Understanding SVMs (1 point)

1. (0.5 points) Explain why a support vector machine using a kernel, once trained, does not directly use the decision boundary to classify points.

   The "kernel trick." Adding new features/variables that are functions of your other input variables can change linearly inseparable problems into linearly separable problems.

   The important property is that kernels look like inner products in a transformed space.

   If I know what the inner product of two transformed items is, then I can directly calculate the inner product without doing the transformation first.

   Briefly speaking, a kernel is a shortcut that helps us do certain calculation faster which otherwise would involve computations in higher dimensional space.

   In general, the larger the margin, the lower the generalization error of the classifier. Therefore, svm doesn't directly use the decision boundary because it can reduce the error.

   A large-margin classifier tends to be more "robust" (resistant to noise in the data, able to generalize)

2. (0.5 points) If the support vector machine does not directly use the decision boundary to classify points, how does it, in fact, classify points. *Hint, what are the support vectors?*

   Hard margin

   If the training data is [linearly separable](), we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them. With a normalized or standardized dataset, these hyperplanes can be described by the equations

   $w \cdot x - b = 1$ (anything on or above this boundary is of one class, with label 1) and $w \cdot x - b = -1$ (anything on or below this boundary is of the other class, with label −1).

   Soft margin

   Allow some instances to fall within the margin, but penalize them

   Non-linear classification

   Non-linear separation • Map the original feature space to a higher-dimensional feature space where the training set is separable. Call this mapping function (kernel function.)

the MNIST data (1 point)

3. (0.5 points) How many images are there in the MNIST data? How many images are there of each digit? How many different people's handwriting? Are the digit images all the same size and orientation? What is the color palette of MNIST (grayscale, black & white, RGB)?

   training set of 60,000 examples, and a test set of 10,000 examples.

	Training + testing
0:	5923+6903
1:	6742+7877
2:	5958+6990
3:	6131+7141
4:	5842+6824
5:	5421+6313
6:	5918+6876
7:	6265+7293
8:	5851+6825
9:	5949+6958

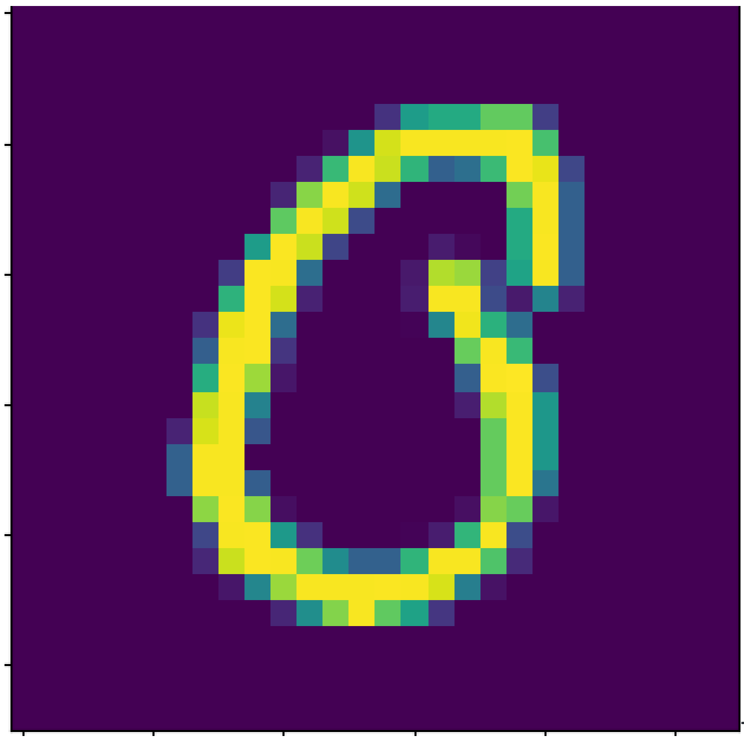
   approximately 250 writers

   The digits have been size-normalized and centered in a fixed-size image. NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio

