University of Granada

Fitting Lineal Models

Machine Learning. Assignment 3

rafael sanjuan aguilera

March, 2019

Contents

Abstract

As an assignment for our Machine Learning module in Computer Engineering, we study two data sets with the objective of applying both theory and practice learnt in the module. With the imposed restriction of only using linear models, we take a look at a classification and a linear regression problem and discuss data processing, possible hypothesis sets, out of sample performance evaluation methodology and the effects in our results. We implement using ML two models, one of them able to classify hand-written digits with very good accuracy and the other is able to predict the noise of produced by the wind hitting an air vehicle.

1 Classification. Hand-written numbers dataset.

1.1 Understanding the problem

The problem at hand consists in the automatic classification of digits, more precisely in low resolution, square shaped, 2D images.

For this task we are using a notably famous data set called *Optical Recognition of Handwritten Digits Data Set.* This data is free and publicly available at the UCI website. [DG17]

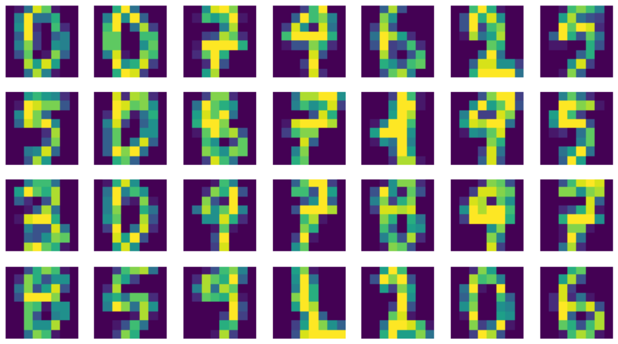

Figure 1: Visualisation of a small subset of trainig data.

Each input data element x from the dataset $\mathcal{D}$ is a 64-dimensional vector. This vectors originate from a reshaped 8x8 matrix where each element is the result of the downsampling of a 32x32 bitmap into non-overlapping blocks of 4x4 and the numbers of pixels are counted for each block. This means each value of the vector is in the range [0, 16], the authors considered this gave invariance to small distortions while reducing dimensionality.

Is note-wordy to point out the data set comes already split in two files for training and testing with ".tra" and "tes" extensions respectably.

1.2 Data preprocessing

As we commented in §1.1 each parameter, block, in a data point x is always in the same [0,16] range, thus, even though normalising the data is a big standard we don't deem it necessary.

Intuition leads us to hipotesize that some blocks of the bitmap will be very underused if relevant at all. Even if we consider the dimensionality of our data handleable with today's tools, we will experiment with dimensionality reduction to see the effects in performance.

For reducing the dimensionality of the input data we have used the technique Principal Component Analysis, PCA, the main reason of his usage is his general good behaviour and popularity of his application for this kind of problems.

PCA aims to find the dimensions of the dataset holding the most relevant information, most variance, and reducing the dimensionality by combining less relevant characteristics.

1.2.1 Processing with PCA

```python
from sklearn.decomposition import PCA
X_pca = np.copy(X_train)

pca = PCA(n_components=0.99)
X_pca = pca.fit_transform(X_pca) # , y_train
print(X_pca.shape)
```

```
>>> (3823, 41)
```

After applying PCA with a very high 99% variance threshold we are surprised to get a heavily reduced dataset, only 41 elements, getting rid of near one third of the pixels in the bitmap.

1.2.2 Visualising variance

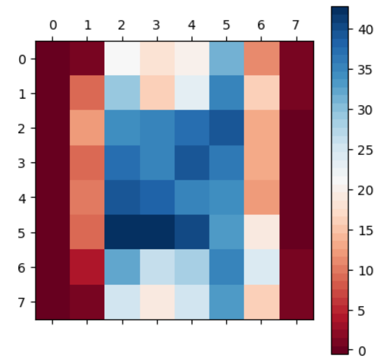

Figure 2: Pixel variance of the training dataset visualised as a heatmap.

