# Uncertainty Estimate with SVM

***Release 0.0.1***

**Marko Djordjic**

**Dec 08, 2019**

# Contents:

CHAPTER 1

# ucf Module

As noted by Murphy[1] Support Vector Machines (SVMs) are not probabilistic models. This package attempts to provide a solution in order to develop SVM which can output the uncertainty estimate alongside prediction, and therefore, at least to some extent, introduce benefits of probabilistic Machine Learning to otherwise non-probabilistic method.

In this specific case, a classification task has been selected to be solved with a SVM, even though the same methodology can be extended to solving regression problems. For further details regarding the data set describing the classification task available at Torres, Ranasinghe, Sample[2]. One characteristic of this data set deserves to be noted; there is a heavy-imbalance across classes with one of the classes accounting for nearly 90% of the data-set.

The strategy to include uncertainty estimates for SVM is based on the approach defined by Osband, Blundell, Pritzel[3]. In addition to the above mentioned approach which is based on boot-strapping, one more innovation is introduced. Ensemble of SVMs which generates prediction and uncertainty estimates, is trained on a collection of data sets which are constructed in such way that all classes are included in exactly same proportion. Further details regarding the influence this approach has on the decision boundary is explained in subsequent pages.

## Examples

```
>>> import pandas as pd
... import numpy as np
... from sklearn.decomposition import PCA
... from sklearn.svm import SVC
... import uncertainty_estimate_with_svm.ucf as ue_svm
```

```
>>> # Generate balanced set.
... ue_svm.reduce_set_to_equal_distribution_of_classes(
...     features_for_training=x_train,
...      targets_for_training=y_train
... )
```

```
>>> # Fit ensemble.
... ensembles = ue_svm.generate_ensemble(
```

(continues on next page)

[1] Murphy, K. (2012). Machine Learning A Probabilistic Perspective, p. 497

[2] Shinmoto Torres, R. L., Ranasinghe, D. C., Shi, Q., Sample, A. P. (2013, April). Sensor enabled wearable RFID technology for mitigating the risk of falls near beds. In 2013 IEEE International Conference on RFID (pp. 191-198). IEEE.

[3] Osband, I., Blundell, C., Pritzel, A. Van Roy1, B. (2016). Deep Exploration via Bootstrapped DQN.

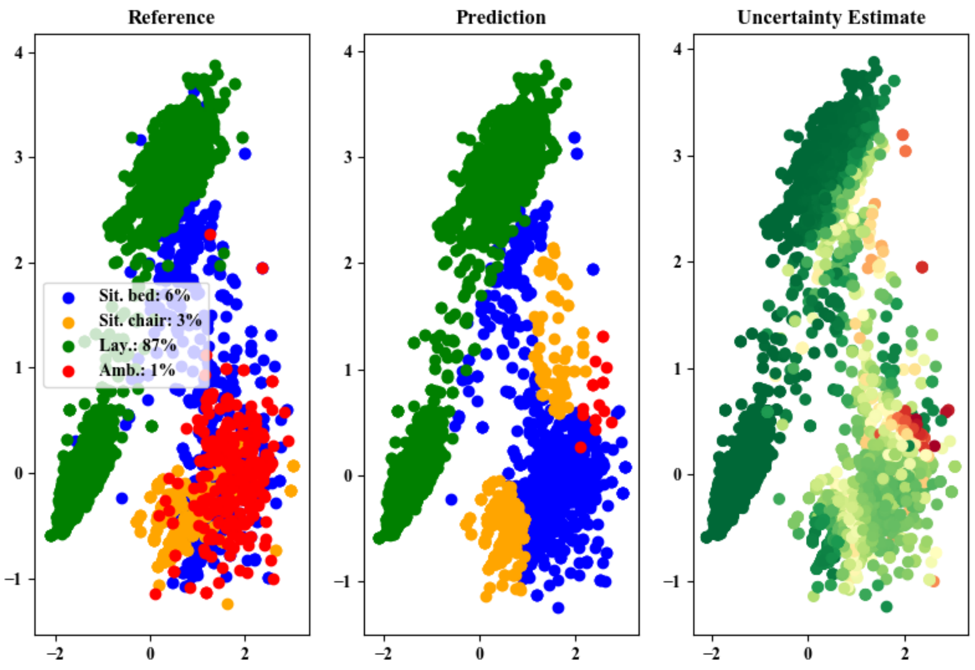
(continued from previous page)

```
...     number_of_estimators=30,
...     features_for_training=balanced_x_train,
...     targets_for_training=balanced_y_train
... )
```

```
>>> # Get ensemble predictions.
... ensemble_predictions, ensemble_uncertainty = ue_svm.generate_predictions(
...     inventory_of_estimators=ensembles,
...     features=x_test
... )
```

```
>>> # Compute ensemble accuracy.
... ensemble_accuracy = (
... np.sum((y_test.astype(int) == ensemble_predictions).astype(int))
...     / len(ensemble_predictions)
... )
... print(ensemble_accuracy)
```