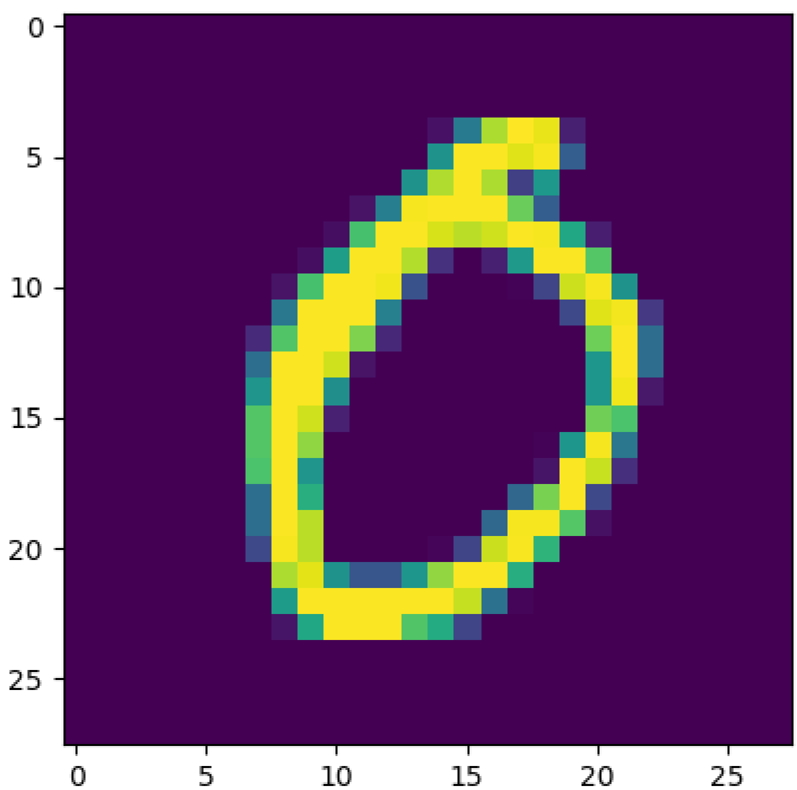
   black and white

4. (0.5 points) Select one of the digits from the MNIST data. Look through the variants of this digit that different people produced. Show us 3 examples of that digit you think might be challenging for a classifier to correctly classify. Explain why you think they might be challenging.

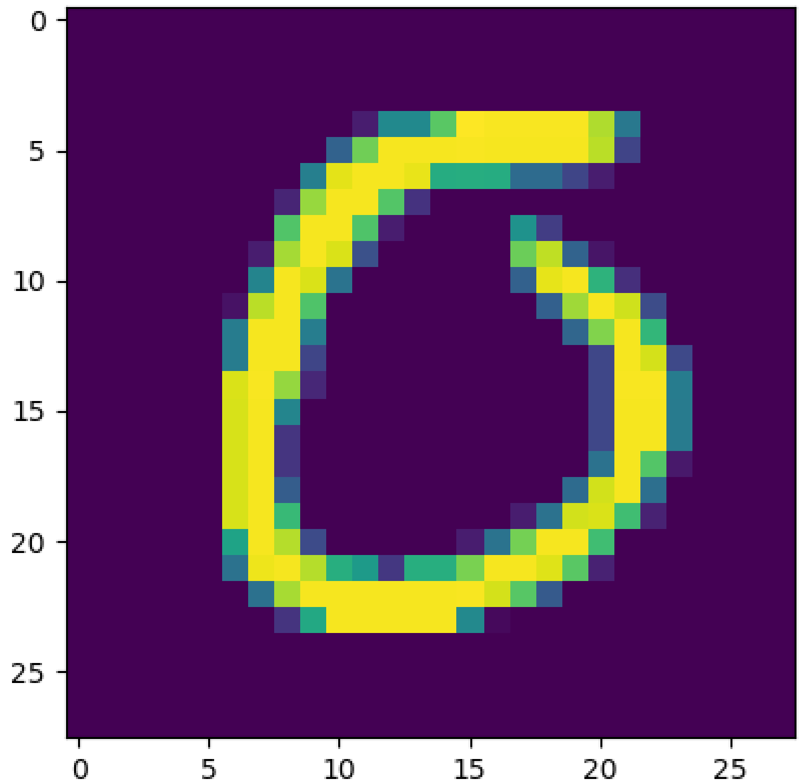
Target to label with 0.



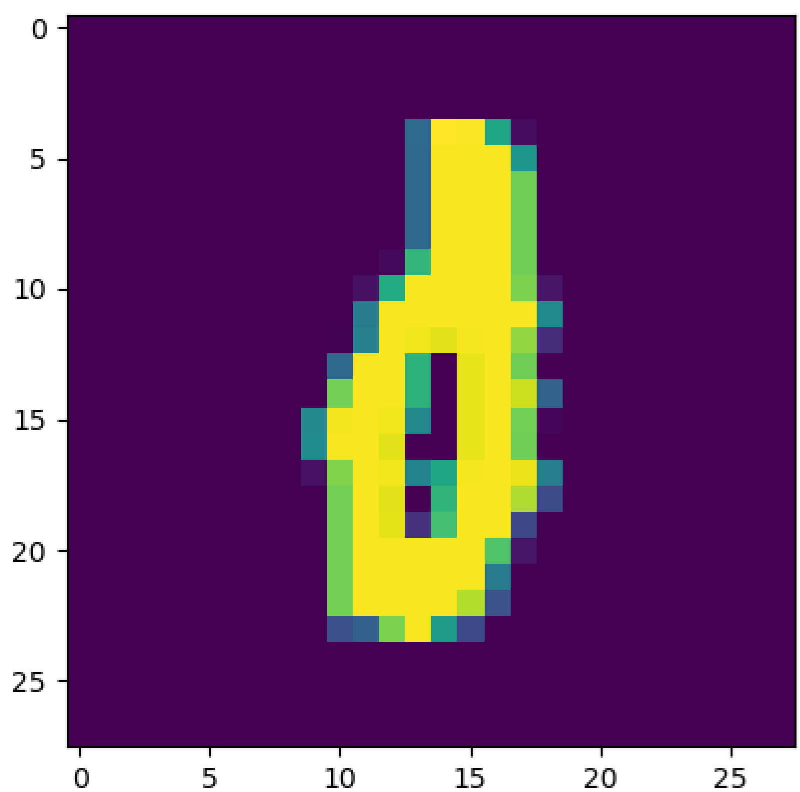
this can be 0 or 6



this can be 0 or 6



this can be 0 or 6



this can be 0 or 1

Estimating training time (1.5 points)

