CS425 Project 5(Report)
Support Vector Classification

Ksenia Burova

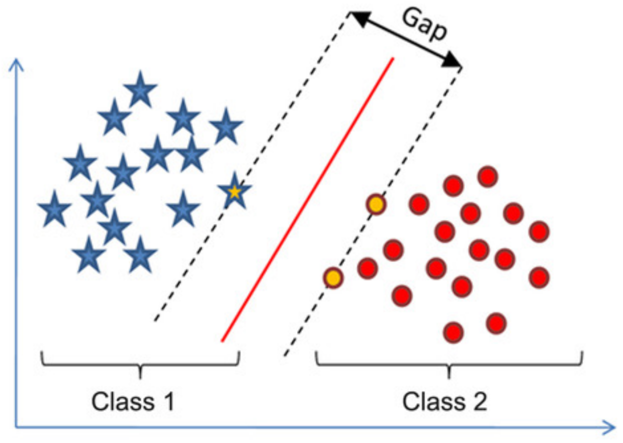
December 8th, 2017

Abstract: In this project, the goal was to experiment with support vector classification (SVC) algorithm. We had to apply this algorithm to three different classification problems: binary classification and two multiclass problems. The data files for each problem were provided. With all given problems, we were to experiment with coarse grid search and with fine grid search for finding best range and optimal hyperparameters. Results for each experiment had to be reported and shown in plots.

1. Support Vector Machine and Classification.

Assume we have data points that are labeled with two different classes. Imagine that we can draw a line between datapoints of first and second class. Therefore, our classes are linearly separable. An example is shown in **Figure 1**.

Figure 1: Linearly separable classes



Obviously, we can draw different lines. But the question is which one is the best. Definitely we would like all datapoints for each class to be from the same side of the line, so we would like our lines to be as much parallel as possible to data planes.

Second question, among all parallel lines, which one is the best. So lines that are too close to our data points have this feature where you believing data that you train too much. And you don't want to believe your data too much. That is called overfitting. At the same time, the middle line is consistent to the data while committing the least to it. In our example in **Figure 1**, that would be the red line, which is also called **an optimal hyperplane**.

Therefore, finding the line of least commitment in linearly separable set of data is a basis behind **support vector machines**.

To talk a little bit more about hyperplane, we can describe it in the following equation:

$$y = w^T x + b,$$

where y value indicates classification label, w's are parameters of the plane, so is b which moves it in and out of the origin. This is how linear classifiers look like even in multiple dimensions with hyperplanes. The decision boundary which is the red line in our example again may not give us positive or negative values for classification by the definition, so no matter what are b and w values are, it is:

$$w^T x + b = 0.$$

For other lines, y will be described by the offset from the middle line. Two possible results could be:

$$w^T x + b = 1, w^T x + b = -1,$$

for lines closest to each class.

We want our decision boundary, or optimal hyperplane, to be as far away from both classes as possible, that means we want to maximize the gap between two lines closest to our classes. Those are the dashed lines in our **Figure 1**. That gap is called **maximum margin**, and lines closest to our classes are **support vectors**. The optimization problem for finding maximum margins is **quadratic programming**. Also linear problem doesn't always seem to be enough, for example, when data isn't really linearly separable, it can be projected to higher dimensional space to be compared there. The change to algorithm is that dot product $x^T y$ is substituted with similarity metric $K(x, y)$ called **Kernel function** that also represents our domain knowledge. The procedure itself is called **Kernel trick**.

2. Tools and Program.

I've chosen to use *python* programming language for my experiments. Since we didn't have to implement our own algorithms, I used *SVC* and *GridSearchCV* modules from *scikit-learn* package. From the same package, I used *StandardScaler* to standardize my data as was suggested in the project write-up, and I also used *train_test_split* module to split my data into testing, validation and training sets. I used *MatPlotLib* package to built plots as in previous projects.

3. Experiments.

In this project, we were to experiment with 3 different data sets of different types and sizes. For all of them we had to perform *coarse grid search* for finding the best range of hyperparameters, then a *fine grid search* for finding the optimal hyperparameters for each problem.

There different kernels that can be used in SVC:

- Linear
- Polynomial
- Radial Basis Function (RBF), defines a spherical kernel
- Sigmoid, has same shape with sigmoid

There are also two parameters used with most of these: *C* and *Gamma*. Linear model doesn't use Gamma parameter.

