Numerical Recipes Report 2
Weather classification using Machine Learning

Simon McLaren 1203258

December 1, 2017

1 Abstract

The purpose of this exercise is to use a Machine Learning algorithm to classify the weather type based on a set of ground observations at a set of weather stations.

2 Introduction

Machine learning techniques are being used in an ever-growing number of fields, including physics and astronomy. Machine learning algorithms are getting more powerful and hence more useful for a wider range of tasks. For example, machine learning techniques can be used to distinguish the signal from background in a physics scenario.

While machine learning covers a large number of different techniques, the primary classifiers (a model used to classify the data) used in this exercise were from the python library sklearn (sci-kit learn).

Machine learning can broadly be divided into 2 categories: supervised learning and unsupervised learning. During supervised learning the algorithm is fed the observed data (for example, the image of a digit) and the corresponding target (the value of the digit). Unsupervised learning is similar, but the algorithm is only fed the observed data but not the target.

This process is known as 'training'. During supervised learning, the algorithm attempts to determine the pattern between the targets and the observed data. During unsupervised learning, however, the algorithm determines the patterns in the observed data alone.

Once the ML (Machine Learning) algorithm has been trained it is tested by unseen data to evaluate its performance.

The ML algorithms in this exercise will be fed weather data, with the corresponding observed weather type, observed from the 24th October to 7th November at MET Office weather stations. The algorithm will be trained on a portion of this data to determine the link between the weather observations and the actual weather and then tested by unseen weather data for evaluation.

3 Algorithms and Methods

3.1 The Weather Data

The weather data was taken from the **UK Met Office DataPoint** service (`https://www.metoffice.gov.uk/datapoint`). Each data point was recorded with the follow-

ing observed features: *Temperature (degrees Celsius)*, *Wind direction (16 point compass)*, *Wind speed (mph)*, *Wind gust (mph)*, *Dew point (degrees Celsius)*, *Screen relative humidity (%)*, *Visibility (m)*, *Pressure (hPa)*, *Pressure tendency (Pa\s)*, *Elevation*, *Latitude*, *Longitude*, *Time since midnight*.

The weather type was recorded for each data point. For the purposes of this exercise, the weather type was split into 2 streams: the *Basic* and *Advanced* stream. The Basic stream consisted of the weather types: *Cloudy*, *Precipitation*, *Clear*. The Advanced stream consisted of: *Clear*, *Partly Cloudy*, *Mist*, *Fog*, *Cloudy*, *Overcast*, *Rain*, *Sleet*, *Hail*, *Snow*, *Thunder*.

In this report, only the Basic stream was used to classify the weather type.

The values for some features had to be changed to a numeric value for input to a classifier. For instance, the 16 point compass values for the wind direction (NNE, NE etc.) were changed to integer values from 0 to 15. The 3 Basic weather types were converted to integers from 0 to 2.

Furthermore, the data for some classifiers had to be standardized for the best results - the data-stream for each feature was scaled to have 0 variance and 0 mean.

3.2 Classifiers used

The following classifiers were used on the weather data:

- The *Decision Tree Classifier* [3] creates a model to predict a target variable by inferring decision rules from the features of the data. See Figure 1 [1] for an illustrative example.

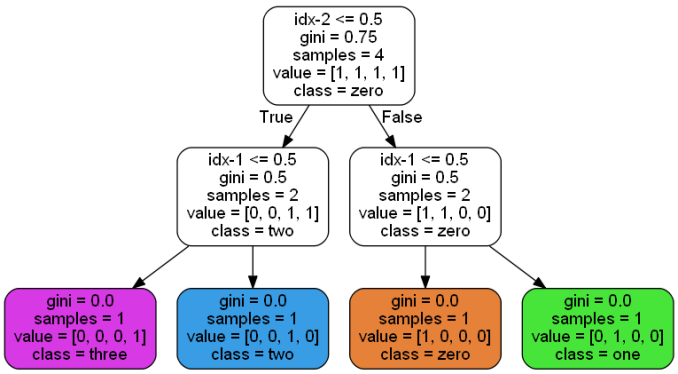

Figure 1: The decision tree classifier

  The decision tree classifier is fast to run and is easy to understand but it's main disadvantage is that there is a danger of overfitting (when the model fits the training data too well).