5. (1 point) Before running any serious experiments, first figure out how long your computer takes to train support vector machines on the MNIST data. Use the default **C** value. Use the linear kernel and train an SVM on three different sizes of data set: 500 examples, 1000 examples, 2000 examples. Record how long it takes to train on each data set. Repeat this timing experiment with a polynomial kernel. Use the default value of 3 for the degree of the polynomial. Repeat the experiment with a radial basis function (RBF) kernel. Report the time it took to train each of your SVMs in a table with 3 rows (1 kernel per row) and 3 columns (for the size of the training set). Rows and columns should be clearly labeled. (*HINT:* Use python's built-in `time` module to time your experiments!)

	2	500	1000	2000
Linear	0.0009984970092773438s	0.14991450309753418s	0.3987913131713867s	1.1453661918640137s
Poly	0.0019741058349609375s	0.3778085708618164s	1.478153944015503s	6.185482978820801s
Rbf	0.0010004043579101562s	0.3038511276245117s	0.9264698028564453s	2.8603618144989014s

6. (0.5 points) In general, SVMs take \$O(n^3)\$ to train, where \$n\$ is the size of the training set. But how does that translate to predicting training time in the real world? Given your data from the previous question, write a formula to estimate in clock time how long it would take to train an SVM on your machine, as a function of the number of training examples, given each of the 3 kernels.

	500	1000	2000	4000
Linear	0.14991450309753418s	0.3987913131713867s Compare to 500 2.66012497078	1.1453661918640137s Compare to 500 7.64012932837	3.3951447010040283
Poly	0.3778085708618164s	1.478153944015503s Compare to 500	6.185482978820801s Compare to 500	30.862067699432373

		3.91244153261	16.3720027968	
Rbf	0.3038511276245117s	0.9264698028564453s Compare to 500 3.04909121154	2.8603618144989014s Compare to 500 9.41369491323	8.837805271148682s

Time = a x^3 + b x^2 + cx +d

Linear: time = -1.13645849e-11*n^3+ 2.05656886e-07*n^2+ 2.09156315e-04*n -4.65730258e-03

Polynomial: time = 2.49303500e-10*n^3+ 7.98529943e-07*n^2+ 5.66614707e-04*n -1.36294206e-01

Rbf: time = -3.07123434e-11*n^3+ 5.66596309e-07*n^2+ 4.49089487e-04*n -5.85036505e-02

Selecting training and testing data (1 point)

7. (0.5 points) Given your formula from the previous question, what size of training set would guarantee a single trial for your SVM takes about 2 minutes? Assume this will determine the size of your training set. Now that you have this, tell us how big your testing set will be.

   Training 6800 examples in svm with kernel polynomial takes 119seconds, with linear takes 7.4 seconds, and with rbf takes 19.5 seconds.

   Training 6500 examples in svm with kernel polynomial takes 105.8seconds, with linear takes 6.9 seconds, and with rbf takes 18.36 seconds.

   Training 6000 examples in svm with kernel polynomial takes 85.87seconds, with linear takes 6.2 seconds, and with rbf takes 16.4 seconds.

   6500 training data should guarantee a single trial to finish about 2 minutes.

   My testing data size will be 6500/4 = 1625 -> 1630

So training: testing = 4:1 = 8:2

8. (0.5 points) Now you have to decide how to make a draw from the data that has good coverage. Think about the goals of training and testing sets - we pick good training sets so our classifier generalizes to unseen data and we pick good testing sets to see whether our classifier generalizes. Explain how you should select training and testing sets. (Entirely randomly? Train on digits 0-4, test on 5-9? Train on one group of hand-writers, test on another?). Justify your method for selecting the training and testing sets in terms of these goals.

   Randomly select the same number of examples of each digit for training and testing data. That is, feed the same amount of examples of 0-9 for training and using the same amount of 0-9 to verify the classifier.

Finding the best hyperparameters (4.5 points)

We want to find the best kernel and slack cost, **C**, for handwritten digit recognition on MNIST using a support vector machine. To do this, we're going to try different kernels from the set {Linear, Polynomial, Radial Basis Function}. Use the default value of 3 for the degree of the polynomial. We will combine each kernel with a variety of **C** values drawn from the set { 0.1, 1, 10 }. This results in 9 variants of the SVM. For each variant (a.k.a. condition) run 20 trials.

What's a trial? In one **trial** you...

- Select testing and traing data using your approach from an earlier question.
- Select the condition: your kernel and value for C
- Train the SVM on the training data until it converges.
- Test the trained SVM on the testing data.

To run mutiple trials for the same condition, you select a new testing and traing set for each trial.

For this assignment, we'll be using classfication error on the testing data as the outcome of a trial. Save this data. We'll ask you to show it to us in different ways.

Note: You will have to do 180 trials (9 conditions, 20 trials per condition). If each trial takes 2 minutes, you will need to dedicate 6 hours to these experiments.