- Gamma can be also called as the 'spread' of the kernel, or **kernel scale**. It is a decision region. If Gamma is low then the curve of the decision boundary is very low and therefore the decision region is too broad. From other side, when Gamma value is high, the curve is high, and we receive islands of decision-boundaries around datapoints.

- C parameter is the **penalty** for misclassifying datapoint. When it's low, we have high bias and low variance,so classification isn't penalized. But when C value is high, bias and variance are opposite, low and high respectively, therefore the classifier gets penalized for misclassified data. When it happens it bends over backwards to avoid misclassified datapoints.

These are the values for C and Gamma that I used in my experiments for coarse grid search, along with Kernels specified above:

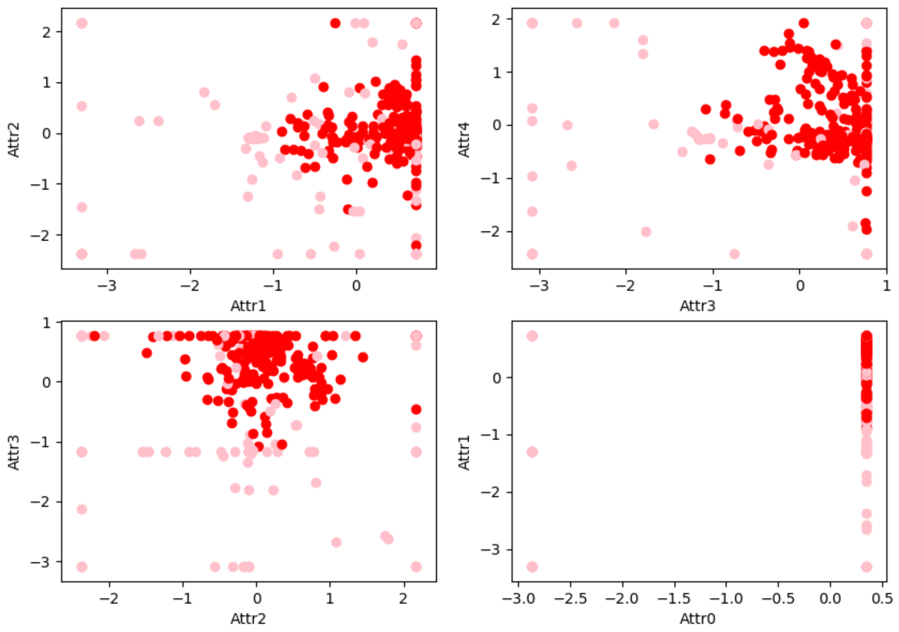
```
C:     [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
Gamma: [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
```

3.1 Ionosphere Dataset

In this experiment, the task is to complete binary classification of interactions of radar signals with electrons in the ionosphere. There were 34 measurements in data file along with labels 'g' or 'b' (good or bad). First, I noticed that there was one column with all zeros and I decided to remove it from the data set to reduce computation overhead. I also replaced 'g' and 'b' with '1' and '0' respectively for easier parsing.

We were not given names for attributes, so I tried to see if there is some possible dependency between several pairs of attributes. Clearly, we can't see any linear separability between chosen attributes as shown in **Figure 2**. But that's normal because attributes probably depend on each other. We still can see that most red datapoints are sort of clustered together, and most pink ones also separated from the red area.

Figure 2: Attributes pairs (2 classes)



After running Coarse Grid Search with chosen parameters and kernels, my best parameters happened to be:

```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

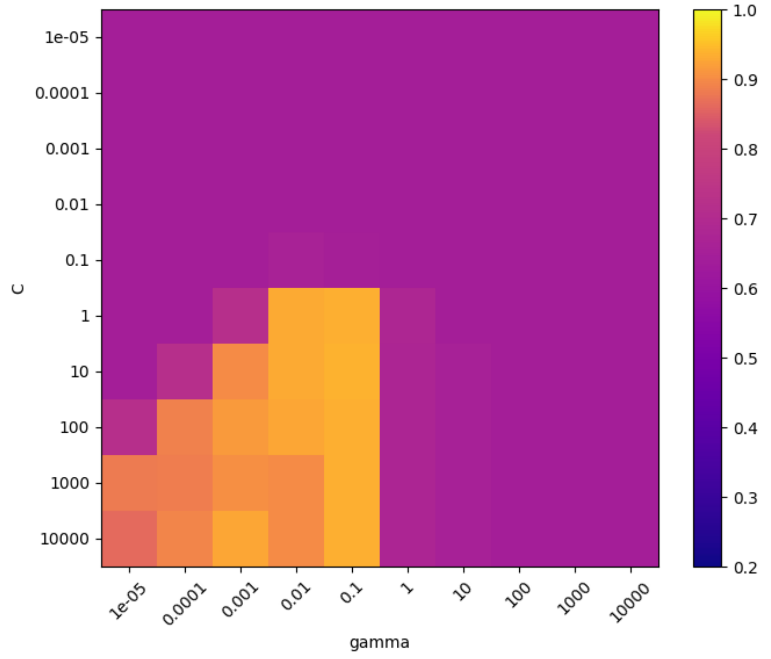
After testing our data on validation/testing set we get a following results table:

```
             precision    recall  f1-score   support

        0.0       0.89      0.97      0.93        34
        1.0       0.98      0.93      0.95        54

avg / total       0.95      0.94      0.94        88
```

Precision is a percentage of correctly predicted instances vs. total number of instances, so technically it speaks for accuracy. And in our case we received 95% accuracy average for both labels. *Recall* is sensitivity which basically means that in 93% we identify good interaction of radar signals with electrons being good, and with 97% we mark bad ones as bad. Overall we also have 94%. *f1-score* is a combination of two metrics above, it is basically a harmonic mean of both, so no wonder why it is 94% as well. *Support* is just number of occurrences for each label, from which we can tell why label '1' was identified better. Possibly in training set we also had more instances of positive labels and so it was trained more and better.

In **Figure 3** we can observe how different combinations of C and gamma parameters performed on current dataset during coarse grid search. We can see that no matter what value we used we never performed worse than 60%. We definitely can tell that gamma = 0.1 is the most optimal since our performance is almost identical there for all C values starting with 1. Technically, with lower gamma values the decision boundary should be less curvy and should cover spread of the data less. That what we observe in that lower left quarter of the plot. With lower gamma values datapoints are more orange. With higher values of C decision boundary becomes less biased and has more variance, and so we can start seeing signs of overfitting. Like for gamma = 0.01, with increasing C values performance becomes worse. We probably overfit there.

Figure 3: Accuracy (Coarse Grid Search)



After the coarse grid search I ran a fine grid search with C values ranging from 2.5 to 50 and with gamma values ranging from 0.025 to 0.5. The result was this:

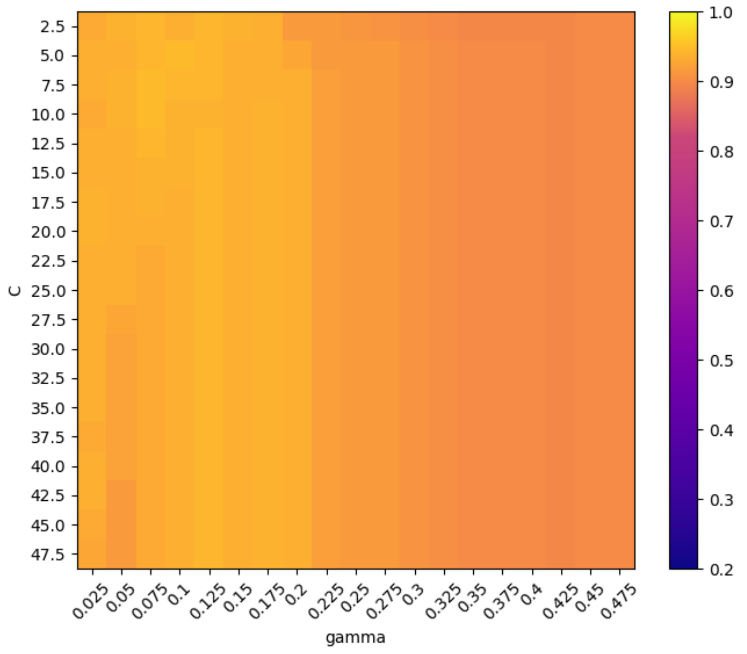
```
{'C': 5.0, 'gamma': 0.1, 'kernel': 'rbf'}
```

The prediction performance of the testing/validation dataset looked as following:

```
             precision    recall  f1-score   support

        0.0       0.92      0.97      0.94        34
        1.0       0.98      0.94      0.96        54

avg / total       0.96      0.95      0.95        88
```

It is slightly better than in previous results. Our optimal gamma value remained unchanged whereas optimal C value was changed to 5. The overall results for all possible combinations are displayed in **Figure 4**.

Figure 4: Accuracy (Fine Grid Search)



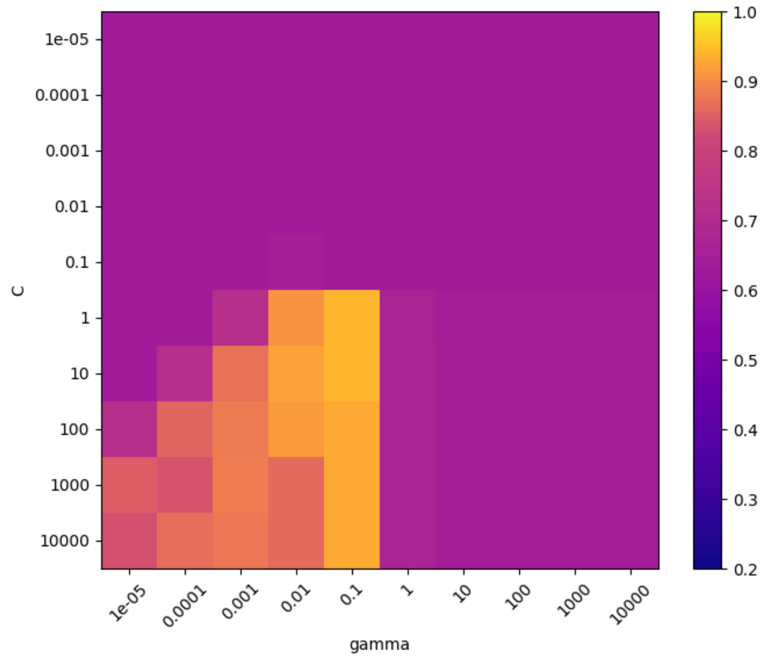
As we can see, in the best range for both hyperparameters, all combinations work pretty well. The lightest column overall in that colormap is probably with gamma = 0.125, although cell with C = 5 and gamma = 0.1 is still lighter. All values near gamma = 0.1 look great, increasing gamma up to 0.4-0.5 start showing signs of overfitting.

In some instances, I was getting C = 1 as the best parameter. The results for those runs were:

```
             precision    recall  f1-score   support

        0.0       0.97      0.89      0.93        36
        1.0       0.93      0.98      0.95        52

avg / total       0.94      0.94      0.94        88
```

It is possible because of data being shuffled differently. Also it is possible because in previous results performance for C=1 and C=10 looks very very alike. Overall performance for this shuffle is worse. *Accuracy* average is lower by 2%. **Figure 5** also shows that performance differs for this shuffle in lower left quarter of the plot. It still reflects best gamma/C parameters combinations there though.

Figure 5: Accuracy (Coarse Grid Search)



The fine grid search result for this shuffle was:

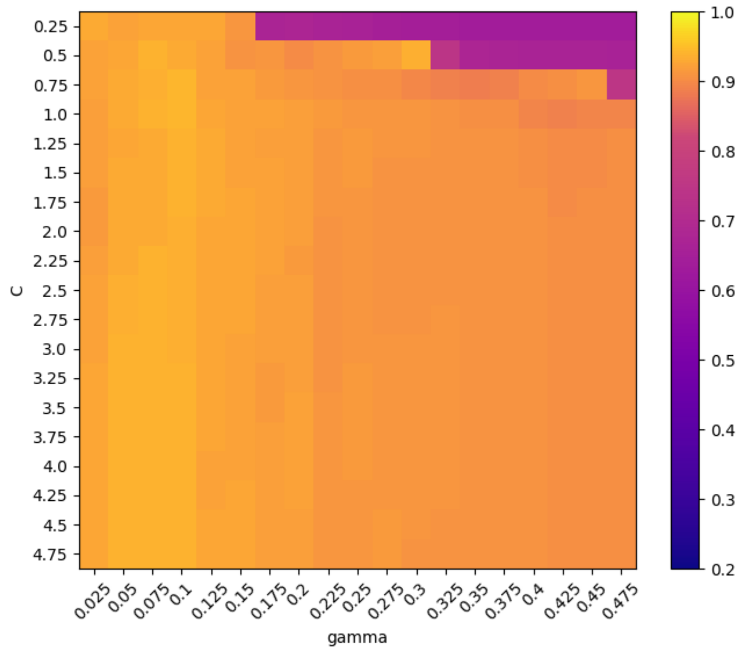
```
{'C': 1.25, 'gamma': 0.05, 'kernel': 'rbf'}
```

Performance results for those values are somehow not better, so probably we shuffled data much better in previous run. Accuracy average is still the same, but sensativity is lower.

