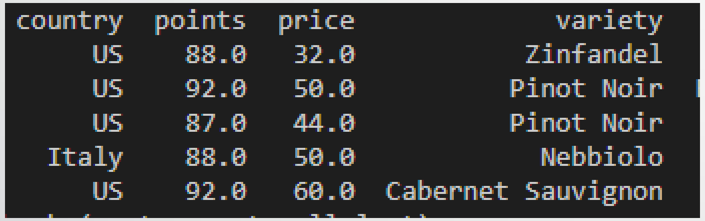
SupportVectorMachine Wine Variety Classifier

Francesco Stucci

Dataset
(https://www.kaggle.com/zynicide/wine-reviews)

Dataset head



```
country  points  price                variety
     US    88.0   32.0              Zinfandel
     US    92.0   50.0             Pinot Noir
     US    87.0   44.0             Pinot Noir
  Italy    88.0   50.0               Nebbiolo
     US    92.0   60.0     Cabernet Sauvignon
```

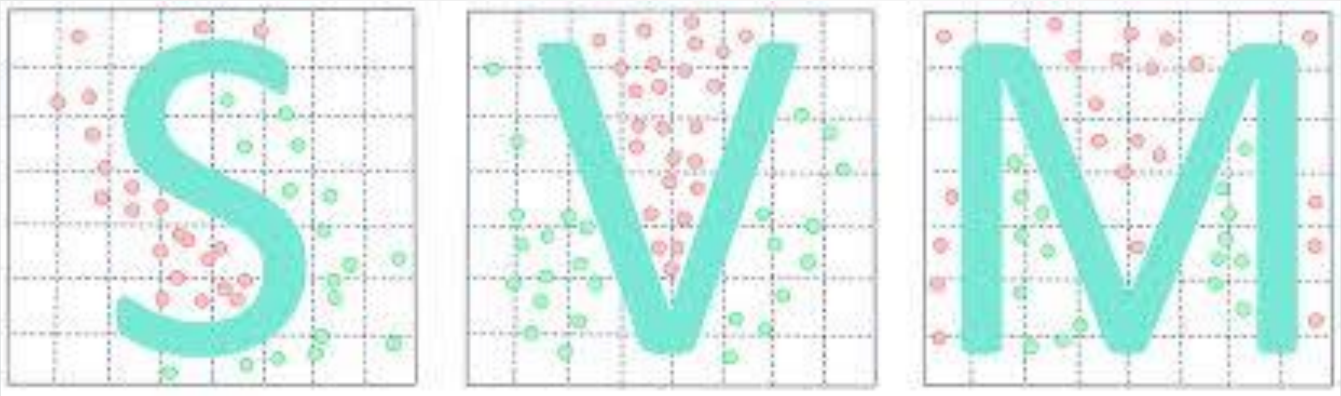
Dataset shape: (104183, 4)

The goal

Our goal is to classify the **Variety** of wine based on the dataset features **Country, Points** and **Price**

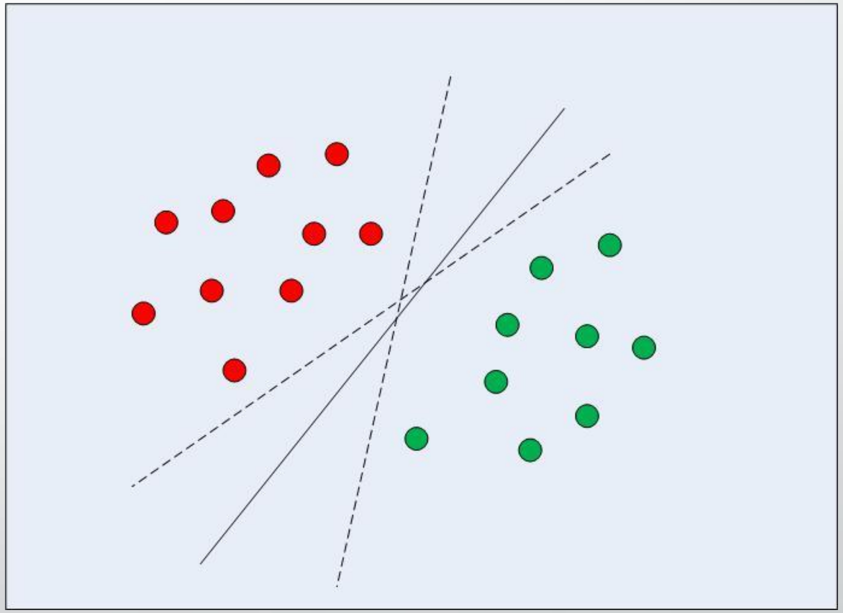
Which model are we supposed to use?

Our dataset consists of **High Sample Size** and **Low Dimension**.
We have more than 100 thousend samples and 54 different wine viriety to classify.
For these reasons I chose the **SupportVectorMachine** model to deal with it



SVM in a nutshell

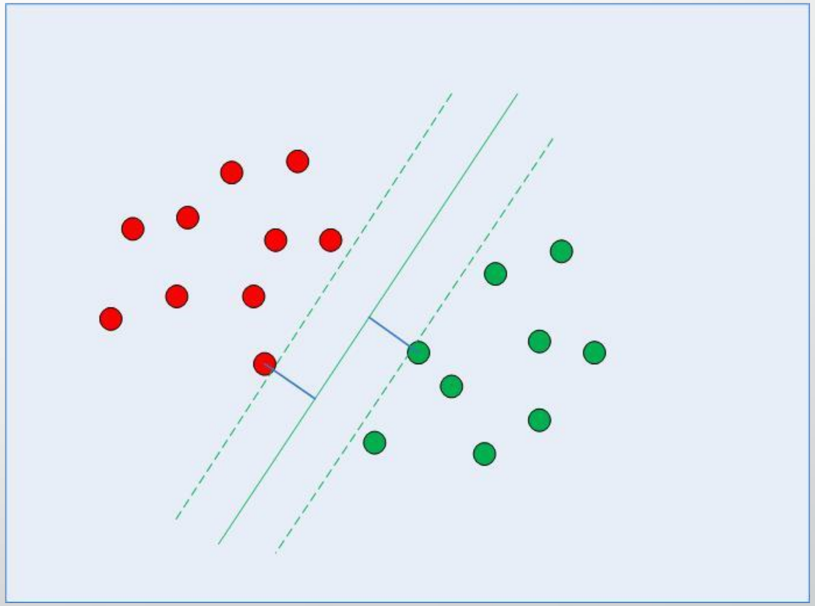
In case of linearly separable data in two dimensions, SVM algorithm tries to find a boundary that divides the data in such a way that the misclassification error can be minimized. As shown in the figure beside, there can be several boundaries that correctly divide the data points. The two dashed lines as well as one solid line classify the data correctly.



