StatML Exam
In a Galaxy Far, Far Away

Arni Asgeirsson lwf986

03/04-2014

Contents

Introduction

- What is this?
  - What are my main focuses in this report?
  - What is the structure of my hand in?
  - What is the structure of this report?
  - My general assumptions
  * That the reader is familiar with basic machine learning concepts (and statistical stuff) and that the reader is familiar with the given assignment text and the given datasets.
  - Have I deviated from the assignment text? If yes, how and why?

  - The math notation I use ..

I focus on discussion my results rather on the deep theory behind my methods, as 10 pages are limited.

As I have used the *sci-kit learn*[1] library extensively through our my code for my machine learning needs, the produced code is rather short and somewhat uninterestingly, and will therefore not dwell into my specific implementations in this report, but rather focus the limited number of pages to briefly discuss the theory behind the used methods and mainly focus on a discussion on my results.

I the reader is interested in the details of my implementations I refer to the code files, I have tried to comment my code to the degree that someone with minor python skills, and a decent understanding of machine learning and linear algebra can understand the code.

1 Predicting the Specific Star Formation Rate

In the following to exercises we will look at linear and non-linear regression for predicting the specific star formation rate (sSFR) for a given galaxy. The file *src/question1.py* shows my own implementation of linear regression. Although even though it is not *sci-kit learn* the implementation is still rather simple and I will not give a code explanation here.[2] In question 2 I have used the *RandomForestRegressor*[3] from *sci-kit learn*.

1.1 Question 1 (linear regression)

Linear regression is when we want to find the weight vector $\bar{w} = (w_0, w_1..., w_M)$ that minimizes $\sum_{n=1}^{N}(y(x_n, \bar{w}) - t_n)^2$, i.e. we want find the line that minimizes the sum of errors, which are the squared distances from our predicted values to the expected values.

When performing linear regression we rewrite y to be $y(\bar{x}, \bar{w}) = \sum_{j=0}^{M} w_j \phi_j(\bar{x})$, where ϕ is some basis function. As we are doing linear regression we define $\phi_0(x) = 1$ and $\phi_i(x) = x$, where $i \neq 0$.

As our data is four dimensional our final regression model will be of the form:

$$\begin{aligned} y(\bar{x}, \bar{w}) &= w_0 + w_1\phi_1(x_1) + w_2\phi_2(x_2) + w_3\phi_3(x_3) + w_4\phi_4(x_4) \\ y(\bar{x}, \bar{w}) &= w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \end{aligned}$$

[1] http://scikit-learn.org/stable/

[2] If the reader were to look through my code, the reader will notice that I actually do use *sci-kit learn* to perform linear regression along side my own implementation. Although this is just to double check that my implementation is correct.

[3] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Now we can move on to find our weight vector $\bar{w} = (w_0\, w_1\, w_2\, w_3\, w_4)^T$. We do that by using the derived formula $\bar{w} = (\Phi^T\Phi)^{-1}\Phi^T\bar{t}$, which is rather simple as when using the linear basis functions, Φ becomes very simple.

When doing the calculations I get that the weight vector is $\bar{w} = \begin{pmatrix} -8.149433493650 \\ -0.794001531121 \\ -1.222959203710 \\ -0.328584747643 \\ -0.786330558664 \end{pmatrix}$, this gives us the final linear regression model

$$y(\bar{x}) = -8.14943349365 - 0.794001531121x_1 - 1.22295920371x_2 - 0.328584747643x_3 - 0.786330558664x_4$$

Which we now can use to predict the specific star formation rate of a new unknown galaxy image.

This model has a mean square error of 0.274754600685 on the training data set and a mean square error of 0.275179630629 on the test data set.

1.2 Question 2 (non-linear regression)

When one is imagining a linear line in some space and then 5000 data points, one quickly starts to wonder what the chances are that the data really is best fit with a linear model. This is where non-linear regression comes in, we suspect that our data is not linear and therefore try to fit a non-linear regression model to our data.

Results

When performing the regression I get a mean-squared error of 0.0429865757597% on the training data and a mean-squared error of 0.0898122295689% on the test data.