Note: There is a tutorial about running python code in parallel included in this repo. Though it is not required, it will make running your experiments much quicker! Look for it here: `code/parallel_tutorial.py`.

9. (1 point) Create a table with 3 rows (1 kernel per row) and 3 columns (the 3 slack settings). Rows and columns should be clearly labeled. For each condition (combination of slack and kernel), show the following 3 values: the testing error measure **e**, the standard deviation of the error **std** and the number of trials **n**, written in the format: **e(std),n**.

e = mean mse

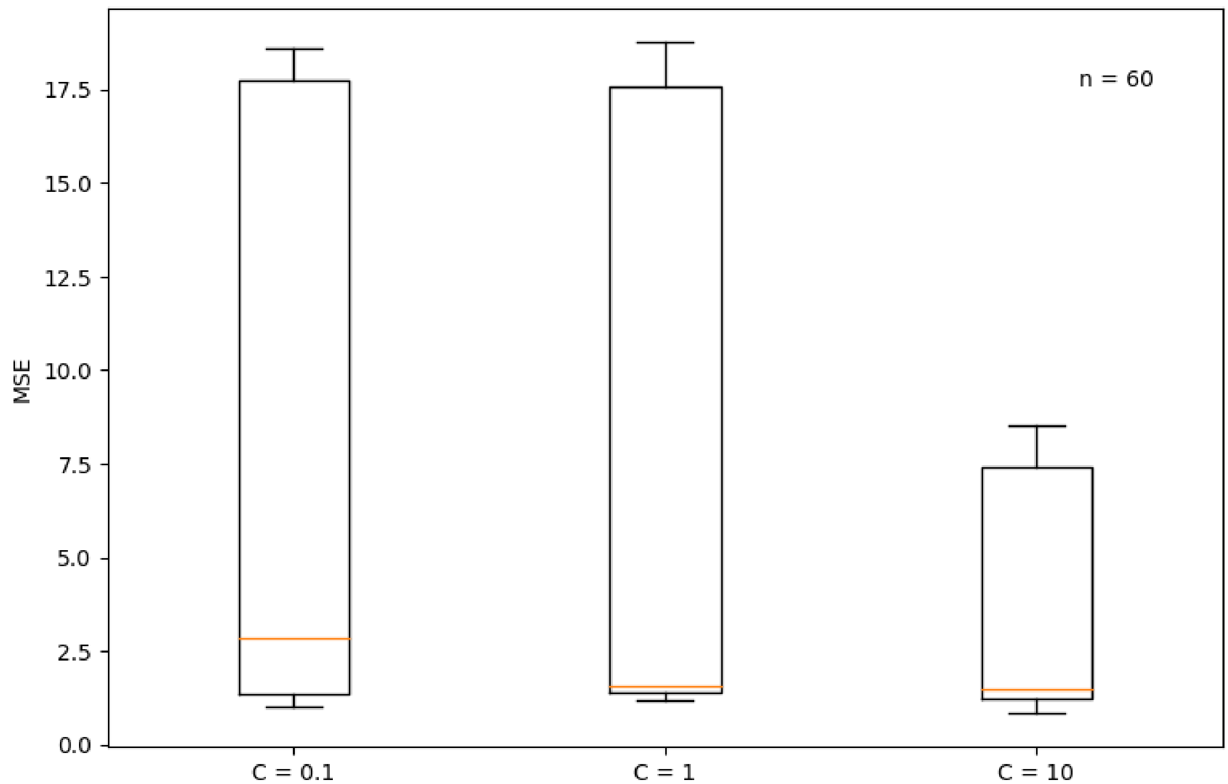
	0.1	1	10
Linear	1.2409509202453988(0.1514733839389112)20	1.4339570552147238(0.1582440669365465)20	1.4698466257668712(0.128996664983984)20
Poly	17.973098159509203(0.37099971874120485)20	17.87668711656442(0.4139239944361266)20	7.728282208588957(0.37424582731728356)20
rbf	2.8021165644171777(0.1911331194562696)20	1.455674846625767(0.18112365335855818)20	1.1054294478527606(0.12819177590460729)20