SVM in a nutshell

SVM chooses the decision boundary that maximizes the distance from the nearest data points of all the classes. It doesn't simply find a decision boundary, it finds the most optimal decision boundary.

The most optimal decision boundary is the one which has maximum margin from the nearest points of all the classes. The nearest points from the decision boundary that maximize the distance between the decision boundary and the points are called support vectors.



How to find the best Hyperplane

SVM problem can be formulated as

$\bar{w} \cdot \bar{x}_i + b \geq 1$ for $y_i = +1$

$\bar{w} \cdot \bar{x}_i + b \leq 1$ for $y_i = -1$

By combining the above equation we get:

$y_i(\bar{w} \cdot \bar{x}_i + b \ - 1) \geq 0 \quad for\ y_i = +1, -1$

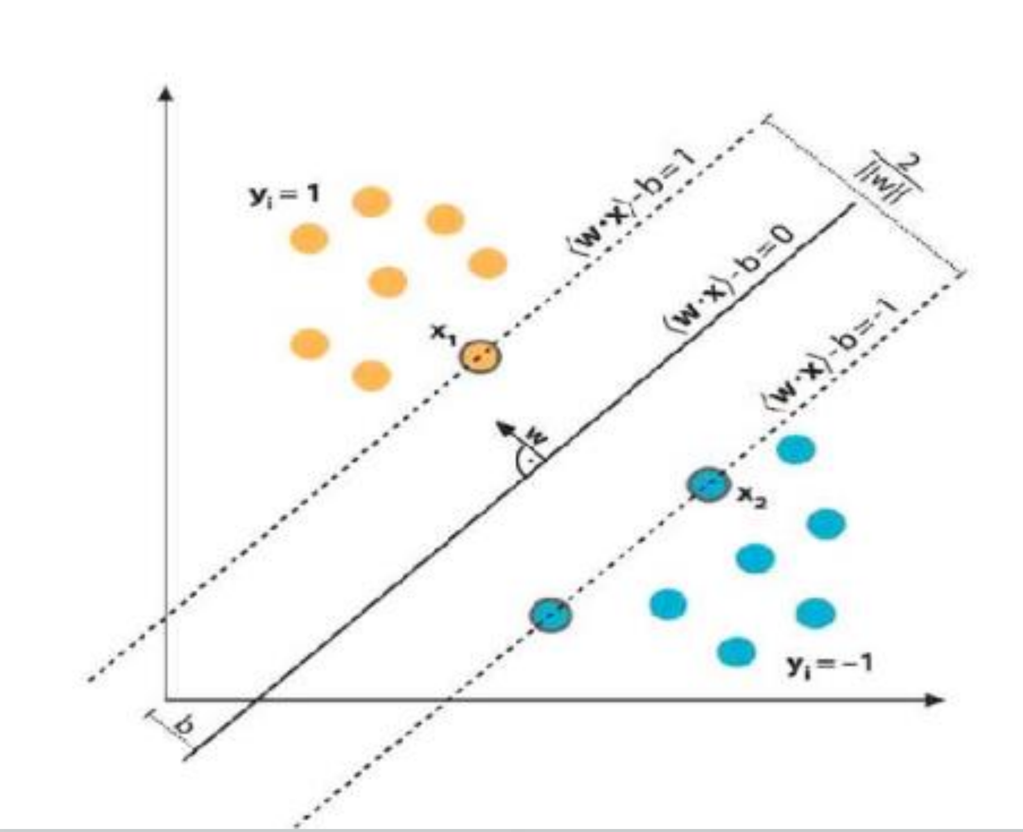
We need to maximize the margin given by:

$$M = (\bar{x}_+ - \bar{x}_-)\frac{\bar{w}}{\|w\|}$$

$$= (1 - b)/\|w\| - (-1 - b)/\|w\| \ = 2/\|W\|$$

Which can be written as

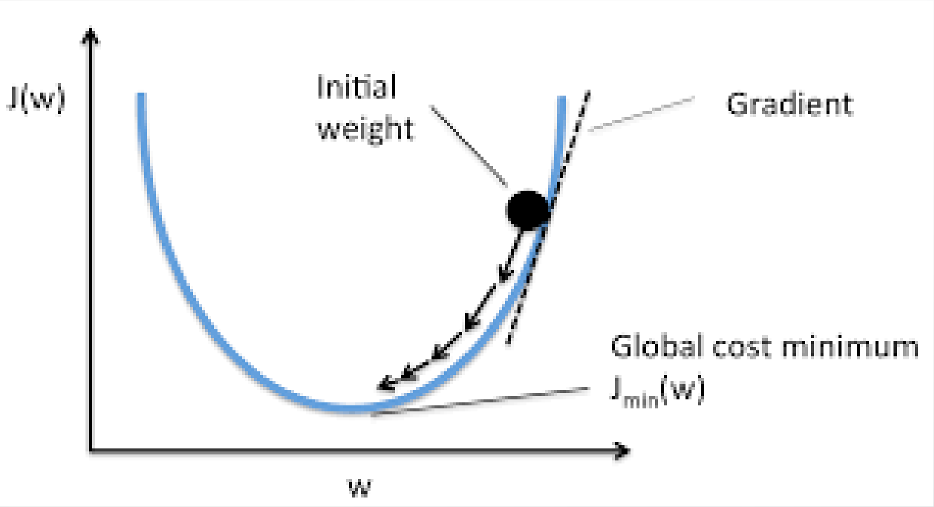
$$\min \frac{\|w\|^2}{2} \quad such\ that\ y_i(\bar{w} \cdot \bar{x}_i + b \ - 1) \geq 0$$