The fact that both the training error and test error are that much less than what we saw in 1.1 when doing linear regression could hint at two things. Either my model has overfitted the data or a non-linear regression model is a more suitable choice for the sSFR data (or a mix of the two). Considering that both the training error and the test error are greatly reduced and the training error is not very close to 0 are venture myself out on a limb and conclude that my non-linear regression model has only overfitted the data a little, if at all, and instead conclude that a non-linear regression model fits the sSFR data better.

2 Stars vs. Galaxies

Now we move on to classification where we want to assign a label l from a finite set of labels L to each data point in the function range, as opposed to regression which we did in the previous assignment. We are given a train and test data set both containing 3000 points either belonging to the star class 0 or the galaxy class 1. Initially we will look at binary classification of the data following by principal component analysis and clustering.

In the following three exercises/questions I have used the *svm*[4], *PCA*[5] and *KMeans*[6] modules from *sci-kit learn* for my machine learning needs.

2.1 Question 3 (binary classification using support vector machines)

Why do we use machine learning for such a task? Well we are able to extract a lot of information from the different kind of images taken of the space objects, al though when they get further and further away, it becomes very difficult to distinguish between the objects when looking at simple features. Instead we can use machine learning, and especially classification, to try and find patterns in the data and a suitable space where the data is linear separable.

I have used the support vector machines (SVM) to try and perform the binary classification of the stars & galaxy data. The specified kernels to be used are the radial Gaussian kernels of the form:

$$k(\bar{x}, \bar{z}) = e^{(-\gamma||\bar{x}-\bar{z}||^2)}$$

As I have used the *svm* module from the *sci-kit learn* library, this kernel is the default one and goes by the keyword *'rbf'*.

Procedure

There are several hyperparameters one could delve in to although we will concentrate on the regularization constant C and the hyperparameter γ for our kernel. It can be very hard to find the best hyperparameters to a given model, especially when the range of possible good hyperparameters and the number of parameters increases, as it simply gets very computationally expensive, and SVMs are already very computationally expensive.

To train my classifier I have used grid search and 5-fold cross validation to find the best values for C and γ, and I have used Jaakkola's heuristic to determine the initial value for γ. Then I train the final C-SVM classifier with the found values for C and γ and use the final classifier to predict the labels for the train and test data. I must also mention that the hole procedure is done on normalized data.

As the assignment text is very clear on how to compute the $\sigma_{Jaakkola}$ and $\gamma_{Jaakkola}$ values, I will refer to my code *src/module/common.py* if the reader is interested in how I coded the heuristic. With no further due my computed values are:

$$\sigma_{Jaakkola} = 0.841689072242$$
$$\gamma_{Jaakkola} = 0.705775578896$$

My initial C value is 10 and using the given combination sets gives me the possible hyperparameters:

$$Cs := \{0.01, 0.1, 1, 10, 100, 1000\}$$

$Gs := \{0.0007057755788959511, 0.007057755788959511, 0.07057755788959511, 0.7057755788959511, 7.057755788$

[4] http://scikit-learn.org/stable/modules/svm.html
[5] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
[6] http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Results

Now after using the above initial C and γ values in a grid search and 5-fold cross validation I find that the optimal combination of hyperparameters from the given sets above are $C = 100$ and $\gamma = 0.00705775578896$.

Now by using these newly found values I train my classifier *clf* with the optimal hyperparameters and is ready to try it out.

Using *clf* to classify the test data set I get an accuracy of 99.5% on the training data and an accuracy of 99.5333% on the test data.

The very high accuracy of the training data does bother and concern me a little bit, as it unfortunately is a very good indication of overfitting, but then again the accuracy of the test data is also very very high, and somehow soothes my nerves, but not quite enough. If the test data is very close in space to the training data, then the accuracy of the test data is not as soothing and my classifier *clf* might be a victim of overfitting, but it is hard to tell without any more data. It might be that a more optimal combination of hyperparameters would be a set that increases the accuracy of the test data, and possible lowers the accuracy of the training data.