```
             precision    recall  f1-score   support

        0.0       1.00      0.83      0.91        36
        1.0       0.90      1.00      0.95        52

avg / total       0.94      0.93      0.93        88
```

The overall fine grid search looks a bit differently from previous run. **Figure 6** shows that top right corner combinations work not as well with this dataset. Whereas having higher gamma values we may overfit our data by making boundaries too precise. Having too low C values doesn't add enough curves to our decision boundaries.

Figure 6: Accuracy (Fine Grid Search)



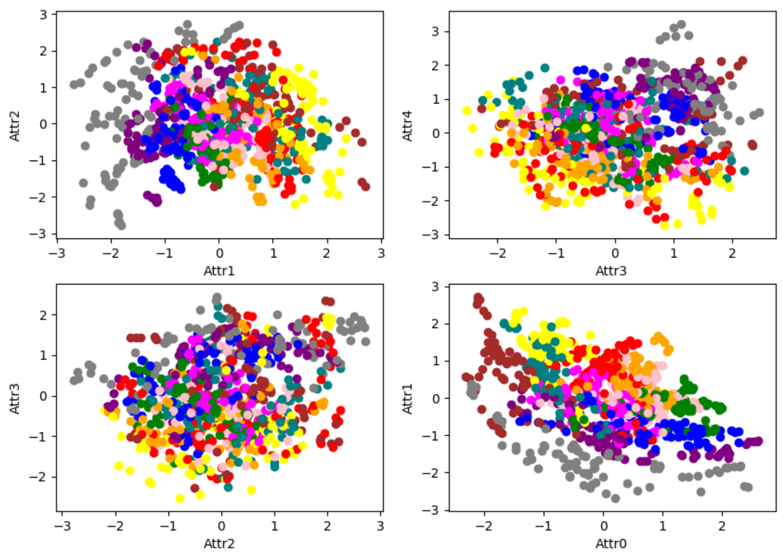
Overall, trying multiple runs and combinations of hyperparameters for this problem, I think C = 5.0 and gamma = 0.1 are the most optimal.

3.2 Vowel-Context Dataset

Second given problem was different from previous in number of classes and number of features for each datapoint. We had 11 classes only 13 features with 3 of them being irrelevant. So after removal, we were left with only 10 attributes. As in previous experiment our data was shuffled and split into training and validation/testing datasets.

Moreover, I also tried to see if there is some possible dependency between several pairs of attributes in this data set. As we can see in **Figure 7**, there are many classes, and they are obviously not linearly separable. But not like in previous experiment, our datapoints for each class are way more clustered in this dataset, for all 4 tested pairs of attributes. So it may mean that overall prediction results should be better in this experiment.

Figure 7: Attributes pairs (11 classes)



I've done several runs for coarse grid search and my best parameter range always looked as following:

```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

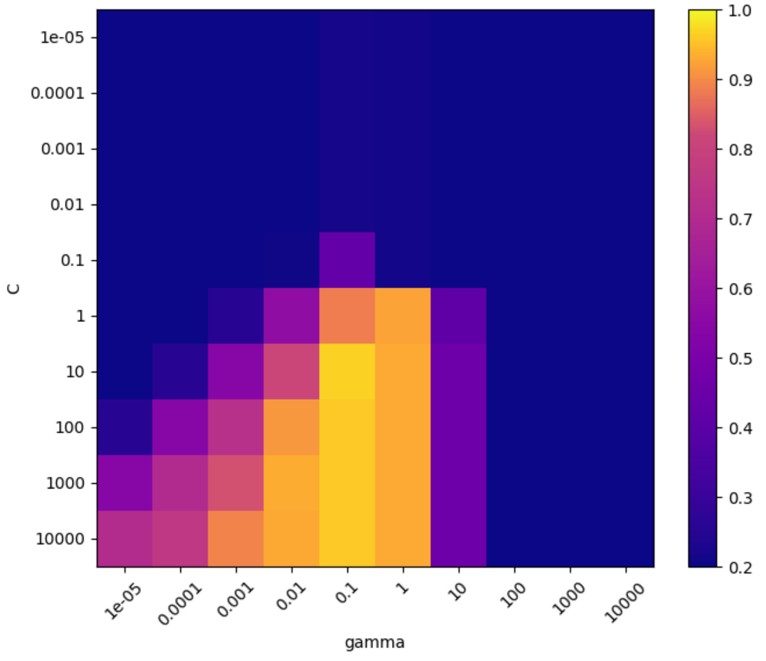
After training my validation/testing set with these parameters I get:

```
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00        23
        1.0       1.00      1.00      1.00        31
        2.0       1.00      1.00      1.00        23
        3.0       1.00      1.00      1.00        27
        4.0       1.00      0.94      0.97        18
        5.0       0.94      0.94      0.94        17
        6.0       1.00      1.00      1.00        21
        7.0       1.00      1.00      1.00        26
        8.0       1.00      1.00      1.00        22
        9.0       1.00      1.00      1.00        17
       10.0       0.96      1.00      0.98        23

avg / total       0.99      0.99      0.99       248
```