$y_i = 1$
$\langle w \cdot x \rangle - b = 1$
$\frac{2}{\|w\|}$
$\langle w \cdot x \rangle - b = 0$
x_1
$\langle w \cdot x \rangle - b = -1$
w
x_2
$y_i = -1$
b

We get the SVM Primal form

In order to solve this convex problem we can use the gradient discendent algorithm



Or we can use Lagrange Multpliers

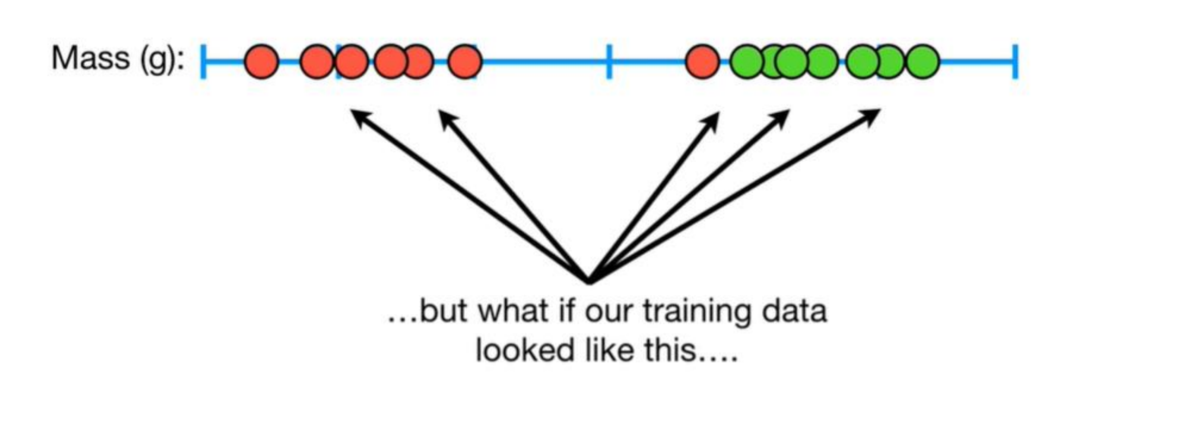
Whereas by solving convex constrained optimization problem with primal formulation could lead not to find any solutions, the Lagrange Multipliers algorithm exploit the dual formulation of the problem which always lead to a solution.
For this reason this method it's usually used.
By using this algorithm we are able to solve our minimization problem and find the best values for $\overline{w}$ and b

Hard Margins and Soft Margins

In the image beside we can notice if we placed the hyperplane between red and green dots, we'd make a mistake!

The previous definition we gave is called **Hard Margins** because it doesn't accept any missclassification.

To allow missclassification we need to add an extra term to our original formula. The result is called **Soft Margins**



The new problem

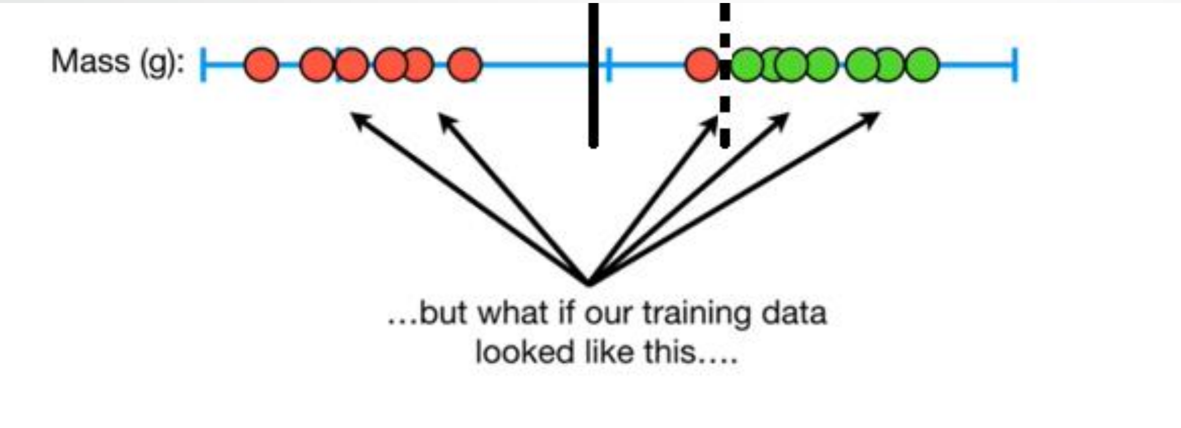
$$\min_{\underline{w},b,\underline{\phi}} \frac{1}{2} \|\underline{w}\|^2 + C \sum_{i=1}^{n} \xi_i$$

$$y_i \left(\underline{w}^T \phi\left(x_i\right) + b \right) \geq 1 - \xi_i \qquad \text{constraint } 1$$

$$\xi_i \geq 0 \qquad \text{constraint } 2$$

By adding this new term, our classifier accept missclassifications.
The larger the C, the less point missclassification is allowed.
The smaller it is, the more points can be missclassified