The mean square error of the training data is 0.005 and the mean square error of the test data is 0.00466666666667.

As we can see SVMs for binary classification can be pretty good, but it comes at the cost of computation time and expensive hyperparameter selection, some of the other methods that could have been used for the is the linear discriminant analysis (LDA) technique or even the perceptron method, although that would require that the data is linear separable which I highly doubt they are in our case, but then we could use kernels with our perceptron to project the data into a space where they indeed are linear separable. I have a hard time expecting those two methods to perform better then the trained SVM (unless the SVM is way overfitted).

2.2 Question 4 (principal component analysis)

Now we will try to use principal component analysis (PCA) on the galaxy training data from the *SGTrain2014.dt* training set.

One way to describe PCA is to say that it is a technique to try and find the meaning of the madness by sorting the variance in the data by size. A little more technically description is to say that PCA uses the eigenvalues and eigenvectors of the data to transform the data so that the greatest eigenvector is aligned with the first axis of the coordinate system, and that the n'th greatest eigenvector is aligned with the n'th axis, hence the PCA can project the data in dimensions equal to or lower than the input dimension. This also means that PCA can only be performed once one a given data set, as doing it multiply times will not result in any new changes as the data is already 'perfectly' aligned with the axis.

Why would one want to use PCA? As hinted above PCA can be a very good tool to extract the most important features of some dataset, to get rid of some noise parameters or visualize some high dimensional data, in a humanly perceptually way. Which is exactly what we want to do in this exercise for the given galaxy data.

Results

Figure 1 shows a plot of the eigenspectrum (the green and squared line) which describes the size of the eigenvalues.

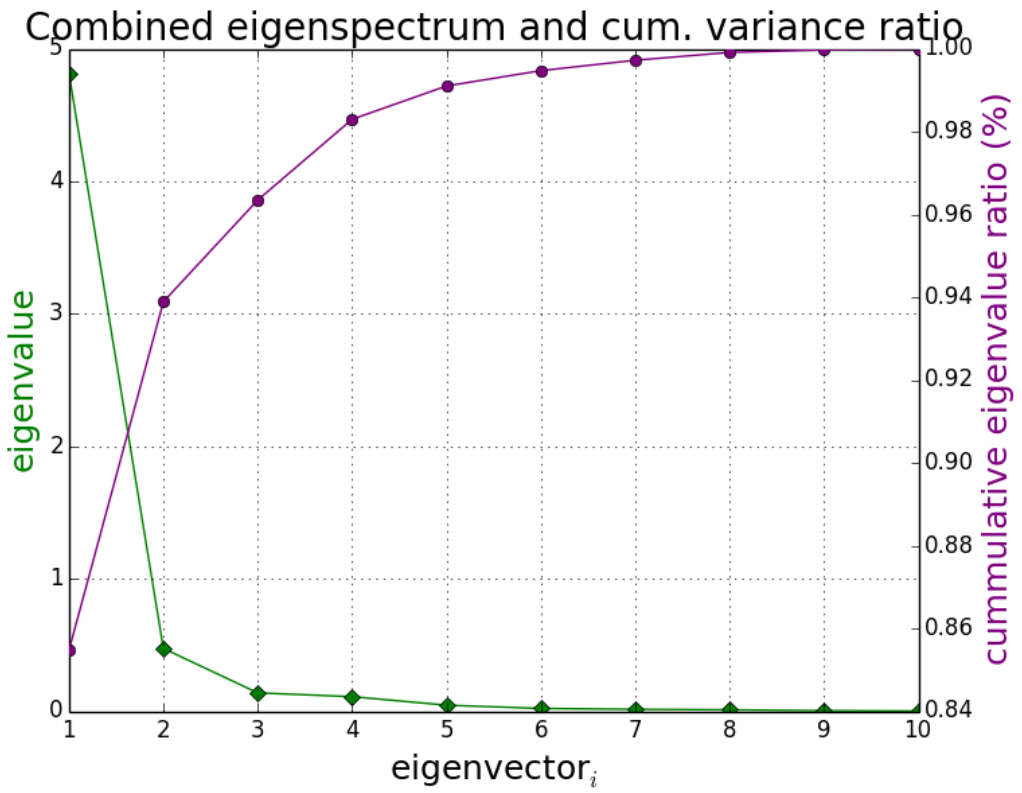

Figure 1:

Figure 1 also shows a plot of how much the eigenvalues/eigenvectors cumulatively cover the variance (the purple and circled line). We can see that by only looking at the first principle component we have already covered about 85% the data. This can be useful to determine how many principal components one should look at when doing dimensionality reduction, as based on the figure one might argue that the added computation time is not worth the last 2%, when going from 4 to 10 principle components, or if one like to live dangerously one could deem those last percentage to be noise. We can see how a diminishing return effect occurs when increasing the number of principal components to the increase (or the loss of decrease) in data size and computation time.

Figure 2 is a great example of using PCA to do dimensionality reduction allowing for the ability to visualize a high dimensionality data set. The input data is reduced to the first two principal components i.e. reduced to 2 dimensions. The plot also shows the eigenvectors (multiplied by the root of their respective eigenvalue) which shows that they are beautifully 'sorted' and aligned with the two axes.

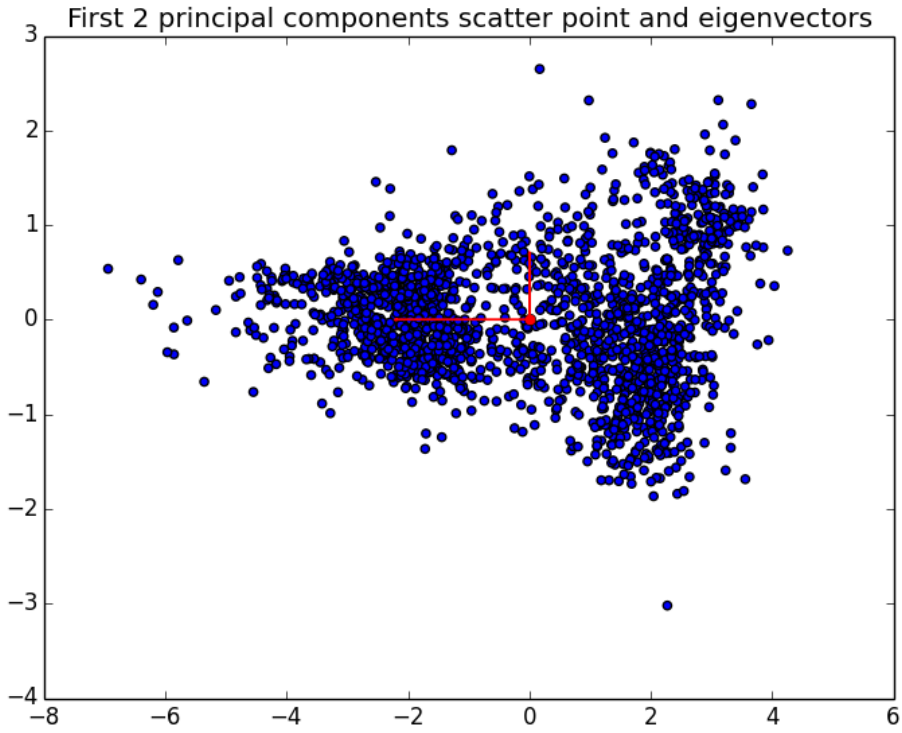

Figure 2:

2.3 Question 5 (clustering)

Now we move on to k-mean clustering, or just clustering by short, which is good tool to extract even more meaning from a data set. Clustering works by finding the n points which describes the data as good as possible. This is done by iteratively assigning each point in the input space to the cluster point which is the closest, then update the cluster points to the mean of the respectively generated cluster group, where the initial n cluster points are selected at random or heuristically.

The *KMeans* module from *sci-kit learn* uses Lloyd's algorithm, which as they say it might be fast but risks falling into local minima. This is because it picks the initial k center points at random. I try to make up for this by making sure that the algorithm is run many times.