Observing the figure 2, we can conclude that both first and last colunms of the dataset 2D images are less relevant data when discriminating the digits. This shouldn't come as any surprise as most people write numbers with bigger height than width, and so this gets reflected in the training data.

1.3 Hypotesis set selection

Although extracting higher abstraction descriptions of our input signal is a very powerful tool for making predictions in images, our input data is way smaller in dimension than most of the images that get handled this way. Many authors like Abu-Mostafa [AMMIL12] have extracted characteristics like symmetry or mean intensity for working with digit classification. Even if this is indeed a good approach, finding this higher level characteristics is not trivial, it's as difficult as finding a good heuristic for the problem at hand.

We have also seen in previous assignments that the aforementioned two characteristics in digit classification can both work wonderfully, for discriminating ones from fives, or terribly, for digits four and eight comparison.

Thus, we set on not making transformations outside of preprocessing in the input data.

1.4 Testing methodology

In order to prevent bias in the training-test split we will use a very common testing split called k-fold cross-validation [M.74]. This will mean splitting the data in k different splits and using k-1 of them for training and the left out fragment for validating the hypothesis obtained with the training data. We will also make sure to keep the ratio of appearance from each class in training and validation.

After fixing our final model adjusting the hyperparameters to get the best performance inside our training data set, we will evaluate our classifier with the previously stored test data split.

1.5 Regularization

We will now talk about applying regularisation to our hypothesis set.

It is our main interest when adjusting any model to get the best performance out of sample, E_{out}, meaning our model generalises well.

In seeking this good generalisation we try to prevent adjusting to the error in our input data, overfitting. Regularising takes part in almost any machine learning problem, is our tool to prevent overfitting by lowering the variance of the model with minimal bias penalisation.

In practice all the linear models used for this experiment have a C parameter in the *sklearn* library set to a default 1.0.

$$C = \frac{1}{\lambda}$$

As his description says in the documentation is the inverse of the regularisation strength. We will set this parameter to soften the adjust of the model to the training data using cross-validation tests in sample as descrived in §1.4

1.6 Models used and hyperparamethers

This classification problem, in contrast with the problems we have faced before, is non binary. Every digit can be binned in one of ten classes, zero to nine.

Powerful tools like Multinomial Logistic Regression and SVM, with a linear kernel, are going to be tested as models for our classification problem. We will measure their E_{val} error as $1 - Acc$ with the discussed metrics and test different regularization strengths.

1.7 Selection and fitting the final model

Classifier	Reg	Mean Acc	*Mean Sen*	*Mean Spe*	*Reg*	*Mean Acc*	*Mean Sen*	*Mean Spe*
LR	0.005	0.957	0.957	0.957	0.5	0.962	0.962	0.962
LR	0.01	0.960	0.960	0.960	1	0.959	0.959	0.959
LR	0.05	0.961	0.961	0.961	5	0.956	0.956	0.956
LR	0.1	0.961	0.961	0.961	10	0.954	0.954	0.954
SVM	0.005	0.946	0.946	0.946	0.5	0.954	0.954	0.954
SVM	0.01	0.945	0.945	0.945	1	0.951	0.951	0.951
SVM	0.05	0.953	0.953	0.953	5	0.961	0.961	0.961
SVM	0.1	0.949	0.949	0.949	10	0.961	0.961	0.961

Table 1: Cross validation mean performance results.

