parameters. Does scaling the data with StandardScaler help? Provide a concise but meaningful interpretation of the results. - Preprocess the data as needed (e.g. are there nominal features that are not ordinal?). If you don't know how to proceed, remove the feature and continue.

3 . Do a default, shuffled train-test split and optimize the linear models for the degree of regularization ($alpha$) and choice of penalty (L1/L2). For Ridge and Lasso, plot a curve showing the effect of the training and test set performance (R^2) while increasing the degree of regularization for different penalties. For ElasticNet, plot a heatmap $alpha \times l1\_ratio \rightarrow R^2$ using test set performance. Report the optimal performance. Again, provide a concise but meaningful interpretation. What does the regularization do? Can you get better results? - Think about how you get the L1/L2 loss. This is not a hyperparameter in regression. - We've seen how to generate such heatmaps in Lecture 3.

4 . Visualize the coefficients of the optimized models. Do they agree on which features are important? Compare the results with the feature importances returned by a RandomForest. Does it agree with the linear models? What would look for when scouting for a baseball player?

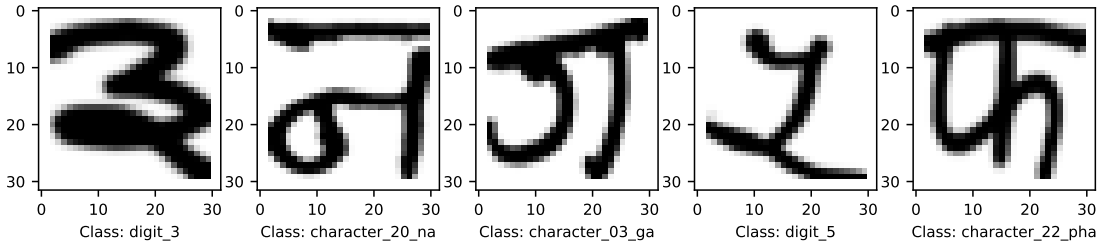
1.2 Nepalese character recognition (5 points, 1+2+2)

The Devnagari-Script dataset contains 92,000 images (32x32 pixels) of 46 characters from Devanagari script. Your goal is to learn to recognize the right letter given the image.

```
In [63]: devnagari = oml.datasets.get_dataset(40923) # Download Devnagari data
         # Get the predictors X and the labels y
```

```
X, y = devnagari.get_data(target=devnagari.default_target_attribute);
classes = devnagari.retrieve_class_labels(target_name='character') # This
```

In [68]:
```
from random import randint
# Take some random examples, reshape to a 32x32 image and plot
fig, axes = plt.subplots(1, 5,  figsize=(10, 5))
for i in range(5):
    n = randint(0,90000)
    axes[i].imshow(X[n].reshape(32, 32), cmap=plt.cm.gray_r)
    axes[i].set_xlabel("Class: %s" % (classes[y[n]]))
plt.show();
```



1. Evaluate k-Nearest Neighbors, Logistic Regression and RandomForests with their default settings.
    - Take a stratified 10% subsample of the data.
    - Use the default train-test split and predictive accuracy. Is predictive accuracy a good scoring measure for this problem?
    - Try to build the same models on increasingly large samples of the dataset (e.g. 10%, 20%,...). Plot the training time and the predictive performance for each. Stop when the training time becomes prohibitively large (this will be different for different models).

2 . Optimize the value for the number of neighbors k (keep $k < 50$) and the number of trees (keep $n\_estimators < 100$) on the stratified 10% subsample. - Use 10-fold crossvalidation and plot k and $n\_estimators$ against the predictive accuracy. Which value of k, $n\_estimators$ should you pick?

3 . For the RandomForest, optimize both $n\_estimators$ and $max\_features$ at the same time on the entire dataset. - Use a nested cross-validation and a random search over the possible values, and measure the accuracy. Explore how fine-grained this random search can be, given your computational resources. What is the optimal performance you find? - Hint: choose a nested cross-validation that is feasible. Don't use too many folds in the outer loop. - Repeat the grid search and visualize the results as a plot (heatmap) $n\_estimators \times max\_features \rightarrow ACC$ with ACC visualized as the color of the data point. Try to make the grid as fine as possible. Interpret the results. Can you explain your observations? What did you learn about tuning RandomForests?