This performance is just surprising, and I think it makes sense, since data file is much larger than in previous experiment. I also think that smaller number of features makes it perform better. For almost all labels we predict our data with 100% **accuracy**. Only couple labels get mistaken in few instances I guess. We can also see similar percentage for **sensitivity** or **recall**. And obviously, **f1-score** is just a mean of those two metrics. We had also bigger number of datapoints to be tested. Numbers for instances per label look similar for all labels which makes it better for training this data.

Figure 8 we can observe how different combinations of C and gamma parameters performed on current dataset during coarse grid search. As with previous experiments, lower left quarter values for C and gamma display better prediction performance. Overall, gamma = 0.1 range has better results. Making gamma bigger gradually leads to overfitting of the data because of decision boundaries being too precise. Whereas lower values of gamma make decision boundary not curvy enough to cover spreads of the data. Moreover, with too low values of C, we do not penalize results enough and allow miss classifications a lot. With C overall bigger than 10 decision boundary becomes less biased and has more variance.

Figure 8: Accuracy (Coarse Grid Search)



In the fine grid search for this experiment, I used C ranges from 2.5 to 50 and gamma values from 0.025 and 0.475. The best result was achieved for:

```
{'C': 7.5, 'gamma': 0.15, 'kernel': 'rbf'}
```

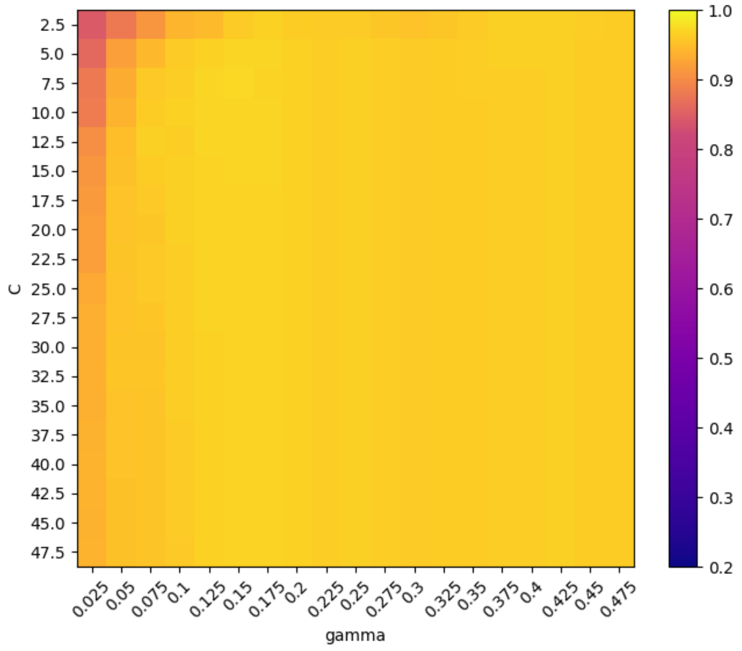
Prediction performance for these values on testing/validation dataset was outstanding:

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       1.00      1.00      1.00        31
         2.0       1.00      1.00      1.00        23
         3.0       1.00      1.00      1.00        27
```