The intuition behind



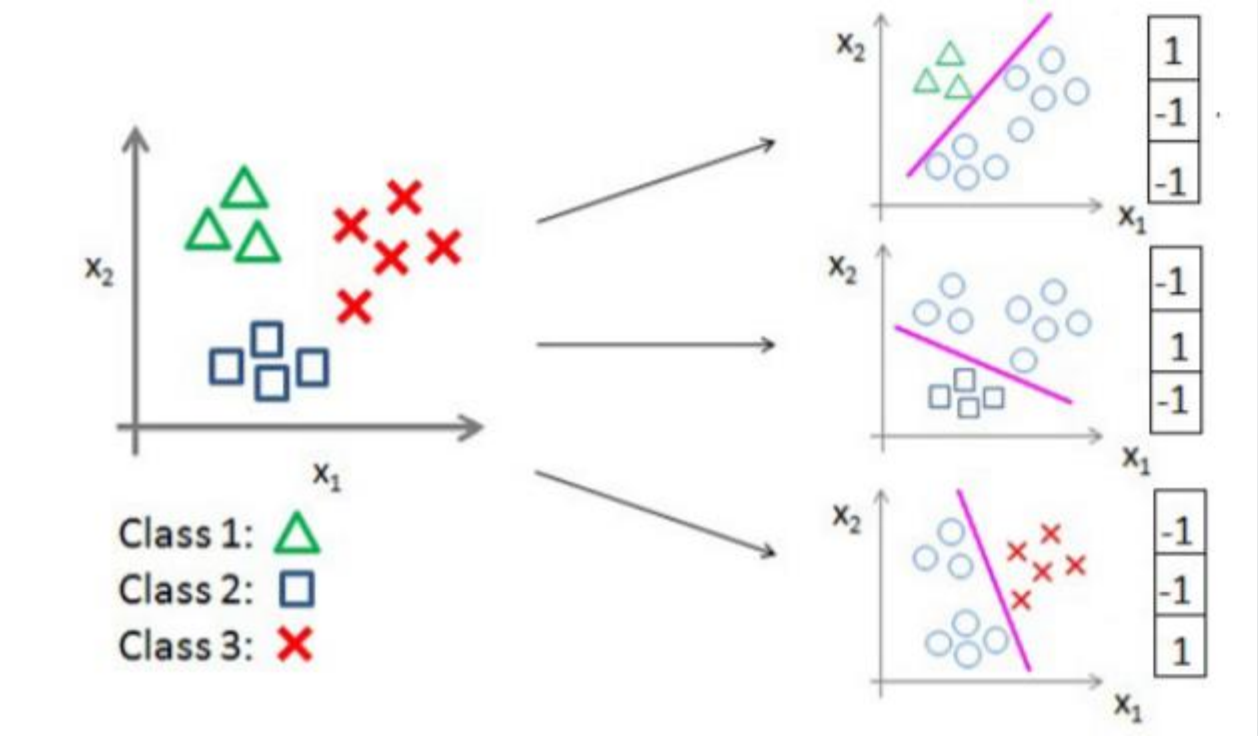
We can say that, even if a red point is misscassified, the solid treshold classifies better our dataset

How to deal with multiple classes

So far we have just considered only 2 different classes. SVM is a binary classifier indeed.
However, our dataset has 54 different classes.
There are two main methods to deal with this inconvenient: **OneVsOne** and **OneVsAll**

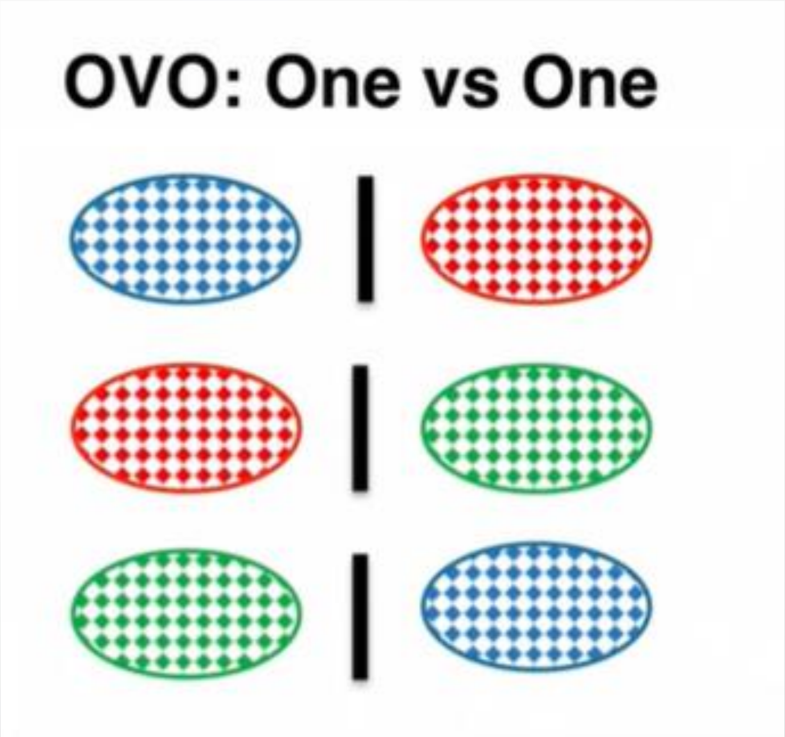
One VS All

In **OVA** we create a classifier for each class, each one has to classify that class in respect to all other points, no matter which class the rest is.
The final prediction will result as the largest number of votes.
This method leads to solve not balanced problem and it can be fine for problme with few classes



One VS One

In **OVO** we create
$c\frac{(c-1)}{2}$ classifiers.
We have much more models, however they are learned for a subset of the orginal data.