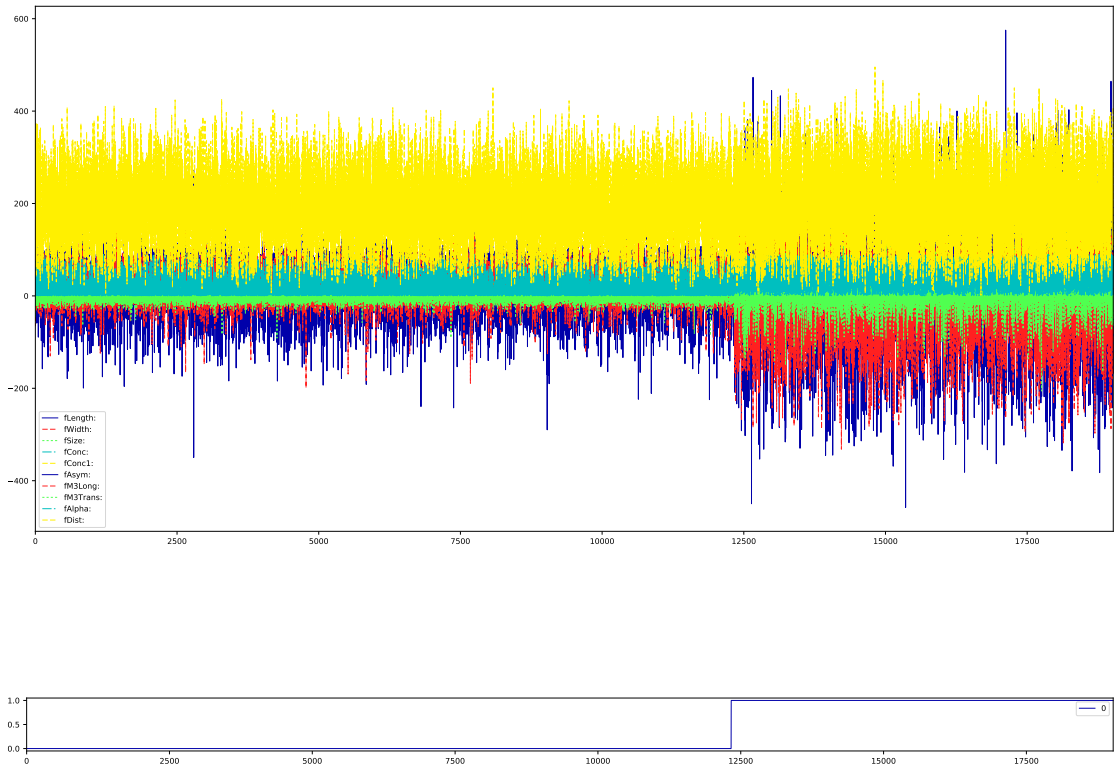
1.3 3. Understanding Ensembles (5 points (3+2))

Do a deeper analysis of how RandomForests and Gradient Boosting reduce their prediction error. We'll use the MAGIC telescope dataset (http://www.openml.org/d/1120). When high-energy

particles hit the atmosphere, they produce chain reactions of other particles called 'showers', and you need to detect whether these are caused by gamma rays or cosmic rays.

```
In [4]: # Get the data
        magic_data = oml.datasets.get_dataset(1120) # Download MAGIC Telescope data
        X, y = magic_data.get_data(target=magic_data.default_target_attribute);

In [6]: # Quick visualization
        X, y, attribute_names = magic_data.get_data(target=magic_data.default_targe
        magic = pd.DataFrame(X, columns=attribute_names)
        magic.plot(figsize=(20,10))
        # Also plot the target: 1 = gamma, 0 = background
        pd.DataFrame(y).plot(figsize=(20,1));
```



1 . Do a bias-variance analysis of both algorithms. For each, vary the number of trees on a log scale from 1 to 1024, and plot the bias error (squared), variance, and total error (in one plot per algorithm). Interpret the results. Which error is highest for small ensembles, and which reduced most by each algorithm as you use a larger ensemble? When are both algorithms under- or over-fitting? Provide a detailed explanation of why random forests and gradient boosting behave this way. - See lecture 3 for an example on how to do the bias-variance decomposition - To save time, you can use a 10% stratified subsample in your initial experiments, but show the plots for the full dataset in your report.

2 . A *validation curve* can help you understand when a model starts under- or overfitting. It plots both training and test set error as you change certain characteristics of your model, e.g. one or

more hyperparameters. Build validation curves for gradient boosting, evaluated using AUROC, by varying the number of iterations between 1 and 500. In addition, use at least two values for the learning rate (e.g. 0.1 and 1), and tree depth (e.g. 1 and 4). This will yield at least 4 curves. Interpret the results and provide a clear explanation for the results. When is the model over- or underfitting? Discuss the effect of the different combinations learning rate and tree depth and provide a clear explanation. - While scikit-learn has a `validation_curve` function, we'll use a modified version (below) that provides a lot more detail and can be used to study more than one hyperparameter. You can use a default train-test split.

```
In [4]: # Plots validation curves for every classifier in clfs.
        # Also indicates the optimal result by a vertical line
        # Uses 1-AUROC, so lower is better
        def validation_curve(clfs, X_test, y_test, X_train, y_train):
            for n,clf in enumerate(clfs):
                test_score = np.empty(len(clf.estimators_))
                train_score = np.empty(len(clf.estimators_))

                for i, pred in enumerate(clf.staged_decision_function(X_test)):
                    test_score[i] = 1-roc_auc_score(y_test, pred)

                for i, pred in enumerate(clf.staged_decision_function(X_train)):
                    train_score[i] = 1-roc_auc_score(y_train, pred)

                best_iter = np.argmin(test_score)
                learn = clf.get_params()['learning_rate']
                depth = clf.get_params()['max_depth']
                test_line = plt.plot(test_score,
                                     label='learn=%.1f depth=%i (%.2f)'%(learn,dept
                                                                         test_score

                colour = test_line[-1].get_color()
                plt.plot(train_score, '--', color=colour)

                plt.xlabel("Number of boosting iterations")
                plt.ylabel("1 - area under ROC")
                plt.axvline(x=best_iter, color=colour)

            plt.legend(loc='best')
```