```
        4.0       1.00      1.00      1.00        18
        5.0       1.00      0.94      0.97        17
        6.0       1.00      1.00      1.00        21
        7.0       1.00      1.00      1.00        26
        8.0       1.00      1.00      1.00        22
        9.0       1.00      1.00      1.00        17
       10.0       0.96      1.00      0.98        23

avg / total       1.00      1.00      1.00       248
```

Definitely we can see improvement. We have 100% accuracy in prediction on average for all labels in this dataset.Performance for data labeled as 5 is much better with these hyperparameters. Overall fine grid search combinations are displayed in **Figure 9**.

Figure 9: Accuracy (Fine Grid Search)



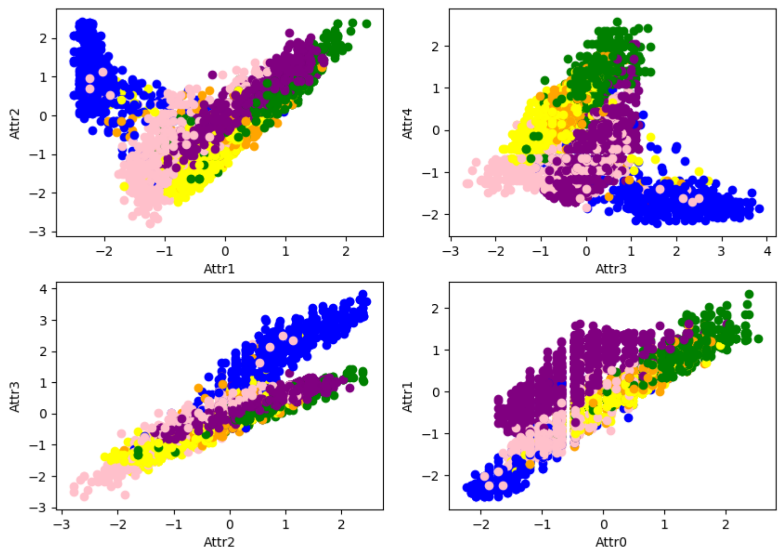
The whole grid shows performance near 0.99-1 value. The very top left corner scored around 0.9. I assume, too low values of both C and gamma do not create decision boundaries precise enough to cover spread of the data well. Plus misclassified points are allowed more in that range of C.

Overall results for this experiment were great. Although dataset was bigger, it didn't take that much longer to run it. I think, having less features made out predictions better since each feature independently had more value than when we have 30 or more features. but that's not always true as well. Also, bigger size of data set and evenly distributed number of instances per each label influenced our training results I think.

3.3 Satellite Dataset

Finally, we have another dataset of a much larger size.We only have 7 classes but 36 features for each instance. We had to determine the type of terrain from multispectral values of pixels in 3x3 neighborhoods in satellite images. Were were given dataset already split into training and validation parts of 4435 and 2000 instances correspondingly. This experiment took much longer to run due to size of data. I also tried to display possible pairs of attributes here after data standardization. I combined both files for better results. Our datapoints are not linearly separable for each combination of attributes but for all classes datapoints are very tightly clustered. I find it unusual for having that many features.

Figure 10: Attributes pairs (7 classes)



After performing a coarse grid search, the best range of hyperparameters looked as following:

```
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
```

The results of predicting labels for testing dataset are displayed below.

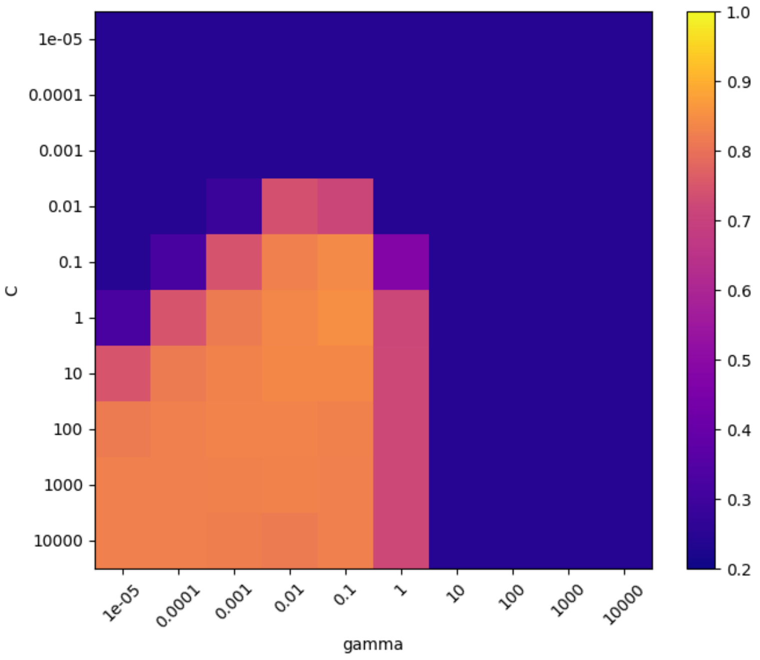
```
           precision    recall  f1-score   support