Results

When performing the 2-mean clustering on the galaxy data we are left with the following two cluster center points.

$$[-0.166893, -0.826119, -1.210908, -1.397877, -1.512199, 0.607583, -0.106192, -0.484871, -0.666617, -0.819270]$$
$$[1.273834, 0.832120, 0.168408, -0.232916, -0.403030, 1.928551, 1.285293, 0.605494, 0.213195, -0.008197]$$

Although it is very hard to image where these cluster points are located in respect with the given data set. We can therefore use our PCA solution from before and project the two 10-dimensional center points onto the first two principal components and get the following two 2-dimensional center points.

$$[-2.12378504, 0.03487029]$$
$$[1.82679699, -0.02999406]$$

These have been plotted in figure 3 along with the scatter point from figure 2.

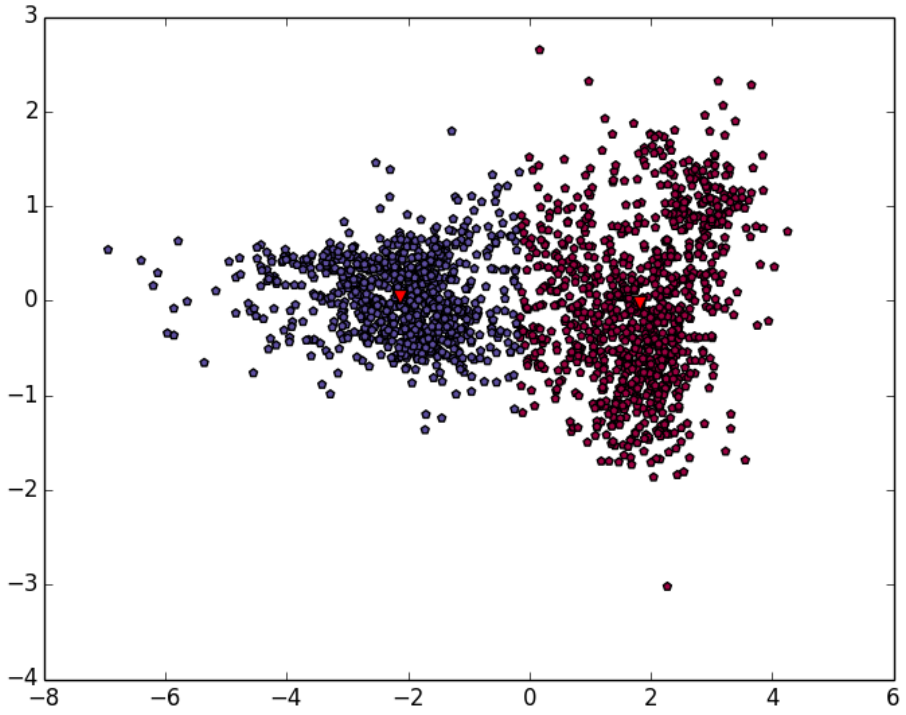

Figure 3:

Of course k-mean clustering will *always* find k clusters even though not one of them might make sense in any useable way, which is in a sense one of the pitfalls of the k-mean clustering. When looking at the scatter point in figure 2 it is relatively easy to spot where the two center points would be positioned at, because we can see the data visualized, although for not PCA how could one imagine the data in higher dimensions of 3? Making it impossible to guess easily how many clusters a data is clustered in. Even with PCA we are limited to the 1-3 dimensional space, which is most likely not enough to visualize all clusters in input spaces that are much larger.

2.4 Question 6 (kernel mean classifier)

We have some input data X which is separated in a set of classes C where X_c denotes the subset of X which is labeled with c where $c \in C$.

$h(x)$ then loops through C and returns the label of class c where the distances is the smallest between x and $\mu(X_c)$.

$$h(x) = argmin_{c \in C} \|x - \mu(X_c)\|$$

Now we want to try and redefine $h(x)$ to use kernel functions. So the goal is to rewrite $h(x)$ to only use the dot product operation when calculating the distance from x to $\mu(X_c)$ and when calculating the mean of X_c.