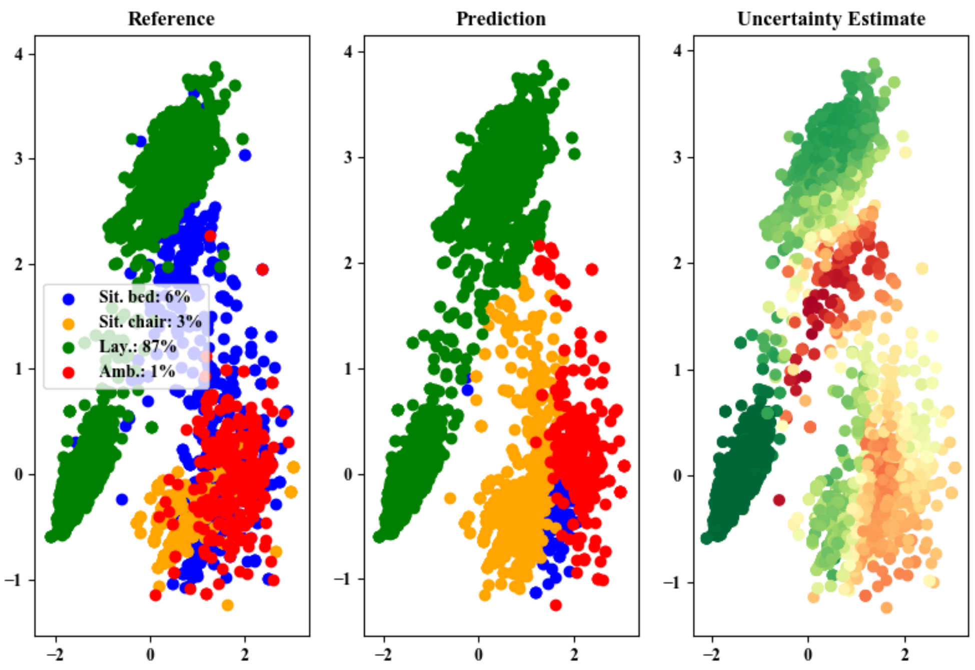
Below we can take a look at the solution produced by the single SVC.



And here is the confusion matrix for the single SVC.

| Reference | Sitting on bed | Sitting on chair | Lying | Ambulating |
|---|---|---|---|---|
| Sitting on bed (n=1028) | 66 | 12 | 1 | 22 |
| Sitting on chair (n=526) | 26 | 69 | 0 | 5 |
| Lying (n=13175) | 1 | 0 | 99 | 0 |
| Ambulating (n=297) | 57 | 0 | 0 | 42 |