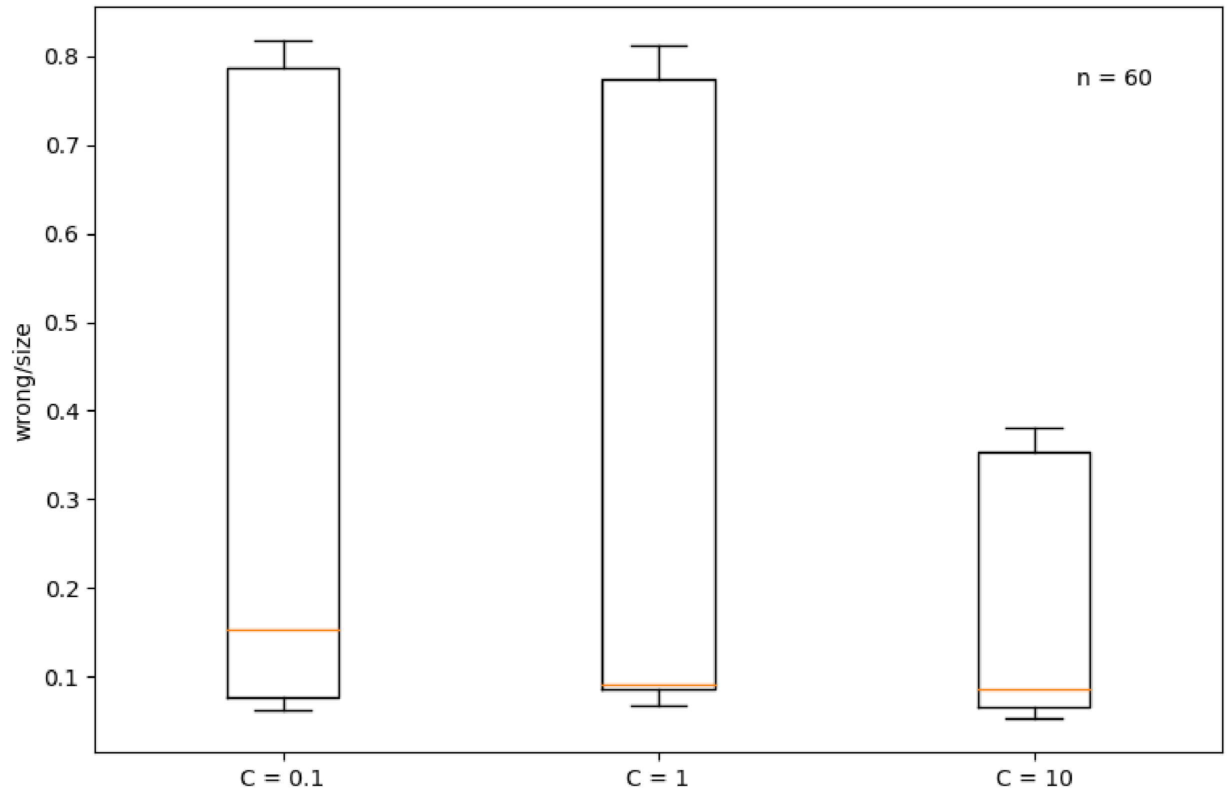
E = wrong/ total

	0.1	1	10
Linear	0.07306748466257669(0.004883292793826162)20	0.08957055214723926(0.006796727656251238)20	0.08819018404907976(0.006949125862286244)20
Poly	0.7948159509202454(0.012872629507248079)20	0.7864110429447853(0.014185722383040441)20	0.3611349693251534(0.012584402683054266)20

rbf	0.15125766871165644(0.007580637278996465)20	0.08429447852760737(0.008022266466677445)20	0.06236196319018405(0.005462951816281436)20

10. (0.5 points) Make a boxplot graph that plots testing error (vertical) as a function of the slack **C** (horizontal). Use average results across all kernels. Indicate **n** on your plot, where **n** is the number trials per boxplot. Don't forget to label your dimensions.





11. (0.5 points) What statistical test should you use to do comparisons between the values of **C** plotted in the previous question? Explain the reason for your choice. Consider how you selected testing and training sets and the skew of the data in the boxplots in your answer. *Note: Your boxplots will show you whether a distribution is skewed (and thus, not normal), but will not show you what the shape of each distribution. There are distributions that are not skewed, but are still not bell curves (normal distributions). It would be a good idea to look at the histograms of your distributions to decide which statistical test you should use.*

    If you have independent samples and data that doesn't follow a normal distribution use this: Mann–Whitney U test.

    Since my testing data is not normal distribution, it is discrete uniform distribution. I choose the same amount of example for each digit. Therefore, I should use Mann-Whitney U test.

12. (0.5 points) Give the p value reported by your test. Say what that p value means.

    MSE

    0.1 vs 1

MannwhitneyuResult(statistic=1595.5, pvalue=0.14214631632792601)

Only 0.14 chance that null hypothesis is true. That is, only 0.14 chance that the result of svm with c= 0.1 and c= 1 are no statistical different.

1 vs 10

MannwhitneyuResult(statistic=1291.0, pvalue=0.003804580749788896)

Only 0.0038 chance that null hypothesis is true. That is, only 0.0038 chance that the result of svm with c= 1 and c= 10 are no statistical different.

0.1 vs 10

MannwhitneyuResult(statistic=1250.5, pvalue=0.0019788877579050494)

Only 0.0019 chance that null hypothesis is true. That is, only 0.0019 chance that the result of svm with c= 0.1 and c= 10 are no statistical different.

E = wrong/size

0.1 vs 1

MannwhitneyuResult(statistic=1684.5, pvalue=0.2730374637771026)

Only 0.27 chance that null hypothesis is true. That is, only 0.27 chance that the result of svm with c= 0.1 and c= 1 are no statistical different.