We start by inserting a dummy root and square

$$h(x) = argmin_{c \in C} \sqrt{\|x - \mu(X_c)\|^2}$$

This allows use to rewrite the square product

$$h(x) = argmin_{c \in C} \sqrt{\langle x, x \rangle - 2 \langle x, \mu(X_c) \rangle + \langle \mu(X_c), \mu(X_c) \rangle}$$

Were $\langle x, y \rangle$ denotes the dot product of x and y. Now we will try to redefine μ, which we can write as:

$$\mu(X_c) = \frac{1}{N_c} \sum_{x \in X_c} x, \text{ where } N_c \text{denotes the number of elements in } X_c$$

If we replace μ with the above notation then we get

$$h(x) = argmin_{c \in C} \sqrt{\langle x, x \rangle - 2 \left\langle x, \frac{1}{N_c} \sum_{x' \in X_c} x' \right\rangle + \left\langle \frac{1}{N_c} \sum_{x'' \in X_c} x'', \frac{1}{N_c} \sum_{x''' \in X_c} x''' \right\rangle}$$

Now we can use the associative and distributive law of dot products to extract those summations of out the dot products

$$\begin{aligned}
h(x) &= argmin_{c \in C} \sqrt{\langle x, x \rangle - 2 \frac{1}{N_c} \sum_{x' \in X_c} \langle x, x' \rangle + \frac{1}{N_c} \sum_{x'' \in X_c} \left\langle x'', \frac{1}{N_c} \sum_{x''' \in X_c} x''' \right\rangle} \\
&= argmin_{c \in C} \sqrt{\langle x, x \rangle - 2 \frac{1}{N_c} \sum_{x' \in X_c} \langle x, x' \rangle + \frac{1}{N_c} \sum_{x'' \in X_c} \frac{1}{N_c} \sum_{x''' \in X_c} \langle x'', x''' \rangle} \\
&= argmin_{c \in C} \sqrt{\langle x, x \rangle - \frac{2}{N_c} \sum_{x' \in X_c} \langle x, x' \rangle + \frac{1}{N_c^2} \sum_{x'' \in X_c} \sum_{x''' \in X_c} \langle x'', x''' \rangle}
\end{aligned}$$

We are given a positive definite kernel function $k : X \times X \to \mathbb{R}$ and we assume that there exists a Hilbert feature space $\mathcal{H}$ and a feature map $\Phi : X \to \mathcal{H}$ which allows use to define the kernel function k as

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

Which allows us to use the kernel definition to rewrite our current formula

$$\begin{aligned} h(x) &= argmin_{c\in C}\sqrt{\langle\Phi(x),\Phi(x)\rangle - \frac{2}{N_c}\sum_{x'\in X_c}\langle\Phi(x),\Phi(x')\rangle + \frac{1}{N_c^2}\sum_{x''\in X_c}\sum_{x'''\in X_c}\langle\Phi(x''),\Phi(x''')\rangle} \\ &= argmin_{c\in C}\sqrt{k(x,x) - \frac{2}{N_c}\sum_{x'\in X_c}k(x,x') + \frac{1}{N_c^2}\sum_{x''\in X_c}\sum_{x'''\in X_c}k(x'',x''')} \end{aligned}$$

This gives us the final formula

$$h(x) = argmin_{c\in C}\sqrt{k(x,x) - \frac{2}{N_c}\sum_{x'\in X_c}k(x,x') + \frac{1}{N_c^2}\sum_{x''\in X_c}\sum_{x'''\in X_c}k(x'',x''')}$$

Where we have redefined the nearest mean classifier to only depend on the value of the kernel function k evaluated on pairs of data points.

One could argue that the square-root operation is redundant as it does not affect the relationship between the found values. One could even argue that the first component $k(x,x)$ is redundant as well and be removing those we arrive at the final formula:

$$h(x) = argmin_{c\in C} - \frac{2}{N_c}\sum_{x'\in X_c}k(x,x') + \frac{1}{N_c^2}\sum_{x''\in X_c}\sum_{x'''\in X_c}k(x'',x''')$$

Note that I did notice some other ways to define the final formula although I like this one as, even though the last two summations might be ugly, they can be easily precomputed and hence somewhat minimize the computation time of the expression.