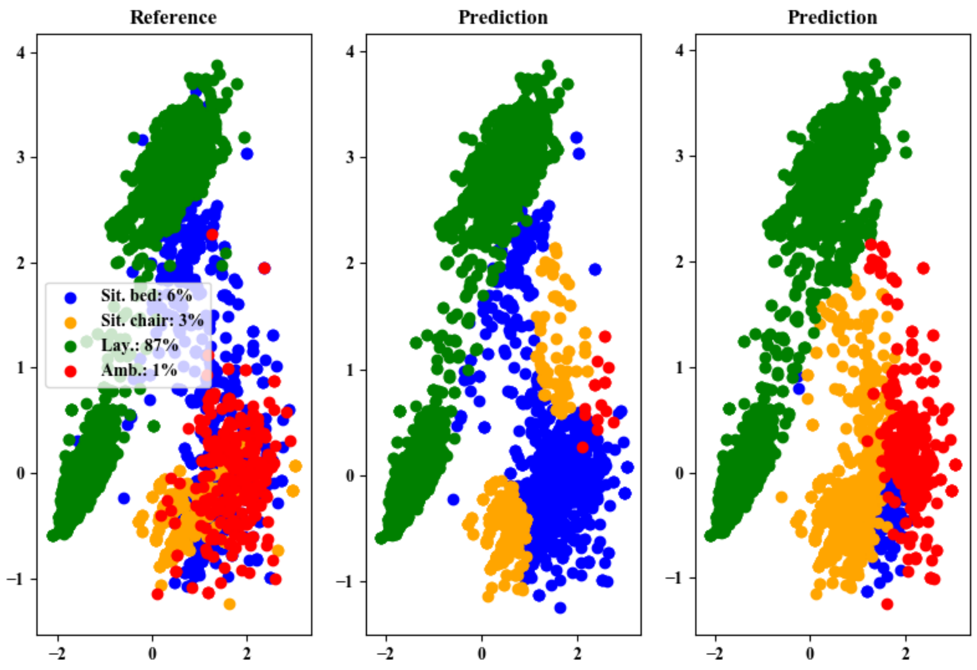
SVC ensemble produces different solution, as we can see from the image below.



Also distribution of classes across the prediction is different.

| Reference | Sitting on bed | Sitting on chair | Lying | Ambulating |
|---|---|---|---|---|
| Sitting on bed (n=1028) | 73 | 3 | 0 | 24 |
| Sitting on chair (n=526) | 44 | 47 | 0 | 8 |
| Lying (n=13175) | 1 | 0 | 99 | 0 |
| Ambulating (n=297) | 50 | 9 | 1 | 41 |

Finally we can take a look a te comparison between a single SVC and an ensemble of SVCs.



There is clearly a different fit, which has been achieved on the basis of training of multiple SVCs on a balanced set, and some of the bias of the model trained on the imbalanced set has be removed. However, even though this allows for training of SVMs on larger training sets, there are two relative problems with this solution: (a) minor but still present loss in accuracy, and (b) longer execution time of the ensemble.

### References

## 1.1 Functions

| | |
|---|---|
| `compute_predictive_entropy`(probability) | Estimate epistemic uncertainty via predictive entropy[1] |
| `generate_ensemble`(number_of_estimators, ...) | Generates a collection of estimators |
| `generate_predictions`(...) | Generate predictions from ensemble |
| `get_all_files_within_folder`(path, ...) | Catalogue all files within a folder according to negative condition |
| `get_all_sub_folders_within_folder`(path) | Catalogue names of all sub-folders within folder |
| `get_data`(folder_wh_data) | Build an inventory of data sets out of individual files within a folder. |
| `make_confusion_matrix`(reference, output, ...) | Compute confusion matrix |
| `plot_comparison`(coordinates, reference, ...) | Plot comparison across different classification solutions. |
| `plot_confusion_matrix`(content[, save_plot, path]) | Plot confusion matrix |
| `plot_individual_classes`(coordinates, ...[, ...]) | Plot class membership in individual plot |
| `plot_solution`(coordinates, original_labels, ...) | Plotting of solution of classification task |

Continued on next page

Table 1 – continued from previous page

| | |
|---|---|
| `reduce_set_to_equal_distribution_of_classes(...)` | Reduces the training set to the size N = K x size of the least frequent class |
| `scatter_plot_with_groups`(coordinates, ...[, ...]) | Produce scatter plot with coloration according to the labels |

### 1.1.1 compute_predictive_entropy

ucf.**compute_predictive_entropy**(*probability*)

Estimate epistemic uncertainty via predictive entropy[1]

**Parameters** **probability** (*numpy.array*) – A numpy.array (N x C) with the probabilities obtained from the underlying classier (soft voting).