1 vs 10

MannwhitneyuResult(statistic=1225.0, pvalue=0.0012814730309348512)

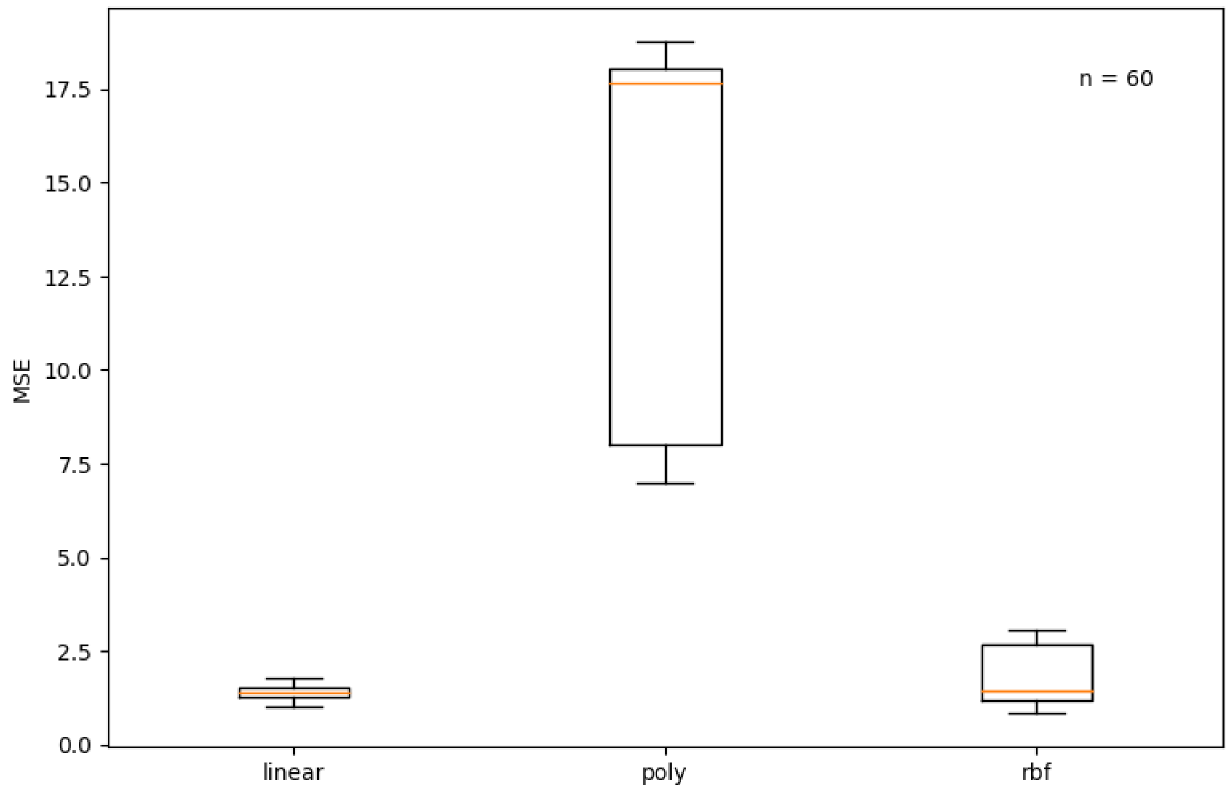
Only 0.00128 chance that null hypothesis is true. That is, only 0.00128 chance that the result of svm with c= 1 and c= 10 are no statistical different.