- The *C-Support Vector Machine* is a type of Support Vector Machine (SVC) Classifier available in sklearn. See [2] for more details. However, the main disadvantage of this classifier is that it performs poorly on large datasets.

- The *Random Forest Classifier* [4] is another type of decision tree classifier, known as an ensembled algorithm. It creates a set of decision trees from randomly selected subsets of the training data. It then aggregates the votes from different decision trees to predict the target variable. In general, it performs better than the decision tree classifier.

- The *Multi-Layer Perceptron Classifier* [5] is a type of neural network. It consists of a network of neurons with weighted connections to other neurons. The main advantage of the MLP classifier is high performance on unseen data - however, it can have large processing requirements for complex networks.

3.3 Optimising the classification process

Each classifier was tested using a *Grid Search* using *Cross-Validation* by the sklearn method *GridSearchCV*.

A grid search is essentially a search over the parameter space of the classifier for the parameters that gives the best score. The parameter space for each classifier was defined manually, such that the grid search did not take an unreasonable length of time, and the scoring method was chosen as $f1\_macro$ in sklearn.

$$f1\_macro = 2\frac{precision * recall}{precision + recall}$$
$$precision = \frac{TP}{TP + FP}$$
$$recall = \frac{TP}{P}$$

where P is the number of positives in the real data, TP is the number of true positives and FP is the number of false positives.

The score is a number from 0 to 1, with 0 being the worst score and 1 the best.

Cross validation (CV) is a method of scoring the data that involves splitting the training data into k smaller folds. Each iteration the classifier is trained on k-1 of the folds, with the final fold set aside for testing and producing a score. The performance of the model reported by CV is then the average of the scores calculated in the loop. The main advantage of scoring using CV is that gives a better indication how well the classifier will generalize to unseen data.

A function was written to perform the grid search for the best parameters for a classifier. The classifier with the best parameters was tested on the test data (which always comprised 20% of the total data) and it's performance evaluated by a confusion matrix, running time (in seconds) and a classification report (which reported on the precision, recall and f1 score of the classifier). The sole scoring method used in all tests was $f1\_macro$, though it is easy to adapt this in the program.

3.4 Feature selection

The correlation matrix in Figure 2 between all features was used to determine some of the feature sets to test.

A correlation matrix identifies the correlation between all pairs of features - the number in each box identifies the correlation between 2 features. 0 indicates the features are not correlated at all; 1 indicates they are perfectly positively correlated and -1 indicates they are perfectly negatively correlated.

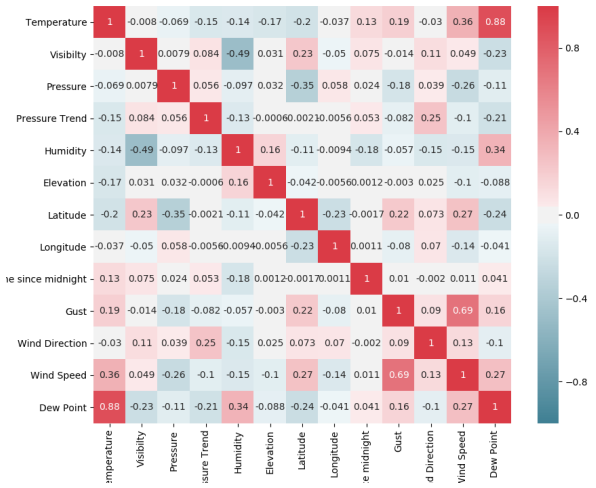

Figure 2: Correlation Matrix of all features

The following feature sets were tested for each classifier.