**Returns** Uncertainty estimate for each prediction.

**Return type** numpy.array

#### Notes

For the computation of uncertainty value equal to zero are replaced with a small constant near zero.

#### References

### 1.1.2 generate_ensemble

ucf.**generate_ensemble**(*number_of_estimators*, *features_for_training*, *targets_for_training*)

Generates a collection of estimators

Each estimator is trained on a sub-set of the training data, and appended to the ensemble. Support Vector Classifier has been selected as the classifier of choice, but can be replaced with any other classifier.

**Parameters**

- **number_of_estimators** (*int*) – How many estimators will be in the ensemble.
- **features_for_training** (*numpy.array*) – Features which will be utilized for training of individual estimators.
- **targets_for_training** (*numpy.array*) – Targets which will be utilized for training of individual estimators.

**Returns** A collection of SVCs trained on different sections of features and targets pairs.

**Return type** list

#### Notes

The function does not shuffle the data. If shuffling is necessary, it has to be done before call to the function.

### 1.1.3 generate_predictions

ucf.**generate_predictions**(*inventory_of_estimators*, *features*)

Generate predictions from ensemble

The function applies 'predict_proba' method to a collection of estimators, in order to get predictions and compute uncertainty estimate via predictive entropy.

**Parameters**

[1] Further details about about predictive entropy available at: *https://en.wikipedia.org/wiki/Entropy_(information_theory)*

- **`inventory_of_estimators`** (*`list`*) – A collection of estimators placed in a list.
- **`features`** (*`numpy.array`*) – Features on which to perform prediction.

**Returns**

- **ensemble_predictions** (*numpy.array*) – Prediction of class membership.
- **uncertainty_estimate** (*numpy.array*) – Uncertainty estimate.

### 1.1.4 get_all_files_within_folder

`ucf.`**`get_all_files_within_folder`**(*path*, *negative_condition*)

Catalogue all files within a folder according to negative condition

**Parameters**

- **`path`** (*`str`*) – Path to the folder.
- **`negative_condition`** (*`str`*) – Exact text contained with the name of the files, which is utilized to identify files which will not be catalogued.

**Returns** Names of all files within folder except file names designated in *negative_condition* parameter.

**Return type** list

### 1.1.5 get_all_sub_folders_within_folder

`ucf.`**`get_all_sub_folders_within_folder`**(*path*)

Catalogue names of all sub-folders within folder

**Parameters** **`path`** (*`str`*) – Path to the main folder.

**Returns** Names of all sub-folder within folder designated in *path* parameter.

**Return type** list

### 1.1.6 get_data

`ucf.`**`get_data`**(*folder_wh_data*)

Build an inventory of data sets out of individual files within a folder.

**Parameters** **`folder_wh_data`** (*`str`*) – Folder in which data files are residing.

**Returns** All the data sets generated from individual data files.

**Return type** list

#### Notes

Function omits files which are having *txt* inside their name.

### 1.1.7 make_confusion_matrix

`ucf.`**`make_confusion_matrix`**(*reference*, *output*, *prediction_labels*)

Compute confusion matrix

**Parameters**

- **`reference`** (*`numpy.array`*) – A vector with reference.
- **`output`** (*`numpy.array`*) – A vector with targets.

- **prediction_labels** (*list*) – Descriptions of labels.

**Returns** Confuision matrix (see Notes).

**Return type** pandas.DataFrame

### Notes

Reference group is placed in row. Proportion of each prediction within the reference group is computed across columns (horizontally).

## 1.1.8 plot_comparison

ucf.**plot_comparison**(*coordinates*, *reference*, *solutions*, *description*, *coloration*, *save=True*, *path=None*)

Plot comparison across different classification solutions.