     1.0       0.98      1.00      0.99       461
     2.0       0.97      0.98      0.97       224
     3.0       0.87      0.97      0.92       397
     4.0       0.77      0.63      0.69       211
     5.0       0.94      0.90      0.92       237
     7.0       0.89      0.87      0.88       470
```

```
avg / total       0.91      0.91      0.91      2000
```

For some labels, *accuracy* performance is much better than for others. For example, instances with labels 1 and 2 get predicted correctly in 97-98% cases, whereas instances with label 4 are often missclassified. That makes overall prediction accuracy 91%. It's not bad but not great as well. *Sensitivity* results are similar.

Overall results for coarse grid search may be seen in **Figure 11**. Again, as with other experiments, combinations of lower values for gamma and higher values for C show better results. For gamma values over 0.1 we overfit our data by making decision boundaries too dependent on individual data points. For C values lower than 0.01 or 0.1 we don't give our classifier enough penalty, so decision boundary doesn't cover spread of the data too well.

Figure 11: Accuracy (Coarse Grid Search)

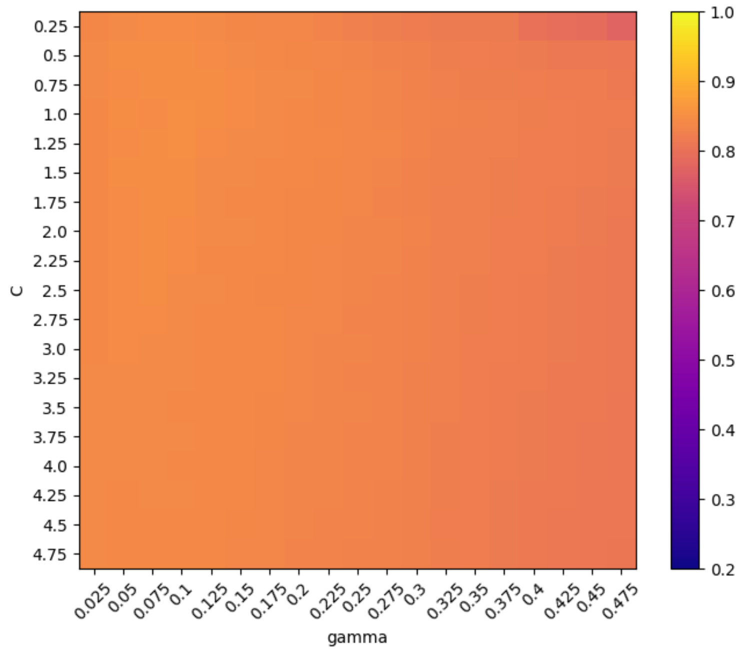


For fine grid search I used C values from 0.25 to 5 and gamma values from 0.025 to 0.5 again. The most optimal parameters were these:

```
{'C': 25, 'gamma': 0.225, 'kernel': 'rbf'}
```

After I tested validation set on it again, prediction results weren't much better. Actually, there were almost the same. In **Figure 12**, we can see overall result for fine grid search. The whole grid has the same shade. There is a very slight difference in the top right corner. So, even though optimal parameters are C=25 and gamma = 0.225, the whole range of best hyperparameters works the same with this dataset.

Figure 12: Accuracy (Coarse Grid Search)



Overall, I think that size of the data and number of features for each instance may influence training and prediction results. But in general, form this experiment we could see, that SVMs are very slow on datasets of large sizes.

4. Conclusions.

Finalizing my experiments, I would like to say that Support Vector Machines are great for dealing with high dimensional data, we don't have to worry about reducing dimensions for training and predicting our data like with some other learning algorithms. SVMs also work the best with smaller datasets. Picking what Kernels to use and what hyperparameters can be already computationally expensive, but if in addition data set is large, training data may take quite some time. This is a part of a reason why we want to narrow best parameters search by doing coarse grid search first I think. Then after we know good parameters range we fine tune optimal parameters by doing fine grid search.