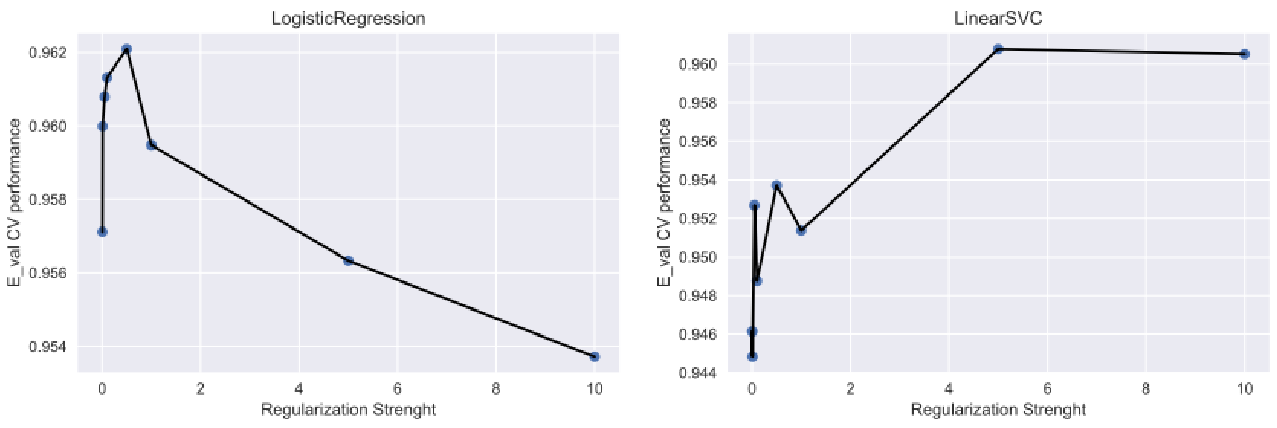

Figure 3: Regularization performance comparison.

We displayed the performance of the CV tests for adjusting models with both LR and linear-SVM in table 1 and as a comparison of Acc in figure 3.

First off, every metric has been displayed up to 3 digit floating point precision but none of them are different until 5 floating point precision, truncating them shouldn't matter to the eyes of our reader, it is clear that these different metrics don't give us distinctive information.

Second important point, we can see that the Linear-SVM performs worse in the CV tests than the Logistic Regression model.

Thus we took the best performing regularization (λ) an decided to fix our final hypothesis as $g = \{LogisticRegression, \lambda = 5\}$

1.8 Performance metrics

Unfortunately this being a multiclass problem the ROC curve becomes a less useful metric due to having to compare multiple graphs, one for each class vs the rest, for every classifier tested.

We opted for evaluating our performance with the *Accuracy* (Acc), *Sensitivity* (Sen) and *Specificity* (Spe) of each model. Sensitivity refers to the rate of correctly classified examples. Specificity refers to the rate of negative examples correctly classified. Accuracy is the total number of correctly classified data divided by the number of examples. This metrics are inspired by the works of [MPA$^+$16] in a recent ML study.

1.9 Out of sample error

As in any machine learning problem, our most important goal is a good out of sample error. In this section we will make use of the previously stored fresh (never seen in our training) data points test set in order to estimate the generalization strength of our model.

```python
# Path to CSV test data file
filepath_test = 'data/optdigits.tes'
# Create numpy arrays with the data
X_test = genfromtxt(filepath_test, delimiter=',')
y_test = X_test[:,-1]
X_test = X_test[:,0:-1]

# Training final model
final_model = LogisticRegression(tol=1e-5, C=(1/5))
final_model.fit(X_train, y_train)
final_model.score(X_train, y_train)
# Testing the Eout
y_pred = final_model.predict(X_test)
acc = final_model.score(X_test, y_test)

print("Eout: %.3f (1-Accuracity)" % (1-acc))
```

```
>>> Eout: 0.052 (1-Accuracity)
```

We have to point out that data preprocessing with PCA has been discarded as it hit performance in the tests by a terrible amount $> 60\%$

1.10 Classification Conclusions

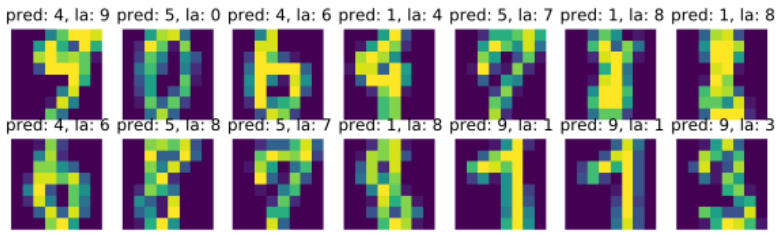

Figure 4: Random subset of mismatched labels

To illustrate the empirical results of our classifier we showed multiple numbers from the tests set, titled with the prediction from our final model and the actual label.