**Parameters**

- **coordinates** (*numpy.array*) – Coordinates of the points.
- **reference** (*numpy.array*) – Labels of the reference.
- **solutions** (*list*) – List containing solutions of classification problem.
- **description** (*dict*) – Description of each label.
- **coloration** (*dict*) – Vector with coloration.
- **save** (*bool*) – Option to save the plot.
- **path** (*str*) – Path where to save the plot.

**Returns** No explicit return. Plot is displayed on the screen, or saved into a file.

**Return type** None

## 1.1.9 plot_confusion_matrix

ucf.**plot_confusion_matrix**(*content*, *save_plot=False*, *path=None*)

Plot confusion matrix

Confusion matrix will always be tabulated and plotted. Optionally, picture of confusion matrix can be saved.

**Parameters**

- **content** (*numpy.array*) – Numpy array with the complete content of the confusion matrix.
- **save_plot** (*bool*) – Indication whether to save the plot. Default set to false.
- **path** (*str*) – Path including the file name where to save the plot.

**Returns** No explicit return. Optionally plot can be saved.

**Return type** None

### Notes

Content must consist all numeric content, as well as headings of all rows and columns.

### 1.1.10 plot_individual_classes

ucf.**plot_individual_classes**(*coordinates*, *class_membership*, *description*, *coloration_mode*, *coloration*, *save=False*, *path=None*)

Plot class membership in individual plot

Point within scatter plots indicating class membership with multiple classes can often overlap, therefore debilitating correct analysis. This function plots all classes independently.

**Parameters**

- **coordinates** (*numpy.array*) – Coordinates of points.
- **class_membership** (*numpy.array*) – Indication of class membership.
- **description** (*dict*) – Description of each label.
- **coloration_mode** (*str*) – Indication of the mode of coloration.
- **coloration** (*numpy.array*) – Vector with coloration.
- **save** (*bool*) – Option to save the plot.
- **path** (*str*) – Absolute path towards the file in which to save the plot.

**Returns** No explicit return.

**Return type** None

### 1.1.11 plot_solution

ucf.**plot_solution**(*coordinates*, *original_labels*, *predicted_labels*, *legend_colors*, *legend_descriptions*, *uncertainty*, *save=False*, *path=None*)

Plotting of solution of classification task

Convenience function to plot: (a) original labels, (b) predicted labels, and (c) uncertainty estimate of the model.

**Parameters**

- **coordinates** (*numpy.array*) – Coordinates of labels
- **original_labels** (*numpy.array*) – Reference one-dimensional encoding of the class membership. One-hot encoding is not supported.
- **predicted_labels** (*numpy.array*) – Predicted one-dimensional encoding of the class membership. One-hot encoding is not supported.
- **legend_colors** (*dict*) – Colors to be utilized for coloration of points.
- **legend_descriptions** (*dict*) – Labels to be utilized for description in plot legend.
- **uncertainty** (*numpy.array*) – Uncertainty of the models estimate of class membership.
- **save** (*bool*) – Option to save the plot. Default set to false.
- **path** (*str*) – Absolute path to the file in which to save a plot.

**Returns** No explicit return. Plot is displayed on the screen, or saved into a file.

**Return type** None

### 1.1.12 reduce_set_to_equal_distribution_of_classes

ucf.**reduce_set_to_equal_distribution_of_classes**(*features_for_training*, *targets_for_training*)

Reduces the training set to the size N = K x size of the least frequent class

**Parameters**

- **features_for_training** (*numpy.array*) – Features which will be used for generating reduced sets.
- **targets_for_training** (*numpy.array*) – Targets which will be used for generating reduced sets.

**Returns** Two separate numpy.arrays for features and targets.

**Return type** numpy.array

### 1.1.13 scatter_plot_with_groups

ucf.**scatter_plot_with_groups**(*coordinates*, *labels*, *legend_colors*, *legend_descriptions*, *save_plot=False*, *path=None*)

Produce scatter plot with coloration according to the labels

**Parameters**

- **coordinates** (*numpy.array*) – Coordinates of points.
- **labels** (*numpy.array*) – Vector indicating class membership of each point.
- **legend_colors** (*dict*) – Colors to be utilized for coloration of points.
- **legend_descriptions** (*dict*) – Labels to be utilized for description in plot legend.
- **save_plot** (*bool*) – Indication whether to save a plot. Defaults to none.
- **path** (*str*) – Path including the file name where to save the plot.