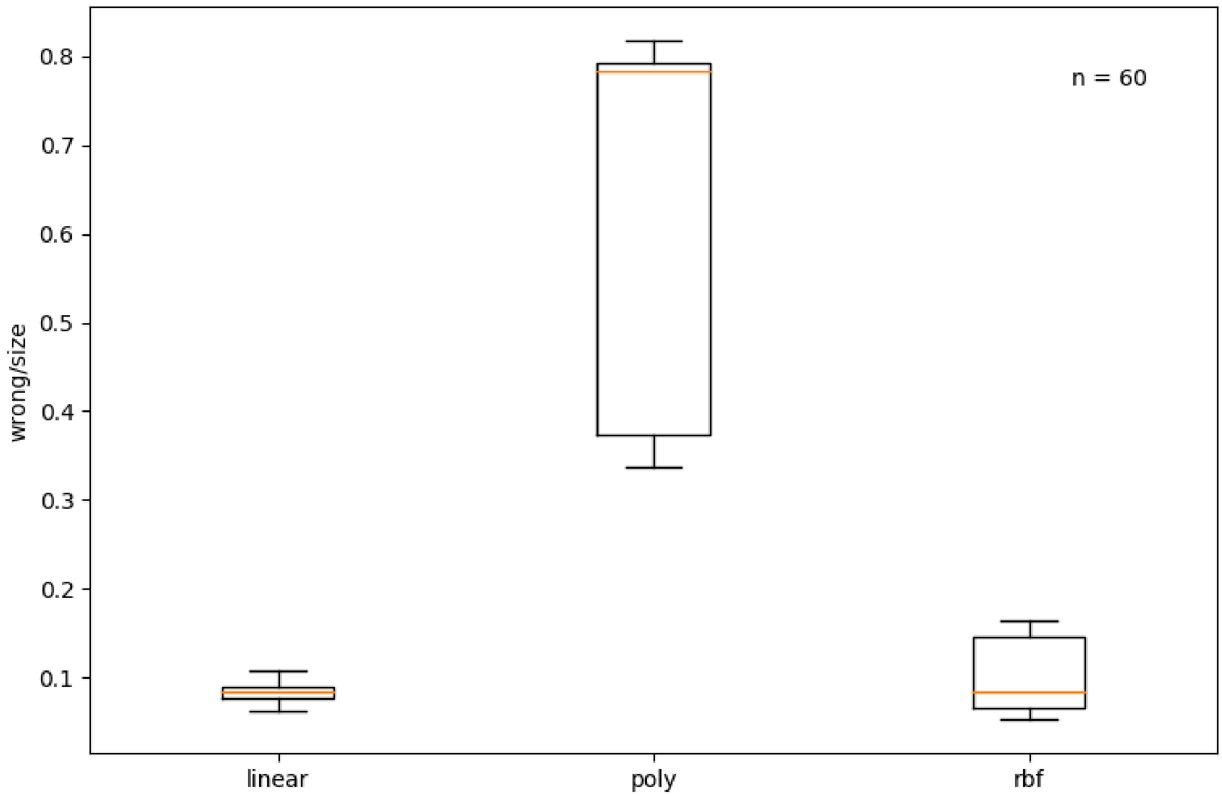
0.1 vs 10

MannwhitneyuResult(statistic=1224.5, pvalue=0.0012716558616631523)

Only 0.00127 chance that null hypothesis is true. That is, only 0.00127 chance that the result of svm with c= 0.1 and c= 10 are no statistical different.

13. (0.5 points) Make a boxplot graph that plots error (vertical) as a function of kernel choice. Average results across all values for C. Don't forget to indicate **n** on your plot, where **n** is the number trials per boxplot. Don't forget to label your dimensions.





14. (0.5 points) What statistical test should you use to determine whether the difference between the best and second best kernel is statistically significant? Explain the reason for your choice. Consider how you selected testing and training sets and the skew of the data in the boxplots in your answer.

    If you have independent samples and data that doesn't follow a normal distribution use this: Mann–Whitney U test.

    Since my testing data is not normal distribution, it is discrete uniform distribution. I choose the same amount of example for each digit. Therefore, I should use Mann-Whitney U test.

15. (0.5 points) What is the result of your statistical test? Is the difference between the best and second best value of **kernel** statistically significant?

    MSE

    linear vs poly

    MannwhitneyuResult(statistic=0.0, pvalue=1.7782854874923613e-21)

    linear vs rbf

MannwhitneyuResult(statistic=1566.5, pvalue=0.11067632821778534)

poly vs rbf

MannwhitneyuResult(statistic=0.0, pvalue=1.7782854874923613e-21)

The best two are linear and rbf.

Yes, the p value is only 0.11. Therefore, the result is statistically significant.

E= wrong/size

linear vs poly

MannwhitneyuResult(statistic=0.0, pvalue=1.7644150243261798e-21)

linear vs rbf

MannwhitneyuResult(statistic=1751.5, pvalue=0.4005276895797677)

poly vs rbf

MannwhitneyuResult(statistic=0.0, pvalue=1.7682882943213812e-21)

The best two are linear and rbf.

No, the p value is only 0.40. Therefore, the result might not be statistically significant. Only about half chance will reject null hypothesis.

16. (0.5 points) Is the combination of kernel and **C** that shows the best error in the table from the previous question the same combination that resulted from considering **C** and kernel independently? Which one do you believe?

    The combination of kernel and C is different from the combination that resulted from considering C and kernel independently. I believe the result from considering C and kernel independently because different C or kernel might have a significant difference of MSE.

    The combination of kernel and C is different from the combination that resulted from considering C and kernel independently. I believe the result from considering C and kernel independently because different C or kernel might have a significant difference of error(wrong/size).

Putting these results in context (0.5 point)

17. (0.5 points) Compare your results with the previous results for SVMs found on MNIST. What is the best kernel reported there? How does your best kernel do compared to that one? *Aside: A Gaussian Kernel is a Radial Basis Function kernel*

    The best result on MNIST is Virtual SVM, deg-9 poly, 2-pixel jittered with error 0.56.

    My best result is rbf with error(MSE) 1.1054294478527606.

    My best result is rbf with error(wrong/size) 0.06236196319018405

Showing us your data.

18. (0.5 points) Put the error from every individual trial into a single table, where the columns are labeled: error, **C** , kernel. Each row will list the error rate (on a scale of 0 to 1) for one trial, the value of **C** for that trial and the kernel for that trial.

MSE

	Linear 0.1	Linear 1	Linear 10	Poly 0.1	Poly 1	Poly 10	Rbf 0.1	Rbf 1	Rbf 10
Column1.1	[1.12515337423312 88	[1.32392638036809 8	[1.5	[18.6128834355828 23	[17.9558282085889 4	[7.75398773006135	[2.88343558282208 57	[1.68834355828220 86	[1.25460122699386 51
Column1.2	1.079141	1.304294	1.309202	17.42454	18.44356	7.158896	2.954601	1.376074	1.026994
Column1.3	1.388957	1.318405	1.330061	17.73067	17.78344	7.976074	2.78589	1.340491	0.892025
Column	1.422086	1.244785	1.560736	18.52638	18.01902	7.00184	2.431288	1.431288	0.983436