We see our tightly dependant to the pixels model falls short when digits like one get slightly shifted to the right, getting confused with nines in our opinion simply because of the placement.

Seeing this results we think there is definitely a big amount of room for improvement, even if a small subset of the digits, like the top-right corner for example, would have a high error rate in human classification in most of the digits displayed a better classifier could definitely have been selected.

2 Linear Regression: Airfoil Self-Noise Dataset

2.1 Understanding the problem

For our second problem we want to make predictions on the sound pressure levels in dB when the wind hits air blades.

To this objective will take a look at the *Airfoil Self-Noise Data Set.* As the previous dataset this data is free and publicly available in the UCI website. [DG17]

As the data set description states: *This data has been obtained through a series of aerodynamic and acoustic tests of two and three-dimensional airfoil blade sections conducted in an anechoic wind tunnel.*

Six attributed where measured in the dataset experiments: Frequency (Hz), Angle of attack (degrees), Chord length (meters), Free-stream velocity (meters per second) and suction side displacement thickness (meters).

Being able to reduce the noise output off air planes by making predictions on how this variables affect the noise would have direct applications for reducing the noise pollution of the air traffic.

With respect to the data set we have a total number of 1503 measurements of these variables in the wind tunnel changing one or multiple variables. Before taking a look at the data we will store a subset for validating our model after fixing our hypothesis:

```python
# Path to CSV training data file
filepath = 'data/airfoil_self_noise.dat'

# Load everything
X = genfromtxt(filepath, delimiter='\t')
# Take the last column as our label data.
y = X[:,-1]
# Remove the column from the set of other characteristics
X = X[:,0:-1]

# Split the data in train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

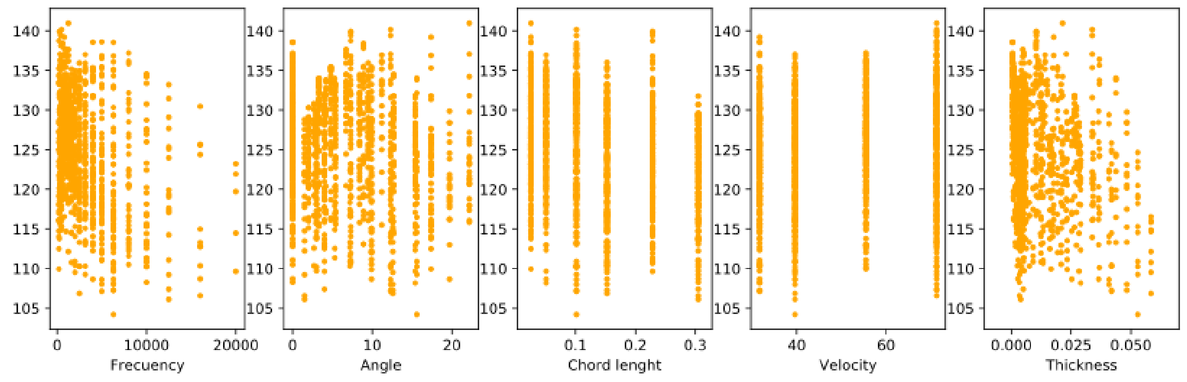

Figure 5: Visualisation of every column related to our target (y)

Now we can take a look at our training split. Plotting every other measured characteristic in relation with our target doesn't reveal much information, there are not apparent relations easily visible to our eye.

2.2 Data preprocessing

As we described in §2.1, the data set was divided in a 80/20% split for training and testing respectively.

As the values on the characteristics of the input data come in different ranges, we will *normalise* the training set. Although we won't process the test data initially we will store the min and max values of the training data normalisation to later on, after our hypothesis has been fixed, apply this range of values to the test set too.