1. Temperature, Visibility, Pressure, Pressure trend and Humidity.

   The initial feature set given for the exercise.

2. All 13 features except Latitude, Longitude, Elevation and Time since midnight.

   Removing features that are fixed for each station (Longitude, Latitude and Elevation) and features that are cyclic (Time since midnight).

3. All 13 features except Dew point, Gust and Longitude.

   Removing the features Dew Point (highly correlated to Temperature) and Gust (highly correlated to Wind Speed).

4. All 13 features.

4 Results

4.1 Decision Tree Classifier

The parameter space used was:
splitter: ['best', 'random']
max_depth: [5, 10, 15, 20, 25, 50]
min_samples_split: [0.0001, 0.001, 0.01, 0.02, 0.05, 0.1]
min_samples_leaf: [0.0001, 0.001, 0.01, 0.02, 0.05, 0.1]
max_features: [0.25, 0.5, 1.]

Feature Set Used				
Score	1	2	3	4
Precision	0.72	0.73	0.76	0.78
Recall	0.73	0.75	0.77	0.78
f1	0.72	0.73	0.77	0.78
running time	0.0019	0.0014	0.0024	0.0029

The confusion matrix of the best performing classifier is shown below. Each row represents instances in a true weather class, while each column represents instances in a predicted weather class. For instance, the value in the 1st row and 2nd column indicates the classifier predicted 603 data points as Cloudy, whose true value was Clear.

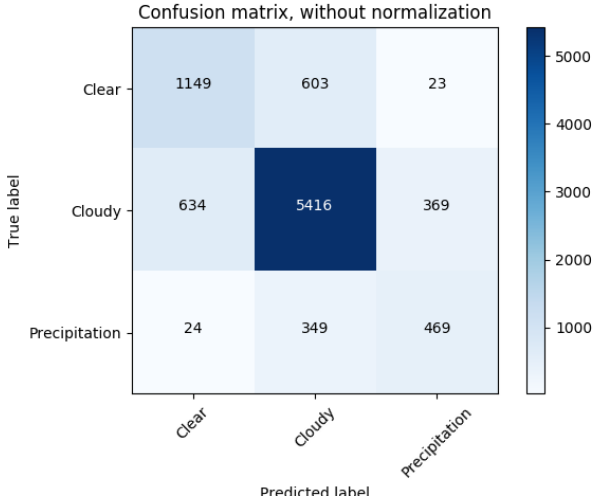

Figure 3: Confusion matrix of best performing feature set

Grid search running time for best performing feature set: 321s.

4.2 Random Forest Classifier

The parameter space used was:
n_estimators: [5, 10, 20, 40]
max_features: ['auto', 'log2', None]
max_depth: [5, 10, 20, 50]
min_samples_leaf: [0.1, 0.2, 0.5, 1]

Feature Set Used				
Score	1	2	3	4
Precision	0.76	0.79	0.83	0.84
Recall	0.78	0.8	0.83	0.84
f1	0.76	0.79	0.83	0.84
running time	0.09	0.09	0.07	0.1

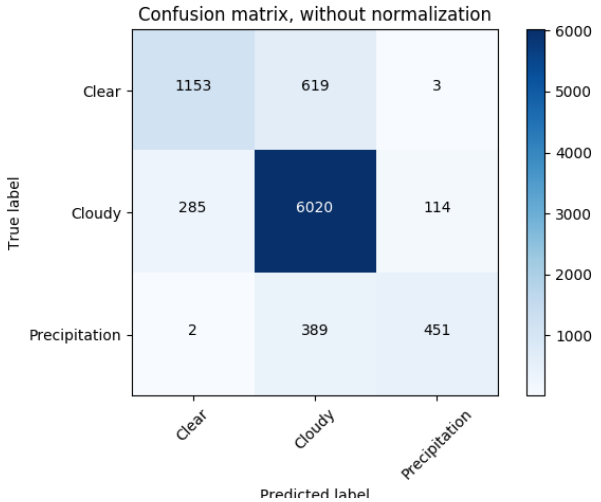

Figure 4: Confusion matrix of best performing feature set

Grid search running time: 465s.