n1.4									
Column1.5	1.219018	1.300613	1.365031	18.01043	17.68037	7.984049	2.42638	1.304908	1.214724
Column1.6	1.44908	1.580368	1.483436	18.21288	17.37914	8.030061	2.911043	1.434356	0.826994
Column1.7	1	1.478528	1.760736	18.15092	17.82331	7.893865	2.991411	1.217178	1.08773
Column1.8	1.03865	1.482822	1.253988	18.06074	17.58098	8.042331	2.952147	1.244172	1.165031
Column1.9	1.517178	1.317791	1.311656	18.10613	18.02086	8.51227	2.741718	1.593252	0.977301
Column1.10	1.085276	1.259509	1.343558	17.64724	18.47975	7.621472	2.89816	1.319632	1.083436
Column1.11	1.368712	1.396319	1.46135	17.91472	17.97117	7.673006	3.011043	1.176687	1.118405
Column1.12	1.262577	1.377301	1.48773	18.08528	17.67914	7.482209	3.064417	1.446012	1.148466
Column1.13	1.354601	1.179141	1.417178	17.82761	17.55092	7.478528	2.642331	1.471166	1.01227
Column	1.236196	1.404908	1.707362	17.71472	17.45951	8.233742	2.776687	1.517178	1.225767

n1.14									
Column1.15	1.174233	1.733742	1.504294	18.0227	17.75337	7.840491	2.65092	1.482822	1.1
Column1.16	1.368098	1.66319	1.525767	17.00736	18.49693	7.408589	2.803067	1.36319	1.069325
Column1.17	1.216564	1.498773	1.470552	17.72699	17.37239	7.707975	2.790798	1.718405	1.233742
Column1.18	1.307362	1.628221	1.526994	18.10736	18.03926	7.360736	2.483436	1.412883	1.354601
Column1.19	1.053374	1.544785	1.509816	18.33252	17.28528	8.023313	2.849693	1.742331	1.150307
Column1.20	1.152760736196319]	1.641717791411043]	1.567484662576687 2]	18.23987730061349 7]	18.75950920245398 7]	7.382208588957055]	2.99386503067484 67]	1.83312883435582 83]	1.1834355828220 86]

Error = wrong/size

	Linear 0.1	Linear 1	Linear 10	Poly 0.1	Poly 1	Poly 10	Rbf 0.1	Rbf 1	Rbf 10

Column1.1	[0.06871165644171 78	[0.08650306748466258	[0.10736196319018405	[0.78957055214723 93	[0.77055214723926 38	[0.36196319018404 91	[0.14846625766871166	[0.06871165644171 78	[0.06748466257668712
Column1.2	0.071166	0.088957	0.082822	0.790184	0.790798	0.355215	0.158896	0.07546	0.066258
Column1.3	0.07362	0.082822	0.08773	0.792025	0.811656	0.364417	0.146012	0.077914	0.055828
Column1.4	0.077914	0.084049	0.085276	0.802454	0.769325	0.358896	0.13865	0.101227	0.060123
Column1.5	0.07362	0.101227	0.094479	0.78589	0.779755	0.33681	0.132515	0.095706	0.060736
Column1.6	0.071166	0.092025	0.084663	0.77546	0.78773	0.379755	0.152147	0.08589	0.065031
Column1.7	0.073006	0.092025	0.090184	0.81227	0.764417	0.372393	0.154601	0.082822	0.052761
Column1.8	0.074847	0.086503	0.088344	0.797546	0.789571	0.376687	0.152761	0.080368	0.064417
Column1.9	0.082822	0.095706	0.084049	0.771779	0.811043	0.357669	0.164417	0.085276	0.055828
Column1.10	0.065644	0.090798	0.089571	0.783436	0.798773	0.373006	0.152761	0.079141	0.054601

Column1.11	0.078528	0.085276	0.096933	0.790184	0.76135	0.352761	0.154601	0.089571	0.060736
Column1.12	0.06319	0.092025	0.081595	0.793865	0.771166	0.363804	0.149693	0.097546	0.066871
Column1.13	0.076074	0.08589	0.082209	0.815337	0.786503	0.36319	0.159509	0.080368	0.062577
Column1.14	0.070552	0.079755	0.081595	0.813497	0.792638	0.373006	0.152761	0.088344	0.060736
Column1.15	0.077301	0.092025	0.08589	0.790798	0.792638	0.352147	0.157055	0.081595	0.064417
Column1.16	0.066258	0.076074	0.080368	0.803067	0.789571	0.358896	0.147853	0.08589	0.060736
Column1.17	0.078528	0.096933	0.092025	0.787117	0.801227	0.381595	0.143558	0.086503	0.064417
Column1.18	0.070552	0.088344	0.079755	0.801227	0.779141	0.341718	0.144785	0.086503	0.07546
Column1.19	0.072393	0.090184	0.096319	0.783436	0.784049	0.357055	0.157669	0.084663	0.069325
Column1.20	0.075460122	0.104294478	0.092638036	0.817177914	0.796319018	0.341717791	0.156441717	0.072392638	0.058895705

	69938 65]	52760 736]	80981 596]	11042 94]	40490 79]	41104 297]	79141 106]	03680 981]	52147 239]