```
def normalise(X):
    for i in range(0, X.shape[1]):
        column = X[:, i]
        min_val = min(column)
        max_val = max(column)

        for j in range(0, X.shape[0]):
            if (max_val - min_val) > 0:
                X[j, i] = (X[j, i] - min_val) / (max_val - min_val)

    return X, min_val, max_val
```

Even if we only have five dimensions to our input data, after applying normalisation we will run PCA to get an idea if all of this variables are critical to our target predictions on noise levels.

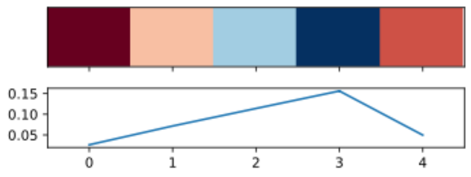

Figure 6: Heatmap of variance.

```python
from sklearn.decomposition import PCA
X_pca = np.copy(X_train)

pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_pca)
print(X_pca.shape)
print(pca.explained_variance_ratio_)
```

Output:

```
>>> (1202, 4)
>>> [0.38172358 0.36972943 0.16828515 0.05374699]
```

The PCA analysis reveals that four out of the five variables amount for 97% of the variance in the data. Even after considering this results, for the moment we will discard the PCA reduction of dimensionality due to the dimension of the data set being small, only five variables.

2.3 Hypotesis selection

For selecting our hypothesis we will proceed with a Structural Risk Minimisation (SRM) framework in order to get a good fit in our training data minimising the risk of overfitting and being penalised with bad generalisation.

2.4 Testing methodology

As in the previous problem discussed in section §1 we will use k-folds cross-validation for testing our performance in sample and once our final hypothesis is fixed we will check his generalisation performance with our previously stored validation set (§2.1).

```python
from sklearn.model_selection import cross_validate
score_metrics = ('neg_mean_squared_error', 'r2')
rid = linear_model.Ridge(alpha=0.01, copy_X=True) # tol=1e-5,
scores_rid = cross_validate(rid, X_train, y_train, cv=5,
                                scoring=score_metrics, return_train_score=False)
```

Code: Example of testing in sample with sklearn

2.5 Regularization

As discussed in the previous problem §1.5 regularization is essential for preventing overfitting and getting a good generalization of our model. To this objective, we will use different regularization strengths in our models testing.

2.6 Models used and hyperparamethers

As this is a regression problem where we have to make real values predictions we will fit two extensions of Linear Regression, a Ridge regression (also know as weight decay) model and a Lasso regression model, both are extensions of Linear Regression adding a regularisation factor, we also tested a Support Vector Regression (SVR) model using a linear kernel.

2.7 Selection and fitting the model

We performed in sample cross-validation tests with different regularization strengths (Reg), the three lineal models and progressively increasing the hypothesis set($\mathcal{H}$) adding non-linear transformations of our data, polinomials to the power of *'Degree'*.

Classifier	*Degree*	*Reg*	MSE	Classifier	Degree	Reg	*MSE*
Ridge	1	0.1	23.570	Ridge	1	0.5	23.576
Lasso	1	0.1	25.014	Lasso	1	0.5	47.840
SVR	1	0.1	23.946	SVR	1	0.5	23.624
Ridge	2	0.1	22.269	Ridge	2	0.5	22.504
Lasso	2	0.1	25.106	Lasso	2	0.5	47.840
SVR	2	0.1	22.816	SVR	2	0.5	23.479
Ridge	3	0.1	21.918	Ridge	3	0.5	22.264
Lasso	3	0.1	25.121	Lasso	3	0.5	47.840
SVR	3	0.1	22.679	SVR	3	0.5	23.095
Ridge	4	0.1	21.763	Ridge	4	0.5	22.110
Lasso	4	0.1	25.121	Lasso	4	0.5	47.840
SVR	4	0.1	22.619	SVR	4	0.5	22.859

Classifier	*Degree*	*Reg*	MSE	Classifier	Degree	Reg	*MSE*
Ridge	1	0.005	23.570	Ridge	1	0.01	23.570
Lasso	1	0.005	23.574	Lasso	1	0.01	23.585
SVR	1	0.005	24.185	SVR	1	0.01	24.178
Ridge	2	0.005	22.261	Ridge	2	0.01	22.260
Lasso	2	0.005	22.282	Lasso	2	0.01	22.452
SVR	2	0.005	23.301	SVR	2	0.01	23.225
Ridge	3	0.005	21.750	Ridge	3	0.01	**21.748**
Lasso	3	0.005	22.251	Lasso	3	0.01	22.403
SVR	3	0.005	23.339	SVR	3	0.01	23.221
Ridge	4	0.005	21.735	Ridge	4	0.01	21.724
Lasso	4	0.005	21.994	Lasso	4	0.01	22.437
SVR	4	0.005	23.229	SVR	4	0.01	23.170

After this results we set our hypothesis as $g = \{Ridge, d_{vc} = 15, \lambda = 0.01\}$. Although we gained a bit of performance by adding the transformations up to fourth degree, 0.024 MSE, in our opinion this increased fit is not enough to justify the added cost.

2.8 Performance metrics

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2. \tag{1}$$

The metric used for comparing our results is the mean squared error (MSE) as defined by equation 1. This metric has been selected for his simplicity and popularity. It's not a perfect metric by any means because the presence of big outlayers has a great effect on it and could result bad fit but we decided in his use.

2.9 Out of sample error

After fixing our final hyphotesis is time to evaluate our results with our test data. We applied the same preprocessing for our test set as our training data and tested our hypothesis with never seen data to finally evaluate the error out of sample.