Since dataset sample size impact a lot on SVM computational time and we have a lot of classes to classify, the **OVO** method is the one which better suits this case

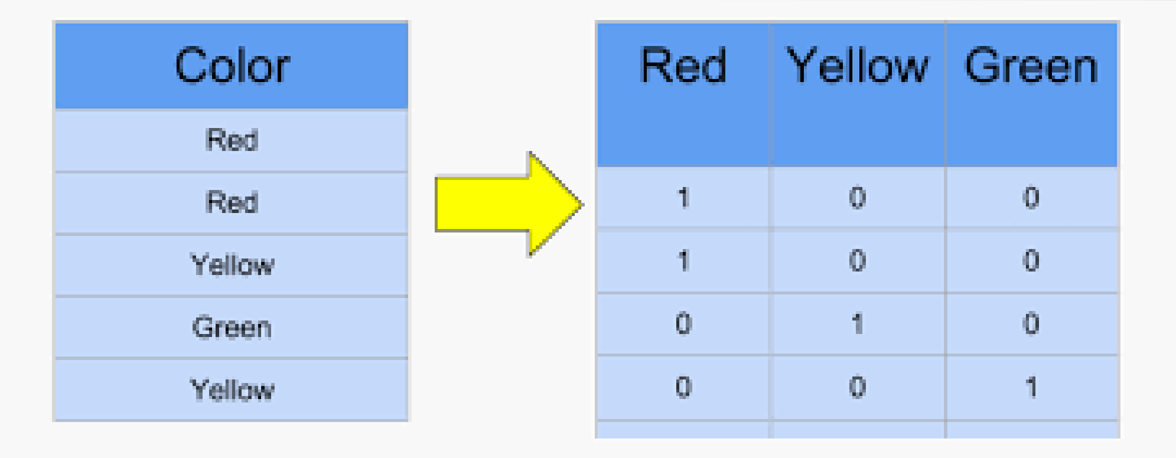
Coming back to our dataset: Preprocessing

Before to start to apply our algorithm to our Dataset, we should check some information about it.
First of all, SVM cannot handle categorical data.
In our features we have **Country** which is a label. We need to transform it into a numerical value in order to make it manageable by SVM

Data Preprocessing
Data Integration
Data Transformation
Data Reduction
Data Cleaning

One-Hot Encoding

Insead of some other models, where we can simply substitute numbers to label, in SVM this will lead to misinterpretation of data. This because numbers are related among them, whereas label not.
To prevent this problem we use the **One-Hot Encoding** method, which add a dimension for each different label as in the beside image.



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Some little adjustment on our specific Dataset

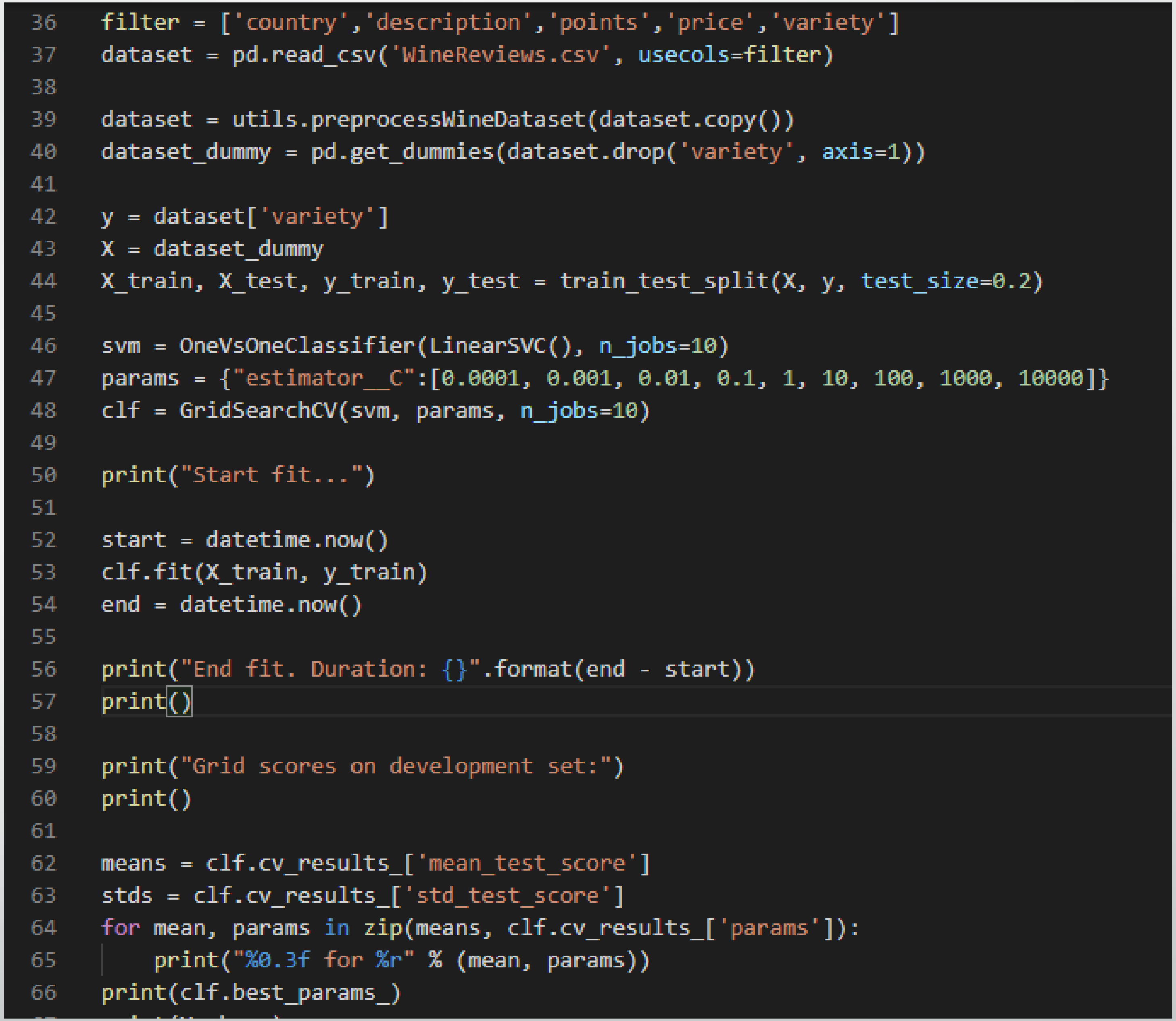
- This dataset was conteining duplicated rows. All replications have been pruned.
- Then there were some wine variety with an insignificant number of reviews, those have been removed too.
- We normilized the translations of some variety to english
- Finally we deleted the null values

We are almost done!