4.3 MLP Classifier

The parameter space used was:
batch_size: ['auto', 50]
activation: ['tanh', 'relu', 'logistic']
learning_rate_init: [0.001, 0.005, 0.01, 0.005]

Feature Set Used				
Score	1	2	3	4
Precision	0.73	0.75	0.78	0.78
Recall	0.75	0.76	0.79	0.79
f1	0.72	0.74	0.78	0.78
running time	0.017	0.03	0.03	0.03

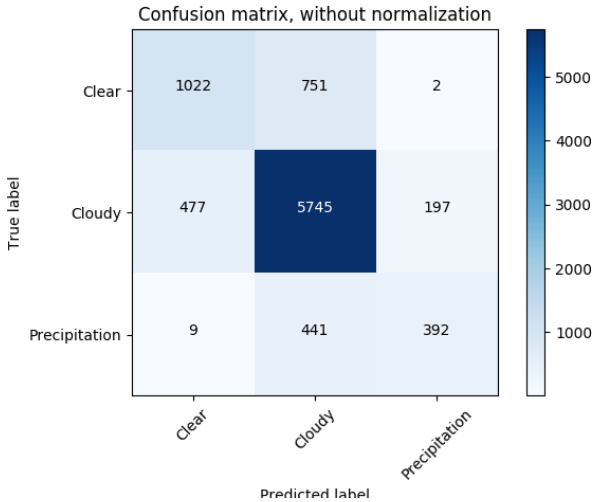

Figure 5: Confusion matrix of best performing feature set

Grid search running time: 2269s.

4.4 C-Support Vector Machine

The parameter space used was:
Only the default parameters for this model.

Feature Set Used				
Score	1	2	3	4
Precision	0.74	0.75	0.77	0.78
Recall	0.75	0.76	0.78	0.79
f1	0.7	0.72	0.75	0.76
running time	3.66	3.97	4.53	6.46

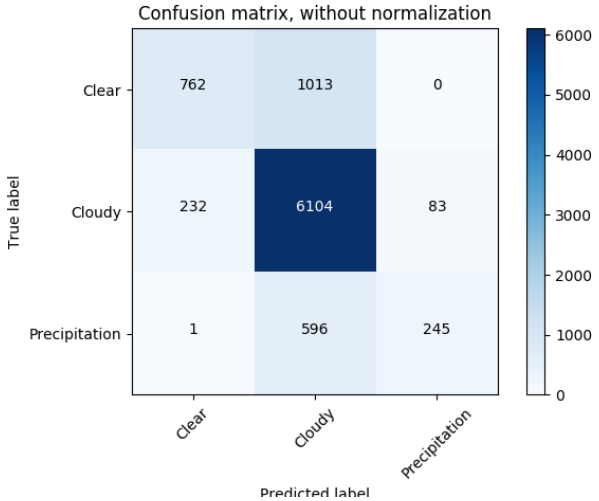

Figure 6: Confusion matrix of best performing feature set

Grid search running time: 244s.

5 Summary and Conclusions

All classifiers improved as more features were added, suggesting that the more features there are the greater the discrimination between the data points - providing more information with which to refine the classifier. Clearly, some features with high correlation would provide only a little new information - which needs to weighed against the longer running time.

By comparing confusion matrices, all models performed well at classifying Cloudy weather correctly. However, the discrepancies came when classifying the other 2 weather types; Precipitation and Clear.

The SVC and Decision Tree classifiers both performed similarly. However, the running time for SVC was far greater - unsurprisingly, as the dataset was very large. The attempted grid search for the SVC classifier was terminated at a running time of over 3 hours. The SVC classifier is therefore unsuitable for this problem.

The Random Forest and MLP classifiers both performed better than the others - identifying more Clear and Precipitation weather types correctly - with the Random

Forest classifier scoring better on the metrics. The grid search for the Random Forest classifier took a shorter time to run than MLP classifier. Therefore, the Random Forest classifier performed the best overall.