```python
# Processing train data, already normalised
X_train_augmented = augmente_degree(X_train, 3)

# Procesing test data
X_test = normalise_range(X_test, min_training_val, max_training_val)
X_test_augmented = augmente_degree(X_test, 3)

# Training final model
rid = linear_model.Ridge(alpha=0.01, copy_X=True)
```

```python
rid.fit(X_train_augmented, y_train)

# Testint the Eout
y_pred = rid.predict(X_test_augmented)
mse = mean_squared_error(y_test, y_pred)
```

```
>>> Eout: 22.551 MSE
```

2.10 Results

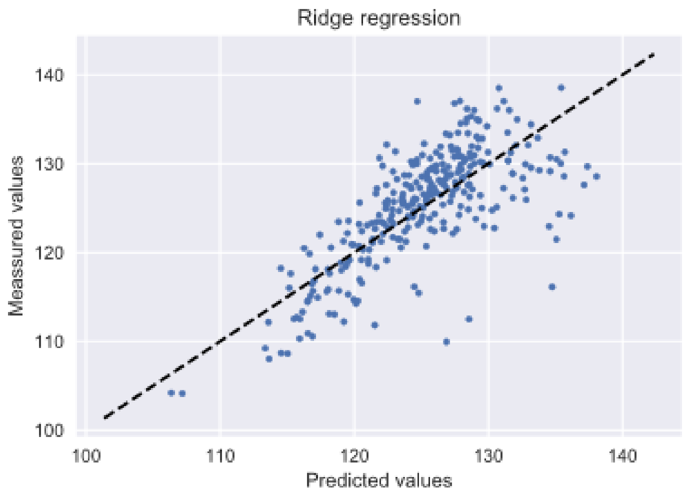

Figure 7: Comparison predicted vs real values

We can say we got very good results within the limitations of the experiment obviously we were limited to lineal models and I'm our opinion they don't seen enough to reflect the complexity of the problem.

We trained a model that generalized decently enough out of sample considering the error in sample wasn't as low as we would have liked it to be.

In figure 7 we can check the relationship between our predictions and the test data split true values. We do observe a tendency and we could say we did learn something.

Through good methodology we obtained a model that can generalize and even if the predictions are not rock solid it can give us a good idea of what the noise levels could be.

References

[AMMIL12] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012.

[DG17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[M.74] Stone M. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111, 1974.

[MPA$^+$16] Pablo Mesejo, Daniel Pizarro, Armand Abergel, Olivier Rouquette, Sylvain Beorchia, Laurent Poincloux, and Adrien Bartoli. Computer-Aided Classification of Gastrointestinal Lesions in Regular Colonoscopy. *IEEE Transactions on Medical Imaging*, 35(9):2051 – 2063, September 2016.