The last step to get the best wine variety classifier, is to tune our C hyperparameter by making some experiments on our dataset.
As said before, it indicates the tollerance of missclassification. Its values cannot be predicted a priori without trying it on our true data.

The final code

Accuracy: **26%** {C = 1}



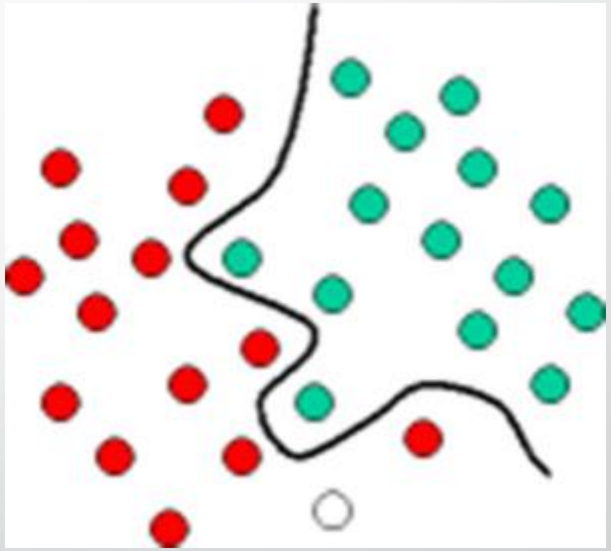
```python
filter = ['country','description','points','price','variety']
dataset = pd.read_csv('WineReviews.csv', usecols=filter)

dataset = utils.preprocessWineDataset(dataset.copy())
dataset_dummy = pd.get_dummies(dataset.drop('variety', axis=1))

y = dataset['variety']
X = dataset_dummy
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

svm = OneVsOneClassifier(LinearSVC(), n_jobs=10)
params = {"estimator__C":[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}
clf = GridSearchCV(svm, params, n_jobs=10)

print("Start fit...")

start = datetime.now()
clf.fit(X_train, y_train)
end = datetime.now()

print("End fit. Duration: {}".format(end - start))
print()

print("Grid scores on development set:")
print()

means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, params in zip(means, clf.cv_results_['params']):
    print("%0.3f for %r" % (mean, params))
print(clf.best_params_)
```

Accuracy: **32%** {C = 0.01}
(out of 10'000 samples)

Can we do better? Yess

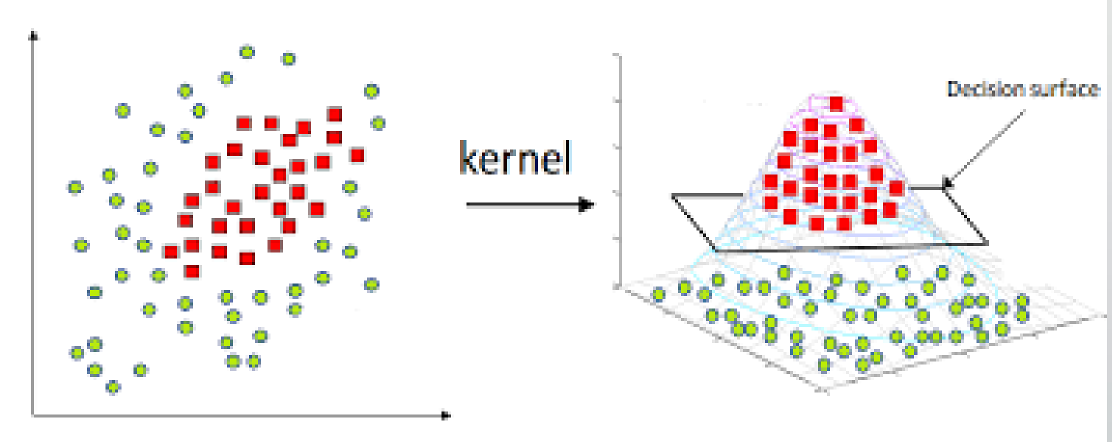
Our dataset is probably not completely linearly separable. We mitigate this fact by aloowing some missclassifications. This leads to some mistake tough. How can we do better?

As shown in the image beside, we'd need such a curved decision boundary to classify correctly all points.
In order to build such a curve, we need to introduce **Kernels**

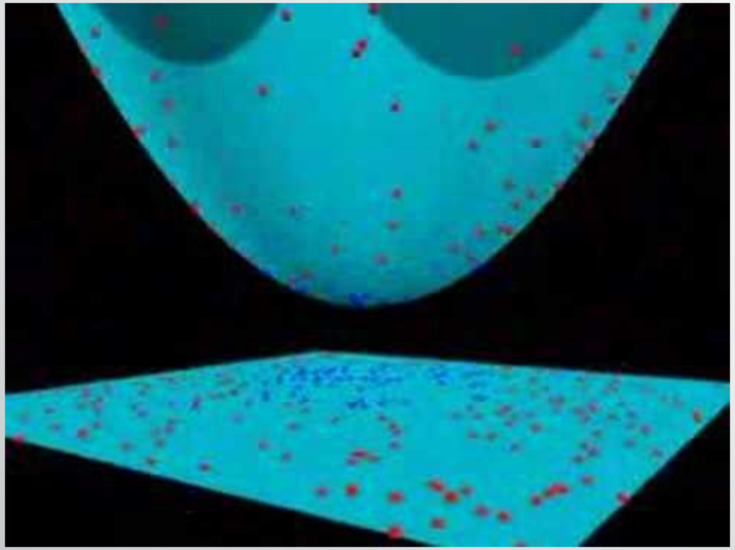


The idea behind Kernels

A **Kernel** is a function which project all points in an extra dimension. Than we are able to create an hyperplane in this new domain and the resulting decision boundry will be curve in the original domain.