3 Variable Stars

In the following I will try and perform multi-class classification on the given Variable Stars data set

3.1 Question 7 (multi-class classification)

I played around with many different options and trained several different models before landing on my final two; linear discriminant analysis[7] (LDA) and random forest[8] for multi classification of the Variable Stars data set.

Note that the data is very scarce and some of the classes have very few data points assigned to them. This might affect the solutions greatly as the small classes might be forgotten.

Linear multi classification

The reason of choice lies in the fact that we had to implement LDA in one of the mandatory assignments and I found it quite fun to implement and work with and found LDA to be one of the ML methods that was somewhat easy to grasp and play with. The LDA build in the mandatory assignment left me an impression that LDA works rather well generally and as it does not have to any hyperparameters to worry about it is a very simple model and easy to work with.

[7] http://scikit-learn.org/stable/modules/generated/sklearn.lda.LDA.html
[8] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

The only thing that bothers me about LDA is that it assumes that all the class covariances are identical, as based on my current ML experience this only happens in custom designed data sets.

When using LDA to classify the Variable Stars data set, the model is fit with the training data, I get an accuracy of 81.5823605707% on the training data and an accuracy of 71.3359273671% on the test data.

Non-linear multi classification

I find the simplicity and easy-achievable efficiency of random forests to be intriguing. Random forests makes few assumptions on the data opposed to many other classifiers and ... and it was one of the classifiers I tried out that gave the best results on the test set (human overfitting much?).

As random forests has *a lot* of hyperparameters I of course used the standard grid search and cross validation techniques to try and optimize these. Some of the main components to decide upon is the number of trees B where i tried different numbers between $1 - 1000$ and I determine the number of max. features m to be $\left\lfloor \sqrt{D} \right\rfloor$ where D is the dimension of the data. As I do not yet have that much experience with random forests I am a bit short on good values for the other parameters, but I have just tried to play around with them.

When using the build classifier I achieve an accuracy of 98.4435797665% on the training data and an accuracy of 77.5616083009% on the test data set.

3.2 Question 8 (overfitting)

3.2.1 Traditional overfitting

Considering that there are so few data points but so many features and classes in the Variable Star data set, one might be concerned of traditional overfitting as there might simply not be enough data points to create a generally well performing classifier.

This affects the classification in such a way that it has trouble finding general meaning of the data and needs to learn the training data more by heart to be able to get reasonable results in the training accuracy. Which is all we have, the training data, so it is the only set of data that we can use to decide wether or not our solution is good enough, and hence we strive to get a good result before stopping, but a good result (say 90% acc.) might have crossed the overfitting boundary without us noticing.

If one wants to maintain the complex classifier then one ought to get more training data.

3.2.2 Parameter tweak overfitting

This has not been such a big concern when fitting my classifiers in question 7. Yet, it is still a very big concern in machine learning in general, as once you move away from simple classifier such as KNN which only have a very few and conceptually simple hyperparameters, and move on to more advanced and complex classifiers, or regression models for that matter, you tend to get an increase in the number of hyperparameters, and tweaking these just right, without a tendency towards overfitting is very hard. When dealing with many hyperparameters it is often hard to distinguish between their meaning when comparing their values with the results of the

build classifier and hence also hard to figure out when the chosen values favor the training data too much, hence overfitting.

3.2.3 Data set selection

This will quickly result in overfitting for many of the reasons described above in 3.2.1. Seeing that we don't have many that training points and it is somewhat hard to get very good results with the created classifiers, one might be very tempted to focus on the training error and try to maximize this and therefore possibly overfit the classifier. One of the golden rules we have learned in this course is that going for 0 empirical risk is not the way to go. It is very easy to get 100% accuracy on the training data, just use 1-nearest neighbor and you are (*almost*) guaranteed a perfect score on your training data. Then you have learned your training data by heart, and at the same time overfitted your classifier.