5.1 Further work and Improvements

The features used were simply the ones given by the Weather observations available. This could be improved upon by selecting new features that are highly uncorrelated (checked by considering the correlation matrix) to existing ones. This could be approached in 2 ways:

- Manually selecting new features calculated from old based on intuition or research into weather data. An example of this is observations at nearby stations, defined to be within a certain distance of the current one.
- Feature unification functions or methods available either in sklearn or elsewhere. For example: `http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html`.

The grid search could be improved to search over a wider range of parameters. A possible way to do this is a rough search over a wide parameter space first to find a 'first best guess' and then a more refined search around this guess. Another improvement on the grid search could be using multiple/different scoring methods during the search to see if they produce better results.

Finally, a wider search over different testing sample sizes, cross validation folds, classifiers and a wider range of feature combinations is an end goal that would require more work and a lot more processing time. Therefore, it could be done when other options have been explored.

References

[1] Decision Tree image
`https://stackoverflow.com/questions/41207923/scikit-learn-decision-tree-export-graphvi`
Accessed 01/12/2017

[2] sklearn SVC Classifier
`http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC`

[3] sklearn Decision Tree Classifier
`http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

[4] sklearn Random Forest Classifier
`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

[5] sklearn Multi-Laper Perceptron Classifier
`http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`

A Code

A.1 Machine learning program

```
## Import and Data loading

import pickle
import numpy as np
import time
import itertools
import matplotlib.pyplot as plt
import json
# Add SKLEARN modules
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import scale

### Function to plot the confusion matrix as a heatmap ###
# From http://scikit-learn.org/stable/auto_examples/model_selection/
# plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

```
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

### Display weather data ###
## Parameters:
# weather object
def display_weather_data(weather):
    print '#'*50
    print '# Inspect the Data'
    print '#'*50
    print '\n'

    # print data
    print 'Weather Data:'
    print weather.data

    # print number of entries
    print 'Number of entries: %s' % (weather.getNrEntries())

    # print target names
    print 'Number of targets: %s' % (weather.getNrTargets())

    print 'Target names: %s' % (weather.getTargetNames())

    # print features
    print 'Number of features: %s' % (weather.getNrFeatures())

    print 'Feature names: %s' % (weather.getFeatures())

### Grid search over the hyperparameters of the estimator for each scoring method ###
## Parameters:
# feature data, target values,
# list of scoring methods, tuned parameters for classifier,
# num of k-folds, the classifier, testing data size
## Returns:
# best parameters, classification reports,
# confusion matrices (for each scoring method)
# total running time, running times for each 'best model'
def grid_search(data, targets, target_names,
                scores, tuned_parameters, cv, classifier, test_size):
    start_time = time.time() # Set start time for grid search
    x = data
    y = targets
    # Make training and testing data sets
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size,
                                                    random_state=0)
# Array to hold best hyperparameters for each scoring method
best_params = []
# Array to hold classification report for each scoring method
classification_reports = []
# Array to hold confusion matrices for each scoring method
cnf_matrices = []
# Array to hold the running times for the model
# with the best parameters for each scoring method
running_times = []

# Evaluate the best parameters using a grid search for each scoring method
for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print ""

    clf = GridSearchCV(classifier, tuned_parameters, cv=cv,
                       scoring='%s_macro' % score)
    clf.fit(x_train, y_train)

    print("Best parameters set found on development set:")
    print ""
    print(clf.best_params_)
    best_params.append(clf.best_params_)
    print ""
    print("Detailed classification report:")
    print ""
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print ""

    best_model_start_time = time.time()
    y_true, y_pred = y_test, clf.predict(x_test)
    best_model_end_time = time.time()

    print(classification_report(y_true, y_pred, target_names=target_names))
    print ""