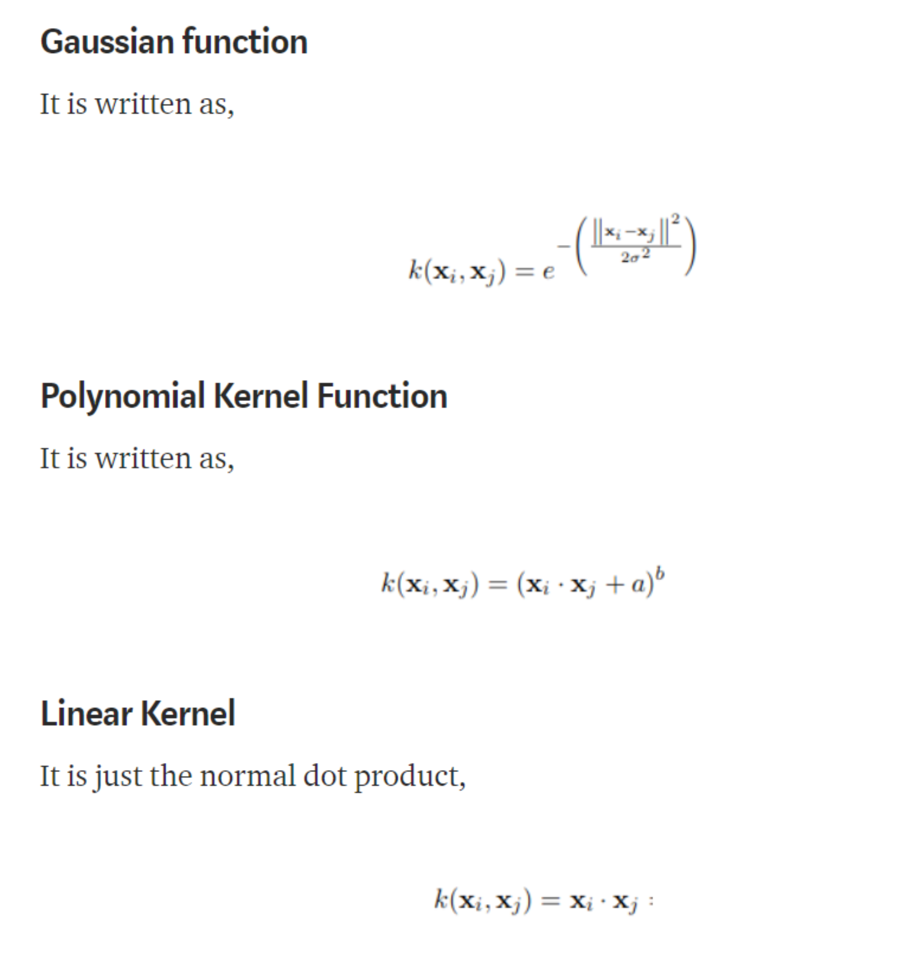
The Kernel Trick



Transforming out data points into higher dimension and the use soft-margin classification method would be fine.
However this needs a lot of space and computational cost.

Hence it's preferred to use the **Kernel Trick** to make our problem computationally solvable.

Most used Kernels



Gaussian function

It is written as,

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)}$$

Polynomial Kernel Function

It is written as,

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + a)^b$$

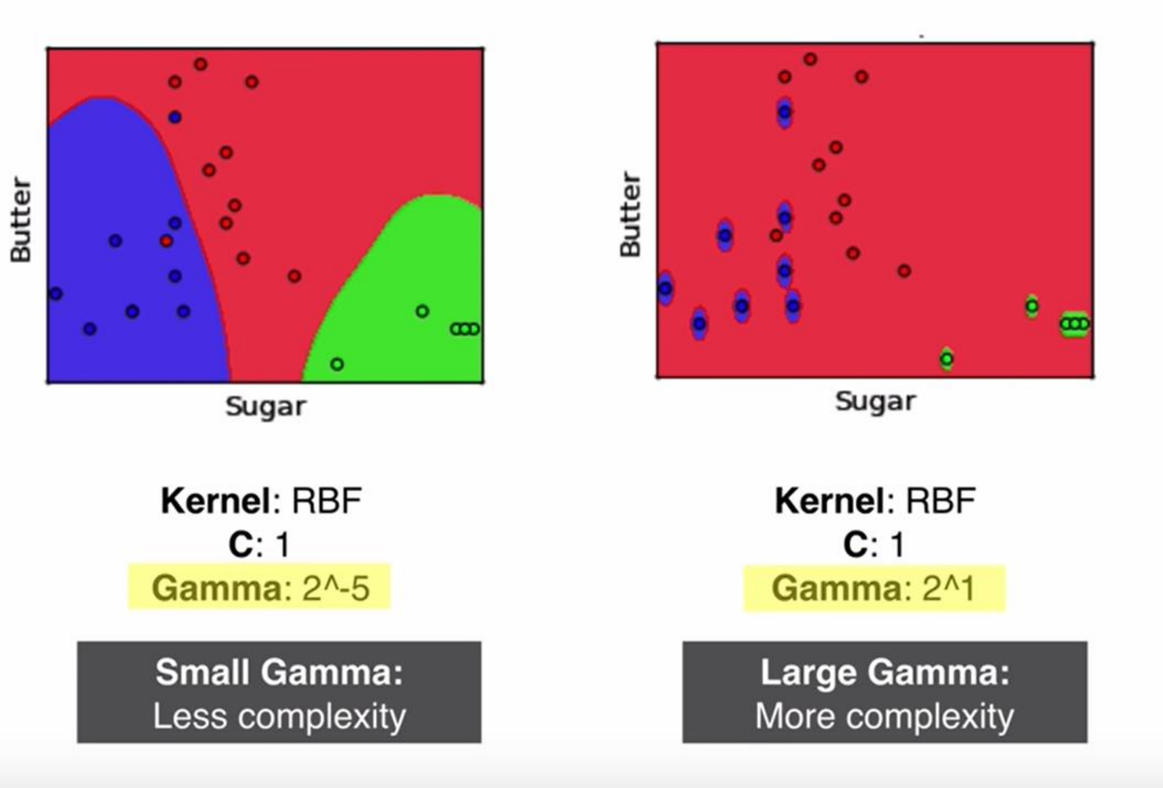
Linear Kernel

It is just the normal dot product,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

A new Hyperparameter: *gamma*

The **Gaussian Kernel** brings with itself a new hyperparameter we need to tune.