**Returns** No explicit return.

**Return type** None

## 1.2 Classes

| | |
|---|---|
| SVC([C, kernel, degree, gamma, coef0, ...]) | C-Support Vector Classification. |
| *TrainingDataSets*(features_and_targets_data_set) | Class for generating training, validation, and testing data sets. |
| product | product(*iterables, repeat=1) –> product object |

### 1.2.1 TrainingDataSets

**class** ucf.**TrainingDataSets**(*features_and_targets_data_set*)

Bases: object

Class for generating training, validation, and testing data sets.

**original_data**

Original features and targets.

**Type** pandas.DataFrame

**indices_of_features**

Numeric indication of position of features in *original_data*

**Type** list

**indices_of_targets**
Numeric indication of position of targets in *original_data*

**Type** list

**train_features**
Unscaled training features.

**Type** numpy.array

**train_targets**
Training targets.

**Type** numpy.array

**validation_features**
Validation features

**Type** numpy.array

**validation_targets**
Validation Targets

**Type** numpy.array

**test_features**
Testing features

**Type** numpy.array

**test_targets**
Testing targets

**Type** numpy.array

**scaled_train_features**
Train features scaled to mean zero and unit variance. Shuffled if desired.

**Type** numpy.array

**scaled_validation_features**
Validation features scaled to mean zero and unit variance.

**Type** numpy.array

**scaled_test_features**
Test features scaled to mean zero and unit variance.

**Type** numpy.array

## Methods Summary

| | |
|---|---|
| *compute_mean_and_standard_deviation*() | Computation of mean and standard deviation of features in the training data set |
| *get_scaled_features*() | Convenience method to return scaled features. |
| *get_targets*() | Convenience method to return targets and features |
| *make_training_data*(train_size, validation_size) | Partition the data into training, validation, and testing sets. |
| *scale_features*() | Standardize features in such manner that their mean is centered to zero, and unit of measurement is set to variance. |
| *shuffle*() | Shuffle scaled features and unscaled targets for training |

### Methods Documentation

**`compute_mean_and_standard_deviation`**()

Computation of mean and standard deviation of features in the training data set

**Returns** No explicit return.

**Return type** None

#### Notes

Values are stored in the 'features_mean' and 'features_standard_deviation' attribute of the class.

**`get_scaled_features`**()

Convenience method to return scaled features.

**Returns**

- **scaled_train_features** (*numpy.array*) – Scaled features for training.
- **scaled_validation_features** (*numpy.array*) – Scaled features for validation.
- **scaled_test_features** (*numpy.array*) – Scaled features for testing.

**`get_targets`**()

Convenience method to return targets and features

**Returns**

- **train_targets** (*numpy.array*) – Targets for training.
- **validation_targets** (*numpy.array*) – Targets for validation.
- **test_targets** (*numpy.array*) – Targets for testing.

**`make_training_data`**(*train_size*, *validation_size*)

Partition the data into training, validation, and testing sets.

**Parameters**

- **`train_size`** (`int`) – Proportion of the training set.
- **`validation_size`** (`int`) – Proportion of the validation set.

#### Notes

Size of the testing set is determined implicitly.

**`scale_features`**()

Standardize features in such manner that their mean is centered to zero, and unit of measurement is set to variance.

**Returns** Standardized features are placed inside appropriate attributes of the class.

**Return type** numpy.array

**`shuffle`**()

Shuffle scaled features and unscaled targets for training

#### Notes

Only scaled training features and targets are shuffled. Validation, and test data sets are not shuffled.

## 1.3 Class Inheritance Diagram

CHAPTER 2

# Indices and tables

- genindex
- modindex
- search

# Python Module Index