    # Evaluation metrics
    classification_reports.append(classification_report(y_true, y_pred,
                                                        target_names=target_names))
    cnf_matrices.append(confusion_matrix(y_true, y_pred))
    # Running time for best model
    running_time_best = best_model_end_time - best_model_start_time
    running_times.append(running_time_best)

end_time = time.time()
running_time_gs = end_time - start_time # Running time for grid search
```

```
    print "Grid search running time = %fs" %(running_time_gs)
    return best_params, classification_reports, \
           cnf_matrices, running_time_gs, running_times

### Write evaluation data to file ###
## Parameters:
# weather object, best parameters, classification reports, total running time,
# running times for each 'best model', output file handle for writing, scoring methods
def write_evaluation_to_file(weather, best_params, classification_reports,
                             running_time_gs, running_times, evalfile, scores):
    evalfile.write('Features used\n\n')
    for w in weather.getFeatures():
        evalfile.write(w)
        evalfile.write('   ')
    evalfile.write('\n\nRunning time for grid search\n')
    evalfile.write(str(running_time_gs))
    for i in range(len(scores)):
        evalfile.write('\n\nScoring method\n')
        evalfile.write(scores[i])
        evalfile.write('\n\nBest Parameters\n')
        evalfile.write(json.dumps(best_params[i]))
        evalfile.write('\n\nClassification report\n')
        evalfile.write(classification_reports[i])
        evalfile.write('\n\nRunning time\n')
        evalfile.write(str(running_times[i]))

### Save confusion matrix image files for each scoring method ###
## Parameters:
# list of confusion matrices, scoring methods, weather object
def produce_confusion_matrices(cnf_matrices, scores, target_names):
    for i in range(len(scores)):
        plt.figure()
        plot_confusion_matrix(cnf_matrices[i], classes=target_names,
                              title='Confusion matrix, without normalization')
        plt.savefig('MLconf_matrix_%s.png' %scores[i])

## Main program
def main():
    # Load the weather data created by FeatureExtraction.py
    weather = pickle.load(open('data/mldata.p'))

    # Define the ML parameters
    scores = ['f1'] # Scoring methods to evaluate (of form 'METHOD_macro')
    cv = 5 # Number of k-folds for cross validation
    test_size = 0.2 # % size of testing data

    # Choose classifier to grid search
    classifier = RandomForestClassifier()
    # Parameter grid for the classifier
```

```
    tuned_parameters_default = {}

    # Scale the data for each individual feature to have 0 variance and 0 mean
    data = scale(weather.data)
    # Extract weather type as a float number for each data point in weather
    targets = weather.target.astype(np.float)
    # List of weather types
    target_names = weather.getTargetNames()

    # Perform the grid searches
    best_params, classification_reports, cnf_matrices, running_time_gs, running_times = \
        grid_search(data, targets, target_names, scores,
                    tuned_parameters_default, cv, classifier, test_size)

    # Write evaluation information to file
    evalfname = "MLeval.txt"
    evalfile = open(evalfname, 'w')
    write_evaluation_to_file(weather, best_params, classification_reports,
                             running_time_gs, running_times, evalfile, scores)
    evalfile.close()

    # Save confusion matrices
    produce_confusion_matrices(cnf_matrices, scores, target_names)

main() # Run program
```

A.2 Correlation matrix program

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

## Main program
def main():
    # Load the weather data you created by FeatureExtraction.py
    weather = pickle.load(open('data/mldata.p'))

    # Define dataframe for weather data
    df = pd.DataFrame(weather.data, columns = weather.getFeatures())

    f, ax = plt.subplots(figsize=(10, 8)) # Define figure

    # Create correlation matrix heatmap for the features in the weather data
    sns.heatmap(df.corr(), mask=np.zeros_like(df.corr(), dtype=np.bool),
                cmap=sns.diverging_palette(220, 10, as_cmap=True),
                square=True, ax=ax, annot=True, vmin=-1.)
```

```
    f.savefig('correlation_matrix.png')

main() # Run program
```