Kernel: RBF
C: 1
Gamma: 2^-5

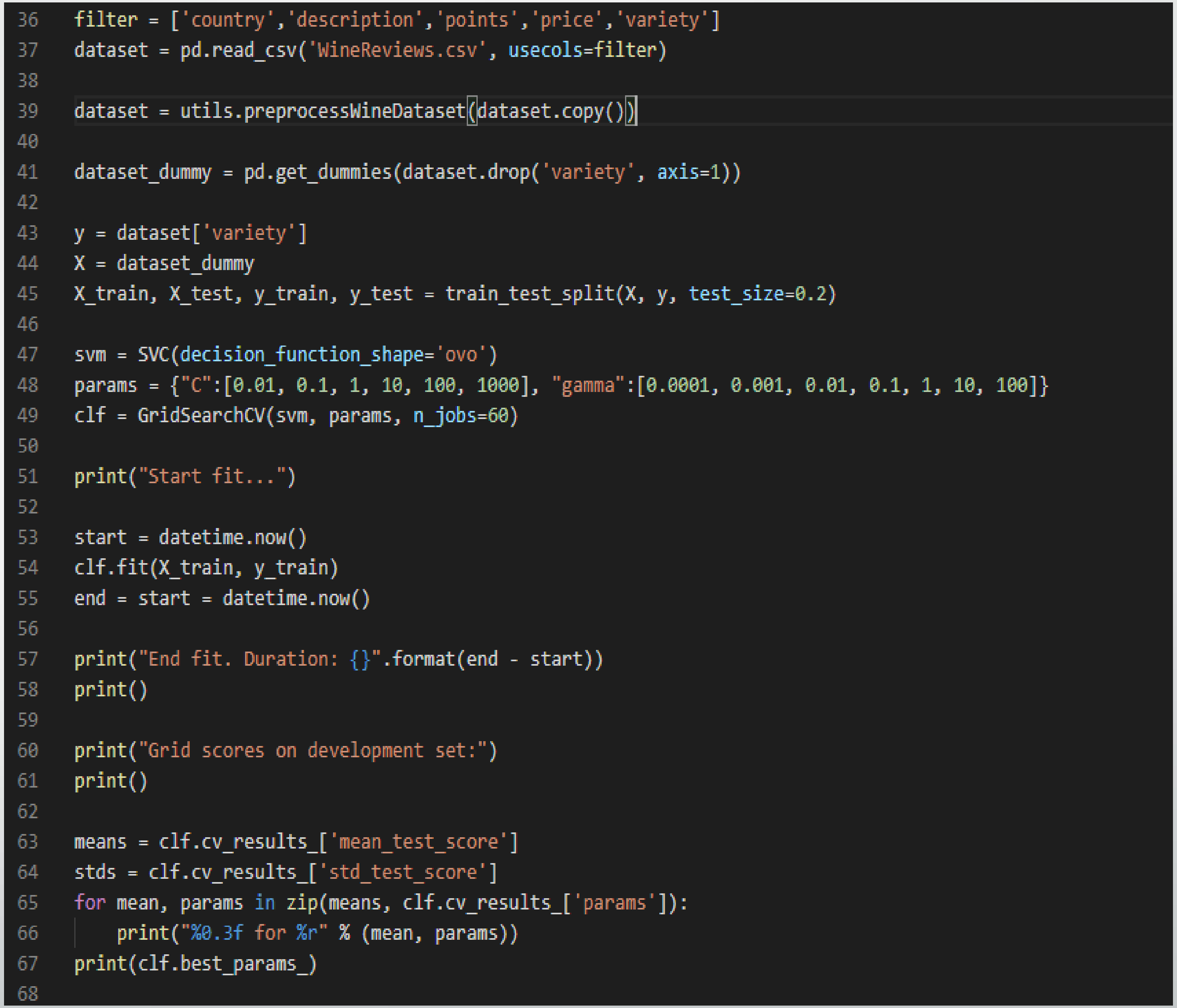
Small Gamma:
Less complexity

Kernel: RBF
C: 1
Gamma: 2^1

Large Gamma:
More complexity

The final code

Accuracy:
29% {C = 1000, gamma= 0.001}



```python
filter = ['country','description','points','price','variety']
dataset = pd.read_csv('WineReviews.csv', usecols=filter)

dataset = utils.preprocessWineDataset(dataset.copy())

dataset_dummy = pd.get_dummies(dataset.drop('variety', axis=1))

y = dataset['variety']
X = dataset_dummy
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

svm = SVC(decision_function_shape='ovo')
params = {"C":[0.01, 0.1, 1, 10, 100, 1000], "gamma":[0.0001, 0.001, 0.01, 0.1, 1, 10, 100]}
clf = GridSearchCV(svm, params, n_jobs=60)

print("Start fit...")

start = datetime.now()
clf.fit(X_train, y_train)
end = start = datetime.now()

print("End fit. Duration: {}".format(end - start))
print()

print("Grid scores on development set:")
print()

means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, params in zip(means, clf.cv_results_['params']):
    print("%0.3f for %r" % (mean, params))
print(clf.best_params_)
```

Accuracy: 36% {C = 1000, gamma= 0.001}
(out of 10’000 samples)